

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Marek Bečvář

Muon Time-of-Passage Estimation for Standalone MDT Trigger

Department of Software and Computer Science Education

Supervisor of the master thesis: RNDr. František Mráz, CSc.

Study programme: Computer Science - Artificial Intelligence

Prague 2026

I declare that I carried out this master thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor, RNDr. František Mráz, CSc., for our many discussions, his valuable advice, comments, and suggestions. Furthermore, I would like to express my gratitude to the ATLAS NGT Work Package 2.2 research team, namely Professor Verena Martinez Outschoorn and Rimsky Alejandro Rojas Caballero, PhD., for their active external supervision, assistance, and trust throughout the completion of this work. Finally, I would like to thank my family and my partner for their endless support through the entire process.

Title: Muon Time-of-Passage Estimation for Standalone MDT Trigger

Author: Bc. Marek Bečvář

Department: Department of Software and Computer Science Education

Supervisor: RNDr. František Mráz, CSc., Department of Software and Computer Science Education

Abstract: The Large Hadron Collider (LHC) at CERN uses Monitored Drift Tube (MDT) chambers to reconstruct muon trajectory and momentum. As part of a HL-LHC upgrade, raw MDT data will be integrated into a new Level-0 muon trigger system to improve its robustness. The trigger requires accurate muon time-of-passage estimation under severe background noise while operating within a strict sub-microsecond budget on FPGA hardware. This thesis evaluates multiple machine learning models for this task, including convolutional and graph neural networks, and a novel Line Graph Neural Network (LGNN). The LGNN reduces computational complexity and enables efficient FPGA implementation. A hardware-aware hyperparameter optimization pipeline is also developed to explore design constraints and select models that maximize efficiency without compromising predictive performance.

Keywords: Machine Learning, Graph Neural Networks, Trigger Systems, ATLAS, Large Hadron Collider

Název práce: Odhad času průletu muonu pro samostatný MDT trigger

Autor: Bc. Marek Bečvář

Katedra: Katedra softwaru a výuky informatiky

Vedoucí diplomové práce: RNDr. František Mráz, CSc., Katedra softwaru a výuky informatiky

Abstrakt: Velký hadronový urychlovač (LHC) v CERNu využívá komory s monitorovanými driftovými trubicemi (MDT) k rekonstrukci trajektorie a hybnosti mionů. V rámci modernizace urychlovače na HL-LHC budou data z MDT nově integrována do systému mionového triggeru, aby se zvýšila jeho robustnost. Tento nový systém vyžaduje přesný algoritmus, který v prostředí se šumem zvládne určit čas průletu mionu MDT detektorem. Algoritmus musí být implementovatelný na FPGA hardwaru a musí splňovat submikrosekundové výpočetní limity. Za tímto účelem práce navrhuje a vyhodnocuje několik modelů strojového učení, včetně konvolučních a grafových neuronových sítí a nové line-graf neuronové sítě, která snižuje celkovou výpočetní složitost a umožňuje efektivní implementaci na FPGA. Dále je vyvinuta procedura pro optimalizaci hyperparametrů, která zohledňuje hardwarové nároky modelů a umožňuje vybírat z architektur maximalizujících efektivitu a přesnost odhadu času průletu.

Klíčová slova: Strojové učení, Grafové neuronové sítě, Trigger systémy, ATLAS, Velký hadronový urychlovač

Contents

Introduction **7**

1 Background **9**
1.1 ATLAS Detector . . . 9
1.1.1 Detector overview . . . 9
1.1.2 Detector structure . . . 10
1.2 Monitored Drift Tubes . . . 11
1.2.1 Principle of a Drift Tube . . . 11
1.2.2 Chamber Geometry . . . 13
1.2.3 Muon Spectrometer Layout . . . 14
1.2.4 Trigger Chambers . . . 15
1.3 The HL-LHC and Phase-II TDAQ Upgrade . . . 16
1.3.1 The High-Luminosity LHC . . . 16
1.3.2 The Phase-II Level-0 Trigger . . . 17
1.4 Field-Programmable Gate Arrays . . . 17
1.4.1 ASIC versus FPGA . . . 18
1.4.2 Architecture . . . 18
1.4.3 Programming FPGAs and High-Level Synthesis . . . 20
1.5 Machine Learning Methods . . . 21
1.5.1 Fast ML . . . 21
1.5.2 ML Architectures . . . 22
1.5.3 Model Quantization . . . 28
1.5.4 Hardware-Aware Neural Architecture Search . . . 30

2 Methodology **31**
2.1 Dataset Introduction . . . 31
2.2 Baseline Method . . . 37
2.2.1 Random Sample Consensus (RANSAC) . . . 38
2.2.2 BCID-Conditioned RANSAC Strategy . . . 40
2.2.3 Baseline Evaluation and Limitations . . . 43
2.3 Convolutional Neural Networks . . . 47
2.3.1 Data Preparation . . . 48
2.3.2 Architectural Exploration . . . 50
2.3.3 Experimental Setup and Evaluation . . . 52
2.4 Graph Neural Networks . . . 59
2.4.1 Graph Building . . . 59
2.4.2 General Message-Passing Model . . . 65
2.4.3 Line Graph Architecture . . . 73
2.5 Model Optimization . . . 81
2.5.1 Multi-Objective Optimization via NSGA-II . . . 81

3 Experiments **86**
3.1 GNN Noise Robustness Evaluation . . . 86

Conclusion **90**

Bibliography **92**

List of Figures **96**

List of Tables **100**

A Code repository **101**

B NSGA-II Pareto Front Results **102**

Introduction

The currently developed High-Luminosity LHC (HL-LHC) represents a major upgrade of the Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN). Designed to achieve a significant increase in luminosity, the HL-LHC will be able to produce larger volumes of particle collision data, increasing the potential for particle physics research. In preparation for the operational environment, all four of the major LHC experiments are currently undergoing massive upgrades to their detectors and data acquisition systems.

The work of this thesis focuses specifically on the muon data acquisition trigger system of the *A Toroidal LHC Apparatus* (ATLAS) detector. ATLAS is one of two general-purpose detectors at the LHC, built to investigate the results of proton-proton and heavy-ion collisions. These high-energy collisions result in a scattering of subatomic particles, a subset of which is detected via highly specialized detector systems. Muons are subatomic elementary particles that, due to their momentum, can penetrate the inner detector layers of ATLAS, requiring a dedicated detector solution in the form of the Muon Spectrometer. This spectrometer tracks muon trajectories primarily through Monitored Drift Tube (MDT) chambers. Upgrading the triggering capabilities of the muon spectrometer to handle the increased particle scattering rates in the HL-LHC environment is a major part of the ATLAS trigger system upgrade.

This upgrade introduces a new Level-0 (L0) trigger system, which will, for the first time, integrate the raw event information collected by the MDT chambers directly into the trigger system. At present, the L0 Muon trigger in certain regions of the ATLAS detector relies exclusively on low-latency Resistive Plate Chambers (RPCs). To fully utilize the high-resolution spatial MDT data, the *Work Package 2.2* research team, as part of the *Next Generation Trigger (NGT)* project, is currently investigating various options to improve the L0 Muon trigger's operational robustness by applying custom machine learning models.

As illustrated in Figure 1, this proposed machine learning pipeline consists of three separate processing steps. The work within this thesis focuses specifically on the central step: developing the time-of-passage (BCID) estimation algorithm, which provides essential temporal information for the final momentum estimation.

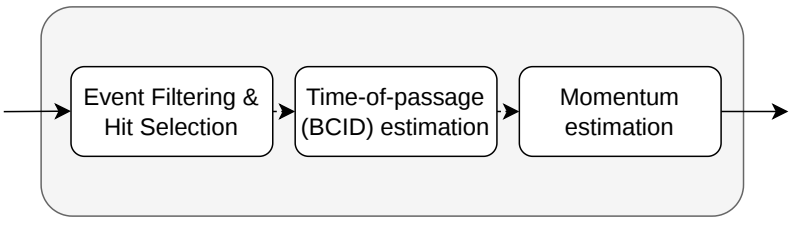


Figure 1 Illustration of the machine learning pipeline for processing the MDT hit data.

MDTs are gas-filled detector tubes. When a muon passes through them, it ionizes the gas along its trajectory, producing free electrons that drift towards a central anode wire and generate an electrical signal in the readout electronics. The time it takes for electrons to reach the wire is called the drift time, which is

strictly correlated with the closest distance between the passing muon's trajectory and the wire, defining the spatial drift radius. By combining the drift radii across multiple adjacent tubes, the true muon trajectory can be reconstructed.

However, the time measurement recorded by the readout electronics is simply the measurement of the arrival of the signal. To determine the actual drift time, the exact time-of-passage of the muon (or the identifier of the particle collision that produced the muon, BCID) must be estimated and subtracted. If this estimation is inaccurate, the resulting computed drift radii are incorrect, which degrades the quality of all subsequent trajectory and momentum estimation steps. Consequently, time-of-passage estimation requires an algorithm with robust predictive performance in an operational environment with background noise. Furthermore, the estimation algorithm must be implementable on the low-level L0 Muon trigger hardware and respect the strict computational restrictions and sub-microsecond inference latencies.

The primary objective of this thesis is to propose, develop, and evaluate a robust time-of-passage estimation algorithm. To achieve this, the work first develops an analytical baseline method that will serve as a benchmark for subsequent evaluations. Secondly, the thesis introduces and optimizes machine learning models suitable for processing the detector event data while respecting the computational constraints of the targeted hardware. Lastly, the thesis evaluates the noise robustness of the proposed models under varying levels of background noise.

The thesis has the following structure. Chapter 1 introduces the ATLAS detector operational environment, specifically focusing on the Monitored Drift Tubes, the High-Luminosity LHC upgrades, and the target hardware of the Level-0 trigger system. Lastly, this chapter concludes with an introduction to relevant machine learning concepts. Chapter 2 describes the methodology of this work, starting with the introduction of data and the definition of an analytical baseline algorithm. It then follows the chronological order of the evaluated machine learning models, moving from Convolutional Neural Networks to Graph Neural Networks, and finally defining a novel optimized architecture. Chapter 3 evaluates a selection of the best-performing models against detector events with artificial background noise. The Conclusion summarizes the research process, the results of this work, and outlines possible future steps. Finally, the Attachments provide access to the repository containing the source code A, as well as the complete results of the NSGA-II hyperparameter optimization experiment B.

MetaCentrum Acknowledgement

Thorough architectural evaluations and large-scale distributed NSGA-II experiments were made possible by using the computational cluster available to Czech students.

Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

1 Background

1.1 ATLAS Detector

The ATLAS (*A Toroidal LHC ApparatuS*) detector is one of the four major experiments located at the Large Hadron Collider (LHC) at CERN [1]. The LHC is the largest and most powerful particle accelerator in the world. It is designed to circulate up to 2808 bunches of 10^{11} protons (p) per beam, providing experimental sites with 40 million proton-proton collisions per second. The interval between these collisions is exactly **25 ns** and is formally referred to as a **bunch-crossing**. Additionally, the LHC can collide heavy ions (A), such as lead nuclei. ATLAS is one of two general-purpose detectors at CERN, alongside CMS, built to specifically probe pp and AA collisions. This section provides a general overview and describes the high-level structure of the ATLAS detector.

1.1.1 Detector overview

Before a detailed description of the specific sub-systems, it is necessary to define the standard ATLAS coordinate system used throughout this work.

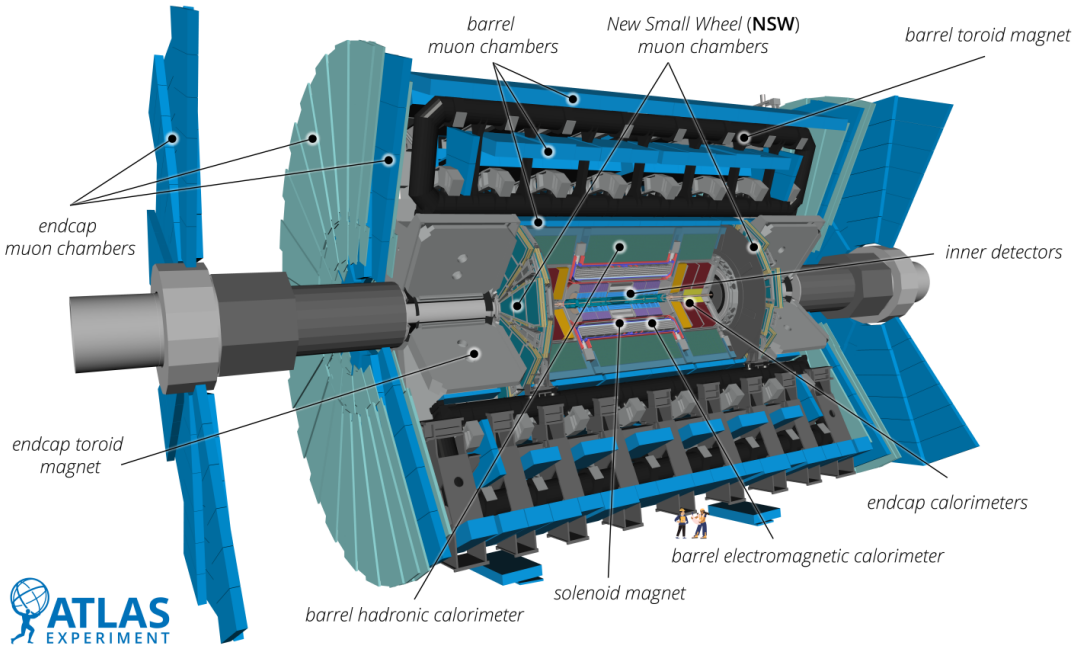


Figure 1.1 Full ATLAS detector schematic of the Run 3 configuration [2].

The detector is nominally forward-backward symmetric with respect to the interaction point, which serves as the origin of the detector's coordinate system [1]. The beamline passing through the detector defines the z-axis, while the x-y plane is transverse to it. The x-axis is defined as pointing from the origin interaction point to the center of the LHC ring, and the y-axis is defined as pointing upward.

Furthermore, spatial directions are commonly described using the azimuthal angle ϕ, measured around the beam axis, and the polar angle θ, measured from

the beam axis. However, in particle physics, the polar angle is more frequently expressed in terms of *pseudorapidity*, defined as $\eta = -\ln \tan \theta/2$. Pseudorapidity describes how far a particle is tilted from the x-y plane and is often used when describing the working ranges of various sub-detectors.

The detector is split into two sides along the z-axis: Side A (positive z) and Side C (negative z). As illustrated in Figure 1.1, the sub-detectors are divided into two distinct geometric configurations: defined as the **barrel** and two **end-caps**. The barrel forms the main cylindrical body around the interaction point, whereas the end-caps are disk-shaped detector structures placed at both ends of the barrel, providing detector coverage of particles at high pseudorapidity.

1.1.2 Detector structure

The whole detector consists of several cylindrical layers of sub-detectors and toroidal magnets around the interaction point, designed to measure different properties of particles emerging from the interaction point.

The inner-most layer is the inner tracking detector, surrounded by a superconducting solenoid. Moving further from the interaction point, there are electromagnetic and hadronic calorimeters, which also act as a shield for the muon spectrometer, incorporating large superconducting toroidal magnets [3].

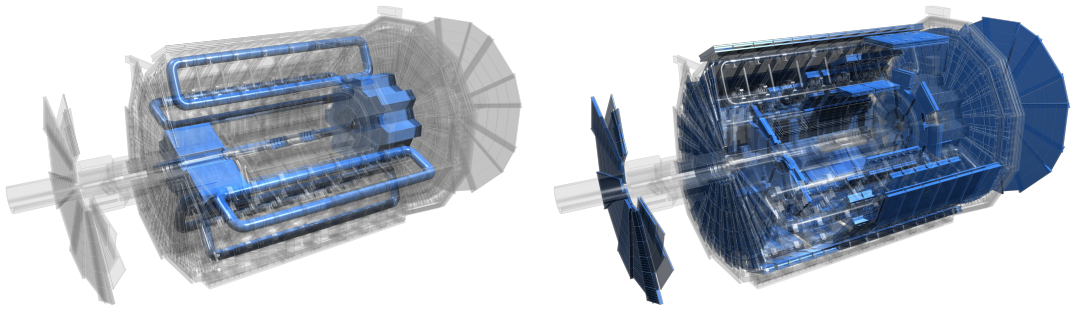


Figure 1.2 ATLAS schematics with highlighted magnet system (left) and Muon Spectrometer systems (right) [4].

Inner Detector The Inner Detector system provides charged-particle tracking in the pseudorapidity range $|\eta| < 2.5$, using the silicon Pixel detectors and Semiconductor Trackers.

Calorimeter systems Farther from the interaction point, **calorimeter systems** cover a wider pseudorapidity range $|\eta| < 4.9$. Calorimeters of different types, such as lead/Liquid Argon or steel/scintillator-tile calorimeters, complement each other in their coverage, optimized for electromagnetic and hadronic energy measurements. The majority of particles deposit their energy and are stopped in the calorimeters. However, **muons** pass through the calorimeter, which is why ATLAS deploys a completely separate system to study muons: the Muon Spectrometer.

Magnet system The ATLAS magnet system (highlighted on the left in Figure 1.2) plays a significant role in the detection and study of charged particles. A large system of magnets provides magnetic fields ranging from 2 to 6 Tesla in different parts of the ATLAS detector. The bending of charged-particle trajectories in this magnetic field is crucial to understanding the particle momentum and studying their properties. ATLAS utilizes a central solenoid magnet near the beamline to bend the trajectories of charged particles, allowing for momentum measurement in the Inner Detector. The two end-cap toroid magnets and 8 separate barrel toroid coils provide a magnetic field for the Muon Spectrometer.

Muon Spectrometer The most interesting part of the detector for our project will be the **Muon Spectrometer** (highlighted on the right in Figure 1.2). Muons are subatomic elementary particles similar to electrons but about 200 times heavier, which allows them to pass through the ATLAS calorimeters. In the current setup, the Muon Spectrometer system houses both the trigger and tracking chambers, providing high spatial resolution. These tracking chambers measure the deflection of muons in a magnetic field generated by the superconducting toroidal magnets. The task of the trigger system is to provide a signal to start the data acquisition system, effectively deciding which collision events to save and which to discard.

In the barrel region, three stations, each consisting of layers of **Monitored Drift Tubes (MDTs)**, are used for the precision tracking task. In the end-caps, the combination of two MDT stations and the **New Small Wheel (NSW)** detector, which replaced the innermost MDT station in the recent update, is used for tracking. The muon trigger system currently utilizes **Resistive Plate Chambers (RPCs)** covering the barrel region and the **Thin Gap Chambers (TGCs)** in the end-cap regions, together with the NSW, which performs trigger and tracking tasks simultaneously.

1.2 Monitored Drift Tubes

As introduced in the detector overview, the primary role of the Muon Spectrometer is to provide precise momentum (p_T) measurements of muons escaping the calorimeters. To achieve this, the spectrometer heavily relies on Monitored Drift Tube chambers (MDTs) [1]. MDTs serve as the precision tracking chambers in both the barrel and end-cap regions, providing high-resolution spatial measurements that enable accurate reconstruction of muon trajectories through the magnetic field. The Muon Spectrometer consists of 1172 MDT chambers with 354 000 drift tubes as of LHC Run 2 [5].

1.2.1 Principle of a Drift Tube

The fundamental building blocks of MDTs are individual drift tubes. Each MDT is a pressurized aluminium tube with an outer diameter of 29.97 mm and lengths ranging from 1 m to 6 m [1, 5]. A gas mixture of 93 % Ar and 7 % CO_2 at 3 bar pressure is fed through the tube at a rate of roughly one tube volume per day. A 50 µm tungsten-rhenium wire at a potential of 3080 V is fixed precisely in the center of each tube, held at each end by an end-plug.

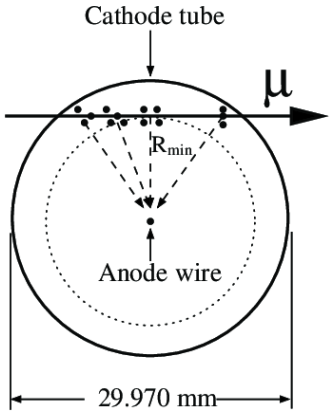


Figure 1.3 MDT tube cross section with an ionizing track [6].

When a muon passes through the tube, it ionizes the gas mixture along its trajectory (illustrated in Figure 1.3). The free electrons then drift toward the central wire, under the influence of the electric field. The intense acceleration of electrons near the wire causes an avalanche effect, amplifying the signal so that the electronics at the end of the tube can read it out.

The signal is read out on the front-end electronics. While the full complexity of analog signal processing is beyond the scope of this work, it is important to note that the signal is routed to a Time-to-Digital Converter (TDC), where the leading and trailing edges of the signal are stored in a buffer along with the tube identification. The TDC operates with a digitization binning of 0.78 ns. This binning introduces a known average quantization error of about 5 µm in the drift distance measurement. The signal arrival time is relative to a specific **bunch-crossing** time (bunch-crossing identifier). This time is measured in 25 ns ticks and is synchronized with an LHC global 40 MHz clock.

The electron drift velocity is highly dependent on the properties of the gas mixture, as well as the environmental factors such as temperature, pressure, and the strength of the local magnetic field. Due to these factors, an online calibration is necessary. In general, the **maximum drift time** of electrons in the drift tube is about **720 ns**.

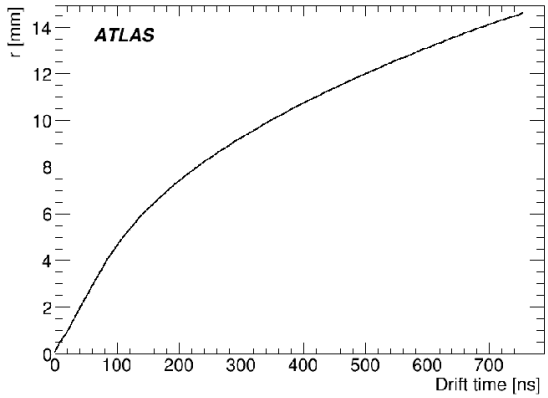


Figure 1.4 MDT $R(t)$ function for Ar/CO_2 gas mixture [5].

The most important value that we want from each tube is the actual **drift time**. The raw time for each hit can be converted to drift time by subtracting the

time of the bunch-crossing and the tube timing offset. This offset is composed of multiple parts: the time-of-flight of the particle from the interaction point, signal propagation time along the wire in the tube, and various fixed electronics and cable latencies. This delay is often referred to as the t_0 offset [5]. By subtracting all offsets from the signal time of arrival, we obtain the **drift time**, which can later be converted to the drift radius (*drift circles*) using the function $R(t)$, the shape of which can be seen in Figure 1.4.

1.2.2 Chamber Geometry

As established in the previous section, a single drift tube provides a highly precise 1-dimensional measurement of a muon's passage. However, the overall goal of the Muon Spectrometer is to determine the momentum of the particle, which is achieved by observing its curved trajectory through the magnetic field. Because a single drift radius contains no directional information and suffers from a left-right ambiguity (it is impossible to know which side of the wire the muon passed on), multiple drift tubes must be mounted together to form a larger structure known as an **MDT chamber**.

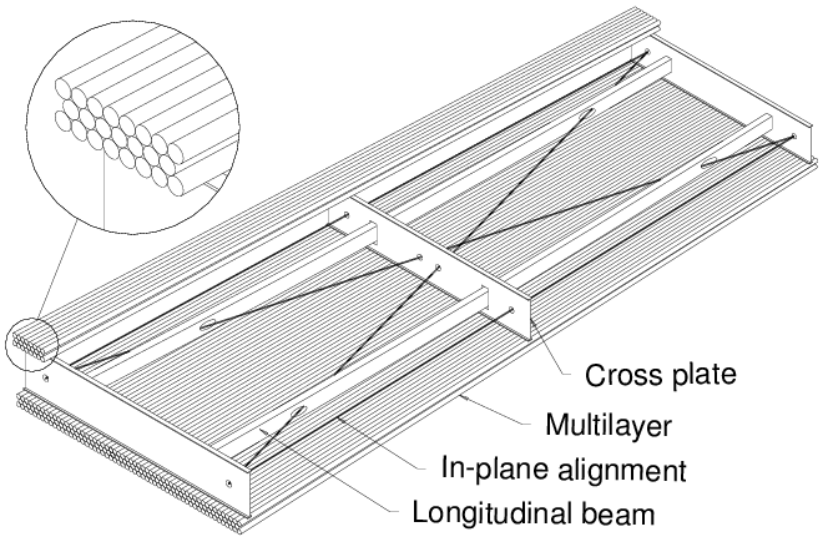


Figure 1.5 Structure of an MDT chamber [7].

The standard layout of an MDT chamber (as seen in Figure 1.5) consists of two distinct groups of tubes called **multilayers**, which are separated by a mechanical spacer frame. To optimize tracking performance across different regions of the spectrometer, the innermost chambers consist of multilayers with four layers of staggered drift tubes, while the middle and outer chambers utilize three layers of tubes per multilayer [1]. When a muon traverses the chamber, it generates a signal in several tubes. Pattern recognition algorithms can then analyze these drift circles to reconstruct a straight **track segment**, defined as the optimal tangent line to the measured radii.

Optical Alignment The spatial resolution of a single drift tube is exceptionally high (roughly 80 µm). However, MDT chambers are massive structures that must withstand the effects of environmental factors such as thermal deformation and

mechanical stress. Any deformations to the structure or tubes would severely degrade the quality of the final momentum measurement.

For this reason, the chambers implement a sophisticated internal alignment system. The system consists of a set of four optical alignment rays for continuous monitoring. The two rays parallel to the tube direction and two in the diagonal direction can be seen in Figure 1.5. This system is designed to operate during production and can detect deformation of a few µm. Further precision alignment systems work on the outside of the chambers, relating the position between neighboring chambers. This optical system continuously monitors the relative positions of the chambers to ensure that the required tracking accuracy is maintained throughout the spectrometer.

1.2.3 Muon Spectrometer Layout

To allow for the measurement of the muon's path and thus calculate its momentum, the system needs to be able to connect individual track segments from multiple chambers through the magnetic field to reconstruct the full trajectory of the particle. To achieve this, the MDT chambers are arranged into layers of distinct **tracking stations**. In the barrel region, these stations form three concentric cylindrical layers around the interaction point. A special layout can be observed in the end-caps, where even the construction of the MDT chambers was altered to a trapezoidal shape, forming large vertical wheels [1, 3].

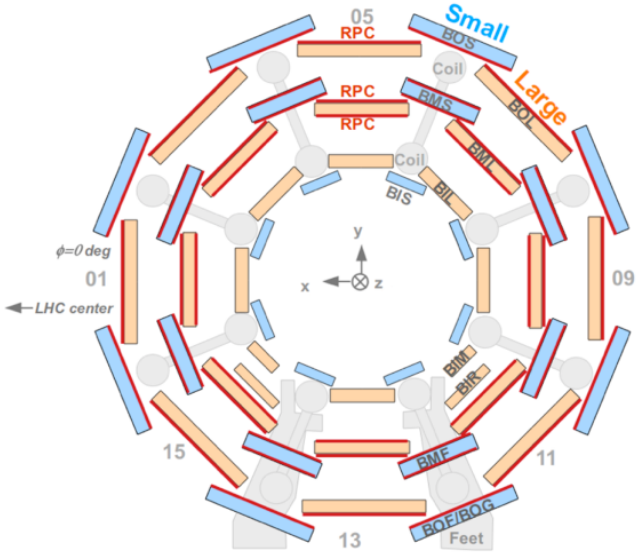


Figure 1.6 ATLAS barrel cross section perpendicular to the beam axis with MDT stations highlighted, small sectors shown in blue and large in orange, the RPC layers are shown as red outlines of MDT chambers [8].

In the barrel, the Muon Spectrometer consists of three concentric tracking layers around the beam axis at distances of approximately 5 m, 7.5 m and 10 m. The symmetry of the toroid magnets is reflected in the symmetrical structure of the station layout. The cross section of the layout transverse to the beam axis can be seen in Figure 1.6. In ATLAS terminology, the names of these stations

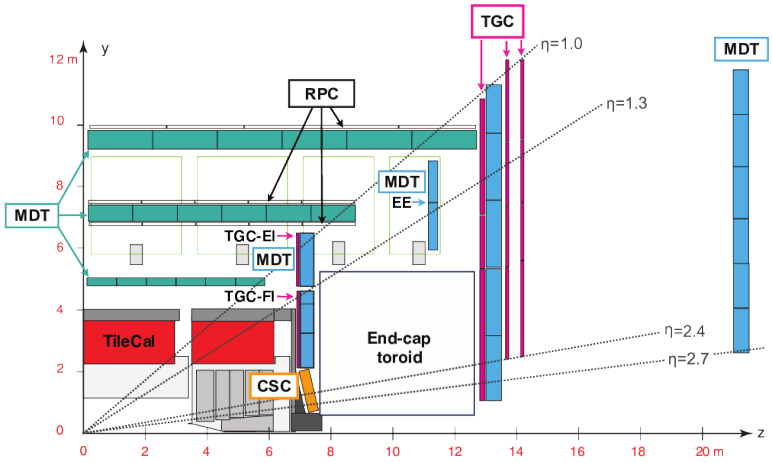


Figure 1.7 ATLAS muon system cross section in the beam axis plane, MDT chambers in barrel highlighted in green, end-cap chambers in blue, RPC and TGC trigger chambers are highlighted alongside MDTs, pseudorapidity lines are present with η values [9].

follow their placement in the inner, middle, or outer stations as **Barrel Inner (BI)**, **Barrel Middle (BM)**, and **Barrel Outer (BO)**, respectively. Special cases can occur near the feet or rails of the ATLAS detector, leading to acronyms such as BOF.

To provide complete ϕ coverage while accommodating the massive barrel toroid magnet coils, the chambers in each layer are divided into 16 distinct sectors. Eight **small (S)** and eight **large (L)** sectors alternate to cover the whole range.

- **Large sectors** (e.g., BIL, BML, BOL) are positioned in the open spaces between the magnet coils.
- **Small sectors** (e.g., BIS, BMS, BOS) are mounted closer to, or directly upon, the structures of the magnet coils themselves.

This structure also allows the sectors to slightly overlap, minimizing the possible "dead zones" of the detector and ensuring that the muon traveling from the interaction point intersects as many tracking layers as possible.

By requiring a muon to leave a reconstructed track segment in the inner, middle, and outer stations, the spectrometer provides three highly precise spatial coordinates. The **sagitta** (defined as the maximum deviation of the curved track from a straight line) in these three segments is induced by the bending power of the magnetic field. Measuring this sagitta is what ultimately allows the system to calculate the muon's momentum.

1.2.4 Trigger Chambers

A fundamental requirement of the Muon Spectrometer is the ability to rapidly trigger data acquisition on passing muons. Although precise MDTs provide exceptional spatial resolution, they are intrinsically slow due to a maximum electron drift time of roughly 720 ns. Consequently, when an MDT registers a muon hit, it is far too late to identify the exact 25 ns bunch-crossing from which the muon originated. For this reason, precision MDTs are paired with fast

trigger chambers capable of providing temporal information within a few tens of nanoseconds after particle passage [1, 10].

In the barrel region, **Resistive Plate Chambers (RPCs)** are utilized, while **Thin Gap Chambers (TGCs)** are used in the end-caps (both are highlighted in Figure 1.6 and Figure 1.7). Both chamber types were designed to reliably identify the bunch-crossing with a probability of $\geq 99\%$, delivering the signal with a spread of 15 ns to 25 ns. Alongside this, they also measure track coordinates in both the bending (η) and non-bending (ϕ) planes, providing the orthogonal azimuthal coordinate that MDTs do not measure.

In the current ATLAS architecture, these chambers serve a key dual purpose. First, they identify the bunch-crossing that originated the muon, providing the essential collision time and t_0 reference (as explained in Section 1.2.1) required to calculate the drift radii of the MDT. Second, they perform a preliminary momentum estimation for the trigger system. All of this information also serves the definition of the Region of Interest (RoI), utilized in later stages for the offline trajectory reconstruction using specific MDTs.

1.3 The HL-LHC and Phase-II TDAQ Upgrade

In the previous section, we introduced the basic architecture of the ATLAS detector, focusing primarily on the current state of the Muon Spectrometer. In this section, we transition to the planned future of the Large Hadron Collider and the necessary upgrades for the muon system.

1.3.1 The High-Luminosity LHC

The High-Luminosity Large Hadron Collider (*HL-LHC*) is a major upgrade project that aims to transform the current accelerator to maximize its performance. This upgrade will enable better understanding and search for rare physics phenomena, such as precision measurements of the Higgs boson, first discovered at LHC in 2012.

Luminosity and Pile-up Instantaneous luminosity (expressed in $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$) is a key indicator of accelerator's performance, indicating the potential number of collisions per surface unit over a period of time. It is worth introducing the integrated luminosity as well (measured in inverse femtobarns fb^{-1}), which corresponds to the potential number of collisions over a given period; one inverse femtobarn is equivalent to 100 trillion (10^{14}) potential collisions.

The original design of the LHC was to deliver instantaneous luminosity of $1 \times 10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$, which was later exceeded in 2016. In the same year, the average pile-up (defined as the number of simultaneous pp collisions within a single bunch-crossing, denoted by $\langle\mu\rangle$) was $\langle\mu\rangle = 24.9$ inelastic pp collisions [11]. At the end of its current operational period in 2025, the LHC is expected to produce roughly $500\,\mathrm{fb}^{-1}$ of data during its lifetime.

During the Long Shutdown 3 of the accelerators, beginning in mid-2026, have plans to transform LHC into the HL-LHC, which should be operational in mid-2030. Later, peak luminosities are expected to be pushed up to 7 times higher than what was originally designed for the LHC. The detector will face an average

pile-up of up to $\langle\mu\rangle = 200$. To support future physics goals, the HL-LHC is projected to produce more than $250\,\text{fb}^{-1}$ of data per year, ultimately collecting up to $4000\,\text{fb}^{-1}$ during its operational lifetime.

1.3.2 The Phase-II Level-0 Trigger

The precision MDT chambers are robust enough to maintain their high-quality track reconstruction under the increased hit rates of the HL-LHC. However, the increased pile-up threatens the viability of the current trigger designs, requiring significant upgrades to maintain acceptable trigger rates at the required momentum thresholds.

The HL-LHC Trigger Requirements The current plans for the ATLAS Phase-II trigger system foresee the introduction of a new single-level hardware trigger: Level-0 Trigger (L0 Trigger). This trigger will be based on muon and calorimeter data with a maximum latency of 10 µs to allow the introduction of more complex trigger algorithms [10, 11]. Furthermore, the L0 Trigger will send events to the higher-level trigger at 1 MHz.

These rate requirements force replacement of a large portion of the current trigger and readout electronics, as the legacy system cannot handle the required 1 MHz rates. Completely off-detector **Field-Programmable Gate Arrays (FPGAs)** will be used instead of the current on-detector integrated circuits, making the trigger more robust and flexible (FPGA architecture will be discussed in Section 1.4).

The Muon L0 Trigger For the goals of HL-LHC, the muon system should be able to trigger the data acquisition efficiently on single muons at a maximum rate of 40 kHz for a threshold of $p_T > 20\,\text{GeV}$, with even lower thresholds of $p_T > 4\,\text{GeV}$ to be viable for special cases (such as *multi-muons*) [10, 11].

As established in Section 1.2.4, the legacy hardware trigger relies entirely on the fast RPC and TGC trigger chambers to perform a preliminary momentum estimation and the classification of the Region of Interest (RoI). Although incredibly fast, their spatial resolution is relatively coarse. Because momentum measurement relies on the calculation of the sagitta of the muon's track, a coarse spatial resolution leads to degraded momentum resolution. Consequently, that is the limiting factor in lowering the rate of **fake triggers** caused by low-momentum muons.

To remedy this, the Phase-II L0 Muon Trigger will incorporate data from the precision MDT chambers directly into the hardware trigger decision. The new front-end readout electronics will allow for **triggerless MDT readout**, making their high-precision measurements continuously available for the trigger decision.

1.4 Field-Programmable Gate Arrays

As established in the last section, the Phase-II trigger upgrade requires replacing a major portion of the electronics. Most notably, a significant amount of **Application-Specific Integrated Circuits (ASICs)** is being replaced by

off-detector **Field-Programmable Gate Arrays (FPGAs)**. In this section, we will evaluate their advantages and disadvantages compared to the legacy ASIC architecture, introduce the fundamental architecture of FPGAs, and demonstrate how they enable the computing required for HL-LHC going forward.

Unlike common processing units, such as Central Processing Units (CPUs) or Graphics Processing Units (GPUs), which execute software instructions on fixed silicon hardware, an FPGA is an electrically programmable integrated circuit that the user can modify to perform a specific task [12]. In comparison to pre-fabricated circuit boards, the FPGA system can actually reconfigure its internal circuitry. After configuration, they allow for near fixed-time processing of the programmed task.

1.4.1 ASIC versus FPGA

As previously established, a significant portion of the legacy ASICs in the ATLAS detector is being replaced with modern FPGA boards. Consequently, it is important to understand the trade-offs between these two architectures.

While this upgrade is a vast improvement in terms of trigger capability, it comes at a hardware cost. FPGAs are substantially slower than ASICs, having speed performance roughly 3 to 4 times slower than that of the legacy system. FPGAs also consume approximately 10 times as much energy and require more than 20 times the chip area. These hardware disadvantages arise directly from the ability of FPGAs to be reconfigured [12].

Despite these performance shortcomings, FPGAs introduce large-scale flexibility to the ATLAS detector trigger system. While ASICs were basically impossible to reuse for anything other than what they were originally manufactured for, an FPGA can be reconfigured and reprogrammed in seconds. The less specialized architecture of FPGAs also comes as a cost-saving factor, as it avoids the engineering and manufacturing costs associated with the fabrication of custom ASICs.

The ability to reconfigure the hardware at will is the definitive advantage of the Phase-II upgrade. It allows researchers to leverage new algorithmic advancements directly in detector systems without the need to manufacture new integrated circuitry. This flexibility ensures that the trigger system will remain up to date for at least the 10-year lifespan of the foreseen HL-LHC experiments.

1.4.2 Architecture

The main advantage of FPGAs lies in their parallel architecture. They allow developers to program custom hardware pipelines designed to process input data in a highly specific way. The internal architecture resembles a grid of tiles, as seen in the illustration in Figure 1.8. It contains logic elements surrounded by programmable routing fabric, both of which can be controlled by the user [12]. When the FPGA is programmed, the routing fabric is configured in such a way that the software program is implemented in hardware.

Configurable Logic Blocks (CLBs) Logic blocks are the fundamental computational units of FPGAs and can be of multiple types. Their purpose is to

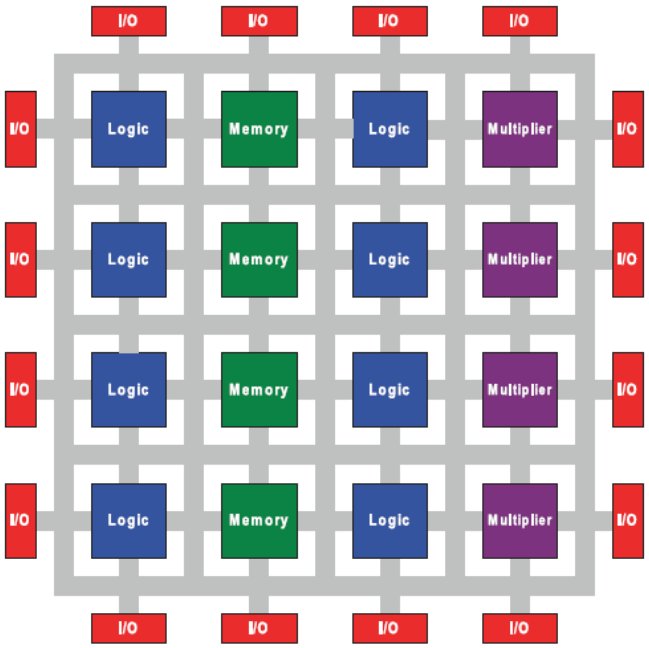


Figure 1.8 Illustration of basic FPGA architecture [12].

provide either the computation or the storage used in the computation. A CLB is composed of **Basic Logic Elements (BLEs)**, which consist of three main components: **look-up tables (LUTs)**, **flip-flops**, and **multiplexers**. Such logic blocks are often referred to as *the soft logic fabric* of the FPGA.

A K-input LUT is essentially a small memory array used to store data, typically used to evaluate logic functions of K input variables. The flip-flop is another primitive memory element, often used to store just a single bit of information, utilized to store output states between clock cycles. However, in modern FPGAs, it is common for general-purpose blocks to also include higher-level functions, such as primitive arithmetic functionality or memory.

By combining numerous simple CLBs in the routing fabric, the user can construct vastly complex digital circuits.

Modern Heterogeneous Elements Although early FPGAs consisted of a purely homogeneous array of soft-logic programmable blocks, modern architecture is commonly heterogeneous.

Constructing complex arithmetic operations or large memory buffers out of basic CLBs is possible, but inefficient. The reprogrammable nature of soft logic blocks consumes excessive power and physical chip area. To fix this, modern FPGAs embed highly specialized blocks (referred to as *hard logic*) directly into the matrix of tiles alongside the CLBs. The most critical additions to modern FPGAs are multi-bit Block RAM (BRAM) modules for fast data storage and Digital Signal Processors (DSPs) optimized for executing large numbers of mathematical operations.

1.4.3 Programming FPGAs and High-Level Synthesis

While the parallel architecture of an FPGA provides substantial performance benefits, it introduces new complexity in terms of its programming paradigm compared to traditional higher-level programming languages. They are traditionally programmed using **Hardware Description Languages (HDLs)**, such as Verilog and VHDL.

Programming in such languages involves an explicit definition of the behavior of the hardware, such as logic gates and data registers. Although this allows for the most accurate optimization, composing complex algorithms purely in HDL requires specialized expertise and is incredibly time-consuming and error-prone.

High-Level Synthesis (HLS) To raise the level of abstraction from HDL to modern, approachable higher-level programming languages, the industry has widely adopted the use of **High-Level Synthesis (HLS)** [13].

Such compilers allow the use of high-level languages like `C` or `C++`. The HLS tools then take this algorithm description, the library of allowed components, and the specific design constraints and synthesize the corresponding low-level HDL, effectively translating the software logic into physical electrical circuitry.

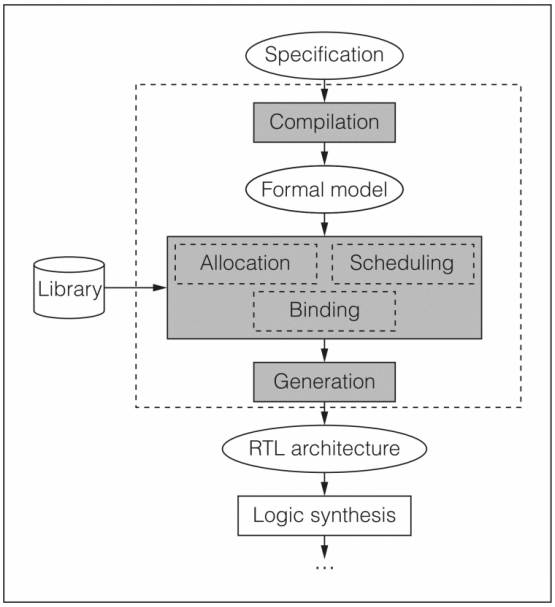


Figure 1.9 HLS design steps [13], RTL (or *register-transfer level*) corresponds to a level of design abstraction for synchronous digital circuits used in HDL.

This modernization in the FPGA architecture is a vital step, allowing physicists and computer scientists to rapidly develop and deploy algorithms without the need to manage the underlying hardware fabric manually. It opens the door for experimenting with more complex pattern recognition algorithms, such as the deployment of **Machine Learning** models in the trigger system.

Deploying Machine Learning to Hardware The abstraction provided by HLS enables developers to deploy complex algorithms on FPGA hardware. However, the efficient translation of trained machine learning models from standard

Python frameworks into `C++` and then into HDL still presents a significant engineering challenge.

Focusing specifically on particle physics and the application of machine learning techniques to a low-latency FPGA environment, an open-source HLS-based compiler package called **`hls4ml`** has been developed [14].

This package works as a drop-in step after training and possible compression of machine learning models using standard means, with frameworks such as `Keras` or `PyTorch`. After the model preparation is complete, the `hls4ml` package translates the model into HLS-compatible `C++` code, while optimizing metrics such as latency and resource usage.

During translation, the package precomputes complex activation functions into on-chip memory and optimizes matrix multiplication to fit efficiently within the FPGA's available DSPs and BRAM.

1.5 Machine Learning Methods

Machine Learning (ML) provides exceptionally powerful pattern-recognition algorithms with the critical ability to generalize to previously unseen data. This capability is a major driving force behind the widespread adoption of ML across various domains. Researchers at CERN already utilize advanced ML models at various stages, such as offline data processing or higher-level triggers at ATLAS.

Crucially, this thesis focuses on the deployment of ML models to one of the most performance-restrictive environments at CERN: the ATLAS Level-0 trigger system. Before delving deeper into the various solutions evaluated in this research, this section first introduces the foundational ML concepts, relevant network architectures, and optimization techniques required to deploy such models within the strict FPGA environment.

1.5.1 Fast ML

As the experiments upgrade to higher luminosities, generating unprecedented amounts of data, leveraging the power of Machine Learning models requires a paradigm shift in how these models are developed. A new approach known as **Fast Machine Learning (Fast ML)** was developed in the scientific community, focusing on the development and embedding of highly optimized ML models directly into detector instruments [15].

For the ATLAS Phase-II upgrade, this corresponds to deploying complex pattern-recognition algorithms onto the FPGAs of the Level-0 trigger. In this environment, machine learning models must operate strictly within a maximum latency budget of 10 µs [10]. Addressing these challenges with strict **µs** latency budgets and **energy-constrained** environments is the primary objective of the Fast ML paradigm.

Achieving ultra-low inference times requires a shift from standard ML practices. Traditional deep learning often produces sophisticated algorithms that prioritize high predictive accuracy at the expense of resource usage and inference time. Fast ML necessitates an approach known as **hardware ML co-design** [15]. This paradigm requires the algorithm designer to be mindful of the targeted

hardware constraints and to utilize it efficiently, while maximizing the algorithm's performance.

1.5.2 ML Architectures

In this section, we will introduce specific ML architectures evaluated during the research conducted for this thesis. This thesis assumes a foundational understanding of general Machine Learning principles, including standard artificial neural networks, backpropagation, and model training principles. For this reason, this text omits rudimentary ML concepts (the reader is referred to the following literature [16, 17]).

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep neural networks originally developed for the classification of 2D image inputs. They are exceptionally well-suited for processing grid-like topological data with a strong spatial correlation between nearby features. Possible examples of such inputs could be time-series (1D), images (2D), 3D scans, or video (3D).

Because particle detectors often produce data that can be mapped to 2D or 3D grid inputs, CNNs have become widely adopted in particle physics research and for many highly demanding tasks at CERN. Notable examples include the CMS project *DeepCore* for high p_T jet tracking [18], as well as the application of specialized *Submanifold Sparse* CNNs for particle classification in a liquid-argon detector of the *ProtoDUNE* experiment [19].

The Convolution Operation The main part of the CNNs architecture is the convolutional layer [20]. Unlike standard neural networks, CNNs rely heavily on spatially correlated features and weight sharing. Both of these advantages are achieved through the use of learnable **filters**, also referred to as **kernels**.

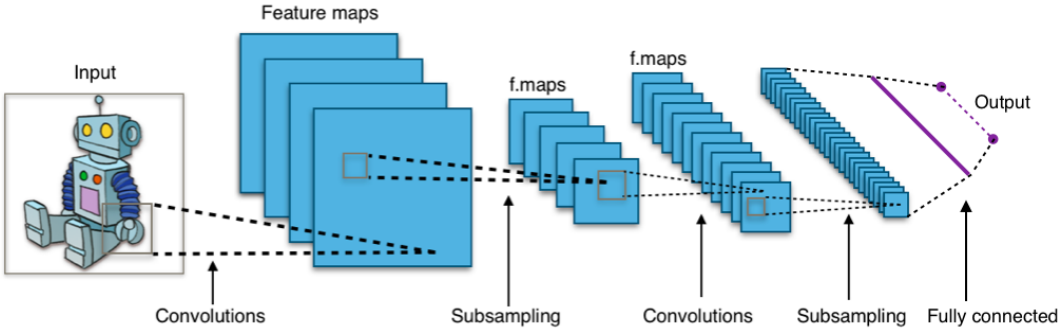


Figure 1.10 Illustration of a typical CNN architecture [21].

A kernel is essentially a small matrix of learnable weights that slides across the input space. At each step (determined by the **stride** parameter), the kernel computes a dot product of its weights and the local input features. The output of this weighted sum across the entire input matrix generates a new matrix representation referred to as a **feature map**.

The feature map is then passed through a **non-linear activation function**, such as the Rectified Linear Unit (ReLU).

The Pooling Operation Following the convolution operation and the non-linear activation function, CNN architectures commonly continue with a **pooling layer**.

Similarly to the convolution operation, this layer scans across the entire feature map using a fixed-size window (*receptive field*) and, at each step, computes a single value to represent the local features.

Multiple variants of the pooling function exist, with **Max Pooling** (outputting the maximum value within the current window) and **Average Pooling** (computing the arithmetic mean) being among the most widely adopted.

Regardless of the specific choice of the operation, these layers improve the **local translation invariance** of the CNN model with respect to small shifts in input data [20]. Furthermore, pooling downsamples the spatial dimensions of the input feature map, which reduces the number of learnable parameters and the computational complexity of subsequent layers.

The Feature Extractor The main body of a CNN architecture is constructed by stacking multiple instances of the above-mentioned blocks, typically composed of a convolutional layer followed by a non-linear transformation and a pooling operation. In many modern architectures, pooling layers are often omitted in favor of *strided convolutions*, which achieve spatial downsampling through the choice of the stride parameter.

These blocks are repeated multiple times throughout the network, extracting important information from the input data into an information-dense, spatially compressed feature map. This part of the CNN is referred to as the **feature extractor**.

Once the feature map has propagated through the feature extractor, the multi-dimensional feature map is flattened into a one-dimensional vector. This vector is then used as input to a standard fully connected neural network (often referred to as a regression or classification *head*, depending on the target task) to compute the network's prediction. An illustration of such an architecture is shown in Figure 1.10.

Graph Neural Networks

As highlighted in the previous section, CNN architectures are generally well-suited for data formatted into dense, regular grids. In these structures, the relationships between data points are strictly defined by their spatial coordinates, such as the adjacency of pixels in an image. Although this uniformity and tight relationships are beneficial for image data, they become fundamentally inefficient when the data are sparse or geometrically irregular.

To address these limitations, it is often highly beneficial to represent such data as a *graph*. Graphs (from *graph theory*) provide a mathematical framework for representing complex geometrical structures and the relationships between data points [16]. Formally, a graph G is defined as a set of nodes (or *vertices*) V and a set of edges E that connect them (producing *neighbors*). Both nodes and edges often encapsulate additional attributes, commonly referred to as node and edge features.

The reader is assumed to have a foundational understanding of basic concepts of graph theory.

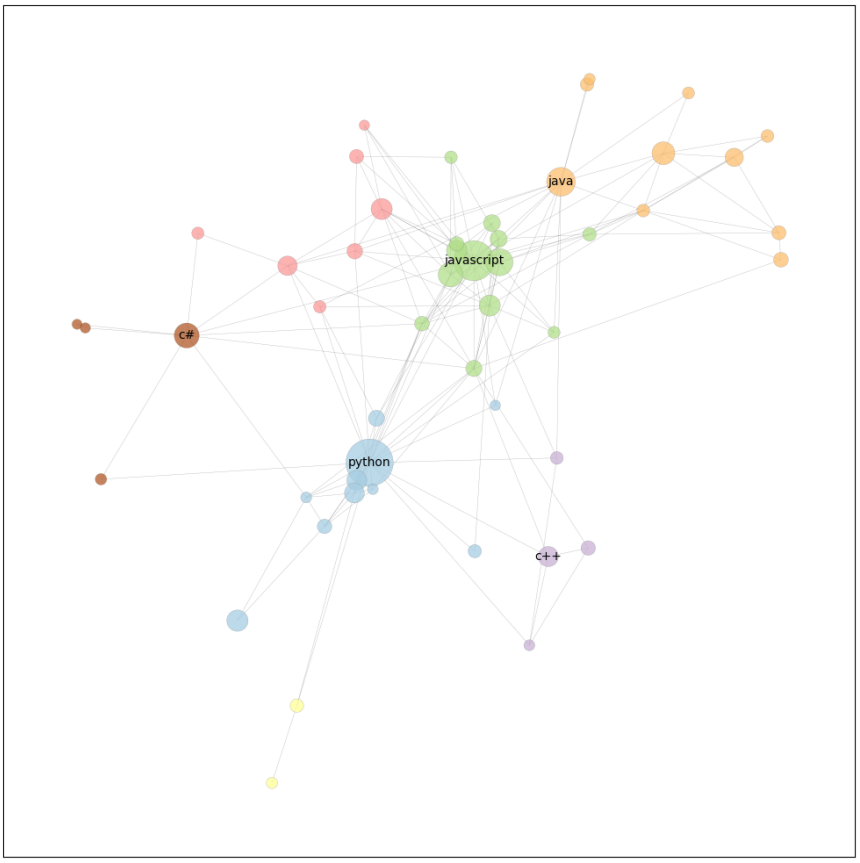


Figure 1.11 Example of structured data visualized as a graph. The graph presents the distribution and interconnection of different programming languages as topics on *Stack Overflow*, with labels for major hubs.

Because graphs do not rely on fixed grids, they are a good representation of a wide range of complex real-world data. Notable examples of fundamentally graph-structured data include the molecular structures of chemicals, the routing of a logistics network, and the relationships within a social network (as illustrated in Figure 1.11).

As with CNNs, there are notable projects at CERN using Graph Neural Networks for particle track classification and filtering. For reference, we can point to the paper about the *Exa.TrkX* project [22] and the technical report on the GNN particle tracking pipeline [23] (illustrated in Figure 1.12).

Graph Ordering To implement any ML method on graphs, we must first represent the graph data as a data structure. A graph is typically represented by a node feature matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$, where N is the number of nodes and D is the dimensionality of the node features. This is accompanied by an adjacency matrix $\mathbf{A} \in \{0,1\}^{N \times N}$, where the element $a_{ij} = 1$ if an edge exists between nodes i and j, and 0 otherwise.

While grid-like data, such as images, possess distinguishing features (e.g., grid coordinates) that fix their ordering, graph nodes generally do not have any absolute

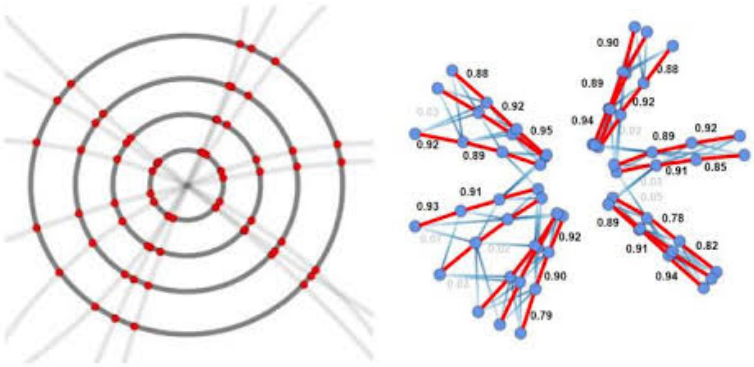


Figure 1.12 Visualization of the GNN particle tracking pipeline [23].

ordering. When constructing the $\mathbf{X}$ and $\mathbf{A}$ matrices, assigning a specific node, such as a single detector hit, to the first row versus the N-th row is a completely arbitrary software choice. However, feeding such matrices into a standard neural network would cause the model to interpret different hit orderings of the same event as two completely distinct inputs.

To formally define what we require from our ML models, let us first consider a permutation matrix $\mathbf{P} \in \{0,1\}^{N\times N}$, defined as a square matrix in which every row and column contains exactly one non-zero entry with a value of 1. By applying this matrix to the node feature matrix $\mathbf{X}$, we permute the node ordering (rows of the matrix), yielding a new matrix

$$\tilde{\mathbf{X}} = \mathbf{P}\mathbf{X}\,. \tag{1.1}$$

To preserve connections between nodes, the adjacency matrix must also be shuffled. This can be achieved by applying the same permutation matrix to both the rows and columns of the original matrix, thus:

$$\tilde{\mathbf{A}} = \mathbf{P}\mathbf{A}\mathbf{P}^{\mathrm{T}}. \tag{1.2}$$

To be able to employ any ML model on graph data effectively, we need to construct it in such a way that its predictions are independent of arbitrary node ordering [16]. Formally, for a task where the model makes a singular prediction about the entire graph, the model function $f(\cdot,\cdot)$ must be **permutation invariant**, ensuring that the output remains the same regardless of the ordering of the input. This property can be mathematically expressed as follows:

$$f(\tilde{\mathbf{X}}, \tilde{\mathbf{A}}) = f(\mathbf{X}, \mathbf{A}). \tag{1.3}$$

Conversely, for a task that requires node-level predictions, the model function $f'(\cdot,\cdot)$ must be **permutation equivariant**. This is a property where shuffling the order of the graph nodes by $\mathbf{P}$ causes the corresponding node-level predictions to be shuffled in the exact same manner:

$$f'(\tilde{\mathbf{X}}, \tilde{\mathbf{A}}) = \mathbf{P}f'(\mathbf{X}, \mathbf{A}). \tag{1.4}$$

By adhering to these two constraints, we can build ML models that decouple the arbitrary software representation from the underlying graph data and are able to make predictions at every level of the representation.

Message-Passing Algorithm To satisfy the aforementioned constraints, modern GNN architectures adopt the **Message-Passing Neural Network (MPNN)** framework. This paradigm draws direct inspiration from the convolutional filters employed in CNNs, effectively generalizing the convolution operation to non-uniform data.

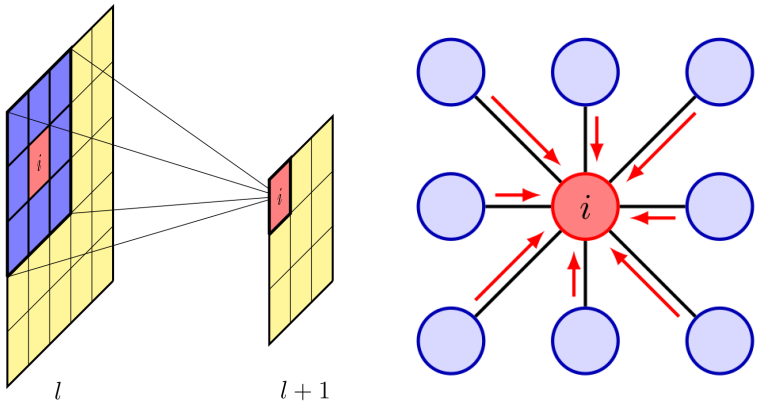


Figure 1.13 Illustration of the similarity between convolution operation and graph message-passing (red arrows symbolizing messages) [16].

However, instead of sliding a fixed, weight-shared filter across an input grid, nodes within a GNN dynamically exchange their feature information (node *embeddings*) along their edge connections (i.e. *passing messages*).

The message-passing algorithm is executed in discrete steps. To formalize the algorithm, let us define $\mathbf{h}_i^{(l)}$ as the feature vector of node i at step l. The initial state at step 0 is equal to the original node feature from the feature matrix $\mathbf{X}$. Each step of the algorithm consists of two successive phases: *Aggregation* and *Update* [16].

Firstly, during the **aggregation** phase, each node gathers messages from its immediate neighbors (where the set of neighbors of node v is denoted as $\mathcal{N}(v)$). A permutation-invariant aggregation function must be used to combine the incoming messages. Most often, a simple function such as an element-wise sum, mean, or maximum (or their combination) is employed. The aggregated message $\mathbf{m}_i^{(l)}$ for node i at step l is computed as:

$$\mathbf{m}_i^{(l)} = \text{Aggregate}\left(\left\{\mathbf{h}_j^{(l)} \mid j \in \mathcal{N}(i)\right\}\right). \tag{1.5}$$

Secondly, in the **update** phase, each node computes its new internal state for the next step. The new state is computed by an update function that combines the node's current state with the newly aggregated incoming messages. This function is typically implemented as a standard Multilayer perceptron (MLP). The new state of node i is updated as follows:

$$\mathbf{h}_i^{(l+1)} = \text{Update}\left(\mathbf{h}_i^{(l)}, \mathbf{m}_i^{(l)}\right). \tag{1.6}$$

These two phases (equations 1.5 and 1.6) result in a single message-passing step during which each node encounters information only from its immediate neighbors. Repetition of the algorithm for multiple steps allows nodes to gather features from increasingly distant regions of the graph, as illustrated in Figure 1.14.

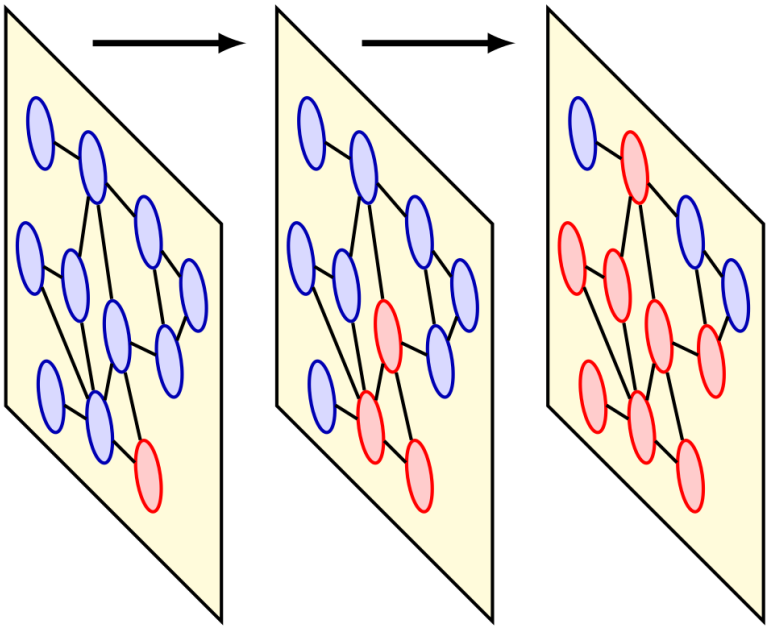


Figure 1.14 Illustration of the growth of the effective *receptive field* (nodes shown in red), with a growing number of message-passing steps (adapted from [16]).

It is important to note that the exact same update function is **reused** in all message-passing steps. Instead of learning a separate set of parameters for each individual step, which would linearly scale the algorithm's memory requirements with the number of steps, the internal weights of the update function (typically an MLP) are shared across all K iterations. This weight-sharing crucially allows the network to effectively expand its receptive field without increasing the overall parameter count, making the architecture highly memory-efficient.

Message-Passing with Edge Embeddings The standard MPNN formulation introduced so far allows the ML model to receive information exclusively from node features. However, in many tasks, critical information is encoded in the connections between nodes. In the context of particle tracking, the relative geometry of the detector, such as distances and orientations, cannot be determined solely from node features. To capture this, the message-passing algorithm must be extended to incorporate **edge features** (or edge *embeddings*) into the update and message-building steps.

Analogously to the node embedding, let us define the edge embedding at step l for an edge connecting nodes i and j as $\mathbf{e}_{ij}^{(l)}$. The message-passing algorithm is extended by introducing a new *Edge Update* phase [16]. The general MPNN equations are then formulated as follows:

$$\mathbf{e}_{ij}^{(l+1)} = \text{Update}_{\text{edge}}\left(\mathbf{e}_{ij}^{(l)}, \mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}\right), \tag{1.7}$$

$$\mathbf{m}_i^{(l+1)} = \text{Aggregate}_{\text{node}}\left(\left\{\mathbf{e}_{ij}^{(l+1)} \mid j \in \mathcal{N}(i)\right\}\right), \tag{1.8}$$

$$\mathbf{h}_i^{(l+1)} = \text{Update}_{\text{node}}\left(\mathbf{h}_i^{(l)}, \mathbf{m}_i^{(l+1)}\right). \tag{1.9}$$

Graph Readout After executing the K message-passing steps, the graph node (and edge) embeddings have been contextualized with information from their expanded receptive fields. In the next step, a mathematical operation is used to extract information from the graph while adhering to the mathematical constraints. This phase is known as the *readout*.

Node- and Edge-Level Tasks: If the objective requires localized prediction, such as noise hit filtering (node classification) or particle trajectory reconstruction (edge classification), the readout function is applied directly to the final embeddings. A variety of readout mechanisms can be employed for this task, such as applying a dedicated MLP to each embedding, or computing the dot product between pairs of node embeddings to calculate the probability of an edge connection [16].

Graph-Level Task: In case the task requires an assessment of the entire graph, such as the determination of molecular properties, the MPNN must aggregate all node states into a single, global graph representation $\mathbf{h}_G$, while satisfying the strict permutation-invariance of the structure. The global graph representation is defined as follows:

$$\mathbf{h}_G = \text{Readout}\left(\left\{\mathbf{h}_i^{(K)} \mid i \in V(G)\right\}\right), \tag{1.10}$$

where $V(G)$ is the set of all nodes (*vertices*) of graph G. This readout operation is commonly implemented using simple functions, such as an element-wise sum, average, or maximum, analogous to the node-level aggregation. Furthermore, advanced attention-based mechanisms have also been utilized in modern architectures.

In the end, the graph representation $\mathbf{h}_G$ is passed through a classification or regression head to produce the final MPNN prediction.

1.5.3 Model Quantization

Standard modern neural networks are, by default, trained and evaluated in 32-bit floating-point number precision. However, as established in Section 1.4 regarding the deployment of ML models on FPGA hardware, all network operations must be mapped onto available Digital Signal Processors (DSPs), Look-up Tables (LUTs), and on-chip memory. Direct translation of full 32-bit-precision operations is basically infeasible due to the strict resource limitations of the hardware.

Consequently, the network must be **quantized**. Quantization is the process of mapping high-precision values (such as 32-bit floating-point numbers) into a smaller set of lower-precision values (such as 8-bit or lower), substantially reducing the memory footprint of the transformed models.

This process has become a standard part of the ML model training pipeline. For example, in models designed for smartphone deployment, quantization is used to save memory on hardware with such limited resources. Quantization generally alters the bit-precision (or *bit-width*) of all network parameters, such as weights and activations, and often replaces floating-point arithmetic with fixed-point arithmetic.

Quantization-Aware Training (QAT) A naive approach, sufficient in less constrained environments, is **Post-Training Quantization** [24]. In this approach, the ML model is first trained and then fed into a quantization pipeline. However,

this transformation inherently introduces rounding errors in the model, which often severely degrade the prediction accuracy.

To remedy this and preserve accuracy, modern pipelines utilize **Quantization-Aware Training (QAT)** [25]. This transformation is being applied during the training process, actively trying to achieve the best predictive accuracy with the smallest possible model.

High Granularity Quantization (HGQ) Traditional QAT frameworks suffer from limited *granularity*, forcing a single uniform bit-width across an entire layer or even the whole network. However, this disregards the possibility that different parameters within a single layer may contribute disproportionately to the final output.

To achieve µs-level inference latencies, while maximizing efficient use of hardware resources, a new open-source framework called **High Granularity Quantization (HGQ)** has been developed [25]. With contributions from CERN researchers, this framework was developed specifically with FPGAs and strict latencies in mind. Unlike traditional QAT frameworks, HGQ treats each parameter of the network independently, allowing for heterogeneous quantization even within a single network layer (hence the *high granularity* name).

The framework achieves this by introducing a hardware-aware regularization term to the loss function. This term estimates the model's resource consumption while running on the FPGA. The dominant part of the resource usage comes from the matrix multiplication operations processed by the model. To accurately estimate the resource usage, the framework introduces a metric called **Effective Bit Operations (EBOPs)** [25], which can be calculated for a model as follows:

$$\text{EBOPs} = \sum_{i,j \in \mathcal{M}} b_i \cdot b_j + \sum_{k,l \in \mathcal{A}} \max(b_k, b_l), \tag{1.11}$$

where $\mathcal{M} = \{\{i, j\}_n\}$ is the set of all multiplication operations between operands with bit-widths b_i and b_j, and $\mathcal{A} = \{\{k, l\}_m\}$ is the set of all explicit addition/subtraction/muxing operations (e.g., bias, average or maximum pooling) between operands with bit-widths b_k and b_l.

This regularization term in the loss function allows both network precision and size to be optimized via gradient descent during training. As a result, the network is optimized to allocate higher precision only for the most critical parameters while effectively pruning the unnecessary ones. The final loss function is then given by:

$$\mathcal{L} = \mathcal{L}_{\text{base}} + \beta \cdot \text{EBOPs} + \gamma \cdot \sum \text{bit-widths}. \tag{1.12}$$

The EBOPs regularization term is included with the coefficient β, which can be used to control the trade-off between the performance of the quantized model and its size. Furthermore, one more L_1 regularization term with coefficient γ is added to account for network parameters, which are not involved in multiplication operations, such as the last-layer's outputs.

The framework promises a near drop-in replacement experience, allowing its users to seamlessly exchange their model's previous Keras implementations with HGQ variants. While the promise of a simple exchange of modules is true in

practice, the end-user still has to go through a non-trivial trial-and-error setup and tuning of quantization parameters to achieve a satisfying result.

1.5.4 Hardware-Aware Neural Architecture Search

As established in Section 1.5.1, during the development of machine learning models for constrained environments, the approach called **hardware ML co-design** must be utilized. Frameworks, such as **HGQ** (see Section 1.5.3), provide the mechanism to train and quantize models into a hardware-feasible footprint. However, a non-trivial step in the ML design pipeline is the selection of architectural *hyperparameters* (such as the number of network hidden layers, their sizes, feature dimensions, etc.).

Traditionally, the process of hyperparameter selection has been done through the means of manual, time-consuming trial-and-error. Given the vast architectural design space and complexity of the environment, such a process is fundamentally inefficient.

To improve hyperparameter selection and discovery, modern machine learning pipelines have started utilizing **Automated Machine Learning (AutoML)** techniques. Furthermore, a specialized branch of AutoML known as **Hardware-Aware Neural Architecture Search (HW-NAS)** is recognized [26], employing architectural search constrained by deployment and hardware limitations, such as strict latency and FPGA memory budgets.

Multi-Objective Optimization By introducing any hardware constraints into the ML design process, the task is immediately transformed into a multi-objective optimization problem. The search algorithm must maximize the model performance while simultaneously satisfying hardware constraints (e.g., minimizing the memory footprint or inference time). Two notable methods for solving such optimization problems are commonly employed in HW-NAS.

Scalarization Method: A traditional approach for multi-objective optimization is to transform the problem into a single-objective optimization problem. This is typically achieved through the means of a weighted sum or other types of weighted functions. Although this transformation simplifies the process, it yields only a single optimal solution for the specific set of weights. To obtain the complete set of optimal solutions (the *Pareto front*), multiple optimization runs with varying weights are required [26].

Evolutionary Method: State-of-the-art HW-NAS frameworks often utilize an evolutionary algorithm, most notably the **Non-dominated Sorting Genetic Algorithm II (NSGA-II)** [26]. This algorithm supports multi-objective optimization out of the box by utilizing *elitism* (a set of the best solutions preserved for the next generation without undergoing genetic mutation or crossover) and *non-dominated sorting*, which enable efficient exploration of the search space. Connecting these two techniques, the algorithm is able to efficiently explore and approximate the entire Pareto front in a single run.

2 Methodology

In this chapter, we will detail the specific methodologies, introduce the datasets, and the baseline and machine learning algorithms utilized in this thesis for muon time-of-passage estimation. Strict hardware constraints, such as the latency and memory budgets of the targeted platform, have necessitated specific steps in the algorithm design process in our work. The reasoning behind these hardware-software co-design choices will be introduced throughout this chapter.

The chapter begins by introducing the Monitored Drift Tube (MDT) events dataset, together with the data preprocessing steps required to format the detector information into normalized inputs for our algorithms in Section 2.1. Next, in Section 2.2, this chapter will introduce a baseline method to provide a performance metric for machine learning approaches, outlined in the following sections 2.3 and 2.4. Finally, Section 2.5 details the hardware-aware optimization techniques employed for model selection and optimization using the Non-dominated Sorting Genetic Algorithm II (NSGA-II).

2.1 Dataset Introduction

To build a foundational understanding of the proposed algorithms, it is necessary to first examine the underlying data structure utilized throughout this work. This section introduces the dataset acquisition process, defines the data features, and provides visualizations of typical model inputs, which should help to establish a clear bridge between the detector geometry and its software representation. Lastly, this section introduces the synthetic noise augmentations utilized in later stages of model evaluation.

Test Vector Generation The data utilized throughout this thesis are fully synthetic. Due to the limitations and complexities of current data-acquisition implementations and the novelty of our task, we were unable to obtain data from real samples from previous detector runs. Instead, the models are trained and evaluated using simulated *test vector* n-tuples. These datasets are generated via a dedicated script that processes Monte Carlo-simulated events, designed to replicate real interactions and electronic responses.

Currently, the data generation pipeline produces only single-muon events. Despite the possibility of rare multi-muon events in real-world running conditions, using a single-muon dataset aligns with the intended operational environment of the studied algorithm. The algorithm is designed to operate on an isolated prefiltered event selected by the upstream filtering stage of the signal processing pipeline.

Test Vector Format The resulting simulated test vectors are saved in multiple Comma-Separated Values (CSV) files, splitting events by physical detector region (e.g., the file `B_A_3` – Barrel region, Side A, Sector 3). Consequently, these files can be easily ingested into any data processing pipeline.

Within the files, each row contains information about a single signal recorded by an individual MDT tube. Multiple such signals can be grouped together based

on a matching event index to reconstruct a complete **event group**.

The following table introduces the test vector features relevant to this thesis:

Feature	Description
Event Index*	A unique identifier grouping MDT hits into a single event.
Coarse Time	The event detection time: an incremental 40 MHz clock counter, from the global ATLAS synchronization clock.
Fine Time	The event detection time: Time-to-Digital Convertor (TDC) counts, providing fine timing precision (1/32 of the 40 MHz clock, ≈ 0.78 ns resolution).
Tube z	The global position of the tube center in the z-axis (along the beamline).
Tube ρ	The global distance of the tube center from the z-axis.
Station Type	An index of the detector station (0 for inner, 1 for middle, and 2 for outer).
Tube Layer	A localized index of the layer within a specific station (the inner station contains 8 MDT layers, the middle and outer stations only 6 layers).
Tube Number	A localized horizontal index of the tube within a specific layer, with the tube closest to the vertical axis of the detector having index 0.
Chamber ID	A unique identifier of the MDT chamber housing the tube.
Electronics Delay	The constant timing delay caused by electronics and signal cables.
Time-of-Flight*	The true time-of-flight of the muon from the interaction point to the tube.
Drift Time*	The true drift time of electrons from the ionizing track to the tube wire, caused by a passing charged particle (see Figure 1.3).

Table 2.1 Summary of test vector features. Features marked with an asterisk (*) denote golden simulation variables not available under real-world conditions.

Data Visualization The event groups extracted from the dataset can be directly visualized, as demonstrated in Figure 2.1.

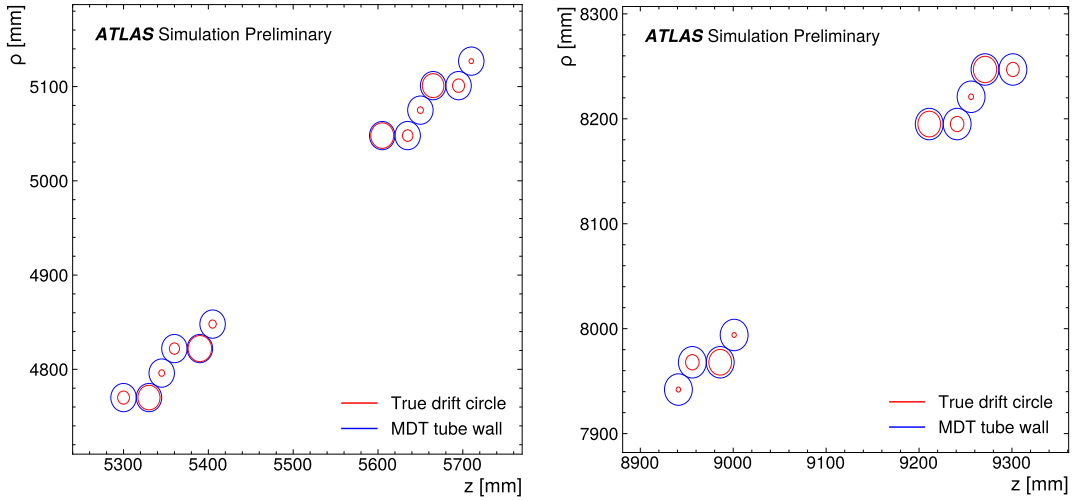


Figure 2.1 Visualizations of representative event groups from the inner (left) and middle (right) stations. The blue circles represent the wall of active MDT tubes, and the red circles represent the true drift radii.

The position of the tubes is given by the actual detector geometry, and the visualizations explicitly display the events in these global coordinates (z, ρ).

These visualizations are representative examples of clean single-muon events. We chose these two events, because they highlight the main difference in how muons are detected across different detector stations: the innermost station (left) utilizes 8 MDT layers, compared to the middle and outer stations (right), where the chambers house only 6 layers.

Furthermore, the figure displays the golden variables available for each active tube in the form of *true drift circles* (shown in red). The radius of a drift circle represents the distance from the central tube wire to the ionizing muon track, and can be derived from the tube's drift time using the $R(t)$ function (as described in Section 1.2.1).

Figure 2.2 provides a view of the entire detector layout in the global coordinate space (z, ρ), as registered by the readout electronics. The colors distinguish the three separate MDT stations. Because the project focuses exclusively on the barrel region of the ATLAS detector, the displayed visualizations do not represent the endcap geometry.

It is important to note that the individual MDT chambers do not form a perfect continuous cylinder with a fixed radius. There is a visible variance in the distance (ρ) from the beamline between different chambers within the same station (e.g., the middle station with chambers situated at two different distances) as shown in the next Figure 2.3. This configuration is forced by the physical geometry of the ATLAS detector, as chambers must fit around other structures, such as the detector legs or the toroidal magnets.

To further introduce the detector layout, Figure 2.3 provides an alternative view of the exact same dataset presented previously in Figure 2.2. While the previous visualization distinguished the three main detector layers in the form of stations, this figure illustrates the individual MDT chambers using separate

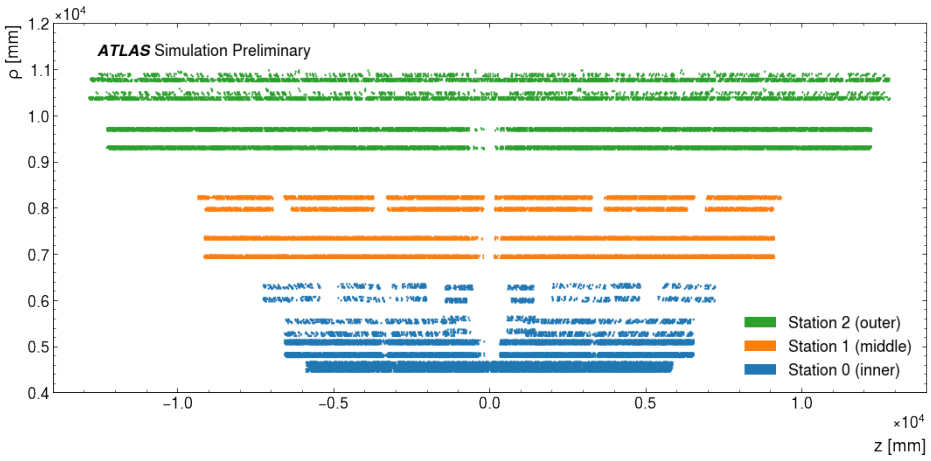


Figure 2.2 Visualization of a subset of the entire muon event dataset, with colors distinguishing the three MDT stations.

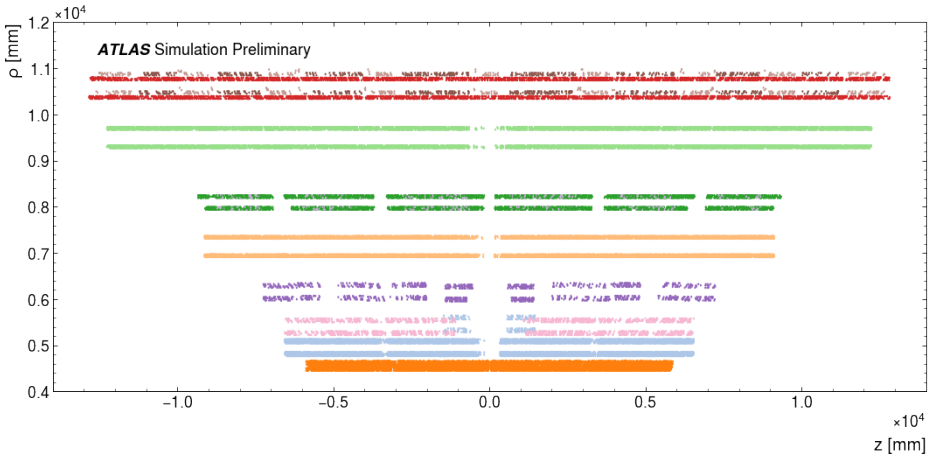


Figure 2.3 Visualization of a subset of the entire muon event dataset, with colors distinguishing individual MDT chambers.

colors. The comparison of the two visualizations illustrates how the individual MDT chambers are arranged to form the geometry of a single station.

Input Space Normalization To optimize the dataset for our algorithms, several transformation and normalization steps can be applied to the data. First, given the forward-backward symmetry of the ATLAS detector along the beamline, the z coordinate range can be trivially normalized to a positive range by applying the absolute value function. This operation basically folds the detector geometry in half while preserving the ordering of the tube numbers intact, as index 0 is always defined as the tube closest to the vertical axis within any given tube layer.

Furthermore, the algorithms can operate on either the global coordinate space (z, ρ), utilized so far in all visualizations, or a discrete local coordinate space formed by the **tube number** and **tube layer** indices. This introduces a major trade-off. The global system has a higher expressive power, providing exact locations within the detector geometry. In contrast, the local system is highly memory efficient, at the cost of sacrificing the precise spatial context, such as

the differentiation between multiple chamber configurations within one station, which would present the tube at the same layer index but different ρ distance. Our experiments have shown that the local coordinate space is often sufficient for algorithms in the context of isolated detector geometry, whereas the global coordinates remain necessary whenever the algorithm must understand the global geometry relations across a larger part of the detector.

In the raw dataset, the tube layer indices l are strictly in the following range depending on the station: $l \in \{0, \ldots, 7\}$ for the inner station, and $l \in \{0, \ldots, 5\}$ for the middle and outer stations. The spatial expressiveness of the local coordinate system can be drastically improved by distinguishing muon stations by the tube layer. This is achieved by offsetting the layer indices of each subsequent station by the total number of layers in all previous stations. With this transformation, the new local coordinate system spans tube layer ranges of $\{0, \ldots, 7\}$ for the inner station, $\{8, \ldots, 13\}$ for the middle station, and $\{14, \ldots, 19\}$ for the outer station.

The final feature space, with both the geometric fold along the beamline and the layer offset transformation applied, is visualized in Figure 2.4.

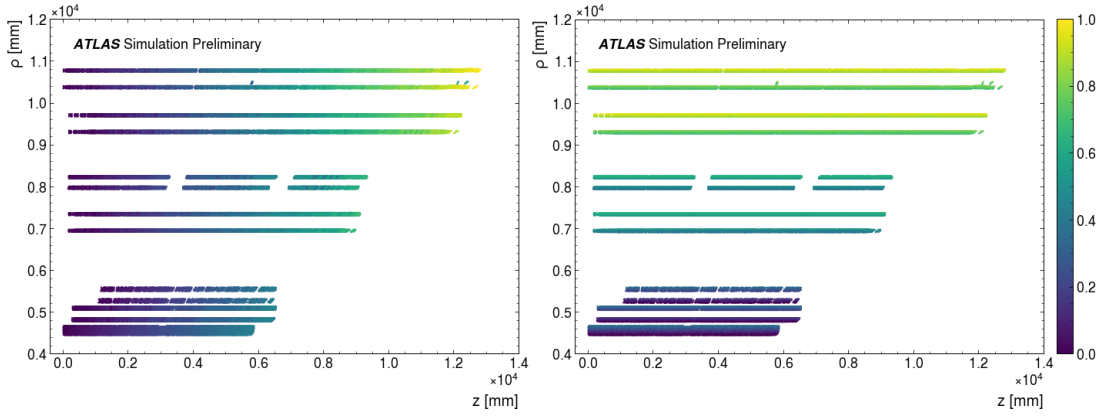


Figure 2.4 Visualizations of the normalized local coordinate space of tube numbers (left) and tube layers (right). The color gradient indicates the final range of values after the normalization.

Note: There is a minor difference in the layout of the MDTs visible in Figure 2.4. Specific tubes, mainly in the inner station, were filtered out from the dataset to correct anomalous assignments of indices, caused by an issue in the data generation pipeline.

Dataset Augmentation To conclude this section, we will introduce the stochastic dataset augmentation utilized in the later experimental stages of this work. As previously established, the dataset contains exclusively single-muon events. A fraction of simulated events contains noise hits generated by the data generation pipeline through Monte Carlo simulation. Such hits can be identified via a dedicated feature flag. In the initial phase of algorithm evaluation, we used this flag to filter these hits out to establish performance exclusively on **clean single-muon events**.

However, the following experimental phases began to evaluate the robustness and susceptibility of different algorithms to varying amounts of noise. Despite the original dataset containing simulated noise hits, they are fixed, and thus the amount of noise in the inputs cannot be parametrically controlled. For this reason, a custom augmentation process was developed, which extends the original examples with generated noise hits, based on a **configurable probability** parameter.

Noise hits within an event group can occur in multiple ways. They are predominantly caused by the high volume of charged-particle scattering inside the ATLAS detector, as well as penetrating cosmic background radiation reaching the detector cavern. In the raw data, such events manifest as entirely new particle tracks or as isolated, uncorrelated noise hits. The implemented augmentation process focuses on emulating specifically these **isolated hits** in the immediate vicinity of the original event (visualized in Figure 2.5). This is based on the assumption that secondary particle tracks, corresponding to background particles, and individual hits further away from the event group will be efficiently rejected and filtered out by the upstream signal filtering process.

Based on these assumptions, the pipeline for the dataset augmentation process is formalized in the following Algorithm 1, and its results are visualized in Figure 2.5.

Algorithm 1 Synthetic MDT Noise Injection

Require: Clean event group E, Noise probability parameter p_{noise}
Initialize $E' \leftarrow E$
Define local MDT neighborhood $\mathcal{N}$ around active tubes in E'
for each tube $t \in \mathcal{N}$ **do**
    Sample random variable $r \sim \mathcal{U}(0, 1)$
    if $r < p_{\text{noise}}$ **then** ▷ Stochastic hit selection
        if t inactive in E' **then**
            Activate t
            Generate timing parameters consistent with E
        else
            either Perturb existing timing parameters of t
            or Deactivate t ▷ Simulates MDT discharge (dead time)
        end if
    end if
end for
Reconstruct augmented E' representation
return E'

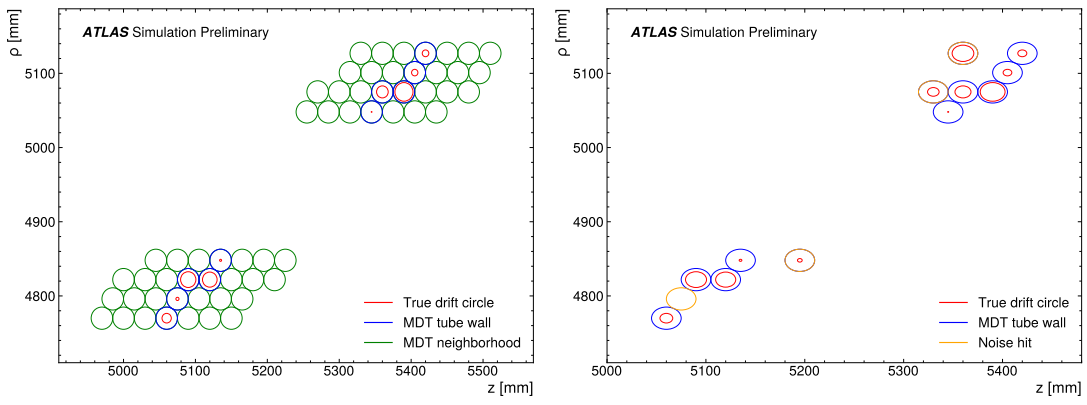


Figure 2.5 Visualization of the synthetic MDT noise injection process. On the left, the defined local MDT neighborhood (green tubes) around the active event group. The original event group is also included in this neighborhood set. On the right, the resulting event group after an augmentation with 5% ($p_{\text{noise}} = 0.05$) noise injection probability. The noise hits (orange) show the potential to activate, deactivate or perturb the tubes in the original event.

2.2 Baseline Method

To be able to evaluate the performance of novel machine learning approaches, it is necessary to first develop a robust baseline method to serve as a performance benchmark. This section will focus on the standard algorithmic (*non-machine learning*) approach selected as this benchmark. Furthermore, we will examine the possible limitations related to the core complexity of the **time-of-passage estimation**.

Core Idea The idea for the algorithmic approach is derived from observations of the visualizations of event groups (such as in Figure 2.1). When the true drift times are known (in figures represented by the radii of the drift circles in red), it is possible to construct a singular common tangent line to all of the drift circles. This serves as a straight-line approximation of the muon's trajectory through the MDT chamber.

It is important to note that modeling the muon trajectory as a straight line in the vicinity of the MDT chamber is a well-established approximation, utilized extensively by multiple teams prior to our work. The justification for the linear model stems from the design of the magnetic field of the ATLAS detector. While the primary toroidal magnets generate a strong magnetic field ranging from 2.0 to 6.0 T, producing the muon's track curvature necessary for the momentum measurement, the magnetic field strength within the MDT chambers of the barrel region averages only 0.6 T [5]. Due to this design, the deflection of high-momentum muons in the small volume of a single MDT chamber is negligible for our measurement.

The following equation shows the relationship between the measured timing components for each signal:

$$\text{Detection Time} = \text{BCID} + \text{ToF} + \text{Drift Time} + \text{Electronics Delay}, \tag{2.1}$$

where BCID (*Bunch crossing identifier*) represents the 25 ns clock synchronized with particle collisions at the interaction point, and ToF denotes the time-of-flight of the muon from the interaction point to the specific MDT that registered the hit. The electronics delay, caused by the propagation of the signal through the cables and the front-end readout electronics, is constant and known ahead of time. Similarly, the ToF can be treated as a constant. For muons at 10 GeV to 100 GeV energies observed by the detector system, the ToF variance across the region of the ATLAS detector is well below the temporal resolution ability of the readout system. Consequently, the above Equation 2.1 can be reformulated to isolate the BCID term:

$$\text{BCID} = \text{Detection Time} - \text{Drift Time} - \underbrace{(\text{ToF} + \text{Electronics Delay})}_{\text{constants}}. \tag{2.2}$$

Because the grouped terms in Equation 2.2 are constant, the fundamental **time-of-passage estimation** task is typically reframed as **BCID estimation** for a given event, since the discrete BCID values are generally easier to understand and interpret as target variables for any predictive model. For this reason, this work will show results predominantly framed as BCID estimation, with the prediction error measured as the deviation from the true BCID value.

Furthermore, an important constraint of this system is that the resulting BCID must be identical for all MDTs that register a hit from the same muon event.

Trajectory Estimation By rearranging Equation 2.2, the drift time term can be isolated for each individual tube. Consequently, by selecting an arbitrary BCID value for a given event and utilizing known signal detection times across all active MDTs,

the exact drift time for each active tube can be inferred (and thus the radii of the corresponding drift circles).

Combining these radii with the previously established fact that a straight line approximation of the muon's trajectory is possible near the MDT chamber, the trajectory can be reconstructed by computing a common tangent line to these generated drift circles.

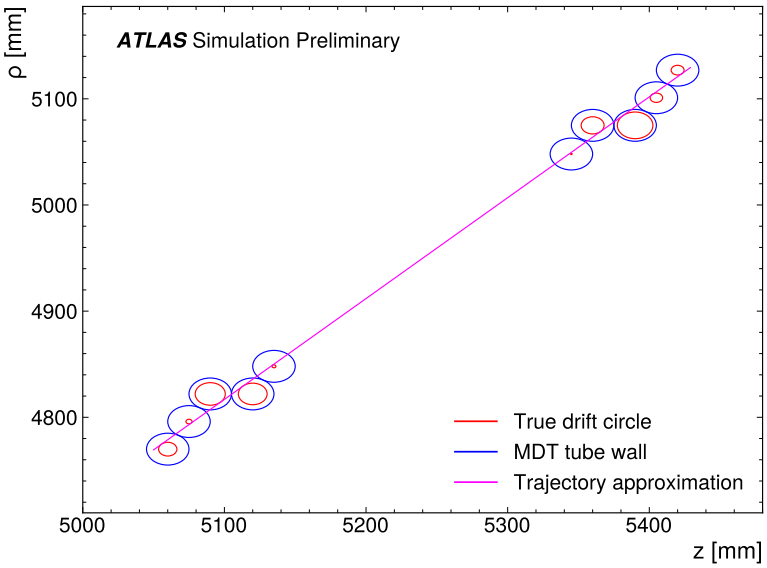


Figure 2.6 Showcase of the trajectory approximation as a straight line common tangent to each drift circle (red radii in each blue tube).

Given a straight line trajectory and a set of active MDTs with their corresponding drift circles, we define the trajectory estimation metric as the sum of shortest orthogonal distances from the trajectory to each drift circle. For clean single-muon events, minimizing this metric yields the true muon trajectory, which is identical to the common tangent line.

By combining the two constraints established above, we can summarize that the task of the baseline algorithm consists of these two parts. First, it must reconstruct the muon's straight line trajectory through the active MDTs. Afterward, given the shortest orthogonal distances between the trajectory and the center of each active tube, it must infer the corresponding drift times to calculate a single BCID value for the observed event, as shown in Equation 2.2.

Given the computational constraints of the project, the algorithm search prioritized simpler approximation methods that remain robust in scenarios with outliers, because events under real-world conditions will always encounter some degree of noise. For these reasons, the Random Sample Consensus algorithm was selected for the initial experiments.

2.2.1 Random Sample Consensus (RANSAC)

RANSAC is an iterative parameter estimation algorithm explicitly designed to be robust against a large number of outliers in the input data [27, 28]. The algorithm operates by generating candidate models of the data from a minimal set of randomly sampled observations. Then it evaluates the entire dataset to find observations that fall

within a predefined error margin to create a *consensus set*. This approach contrasts with other model estimation techniques, which initially try to use all available observations and attempt to remedy the inclusion of noisy observations afterward.

The general RANSAC algorithm can be written as follows [28]:

Algorithm 2 General RANSAC

Require: Set of observations D, Error margin ϵ, Consensus threshold τ

1: Randomly sample a minimal subset of observations from D
2: Solve for candidate model M using this subset
3: Find all *inliers* from D that fall within the ϵ-boundary of M
4: **if** Number of inliers $\geq \tau$ **then**
5: Re-estimate model M using the complete set of inliers
6: **return** M
7: **else**
8: Return to step 1 ▷ Resample observations
9: **end if**

Implementation In our specific implementation, the algorithm randomly samples three active MDTs from an event group to generate a straight line candidate trajectory. The error margin (ϵ) for the classification of inliers is generally selected between 1.5 to 2.0 times the MDT radius, to account for the case of two adjacent tubes within a single layer.

The core resampling loop, as described in Algorithm 2, is repeated for a predefined number of iterations, tracking the best candidate model and the size of its consensus set. At the end of this process, the model with the maximal set of inliers is selected. The number of resampling steps can be tuned based on the expected amount of input noise, as a higher number of steps significantly lowers the chance of an outlier being selected for the model estimation. This hyperparameter introduces a trade-off between accuracy and computational complexity of the algorithm.

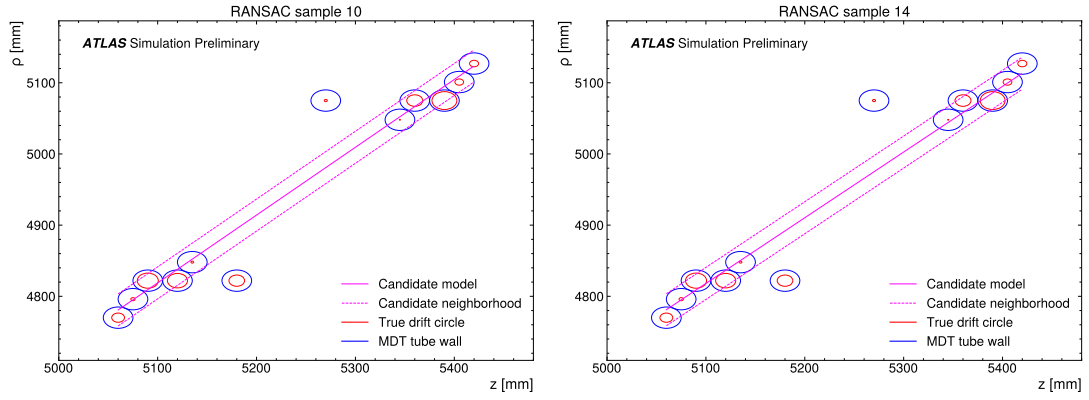


Figure 2.7 Visualization of two independent RANSAC resampling iterations. Both candidate models correctly classify all true MDT hits from the presented event group with injected noise.

As demonstrated in Figure 2.7, the RANSAC implementation produces a variety of candidate solutions. Because the algorithm constructs the straight line model from the centers of the sampled MDTs, multiple different candidate solutions can contain

all true hits within their ϵ margin while still differing from the true muon trajectory. Furthermore, because of the algorithm's design, it is difficult to introduce an accuracy metric (such as the previously discussed deviation from a common tangent) into the stochastic sampling technique to actively guide the search toward the true trajectory.

However, the solution variance of the straight line model can be remedied by utilizing the geometric constraint introduced earlier. Instead of fitting a generic straight line to the centers of three MDTs, a constrained trajectory can be constructed as a common tangent to their drift circles. By explicitly connecting the trajectory approximation with the drift times, the following algorithm was developed.

2.2.2 BCID-Conditioned RANSAC Strategy

Using the constraints established in the previous section, this search strategy substantially reduces the size and dimensionality of the search space during the trajectory approximation. Because the muon's trajectory can be deterministically modeled as a common tangent to the MDT drift circles, rather than modeling a general straight line trajectory using a set of three points, this strategy reduces the minimal sample size to only two tubes.

Following the RANSAC property of outlier robustness by selecting the minimal set of observations required for model estimation, this strategy stochastically selects a pair of active MDTs. Given these two tubes and their corresponding drift circles, exactly four distinct common tangent lines can be constructed. While the need to evaluate four tangent lines brings minor computational overhead, this approach provides a critical improvement. If both sampled MDTs belong to the true muon track, the geometric constraint guarantees that one of the four tangents aligns with the true trajectory.

To formalize the trajectory evaluation metric established previously, we compute the shortest orthogonal distance from the candidate trajectory to each drift circle. For a given candidate trajectory, defined by the general linear equation $ax + by + c = 0$, the orthogonal distance d_i from the line to the center coordinate (x_i, y_i) of the tube i is defined as:

$$d_i = \frac{|ax_i + by_i + c|}{\sqrt{a^2 + b^2}}. \tag{2.3}$$

Using this orthogonal line-point distance, we can define the overall trajectory approximation error E as the sum of squared residuals between the distance d_i and the drift radius r_i of each tube:

$$E = \sum_i (d_i - r_i)^2, \tag{2.4}$$

where the summation goes over all active tubes in the current event group. This error function is minimized by the trajectory that aligns with the true common tangent of all drift circles.

Temporal BCID sampling The candidate trajectory generation and evaluation methods formalized above both operate under the assumption that the exact drift radii are already known. However, under real-world conditions, these are completely unknown ahead of time and must be inferred. Because the algorithm utilizes the RANSAC approach of stochastically selecting the minimal subset required for the tangent creation, this step of the algorithm is strictly fixed without any additional free variables. As a result, the only variable remaining in the entire pipeline is the discrete BCID of the event, which is the primary target of this task.

By selecting a specific BCID hypothesis and substituting it into the reformulated Equation 2.2, the exact drift time for each tube can be computed and immediately converted to the drift radius using the $R(t)$ function described in Section 1.2.1. Because this single temporal variable characterizes the full geometric structure of the candidate trajectory, selecting an efficient BCID sampling strategy is a critical part of the algorithm's design.

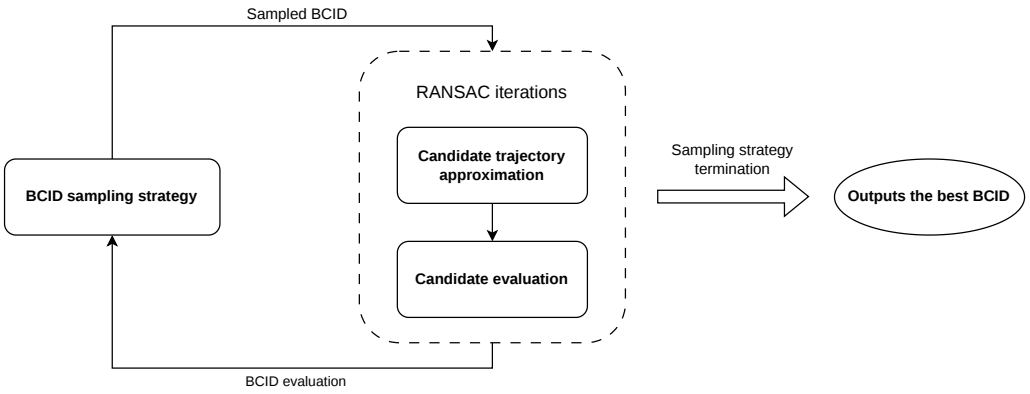


Figure 2.8 Simplified illustration of the baseline algorithm pipeline.

To address this design challenge, two BCID sampling strategies were developed to enable this pipeline to serve as both a baseline and an indication of the inherent performance limitations imposed by the complexity of the task.

Exhaustive Search Strategy To estimate the performance limits of the trajectory evaluation, we first introduce a simple brute-force exhaustive BCID search strategy. This strategy evaluates every discrete BCID hypothesis within the feasible range for a given event. This range can be derived by reformulating the timing constraint from Equation 2.2:

$$\text{BCID} + \text{Drift Time} = \underbrace{\text{Detection Time} - (\text{ToF} + \text{Electronics Delay})}_{\text{Known at the time of detection}}. \tag{2.5}$$

Because the right-hand side of Equation 2.5 is a known constant for any detected signal, the feasible range of BCID values is bounded by the inherent drift time range of the MDTs. As introduced in Section 1.2.1, the maximum drift time of electrons in MDTs is approximately 720 ns. The extremes of the BCID search window are thus defined by the following two geometric cases:

- If a muon passes directly through the center of an MDT, the drift time is 0 ns, which corresponds to the **latest** possible BCID from which this event could have originated.
- In the opposite extreme case, if a muon passes through the absolute perimeter of the tube, the drift time is maximized at approximately 720 ns, defining the **earliest** possible BCID value.

To account for the physical effects of the magnetic field inside the ATLAS detector, which can slow down the drift of electrons in the MDTs, the 720 ns search window is typically widened in practice. In the experiments presented in this work, we will be uniformly working with a search window of 800 ns (corresponding exactly to **32 discrete BCID values**).

As illustrated in the algorithm pipeline in Figure 2.8, the process computes the BCID search window and then systematically samples each feasible BCID hypothesis, which is passed into the BCID-conditioned RANSAC loop. In this loop, the algorithm performs multiple iterations of stochastically sampling MDT pairs to generate four distinct tangent models based on the corresponding drift circles. These candidate trajectories are then evaluated against the rest of the active MDTs using the trajectory approximation error metric (E) defined in Equation 2.4. In the end, each BCID hypothesis is assigned a score based on the minimum error E achieved during the RANSAC iterations. Upon reaching its termination condition, the pipeline returns the BCID value with the lowest overall error, along with the corresponding optimal trajectory.

Iterative Search Strategy Due to computational constraints of the main objective, the goal is to minimize the number of evaluated BCID samples while preserving the ability to identify the optimal BCID value.

To achieve this, an iterative search strategy was developed that draws inspiration from gradient descent methods, minimizing the error function E by stepping the BCID hypothesis in the direction of decreasing error. However, because the BCID is a discrete variable, the error function is non-differentiable, and thus the strategy must utilize a discrete *finite difference approximation* of the local gradient to determine the correct next search direction while keeping the total sample count to a minimum. The search strategy executes in two distinct phases:

1. In the first phase, the strategy selects and evaluates a small set of evenly-spaced discrete BCID values across the search window. These values are evaluated using the BCID-conditioned RANSAC approach established earlier, and the optimal BCID sample S is selected for the second phase.

2. In the second phase, the strategy samples two BCID values, A and B, adjacent to S (such that $A < S < B$). By comparing the three error function values $E(S), E(A)$, and $E(B)$, the local gradient can be approximated using the finite difference approximation, allowing the strategy to select the direction of the steepest descent. The algorithm then performs a limited number of steps in this optimal direction. If an update step results in an increase in error E, the preceding BCID value is a **local minimum**, the process terminates, and this optimum is returned as the final solution.

As demonstrated in Figure 2.9, this implementation of the iterative strategy evaluates six evenly-spaced samples in the initial phase, followed by up to five steps in the gradient descent phase. Our experimental observations indicate that, with this configuration, the iterative strategy achieves performance comparable to that of the exhaustive strategy while using about one-third of the resources. Further evaluations and comparisons of these baseline algorithms are shown in the following section.

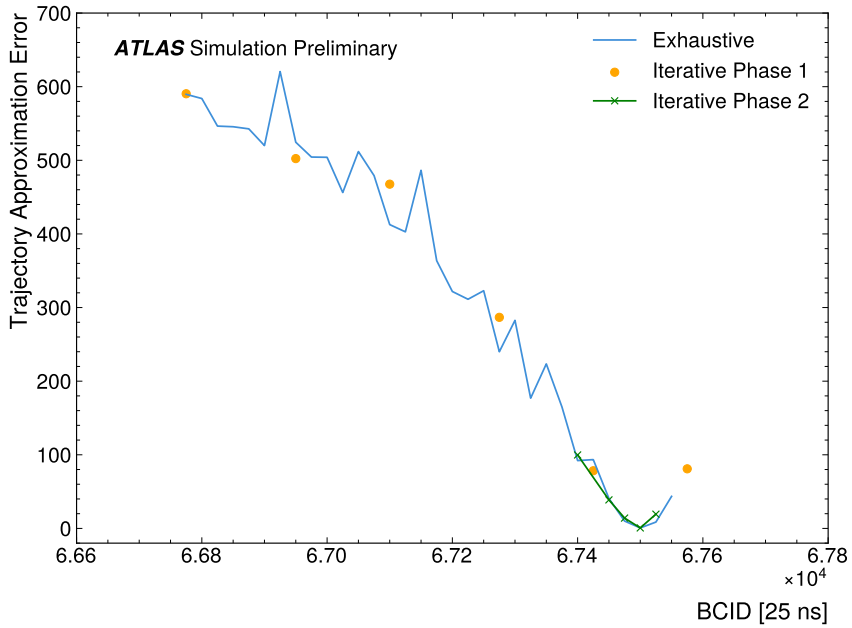


Figure 2.9 Comparison of the trajectory approximation error (E) across discrete BCID samples from a single event group. The brute-force exhaustive search strategy (blue line) shows the full error landscape. The iterative search strategy minimizes the number of evaluations by first evaluating a small subset of BCID values (Phase 1, orange dots), followed by a discrete gradient approximation that takes a limited number of steps towards the local minimum (Phase 2, green crosses).

2.2.3 Baseline Evaluation and Limitations

In previous sections, the theoretical and algorithmic foundations of the BCID-conditioned RANSAC algorithm were established. This section focuses on evaluating its performance, comparing the introduced BCID search strategies, and demonstrating the inherent limitations related to the complexity of the task. The evaluation utilizes a single-muon simulated dataset containing 100 000 unique simulated events generated from test vectors. A high-level summary of this dataset is provided in the following Table 2.2.

Dataset Parameter	**Value**
Cache Identification String	`class_local_num_per_station_100k`
Unique Simulated Events	100 000
Total Extracted Event Groups	233 514
Station 0 Event Groups	89 127
Station 1 Event Groups	70 876
Station 2 Event Groups	73 511
Train/Test Split	80/20

Table 2.2 Summary of the single-muon 100 000-event dataset utilized for benchmarks and comparison of the baseline algorithms.

It is important to note that while the dataset described in Table 2.2 contains 100 000 unique event indices, each of the events can interact with multiple stations. As our algorithms work on a per-station level, always receiving an event group in a single

station, the resulting number of total event groups (*total inputs*) is larger than 100 000.

Benchmark process During the benchmarking process, data is passed to the algorithms in a unified manner, where each algorithm always evaluates the complete event group simultaneously. While a sequential event-streaming approach would more closely resemble real-world conditions, this method was discarded because it introduces significant architectural complexity and difficulties when comparing the performance of baseline algorithms with advanced machine learning models, particularly during important noise-robustness experiments that require a non-trivial noise-injection procedure. Consequently, to maintain a unified experimental approach across different models, all experiments presented in this work artificially provide the entire event group to the algorithms as a single batch.

BCID Deviation Metric As previously established, the primary objective is to replace continuous time-of-passage estimation with discrete BCID estimation to improve the interpretability of the results. Because of this, it is beneficial to define an evaluation metric, called **BCID deviation** (denoted as ΔBCID), which is calculated as follows:

$$\Delta\text{BCID} = \text{Predicted BCID} - \text{Target BCID}. \tag{2.6}$$

Let $\Delta\text{BCID}(x)$ represent the BCID deviation of a model for a given input event group x from the dataset X, where $|X|$ denotes the total number of event groups. To evaluate model performance, let us define two useful accuracy metrics, $\Delta^0\text{BCID}$ and $\Delta^1\text{BCID}$ as:

$$\Delta^0\text{BCID} = \frac{|\{x \in X \mid \Delta\text{BCID}(x) = 0\}|}{|X|}, \tag{2.7}$$

$$\Delta^1\text{BCID} = \frac{|\{x \in X \mid |\Delta\text{BCID}(x)| \leq 1\}|}{|X|}. \tag{2.8}$$

These represent the proportion of exact predictions and predictions within a single BCID of the true value, respectively.

BCID Search Strategy Comparison The initial experiment evaluates and compares the predictive accuracy of the brute-force exhaustive and the optimized iterative BCID search strategies. The BCID deviation metric (ΔBCID) is computed for each input event group to evaluate the predictive accuracy of the model. The distributions of the deviations are illustrated in the following Figure 2.10.

Figure 2.10 also shows that the exhaustive approach naturally results in a higher accuracy compared to the iterative strategy. However, it is important to reiterate that the optimized iterative algorithm consumes roughly only one-third of the computational resources required by the exhaustive approach. The absolute accuracies for this experiment can be seen in Table 2.3.

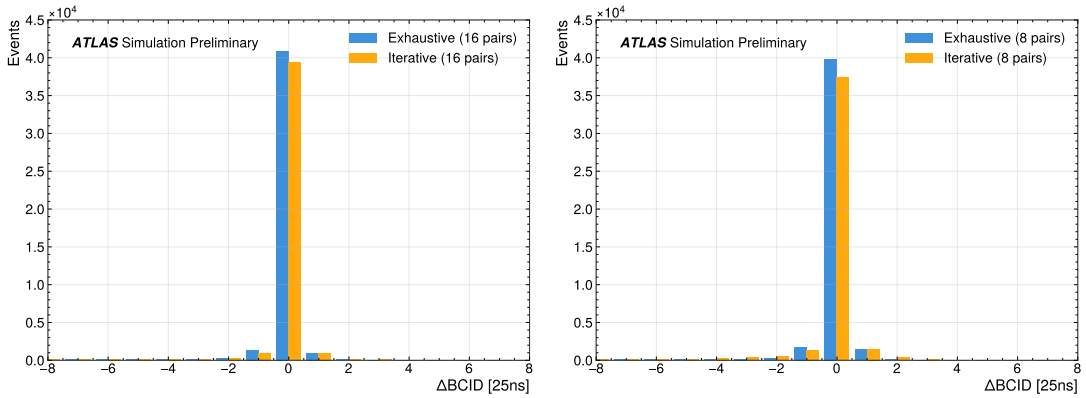


Figure 2.10 Histogram of the predicted BCID deviation (ΔBCID) relative to the true BCID of the event for the two baseline search strategies: Exhaustive (blue) and Iterative (orange). The evaluation using the BCID-conditioned RANSAC algorithm was performed using 16 stochastically sampled MDT pairs (left) and 8 sampled pairs (right) per each iteration.

Strategy	**Pairs Sampled**	Δ^0BCID	Δ^1BCID
Exhaustive	16	87.61%	92.54%
Exhaustive	8	85.37%	92.26%
Iterative	16	84.32%	88.45%
Iterative	8	80.08%	86.04%

Table 2.3 Comparison of the BCID estimation accuracy between the exhaustive and iterative search strategies. The evaluation was performed on a test set of $|X| = 46\,700$ event groups introduced in Table 2.2.

Baseline Algorithm Pathological Regions Using the global coordinate information of each MDT, the evaluated event groups can be mapped back into the geometry of the ATLAS detector. This reconstruction is beneficial because it allows us to visualize the performance of the baseline algorithm in specific regions of the detector and where the performance may be inherently degraded.

As observed in Figure 2.11, misclassifications (yellow and red colors) are not distributed uniformly. The visualization points to a central region of the detector, where the baseline algorithm repeatedly struggles to predict the correct BCID value. The same trend can be supported using the Figure 2.12.

The root cause of this performance limitation comes from the geometric nature of the baseline RANSAC algorithm and the specific metric utilized to measure the accuracy of the candidate trajectory. At trajectory angles between approximately 23° and 40° relative to the vertical axis, the muon path is specifically aligned with the regular grid structure of the tubes in the MDT chamber. Such scenarios create a degenerate state in which multiple different BCID hypotheses generate drift circle configurations that all perfectly align with the common tangent trajectory (such an event group can be seen in Figure 2.13). Because of this, the resulting landscape of the trajectory approximation error function (E) becomes nearly flat with multiple comparable local minima. In the context of this work, these cases are called **pathological** cases.

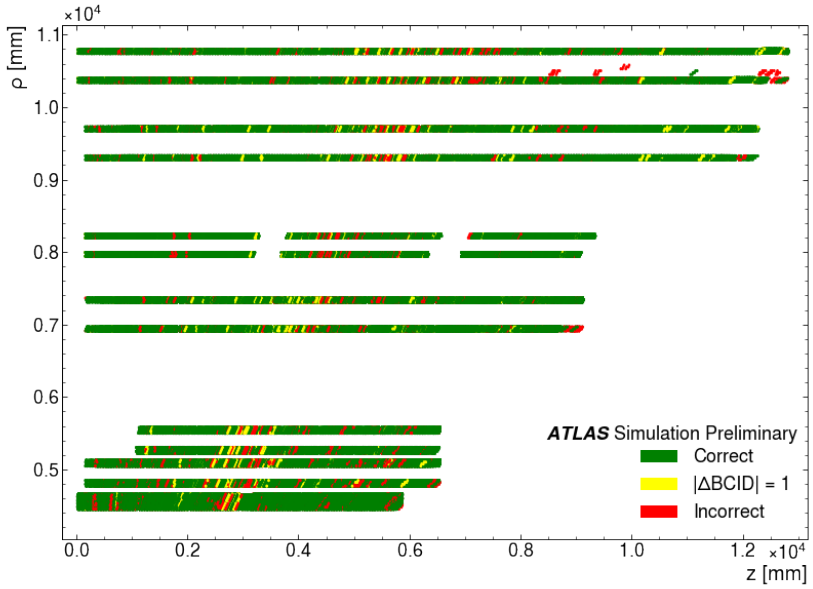


Figure 2.11 Illustration of the spatial distribution of BCID estimation accuracy in the ATLAS detector. The MDT tubes are color-coded based on the baseline algorithm's prediction accuracy on their respective event groups, with green indicating correct predictions ($\Delta\text{BCID} = 0$), yellow indicating close predictions ($|\Delta\text{BCID}| = 1$), and red indicating incorrect predictions with a BCID deviation greater than one.

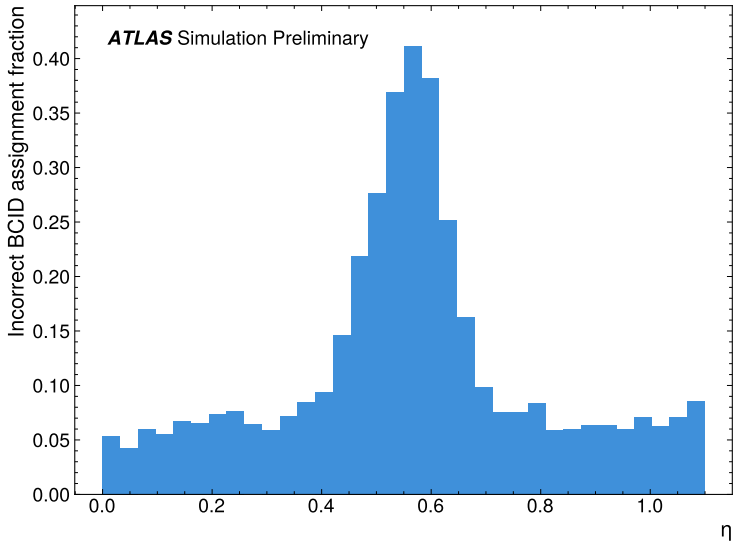


Figure 2.12 Visualization of the fraction of incorrect BCID assignments as a function of pseudorapidity (η). The histogram indicates degraded predictive performance of the baseline algorithms in the central $0.4 < \eta < 0.7$ region (approximately from 23° to 40° off the vertical axis). The results correspond to the experiment from the color-coded visualization in Figure 2.11.

Motivation for Machine Learning Architectures Because the baseline algorithm evaluates hypotheses based only on the geometric trajectory approximation error (E), it is not capable of distinguishing between the true trajectory and other geometrically valid trajectories. This inherent geometric limitation motivates the exploration of

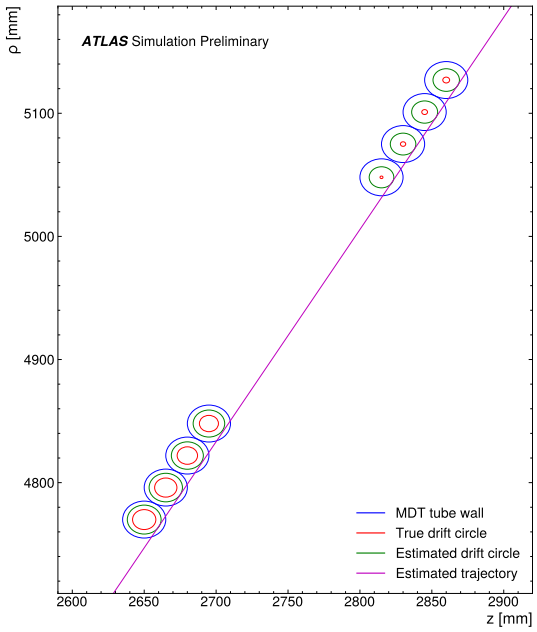


Figure 2.13 Visualization of a **pathological** event group on which the geometric baseline algorithms fail. Due to the alignment of the muon path with the arrangement of tubes in the MDT chamber, an incorrect BCID hypothesis produces estimated drift circles (green) that perfectly align with the estimated trajectory (magenta). This configuration returns a near zero trajectory approximation error, which is why the algorithm selects it as the optimal solution.

more advanced predictive architectures that might be capable of finding a more complex structure beyond a purely geometric approach.

Furthermore, the baseline algorithm has multiple inherent hardware-level constraints. The algorithm relies heavily on multiple procedures, such as the common tangent or orthogonal line-point distance computations (Equation 2.3), which require complex mathematical operations, including division, square root, and pseudo-random number generation. These operations are computationally resource-intensive and difficult to implement efficiently on FPGA architectures [29, 30]. This limitation provides a further motivation for the experiments with machine learning models, which can approximate these complex tasks while relying mainly on a series of matrix multiplications and simple non-linear activation functions, both of which are simpler to optimize for deployment on FPGAs.

Noise Robustness Experiments The final experiment of the evaluation framework assesses the robustness of the models against increasing amounts of artificial noise injected into the input event groups. The detailed description of the method and the results of these noise-robustness evaluations are presented in a later Chapter 3, which compares the developed geometric baseline algorithms with the advanced machine learning models.

2.3 Convolutional Neural Networks

As demonstrated at the end of the previous section, the implementation of the baseline algorithmic approach is constrained by its susceptibility to complex pathological

cases and by its reliance on resource-intensive operations that are highly inefficient to implement on the FPGA hardware. The transition to advanced machine learning models is motivated by their inherent ability to overcome both of these limitations.

Deep learning architectures can approximate complex models using a limited set of mathematical operations, such as matrix multiplications and non-linear activation functions, that are well-defined and efficiently implementable on FPGA hardware. Together with advancements in model quantization pipelines, such as HGQ (introduced in Section 1.5.3), these complex models can be further compressed to fit the strict memory and latency restrictions of the target hardware.

This section begins a detailed description of the various machine learning architectures researched for this task. It first introduces the initial motivation behind selecting Convolutional Neural Networks (CNNs), followed by a definition of the data preparation and augmentation pipeline. The section concludes with the introduction of relevant architectures and the evaluation of corresponding experiments.

CNN Motivation During the initial architectural assessment, primary consideration was given to two task-specific constraints: the variable input length caused by the variable number of MDT hits per single muon event, and the geometric layout of the event groups, which was already established during the development of the baseline algorithm.

Given these parameters, Convolutional Neural Networks (CNNs, formally introduced in Section 1.5.2) were selected for evaluation as the initial candidate for a viable machine learning architecture. This choice was supported by their proven use in similar physics applications at CERN. Notable use-cases include the CMS *DeepCore* project for high p_T jet tracking [18], and the project on particle classification in the liquid-argon detector of the *ProtoDUNE* experiment [19]. The latter is particularly relevant for this work, as it utilized a specialized *Submanifold Sparse* convolutional layer to efficiently process input data with a high degree of sparsity.

2.3.1 Data Preparation

To process MDT input event groups with a standard CNN architecture, the variable-length signal data must first be transformed into a unified representation of fixed dimensions. The MDT chamber cross-section was mapped onto a discrete 2D rectangular grid. Initial experiments evaluated multiple pixel resolution configurations, such as 32×32, 16×16. However, these configurations were prone to information loss during the transformation, as the mapping was able to contract multiple physical tubes into the same grid pixel. To remedy this, further evaluations of various grid dimensions were performed. Rectangular resolutions of 20×8 and 24×8 (demonstrated in Figure 2.14) were selected as the optimal trade-off between grid size and the probability of information loss occurring, with approximately 2% and 0.007%, respectively. In this representation, spatial information is encoded in the coordinates of the activated pixels, while a single channel stores the timing information of the corresponding MDT hit.

To normalize the input feature, the corresponding timing information is transformed using the following formula:

$$\tilde{t}_i = \frac{t_i - t_{\min}}{t_{\text{drift}}} \quad \forall i \in \text{Active Event MDTs}, \tag{2.9}$$

where t_i is the signal detection time of the tube i, $t_{\min}$ is the earliest recorded signal detection time within the current event group, and t_{drift} is the maximum electron drift time in the tube, which corresponds to approximately 720 ns. This normalization is computed independently for each event group to preserve their relative timing information.

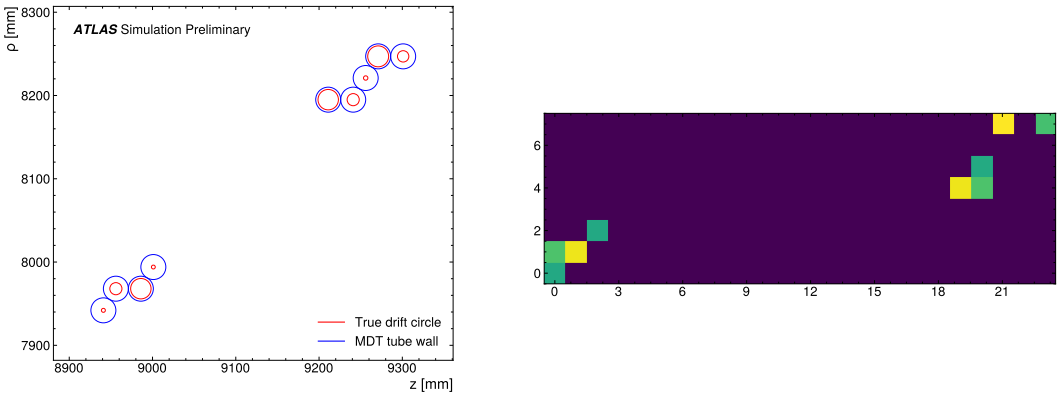


Figure 2.14 Demonstration of mapping of an even group to a discrete 2D rectangular grid, where a single channel stores the signal detection information of the corresponding MDT hit.

Input Sparsity Although the optimized rectangular grid mapping significantly reduced information loss, it highlighted another challenge inherent in grid representation, which is a high degree of input data sparsity. Because the trajectory of a muon intersects only a small number of MDTs within a chamber, the number of active tubes per event is generally limited. Consequently, the rectangular mapping constructs the input image with a vast majority of empty zero-valued pixels. Standard CNN architectures are generally not well-suited to handle sparse inputs, as large regions of zero values severely degrade the feature extraction.

Spatial Blurring To combat the sparsity of the grid representation, the application of spatial blurring techniques was explored. The objective was to artificially expand the spatial representation of an individual hit such that a denser spatial context is provided to the convolutional kernels. However, this approach introduced another trade-off between input density and the degree of corruption of the signal timing information, where active pixels will no longer strictly correspond only to the actual muon hits. Multiple blurring kernels were evaluated and their application is demonstrated in Figure 2.15.

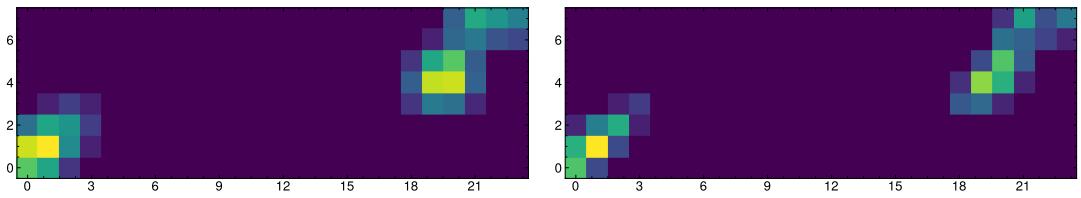


Figure 2.15 Application of two different blurring kernels on the event group previously shown in Figure 2.14. The left input utilizes the standard 3×3 Gaussian blur kernel, the right input utilizes a custom 3×3 *directional* kernel (both of which are defined in Equation 2.10).

$$K_{\text{Gauss}} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad K_{\text{Directional}} = \frac{1}{12}\begin{bmatrix} 0 & 1 & 2 \\ 1 & 4 & 1 \\ 2 & 1 & 0 \end{bmatrix} \tag{2.10}$$

Two specific 3×3 blurring kernels were further evaluated: a standard Gaussian kernel and a custom *directional* kernel (formally defined in Equation 2.10). In contrast

to the uniform Gaussian blur, the directional kernel was constructed to explicitly favor the typical direction of a muon trajectory through the MDT chamber, propagating the information primarily along this direction.

2.3.2 Architectural Exploration

Having established the data preparation pipeline, an evaluation of various CNN architectures was performed. The initial focus was directed towards well-established and optimized state-of-the-art architectures, which are aligned with the computational and memory restrictions of this task. Specifically, minimal variants of *ResNet* and *MobileNet* were selected for the assessment.

The following paragraphs briefly introduce the characteristic ideas of each selected architecture. The text focuses on specific defining features relevant to this work, while the reader is referred to the corresponding technical papers for a detailed description.

Residual Networks (ResNet)

ResNet [31] is a foundational architecture in the field of computer vision that established a modern paradigm for training deep convolutional networks. At the time of its introduction, it outperformed the competition in basically every computer vision challenge by allowing the training of deeper networks than was previously feasible.

The foundational paper presents a degradation phenomenon in standard deep models, in which, beyond a certain number of layers, deeper networks stop improving their accuracy compared to their shallower counterparts, with subsequent deepening decreasing their performance further. ResNet overcomes this bottleneck by introducing a *deep residual learning* framework with its basic building block (illustrated in Figure 2.16).

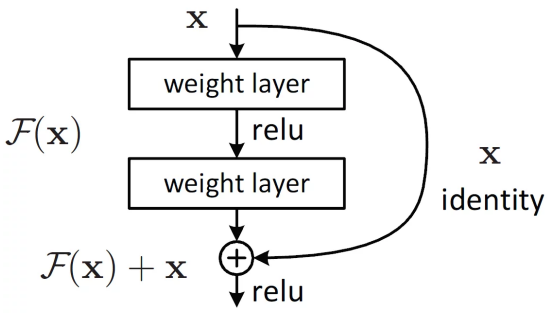


Figure 2.16 Diagram of the building block of *residual learning* [31].

As illustrated in Figure 2.16, the building block utilizes *shortcut connections* (also referred to as *skip connections*) that skip one or more convolutional layers, performing either identity mapping or a simple projection via a 1×1 convolution when dimensional matching is required. These connections are the main mechanism for the improvements in the training of deeper networks. By providing a path for gradient flow unrestricted by weights or non-linear activations, they combat the vanishing gradient problem during backpropagation and generally stabilize the training process of deep architectures.

This work evaluates multiple architectures that take advantage of the ResNet paradigm of residual building blocks with skip connections.

MobileNet

MobileNet is a family of CNN architectures for computer vision tasks, specifically optimized for low latency and a small memory footprint. This efficiency enables

their deployment in resource-constrained environments, such as mobile hardware and embedded systems, which was the primary motivation for selecting MobileNets as a candidate for this evaluation.

Since its introduction, multiple versions of the MobileNet architecture have been published, iteratively improving on the previous models. This evaluation focuses on ideas introduced in the first three iterations (*MobileNetV1* [32], *MobileNetV2* [33], and *MobileNetV3* [34]), since subsequent versions focus primarily on neural architecture search applications, rather than major structural improvements. The following text outlines the distinct improvements introduced by the three iterations, a combination of which will form the evaluated configuration in this work.

MobileNetV1 The innovation of the initial architecture is the replacement of standard convolutional layers with *depthwise separable convolutions* [32]. This procedure factorizes the convolution into two steps: a depthwise convolution, which applies a single convolutional filter independently to each input channel, and a 1×1 convolution (referred to as *pointwise convolution*), which combines the filtered outputs of the depthwise convolution.

A standard convolution has the effect of simultaneously filtering features and their subsequent combination across channels. The depthwise separable convolution separates these two steps, allowing a radical reduction in computations by a factor of 8 to 9 (while using 3×3 convolutional kernels) compared to the standard convolution.

MobileNetV2 The second iteration [33] builds on top of the depthwise separable convolutions and the paradigm of shortcut connections from the ResNet architecture by introducing the *inverted residual with a linear bottleneck* block.

Standard residual blocks (such as those in deeper variants of ResNet) that utilize a bottleneck architecture project a high-dimensional input representation (large number of channels) into a low-dimensional space for multiple convolutional blocks before expanding it back to a higher dimension. The inverted block reverses this, first projects to a higher dimension using an efficient 1×1 convolution (increase in size controlled by an *expansion factor* parameter), then performs the lightweight depthwise separable convolution, and finally projects the features back to a lower-dimensional space (illustrated in Figure 2.17).

Crucially, the non-linear activation function (typically ReLU) is removed from the final projection forming the *linear bottleneck*. This architectural choice was implemented to prevent information loss caused by the application of non-linearity to low-dimensional spaces (for the rigorous reasoning, the reader is referred to the foundational publication [33]).

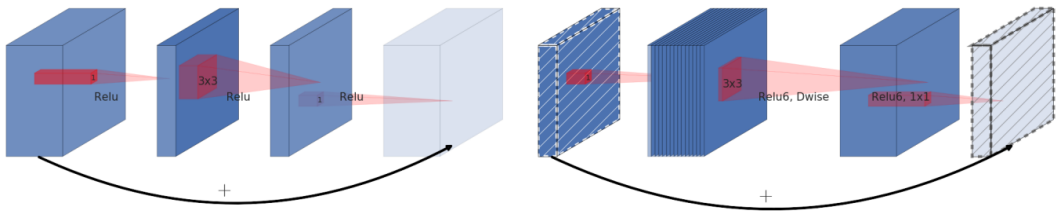


Figure 2.17 Comparison of the standard residual block (left) and the inverted residual block with linear bottleneck (right). The thickness of each block indicates the relative number of channels, and hatched blocks highlight layers utilizing only a linear activation function [33].

Furthermore, the structure of the inverted residual makes the architecture well-suited for deployment in memory-restricted environments. Because the entire block is bounded by the 1×1 projections, the system never has to fully materialize large intermediate tensors, allowing the computation of the entire block in minimal parts that fit in a small memory footprint.

MobileNetV3 Lastly, in the third iteration of the MobileNet architecture [34], the core inverted residual block is expanded with a *Squeeze-and-Excitation* (SE) module. The SE is a lightweight attention module added into the inverted bottleneck block after the depthwise convolution, introducing a channel-wise attention mechanism to highlight important features with only a minimal parameter overhead.

Furthermore, this iteration addresses the computational cost and quantization-friendliness of the model and utilizes non-linear functions. The MobileNetV3 block replaces expensive non-linear operations, such as swish (utilized for intermediate activation) and sigmoidal function (in SE block), with their piece-wise linear approximations *hard-swish* and *hard-sigmoid*.

However, it is important to note that the inclusion of the SE module introduces a performance bottleneck compared to the second version. Because the attention mechanism relies on global average pooling, it must process the full expanded block in the middle of the inverted bottleneck block before the attention weights can be applied. This disrupts the computation by minimal parts that originally made the inverted bottleneck well-suited for deployment on memory-bound devices.

Subsequent evaluations will be performed using both the MobileNetV2 building blocks and the blocks augmented with the SE module.

2.3.3 Experimental Setup and Evaluation

To evaluate the performance of the selected CNN architectures in a controlled and efficient manner, the experiments will be divided into two parts. The initial phase isolates the impact of various data representations. Two ResNet-style architectures with varying degrees of complexity are evaluated against all three input image configurations established earlier: the clean mapping and two types of blurring kernels. In the second phase, the best-performing data representation will be selected, and the results will be compared against optimized architectures with building blocks of the MobileNet architecture.

Prediction Head and Model Target In standard CNN architectures, the majority of the networks function as a specialized feature extractor, with only the last layers (commonly referred to as a *prediction head*) mapping the extracted representation to a target prediction. This work experimented with both classification and regression heads for the final layer of machine learning models. Focusing on **BCID estimation**, the final output of the model must encode the prediction bunch crossing identifier of the input event group.

Because the BCID is a strictly discrete value, the architecture of the prediction head utilized the approach established in the baseline method. As discussed in Section 2.2.2, a BCID search window is defined by the latest possible BCID from which an event could have originated and the maximum electron drift time inside MDTs. Due to the trivial inference of the search window boundaries, a similar approach was employed to encode the machine learning model target.

For BCID estimation, the model performs classification into **32 classes**. These classes represent the specific BCID offset of the true BCID from the end of the search

window (i.e., the latest feasible BCID for the given event group). Consequently, the classification space is ordered, with class 0 indicating that the event occurred at the last possible BCID within the window, while class 31 represents the earliest possible BCID of the search window.

Gaussian Label Smoothing In standard classification tasks, target variables are typically represented as one-hot encoded vectors. During the evaluation of the model, all incorrect predictions are penalized uniformly, as predictions to nearby classes do not yield any benefit during typical classification.

However, in BCID classification, the target classes represent an ordered sequence of bunch crossing times. Consequently, the incorrect prediction can be evaluated by its distance from the true class. To implement this property of the target variable into the model's training process, and thus lowering the complexity of the classification task, a special kind of target transformation technique, known as *Gaussian label smoothing*, was implemented.

This transformation replaces the one-hot target vector with a discrete probability density function. A Gaussian curve, centered on the ground truth class index, is computed across all output class indices. The spread of this curve can be tuned using a standard deviation parameter σ. The resulting vector (demonstrated in Figure 2.18) is normalized so that it remains a valid probability distribution, utilized to compute the cross-entropy loss during model training.

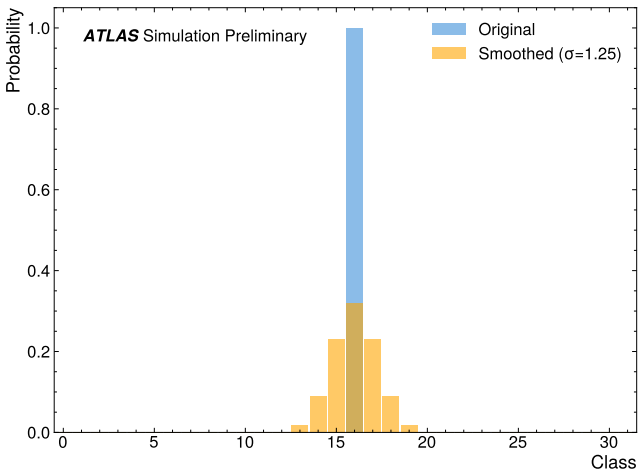


Figure 2.18 Comparison of the original one-hot target vector (blue) and the Gaussian label smoothing transformed target vector (orange) with the standard deviation parameter $\sigma = 1.25$.

The following experiments utilize the identical 100 000 event single-muon dataset established in the baseline methodology (referenced internally as `single_event_cnn_100k_24X8`). Furthermore, the transformation of event groups into 24×8 grid representations utilizes the data preparation pipeline defined previously in Section 2.3.1.

All following architectures were implemented and evaluated using the Keras API of the TensorFlow machine learning framework [35, 36]. Network optimization was performed via the *AdamW* optimizer. The optimizer was configured with a scheduled learning rate based on the *Cosine Decay* strategy, with the initial learning rate set to 0.001, decaying smoothly to 0.0001 ($\alpha = 0.1$) over the duration of the training process.

Furthermore, a weight decay of 0.001 was applied to improve model generalization. As established previously, all models were trained to minimize the *Categorical Cross-Entropy* loss, with the custom *Gaussian label smoothing* applied during the data preparation phase. During the training process, a batch size of 32 or 64 was selected. Training was typically performed for a total of 40 epochs, as initial experiments indicated model convergence with no significant improvements beyond this point. Model performance was evaluated using both the exact (Δ^0BCID) and the off-by-one (Δ^1BCID) accuracy metrics 2.2.3.

ResNet Evaluations

The initial experiments were designed to evaluate the feasibility and memory requirements of CNN architectures for this task. Following an exploratory phase, two ResNet-style architectures (detailed in Table 2.4) were selected. The larger high-capacity variant was designed to prioritize accuracy over computational efficiency (*without regard to the final hardware constraints*) and serves as a theoretical upper bound for the predictive performance of the ResNet architecture. In the second case, the smaller optimized architecture was designed to assess the viability of CNN deployment within the strict environment of the targeted hardware.

In the first phase, the larger ResNet architecture was evaluated across the three distinct input image configurations for the 24×8 grid: the clean representation (no blurring), the Gaussian blur kernel, and the directional blur kernel. The validation accuracy curves during the training process are visualized in Figure 2.19.

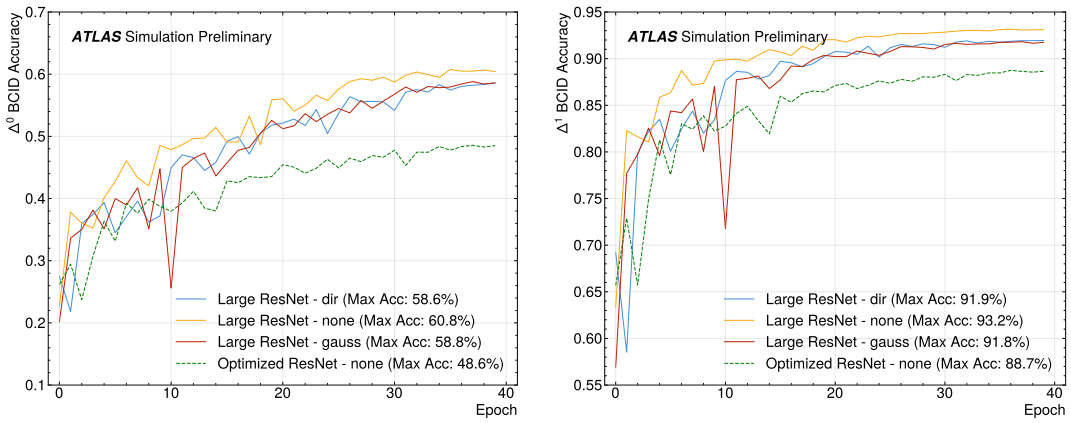


Figure 2.19 Visualization of the progression of the Δ^0BCID (left) and Δ^1BCID (right) validation accuracies during the training process of the Large and Optimized ResNet architectures defined in Table 2.4. The three solid lines distinguish between the three different input image configurations. The dashed green line shows the validation accuracy trajectory of the Optimized ResNet model, trained on the selected highest-performing clean input representation.

As demonstrated in Figure 2.19, the application of blurring kernels intended to remedy input sparsity negatively affected the predictive performance of the model. This degradation is most likely explained by the blurring process obscuring the precise timing and geometrical features, which are crucial for accurate BCID estimation. Because of this result, clean (without blurring) event group representation will be utilized in all subsequent experiments. The same Figure 2.19 additionally visualizes the training trajectory of the smaller optimized architecture, evaluated under identical conditions.

Layer	Output Size	Large	Optimized
Input	24×8	-	-
Initial Convolution	24×8	3×3, 16 channels	3×3, 8 channels
Residual Block 1	24×8	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 8 \\ 3 \times 3, 8 \end{bmatrix} \times 1$
Max Pooling	12×4	2×2, stride 2	2×2, stride 2
Residual Block 2	12×4	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 8 \\ 3 \times 3, 8 \end{bmatrix} \times 2$
Residual Block 3*	6×2	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 1$
Global Avg Pool	1×1	-	-
Hidden Fully Connected	1×1	128 units	64 units
Fully Connected	1×1	32 classes	32 classes
Total Parameters	-	**60.9k**	**10.7k**

Table 2.4 Definition of the two ResNet configurations evaluated in the initial experiments. The bracketed notation denotes the kernel dimensions and output channels of the internal block of convolutional layers, followed by the number of block repetitions. Both architectures apply one full-resolution residual block before the initial spatial subsampling (max pooling) to maximize the feature extraction from the full input representation. Following this initial step, the architecture follows the standard ResNet design paradigm. *The first convolution in Residual Block 3 utilizes a stride of 2 to perform the final spatial downsampling.

The target of the evaluation of this optimized architecture was to assess the loss of performance associated with the compression of the model towards a constrained hardware environment. Initial hardware estimates approximate that models feasible for FPGA deployment should be bounded to at most 10 000 trainable parameters. For this reason, the optimized model was designed to assess CNN performance within this parameter range.

As demonstrated by the experimental results, CNN architectures were able to extract meaningful features from the input representation and classify the exact true BCID value with an accuracy of 60.8% for the best-performing Large model. The accuracy of the Optimized architecture decreased to 48.6% following a substantial reduction in the parameter count. However, as illustrated in the right plot of Figure 2.19, the off-by-one (Δ^1BCID) accuracy reached at most 93.2% and 88.7% for the Large and Optimized models, respectively. These results confirm that the models are capable of learning the underlying task, and suggest that stronger models have the potential to further decrease the off-by-one prediction errors. The same trend, where the majority of errors fall within a ± 1 BCID window, is visualized in the BCID deviation histogram in Figure 2.20.

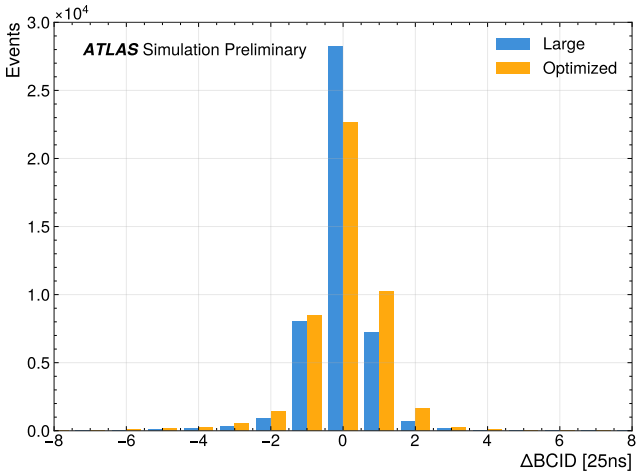


Figure 2.20 Histogram of the predicted BCID deviation (ΔBCID) relative to the true BCID for the Large (blue) and Optimized (orange) ResNet architectures. The distribution demonstrates that the models are not guessing the true class randomly and are able to roughly estimate the true BCID value, suggesting that stronger predictive models have the potential to further improve their predictive accuracy.

MobileNet Evaluations

Following the initial experiments with ResNet-style architectures, the second phase evaluates models utilizing MobileNet blocks (introduced previously in Section 2.3.2). These building blocks are designed to drastically reduce the computational complexity and total number of operations at equivalent model sizes compared to standard convolutional layers. Mirroring the design strategy of the initial experiment, two configurations presented in Table 2.5 were evaluated.

During the translation of the ResNet architecture into MobileNet equivalents, it was necessary to configure the *expansion factor* parameters. This hyperparameter dictates the ratio between the block's input channels and the dimensions of its internal feature representation. Although the foundational MobileNet papers [33] typically use an expansion factor of 6, a factor of 4 was selected for this implementation, as the resulting model size more closely matches that of the corresponding ResNet counterparts. Furthermore, max pooling layers were removed from the architectures. In line with the approach of modern CNN design paradigms, MobileNets exclusively utilize strided convolutions (stride > 1) for spatial downsampling.

The goal of this design process was to implement architectures with parameter counts comparable to those of their ResNet counterparts, while utilizing the computationally efficient inverted residual blocks. The following evaluation, conducted under experimental conditions identical to those of ResNets, compares the performance of the two architectural designs.

As demonstrated in Figure 2.21, MobileNet-like architectures achieve predictive accuracies that are highly comparable to the ResNet baselines from the initial experiment, while using a fraction of the total computations due to the efficiency of depthwise convolutions.

Further modifications to the network design, such as reducing the *expansion factor* of the inverted residual blocks, allow for an overall reduction in model size. However, even for the specialized blocks, the fundamental machine learning fact still applies. The total number of trainable parameters strongly correlates with the predictive capacity of the model itself. A substantial reduction in parameter count inevitably degrades the overall

Layer	Output Size	Large	Optimized
Input	24×8	-	-
Initial Conv	24×8	3×3, 16 channels	3×3, 8 channels
IRB Stack 1	24×8	$[16, 4, 16, 3, 1] \times 1$	$[8, 4, 8, 3, 1] \times 1$
IRB Stack 2	12×4	$[16, 4, 16, 3, 2] \times 1$	$[8, 4, 8, 3, 2] \times 1$
IRB Stack 3	12×4	$[16, 4, 16, 3, 1] \times 2$	$[8, 4, 8, 3, 1] \times 1$
IRB Stack 4	6×2	$[16, 4, 32, 3, 2] \times 1$	$[8, 4, 16, 3, 2] \times 1$
IRB Stack 5	6×2	$[32, 4, 32, 3, 1] \times 1$	-
Global Avg Pool	1×1	-	-
Hidden Fully Conn.	1×1	128 units	64 units
Final Fully Conn.	1×1	32 classes	32 classes
Total Parameters	-	**54.6k**	**10.0k**

Table 2.5 Definition of the MobileNet configurations mirroring the two selected ResNet-style architectures. Each **IRB** denotes an Inverted Residual Block, parametrized by five values: [*input channels*, *expansion factor*, *output channels*, *kernel size*, *stride*]. IRB blocks with strides greater than 1 are utilized instead of spatial downsampling layers, such as max pooling. Furthermore, each IRB block implements a *Squeeze-and-Excitation* (SE) attention block to improve channel-wise feature extraction.

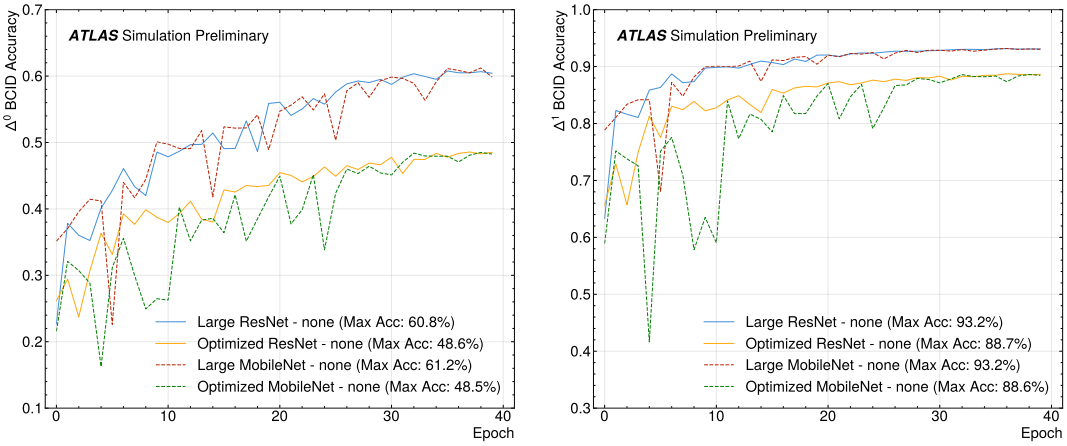


Figure 2.21 Visualization of the trajectories of validation accuracy during the training process of ResNet baselines (solid line) and their MobileNet equivalents (dashed lines). The left figure shows the exact (Δ^0BCID) accuracy of models, while the right figure shows the off-by-one (Δ^1BCID) accuracy. All models were trained using the clean input group representation (with no spatial blurring effects), as established in the initial experiment.

performance of the model. This phenomenon is demonstrated in the following evaluation presented in Figure 2.22, where the Large MobileNet architecture was restricted by decreasing all internal expansion factors from the baseline 4 to smaller values.

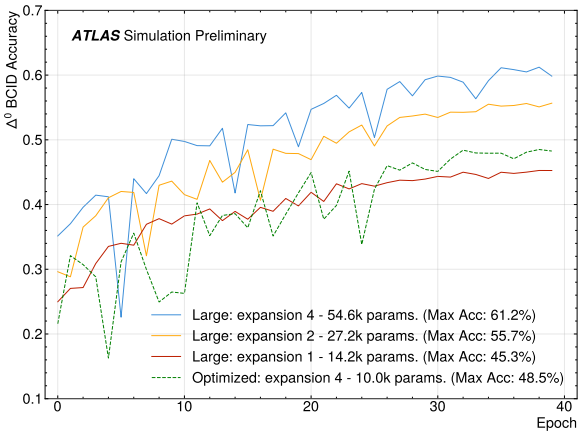


Figure 2.22 Illustration comparing the validation (Δ^0BCID) predictive performance of the Large MobileNet architecture (defined in Table 2.5, when systematically restricted by decreasing all internal expansion factors to 2 and 1. The legend displays the resulting total parameter count for each configuration. For the baseline, the performance of the Optimized model is also plotted (dashed green line).

As demonstrated in Figure 2.22, while restricting the expansion factor within the Large MobileNet architecture reduces the total number of parameters, it degrades the overall predictive accuracy of the model. Reducing the expansion factor from 4 to 2 approximately halves the model size to 27.2k parameters and lowers the maximum accuracy achieved to 55.7%. Subsequent reduction of the expansion factor to 1, which effectively removes the expansion phase from the inverted residual block, results in a model with 14.2k parameters that achieves an accuracy of only 45.3%. It is important to point out that the Optimized architecture, despite containing fewer parameters than the most restricted Large model, achieves a higher maximum accuracy of 48.5%. This result demonstrates the importance of the expansion phase within the MobileNet blocks, as well as highlights a potential limitation introduced by applying depthwise convolutions to highly sparse data.

Limitations of CNN Architectures

All evaluations point to the fundamental machine learning principle that a model's total parameter count strongly correlates with its predictive capacity. This was demonstrated throughout the experiments, where smaller architectures corresponded to lower predictive accuracy. While the implementation of specialized computational building blocks, such as MobileNet's inverted residual blocks, reduces the computational requirements of an architecture at a given size, it does not mitigate the model's capacity bottlenecks when scaling the total parameter count down, targeting hardware deployment.

We hypothesize that extreme input sparsity fundamentally limits the efficiency of the models. The architecture must spend a significant amount of computational resources processing completely empty regions, which seriously hinders the feature extraction process. For these reasons, the work moves away from convolutional neural networks towards an alternative machine learning architecture that is better suited to handling

sparse geometric data. The architecture will be further introduced and evaluated in the following Section 2.4.

2.4 Graph Neural Networks

As established in the previous section, convolutional neural networks were fundamentally limited by input sparsity, an artifact introduced during the 2D data preparation pipeline. The resulting raw event group representation was almost entirely empty, with only a small number of active pixels surrounded by large neighborhoods of zero values.

To address this exact problem, this work pivots towards **Graph Neural Networks (GNNs)** (formally introduced in Section 1.5.2). Mirroring the structure of the CNN section, this section begins by introducing the core motivation behind the graph-based approach. Subsequently, the graph-building process (Section 2.4.1), which is the primary data preparation phase for GNNs, is detailed. Section 2.4.2 then introduces a model based on the standard *message-passing* paradigm before defining a novel, custom GNN architecture designed specifically around the restrictions of this task. The section concludes with an evaluation of the predictive performance of the final models, alongside approaches implemented to enhance model robustness against input noise.

GNN Motivation The primary motivation for transitioning from Convolutional Neural Networks to Graph Neural Networks lies in their theoretical relationship (illustrated in Figure 1.13). The standard 2D CNN can be understood as a specialized, restricted case of a GNN, processing a fixed grid of nodes, connected by edges based on the size of the kernel. In the case of CNNs, the neighborhood of a given pixel is defined by its position in the pixel grid and the size of the convolutional kernel. For example, the standard 3×3 convolutional kernel centered on a given pixel forces the exchange of information to flow between the central pixel and its 8 immediate neighbors.

GNNs generalize this approach by allowing for the dynamic building of arbitrary neighborhoods. Compared to the CNN data preparation approach, which forced the data into a rigid 24×8 grid with only a small minority of active pixels, the construction of arbitrary neighborhoods connects only the nodes that contain meaningful features. Formally, GNNs efficiently represent input event groups using a graph $G = (V, E)$, where V is a set of nodes (vertices) that exclusively represent active MDTs of the event group, and E is a set of edges defining relations between distinct tubes.

By dynamically constructing the neighborhood, the representation discards all inactive MDTs, allowing the network to focus its expressive power purely on the features within the active tubes. Furthermore, the model's computational resources are no longer wasted on empty regions and are utilized exclusively for processing the actual muon hits, eliminating the severe sparsity bottleneck that the CNNs suffered from.

Furthermore, the transition to graph-based architectures is supported by their recent successful applications at CERN for tasks including particle track classification and event filtering. GNNs have been utilized in multiple projects, such as the *Exa.TrkX* project [22] and the *GNN4ITK* particle tracking pipeline [23].

2.4.1 Graph Building

To process the raw event data with a Graph Neural Network, the input data must first be transformed into a graph structure on which the network operates. Due to the **permutation invariance** and **permutation equivariance** properties of GNNs (as established in the previous Section 1.5.2), the graph design is basically unrestricted,

allowing the graph construction process to be highly flexible and dependent only on the specific task.

Because of the dynamic nature of this construction, together with the strict latency-constrained environment of the target hardware, careful consideration of the computational complexity of the construction algorithm is necessary. Furthermore, the algorithm must restrict its use of complex mathematical operations, such as square roots and divisions, which would be expensive to compute on FPGAs, a limitation previously highlighted in the context of the baseline geometric algorithm. Additionally, based on the past experience of the experimental team, an upper node processing limit of **16 MDT hits** per event per single detector station was established to preliminarily satisfy computational constraints.

The primary objective of the graph building pipeline is to receive a raw event group and transform it into a graph representation of the event, where both graph nodes and edges are allowed to contain relevant features. The output of this process is generally a description of the graph nodes, such as a **node feature** matrix (containing the attributes for each vertex), together with a definition of the graph interconnections, typically achieved via an **adjacency matrix** or an edge list, sometimes accompanied by a list of corresponding **edge features**.

Node Definition (V)

The graph construction process begins by defining the set of nodes V, represented as a node feature matrix $X \in \mathbb{R}^{N \times D}$, where N is the number of active MDTs in the event group, and D is the dimensionality of the node features. In this matrix, each row corresponds to an individual node, and the columns correspond to its distinct features.

To build a lightweight and FPGA-friendly representation, a minimal set of three features was selected to clearly define each active MDT:

- **Tube Number:** The horizontal index of the tube within a specific layer of MDTs.
- **Tube Layer:** The vertical index of the layer within a specific detector station (as detailed in Table 2.1).
- **Signal Detection Time:** The relative activation time of the corresponding MDT.

As described in the input space normalization in Section 2.1, both positional features (**Tube Number** and **Tube Layer**) are normalized into the $[0, 1]$ range. Based on observations of initial experiments, this normalization was further enhanced with a **Tube Layer** offset transformation, which significantly improves the spatial expressiveness of the local coordinate system. The resulting normalized local coordinate space is visualized in Figure 2.4.

The temporal feature (**Signal detection time**) is normalized across the entire event group using the same scaling approach (Equation 2.9) established for CNN architectures in the previous section.

This 3-tuple of features fully describes the position and relative detection time of each active drift tube within the event group. The transformation from the raw geometry to the discrete graph node representation is visualized in the following Figure 2.23. The corresponding feature values for this event are detailed in Table 2.6.

From a **computational complexity** point of view, the preparation of the node feature matrix is highly efficient, operating with a linear time complexity of $\mathcal{O}(N)$, where N is the number of active MDTs. The normalization procedures rely only on

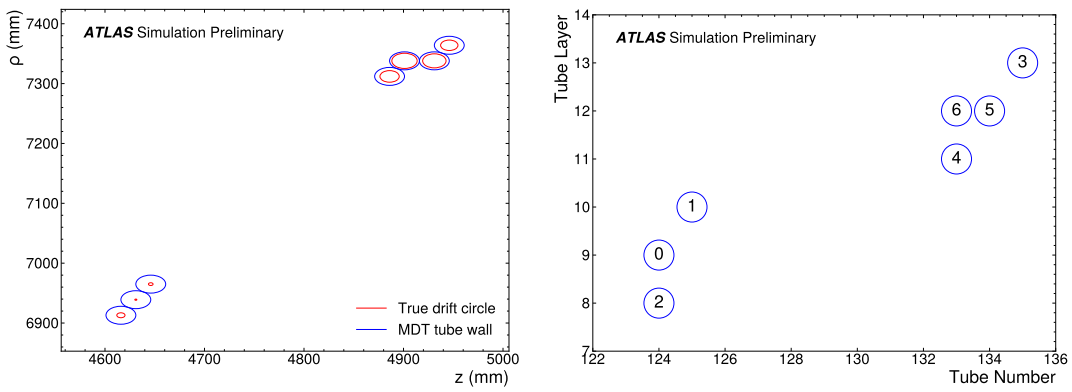


Figure 2.23 Visualizations of an original input event group in global coordinates (left) and its graph node representation in the local coordinate system (right). The node indices (0–6) represent the chronologically ordered sequence of MDT signal detection times. The specific node feature values corresponding to this event are detailed in Table 2.6.

ID	Tube Number	Tube Layer	Detection Time
0	0.292453	0.450000	0.000000
1	0.294811	0.500000	0.038512
2	0.292453	0.400000	0.094630
3	0.318396	0.650000	0.330106
4	0.313679	0.550000	0.406030
5	0.316038	0.600000	0.599692
6	0.313679	0.600000	0.710827

Table 2.6 Table of processed node features corresponding to the raw input event group visualized in Figure 2.23, demonstrating the processing of spatial and temporal features of MDTs.

trivial mathematical operations, where crucially, any required scalar division operates on a constant and thus can be precalculated as a multiplication by a reciprocal, which complies with the established FPGA latency constraints.

Edge Construction (E)

Following the definition of the node feature matrix, the next step in the graph building pipeline is to establish the procedure for constructing the set of edges E. These interconnections define the topological neighborhood of each node, which is the basis for the GNN's fundamental message-passing algorithm.

The design of the edge construction algorithm has gone through multiple iterations throughout the course of this work. In the initial phase, a naive tube proximity-based approach was implemented and evaluated. In this first design, active drift tubes were connected to their spatial neighbors based on a predefined distance threshold, evaluated in either the global coordinate system or the local integer space. Additionally, special heuristic rules were required when connections had to cross the multilayer gap of the detector station.

However, this combination presented multiple critical flaws. Due to data imperfections, such as during the introduction of injected background noise hits, the

distance-based approach became highly unstable, often resulting in the generation of disjoint graph components. This is a critical issue for GNNs, as in that case, the message-passing algorithm was unable to propagate information along the complete muon trajectory, causing the model to possibly miss vital event features. Furthermore, the necessity to compute pairwise tube distances introduced computational overhead, which could lead to latency issues during hardware deployment.

To resolve both of these flaws, a deterministic sorting-based approach was developed. Instead of calculating pairwise distances, the active MDTs are first sorted based on their local coordinates: primarily from bottom to top by their layer index, and secondarily from left to right within each layer by their tube number. An example of this ordered node representation is visualized in Figure 2.24.

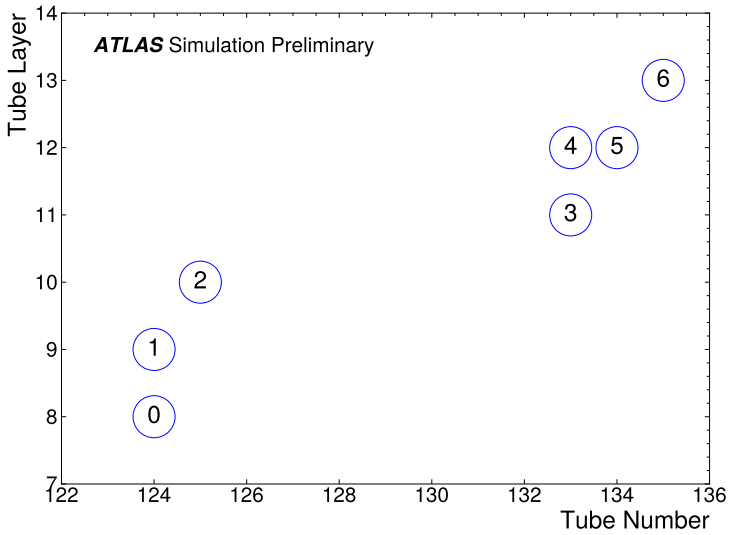


Figure 2.24 Demonstration of the sorted node representation of the input event group previously displayed in Figure 2.23. The nodes are ordered primarily from bottom to top based on their layer index, and secondarily from left to right within the same layer based on the tube number.

Once the active tubes are ordered, a deterministic rule is applied to create the interconnections: each tube is connected via a bidirectional edge to the two following tubes in the sorted sequence (or fewer, for tubes at the end of the sequence). Because of the initial spatial ordering, this rule guarantees that each tube connects to its immediate neighbor along the muon trajectory.

The reasoning for connecting to the two next closest neighbors stems from the scenario where a muon trajectory passes through a single tube in one layer, but intersects two adjacent tubes in the subsequent layer. In such cases, these dual connections improve the flow of information toward the double hit, which theoretically yields richer trajectory features for the message-passing algorithm.

This edge construction procedure builds highly uniform topological neighborhoods, which, in theory, helps the message-passing algorithm, as node updates consistently collect messages from the same number of neighbors (with minor exceptions at the boundary nodes). Furthermore, limiting the number of connections reduces the latency requirements of the following edge feature computation procedure defined next. The resulting connected graph representation is demonstrated in Figure 2.25

It is important to note that later experiments with a custom GNN architecture, introduced in Section 2.4.3, demonstrated that building node connections exclusively

between immediate neighbors is also a viable approach. Based on experimental results, this simplification did not degrade the predictive performance of the model while further reducing computational costs.

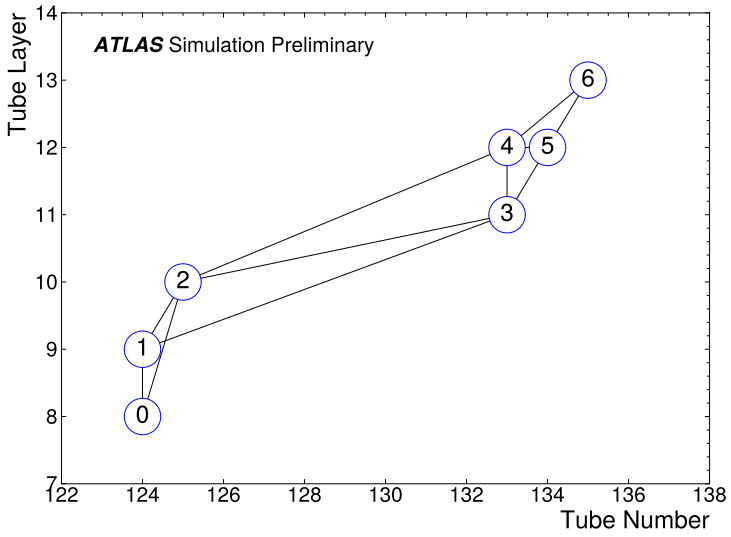


Figure 2.25 Demonstration of the resulting interconnected graph representation of the input event group previously displayed in Figure 2.23.

Edge Feature Definition

Encapsulating features inside graph edges is an efficient approach to providing the GNN model with critical information about the relations between two connected nodes. During the edge construction process, a set of features is computed for each edge based on its two connected nodes. It is important to note that the implementation in this work utilizes bidirectional edges, which require the computation of features in both directions. However, because the spatial and temporal differences are symmetric, the computation of the reverse direction requires only a sign inversion, which does not introduce additional computational overhead.

For a given directed edge connecting a source node i to a target node j, a set of three features is computed:

- **z-direction:** The z-component of the normalized direction from node i to node j.
- **ρ-direction:** The ρ-component of the normalized direction from node i to node j.
- **Drift Time Difference:** The approximate difference between the drift times of the nodes i and j.

The first two features (**z direction** and **ρ direction**) represent the direction of the connection in the global coordinate space, providing the GNN model with crucial information about the overall geometry of the problem. The features are computed as follows:

$$z_{\text{dir}} = (z_j - z_i) \cdot C, \qquad \rho_{\text{dir}} = (\rho_j - \rho_i) \cdot C, \tag{2.11}$$

where C is a normalization term, and (z_i, ρ_i) and (z_j, ρ_j) are the spatial coordinates of node i and node j, respectively.

The primary challenge in this formulation arises from the normalization term C. In an environment without computational restrictions, this term would simply be the inverse of the standard Euclidean distance between the two tubes ($C = 1/\sqrt{(z_j - z_i)^2 + (\rho_j - \rho_i)^2}$), producing a true normalized vector. However, this would require the computation of highly expensive square root and division operations for each constructed edge, which is strictly infeasible under the current computational restrictions of the targeted FPGA deployment.

Significant effort was made in optimizing the computation of this normalization term C, as multiple preliminary experiments suggested the importance of the directional feature pair for the GNN model. Multiple hardware-friendly alternatives were evaluated, such as no normalization ($C = 1$) or scaling by a precomputed maximum or average connected tube distance ($C = 1/\text{Average distance}$).

Additionally, a hybrid approximation approach was evaluated, combining a small precomputed Look-Up Table (LUT) of inverse Euclidean distances with a hardware-efficient *fast inverse square root* approximation algorithm [37]. This algorithm utilizes exclusively efficient arithmetic operations and bit-level memory manipulations to compute the approximation of the inverse square root at a substantially lower computational cost on FPGAs compared to the standard inverse square root computation. To optimize accuracy and efficiency, the small precomputed LUT stores the exact inverse Euclidean distances for nearby tubes within the regular grid of the same multilayer. The inverse square root approximation is then utilized only for complicated cases, such as edges that cross the multilayer gap or nodes connected at a distance outside of the precomputed window. This hybrid approach allows for an efficient and highly accurate approximation of the normalization term C.

The impact of these normalization strategies on the predictive accuracy of the GNN models is illustrated in the following Figure 2.26.

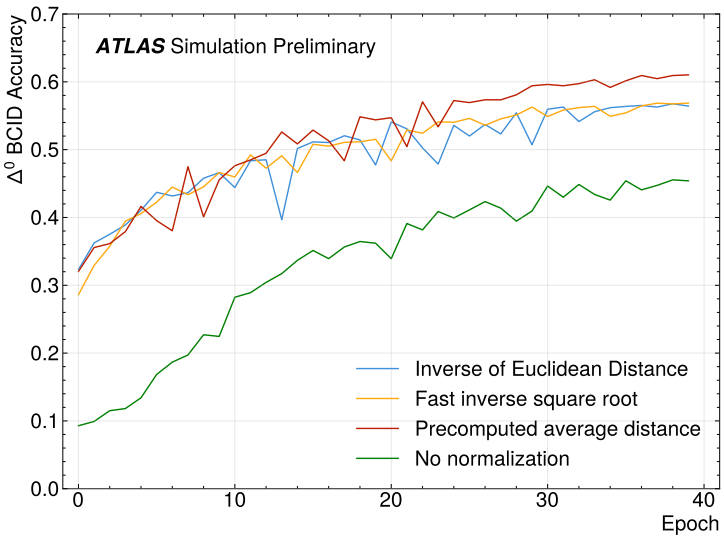


Figure 2.26 Visualization of the validation accuracy trajectories during training of four general message-passing models, differentiated only by the normalization approach to the z and ρ directional edge features. The four share identical architectures and were trained on an otherwise identical dataset consisting of 25 000 distinct events (a subset of the 100 000 event dataset utilized in previous experiments).

As demonstrated in Figure 2.26, the efficient hybrid approximation algorithm successfully achieved highly comparable results to the exact inverse Euclidean distance normalization, validating it as a viable FPGA-friendly alternative for this use-case. However, the evaluation also revealed a counter-intuitive outcome, where the imperfect and substantially simpler normalization term, utilizing a single precomputed average connected tube distance, outperformed both the exact and approximated inverse Euclidean distances.

We believe that this behavior can be explained by analyzing the magnitudes of the resulting directional feature vectors. Normalization by a global average value yields directional vectors of magnitudes less than one for nearby tubes, and greater than one for longer connections, such as those crossing the multilayer gap. In such cases, the GNN model receives a stronger signal for these longer edges. This proves to be beneficial, as these longer connections theoretically encapsulate richer temporal information, as the increased distance between the tubes results in more obvious time-of-flight differences, providing new distinct data points of the particle's trajectory. In other words, this simpler normalization approach acts as a trivial attention mechanism.

The third feature (**Drift Time Difference**) captures the temporal relationship between the two hits as the difference between the drift times of tube i and tube j. It is a crucial observation that while the true drift times are unknown prior to bunch crossing identification, the exact difference between the two drift times can be determined utilizing a modification of Equation 2.5:

$$\begin{aligned} c_i &\equiv \text{ToF}_i + \text{Electronics Delay} \\ c_j &\equiv \text{ToF}_j + \text{Electronics Delay} \\ \cancel{\text{BCID}} + \text{Drift Time}_i &= \text{Detection Time}_i - c_i \\ \cancel{\text{BCID}} + \text{Drift Time}_j &= \text{Detection Time}_j - c_j \\ \Delta\text{Drift Time}_{ij} &= (\text{Detection Time}_j - c_j) - (\text{Detection Time}_i - c_i) \end{aligned} \tag{2.12}$$

Similarly to the node feature definition, this difference is subsequently normalized by the maximum possible drift time constant (from Equation 2.9), ensuring a proper range of the feature values before GNN operations. This feature should allow the model to effectively capture the local temporal properties between the different active tubes.

Graph Building Conclusion

With the definition of node features, graph topology, and corresponding edge features complete, the full graph representation $G = (V_{\text{attr}}, E, E_{\text{attr}})$ for a given input event group is established. As introduced in the previous sections, the MDT hits are transformed into a graph structure, where the node features fully describe spatial and temporal information, and the edge features encode the relative geometry and drift time relationship between connected tubes. Additionally, every phase of this process is fully deterministic and designed to satisfy the strict limitations of the target FPGA hardware. The resulting graph representation is thus complete and prepared for feature extraction via the GNN model.

2.4.2 General Message-Passing Model

Following the definition of the **Message-Passing Neural Network (MPNN)** framework, established in Section 1.5.2, and the graph building pipeline introduced in the previous Section 2.4.1, this section focuses on the specific architecture implemented for the **BCID estimation** task. The implementation follows the standard approach of

constructing an MPNN model, focusing on architectural design to balance predictive performance with minimal parameter count to comply with the computational and latency constraints of the target environment.

As established previously, the architecture executes a fixed number of message-passing steps, in which nodes gather messages from their neighboring nodes and corresponding connecting edges, and iteratively update their own embedding vectors. Similarly to the CNN architectures evaluated earlier, the model performs classification into **32 classes**, where each class represents a 25 ns bunch crossing window, matching the target definition introduced in Section 2.3.3. Consequently, following the sequence of message-passing steps, the architecture performs a global graph readout followed by a classification head.

Message Generation and Aggregation During each message-passing step, the model first computes the messages along every directed edge. For an edge connecting a *sender* node j to a *receiver* node i, the incoming message $\mathbf{m}_{ij}$ is generated from a vector composed of concatenated sender features $\mathbf{h}_j$, receiver features $\mathbf{h}_i$, and the corresponding edge features $\mathbf{e}_{ij}$. This vector is subsequently processed by an asymmetric two-layer Multilayer Perceptron (MLP), with the first layer twice the size of the second to improve the model's capacity, utilizing hardware-aware capped Rectified Linear Unit (ReLU6, bounds the maximum output value to 6) activation functions, which is frequently utilized in efficient MobileNet architectures.

$$\mathbf{m}_{ij}^{(l)} = \mathrm{MLP}_{\mathrm{msg}}\left(\mathbf{h}_j^{(l)} \parallel \mathbf{h}_i^{(l)} \parallel \mathbf{e}_{ij}\right), \tag{2.13}$$

where $\parallel$ denotes the vector concatenation operation, and l is the index of the current message-passing iteration.

Once all individual messages are computed, they are aggregated at each corresponding receiver node. To preserve the absolute signal strength of the accumulated messages, which scales with the degree of a given receiver node, an element-wise sum aggregation is utilized. Furthermore, because of the deterministic graph construction pipeline, the maximum degree of each node is known in advance. This allows for heavy optimization efforts, such as static memory allocation and loop unrolling, prior to the hardware deployment. The element-wise sum aggregation operation is defined as follows:

$$\mathbf{m}_i^{(l)} = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}^{(l)}, \tag{2.14}$$

where $\mathcal{N}(i)$ denotes the set of neighboring nodes of node i.

Node State Update Before the initial message-passing step, the raw node features $\mathbf{h}_i^{(0)}$ are projected into the internal dimension of the model using a single dense layer. In the subsequent update phase, during step l, the current hidden node state $\mathbf{h}_i^{(l)}$ is concatenated with the aggregated message vector $\mathbf{m}_i^{(l)}$ and passed through a dedicated node update MLP, which again consists of two asymmetric dense layers with ReLU6 activations.

A residual connection (similar to those in ResNet architectures) that adds the previous node state to the output of the update MLP is typically implemented to stabilize the training across multiple message-passing steps. Subsequently, a layer normalization is applied to improve the stability of the hidden node representation:

$$\mathbf{h}_i^{(l+1)} = \text{Batch Normalization}\left(\mathbf{h}_i^{(l)} + \mathrm{MLP}_{\mathrm{update}}\left(\mathbf{h}_i^{(l)} \parallel \mathbf{m}_i^{(l)}\right)\right) \tag{2.15}$$

Graph Readout and Classification After the K message-passing steps, the final node embeddings $\mathbf{h}_i^{(K)}$ contain the properties extracted from a given tube and the context of its surrounding neighborhood of nodes and connected edges. Because the BCID estimation task requires a single prediction for the entire graph, all node embeddings are pooled into a single global graph representation $\mathbf{h}_G$.

Preliminary evaluations demonstrated performance improvements when the pooling operation retained as much structural information as possible. For this reason, the graph readout utilizes two distinct aggregation methods: an element-wise mean and an element-wise maximum. Both operations, formalized in the following equations, are applied across each feature dimension d for all nodes V within the given graph:

$$\forall d: \quad h_{G,\text{mean},d} = \frac{1}{|V|} \sum_{j \in V} h_{j,d}^{(K)}, \quad h_{G,\max,d} = \max_{j \in V} h_{j,d}^{(K)}. \tag{2.16}$$

The resulting aggregated graph embeddings are concatenated into a single global graph representation:

$$\mathbf{h}_G = \mathbf{h}_{G,\text{mean}} \parallel \mathbf{h}_{G,\max}. \tag{2.17}$$

Finally, this global representation is passed into a classification head, consisting of a readout MLP with ReLU6 activations and a small dropout layer, culminating in a dense classifier layer with a softmax activation to produce the probability distribution over the 32 BCID classes.

Training Setup

Following the approach established for CNN models, all GNN architectures were implemented using the TensorFlow framework and the Keras API. The models were optimized using the *Adam* optimizer, minimizing the *Categorical Cross-Entropy* loss function across the 32 BCID classes, with classification targets augmented by the identical *Gaussian label smoothing* technique introduced in Section 2.3.3.

The optimizer was configured with a *Cosine Decay* learning rate scheduler, with a conservative initial learning rate of 0.0005, because initial experiments suggested possible training instability in message-passing models at higher rates. The learning rate then decays to 10% ($\alpha = 0.1$) of its initial value over all epochs of the training process.

Similarly to the CNN evaluations, the standard training schedule was performed over 40 epochs, with exceptions for specific fine-tuning runs detailed in later experiments. During training, the performance of the models was again evaluated using both the exact (Δ^0BCID) and the off-by-one (Δ^1BCID) accuracy metrics, defined in Section 2.2.3

Depending on the evaluation setup, various subsamples of the large 178 000 event dataset were utilized. For this reason, each experiment will explicitly state the size of the dataset used for training and evaluation. These raw event datasets were first processed into the corresponding graph representations using the graph building pipeline, established in Section 2.4.1, before being passed to the optimizer in batches of 16 to 64.

Hyperparameter Evaluation

The previously established MPNN architecture is fully characterized by four independent structural hyperparameters:

- **Node Dimension:** The dimensionality of the model's internal node state representation. This parameter also configures the asymmetric architecture of the two

dense layers of the node update MLP, with the first layer scaled to twice the size of the node representation.

- **Edge Dimension:** The dimensionality of the model's internal message vector representations. Similarly to the node dimension, the parameter configures the asymmetric architecture of the two-layered message MLP.
- **Message-Passing Steps (K):** The total number of message-passing iterations performed by the model, defining the receptive field of each graph node.
- **Readout Size:** The configuration of the final graph readout MLP, defining both the number of hidden layers and their dimensionality.

To determine the optimal configuration, the hyperparameter evaluation was divided into two phases: the message-passing depth optimization and a wider architectural search.

Message-Passing Depth (K) Following the initial model assessment and the setting of preliminary architectural hyperparameters, the first phase of the evaluations focused on selecting the optimal number of message-passing steps. The experiment fixed the internal node and edge dimensions of the MPNN at 32, and the readout MLP was configured to a single 64-node dense layer followed by the 32-class classification head. These parameters were established during the initial model assessment to ensure a sufficiently strong model that would expose any performance gains or degradations connected to the selected depth parameter

This evaluation was performed using the 50 000-event dataset. A total of ten independent models were trained, evaluating the message-passing iteration parameter in the range $K \in \{1, 2, \ldots, 10\}$. The validation accuracy trajectories for each architecture during training are visualized in Figure 2.27.

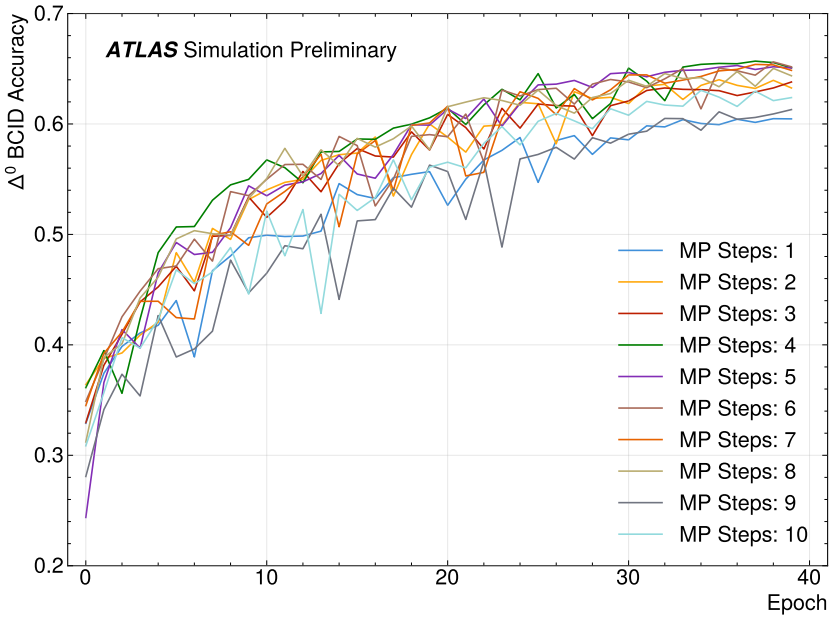


Figure 2.27 Visualization of the validation Δ^0BCID accuracy trajectories during the training process for multiple MPNN architectures differentiated by the number of message-passing steps performed.

As demonstrated in Figure 2.27, the number of message-passing steps has a significant influence on both the training stability and the final predictive performance of the model.

Shallower architectures, such as $K \leq 2$, exhibit lower accuracy and higher performance variance during training. This demonstrates the necessity of allowing the MPNN model sufficient information propagation through the graph structure before the prediction can be made.

Both performance and stability improve as the receptive field of the nodes expands, with the highest performance achieved for the $K = 4$ and $K = 5$ configurations. However, by executing further steps ($K \geq 6$), the performance shows diminishing returns, and subsequent steps further degrade the predictive performance. This is a known phenomenon in deep graph neural networks called *over-smoothing*. Repeated aggregation causes node representations to converge towards a uniform value, averaging out valuable localized information from the model.

Based on the results, the $K = 5$ configuration was chosen for its balance of model performance and training stability, and will be utilized in subsequent architectural evaluations.

Architectural Search After selecting the optimal number of message-passing steps ($K = 5$), a broader evaluation of different internal architectural configurations of the MPNN model was performed.

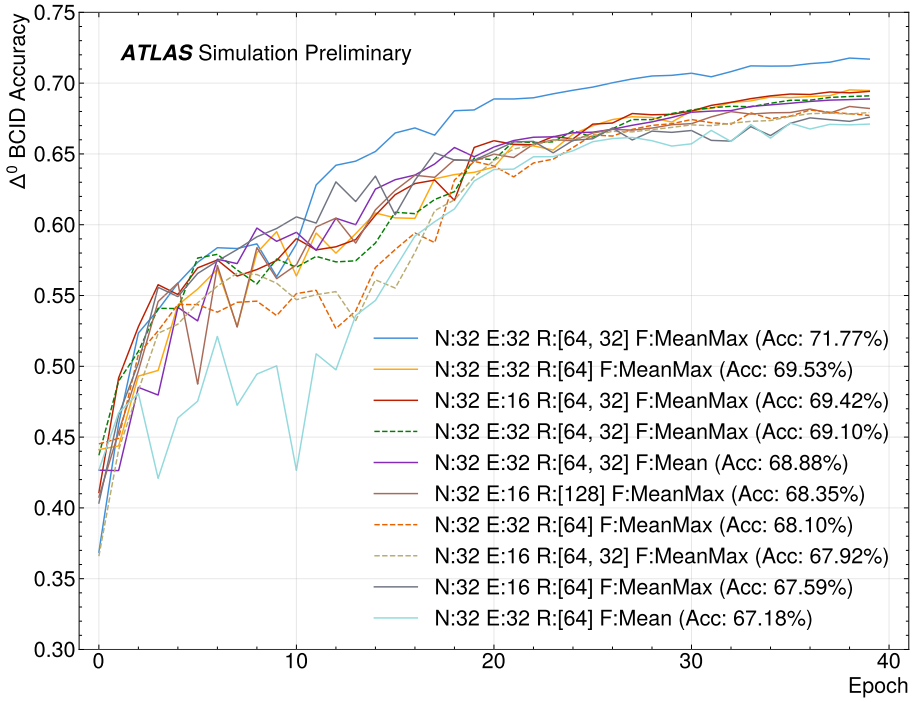
Firstly, a grid search explored the rest of the primary structural hyperparameters, evaluating different combinations of internal node dimensions, edge dimensions, and readout MLP layer sizes. A high-performing subset of these architectures was further evaluated using varying graph readout operations, comparing isolated aggregation operations against the previously established concatenated mean and max operations. Lastly, variations in the activation functions within the two-layered MLPs were tested, including replacing the last layer's ReLU6 with a linear activation to form a linear bottleneck, resembling the MobileNet blocks, to experiment with the preservation of feature variance before aggregation operations.

In total, approximately 50 distinct structural configurations were trained and evaluated utilizing the complete 178 000-event dataset. It should be noted that the grid search was performed using a preliminary dataset version that utilized the hybrid inverse square root approximation (established in Section 2.4.1) for normalization during the graph-building phase, before the better-performing normalization was established. However, since this affects the input features of all evaluated models equally, it should not affect the comparative structural configuration analysis. The training trajectories of the 10 highest-performing architectures are visualized in Figure 2.28.

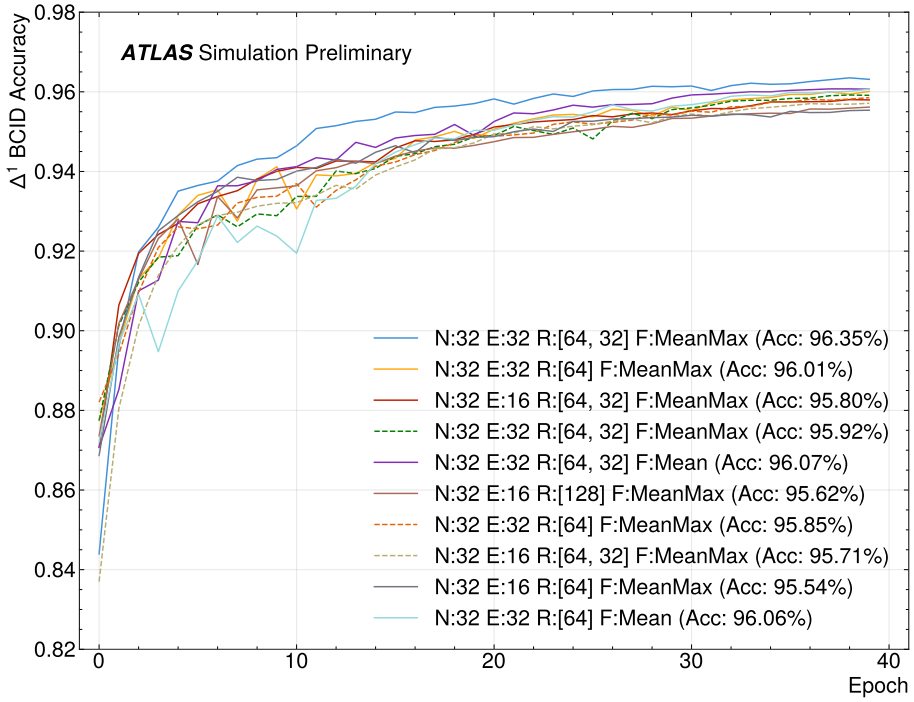
Based on the results presented in Figure 2.28, the best-performing configuration utilized node and edge dimensions of 32, ReLU6 activation functions within the update MLPs, concatenated mean and maximum graph aggregation functions during graph readout, and a two-layer readout MLP configured with 64 and 32 units. Lastly, the configuration used the $K = 5$ message-passing steps established in the previous depth evaluation. Most importantly, due to the weight-sharing mechanism facilitated by the recursive message-passing layers, this entire configuration maintains a small memory footprint of only 20 160 parameters.

This model achieved a maximum exact accuracy (Δ^0BCID) of 71.77% and an off-by-one accuracy (Δ^1BCID) of 96.35%.

The performance comparison in Table 2.7 displays the performance of the configured MPNN model against all methods previously established in this work. Its ability to outperform classical geometric approaches in the off-by-one metric, while maintaining a relatively small memory footprint, highlights the capabilities of the GNN-based approach.



(a) Exact Accuracy (Δ^0BCID)



(b) Off-by-One Accuracy (Δ^1BCID)

Figure 2.28 Validation accuracy trajectories for the top 10 evaluated MPNN configurations. The top figure (a) displays the exact prediction accuracy, while the bottom figure (b) displays the off-by-one accuracy. The legend (sorted by the exact accuracy for both figures) details the model configuration parameters: Node dimension (N), Edge dimension (E), Readout size (R), and the readout pooling function (F). The solid and dashed lines then differentiate between the utilization of ReLU6 or Linear activation function in the second layer of update MLPs. The best validation accuracy achieved over the 40 training epochs is displayed in parentheses for each model.

Model	Δ^0BCID	Δ^1BCID
Exhaustive (16)	87.6%	92.5%
Iterative (16)	84.3%	88.5%
CNN (Large)	60.8%	93.2%
CNN (Optimized)	48.6%	88.7%
MPNN	71.8%	**96.4%**

Table 2.7 Performance comparison of the three methodologies evaluated so far for the BCID estimation task. While the geometrical Exhaustive approach retains the highest exact accuracy (Δ^0BCID), the evaluated MPNN architecture demonstrates improved performance in the off-by-one accuracy (Δ^1BCID), significantly outperforming both classical algorithms and all established CNN configurations.

Station-Specific Fine-Tuning

Although the established MPNN architecture demonstrates definite performance gains over the baseline models, the significant difference between its exact and off-by-one accuracies suggests an opportunity for further optimization. Because the model is highly capable of predicting the bunch crossing window within a single 25 ns time bin (over 96% of cases), further specialization of the model might improve the exact predictive accuracy.

As established before, the muon spectrometer in the barrel region of the ATLAS detector consists of three distinct muon stations that occupy a region at a specific distance from the interaction point. Theoretically, each station observes distinct muon tracks, background rates, and spatial distributions of MDT hits that differ significantly from the other two stations. Consequently, these differences might yield specific distributions of event features that machine learning models can learn and use for predictions.

With the goal of exploiting these distinct feature distributions, the subsequent experiment investigates the possibility of separating the global MPNN model. The established architecture is first pre-trained on the entire dataset to train a robust baseline. The model is then duplicated into three independent networks, each fine-tuned exclusively on a dataset restricted to events from a single specific detector station.

Inner Station Fine-Tuning Evaluation To evaluate the proposed model separation, a subset of the best-performing configurations identified during the architectural search was subjected to a fine-tuning training phase. Following the initial 40 pre-training epochs, during which the models had seen the complete dataset comprising events from all three detector stations, the models underwent an additional 20 epochs of fine-tuning on a restricted dataset of events from only the innermost station. A rigorous dataset preparation pipeline was implemented to prevent any data leakage between the pre-training and fine-tuning datasets and to ensure the validity of the subsequent experiments.

Furthermore, the hyperparameters of the network optimizer were reconfigured for the fine-tuning phase in line with typical transfer learning procedures. The initial learning rate of the *Cosine Decay* scheduler was reduced to 0.0001 (20% of the initial pre-training learning rate), with the decay target again at 10% ($\alpha = 0.1$).

The validation accuracy trajectories for this fine-tuning process are visualized in Figure 2.29.

The results presented in Figure 2.29 demonstrate a definite performance gain within

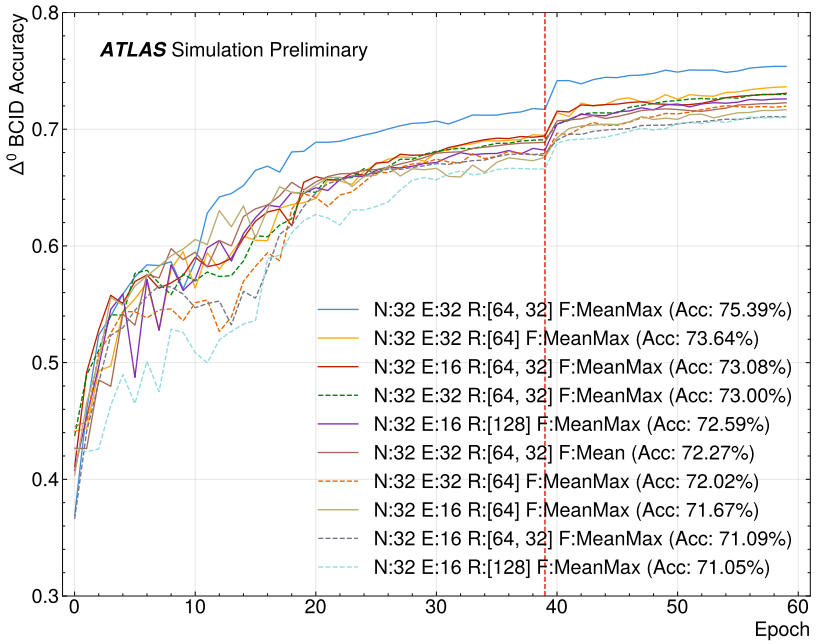


Figure 2.29 Validation exact accuracy (Δ^0BCID) trajectories for the top ten MPNN architectures during the two-stage training process (pre-training, fine-tuning). The vertical red dashed line marks the transitional epoch from global pre-training to Inner station-specific fine-tuning.

the innermost station. Following 40 pre-training epochs and the transition to the fine-tuning phase, all evaluated architectures showed improvements to their predictive performance, measured by the exact accuracy (Δ^0BCID). Specifically, the accuracy of the best-performing configuration increased from the baseline of 71.8% to a peak of 75.4%, solely due to this spatial specialization.

Station Specialization Following the successful evaluation of the fine-tuning approach at the innermost detector station, the fine-tuning was subsequently performed at all stations. To utilize the current optimal graph-building pipeline, the evaluation dataset was updated to use the optimal edge feature normalization technique.

The selected MPNN configuration (utilizing node and edge dimensions of 32, a $[64, 32]$ readout MLP, concatenated mean and max pooling, and ReLU6 activation functions) was first pre-trained on the full 178 000-event dataset, followed by duplicating the model into three independent instances ahead of station-specific fine-tuning on datasets restricted to the Inner, Middle, and Outer detector stations.

The resulting predictive performances of these specialized station-specific models are summarized in Table 2.8.

Summary of MPNN Evaluations

All previous evaluations of the Message-Passing Neural Network have demonstrated the effectiveness of graph-based event representation for the BCID estimation task. Furthermore, by transitioning from a single global model to a station-specific, fine-tuned implementation, the architecture improved its predictive performance by focusing on the distinct feature distributions of each detector station.

Detector Station	Δ^0BCID	Δ^1BCID
Global	73.39%	96.59%
Inner	78.05%	97.27%
Middle	72.78%	96.32%
Outer	71.04%	96.46%

Table 2.8 Performance summary of the specialized station-specific MPNN models across the three distinct MDT detector stations. The global model represents the accuracy baseline achieved during pre-training on the entire dataset before station-specific fine-tuning. The Inner station was the only one to exhibit significant performance improvements; the other two stations showed stale predictive performance, with a small degradation of model's accuracy.

While the memory footprint of the best-performing model is highly efficient at approximately 20 000 parameters, hardware deployment requires reducing the structural complexity as much as possible, even before applying quantization frameworks such as HGQ (introduced in Section 1.5.3). During this thorough structural review, several critical hardware deployment bottlenecks were found.

Firstly, the inherent structure of the graph representation yields a standard MPNN model with a dynamic memory footprint, as the graph size varies with the number of active MDT hits in the input event group. This variability presented challenges already during the TensorFlow implementation, as dynamic tensor shapes do not allow efficient model precompilation procedures. While this issue can be remedied during the FPGA implementation phase by using input padding, it introduces computational inefficiency that can negatively affect the final model performance.

Secondly, the use of edge features, which are significant for the effectiveness of the model predictions, forces the message-generation process to dynamically gather node features based on specific adjacency indices. Such dynamic memory access operations significantly increase the complexity of the FPGA implementation and the overall latency of the model. This behavior severely complicates the implementation and synthesis of a highly memory-efficient and low-latency FPGA model.

To mitigate these hardware challenges, this work introduces a custom **Line Graph Neural Network** architecture, detailed in the following section 2.4.3.

2.4.3 Line Graph Architecture

The **Line Graph Neural Network (LGNN)** architecture is introduced to directly address multiple bottlenecks that severely complicate the deployment of general Message-Passing Neural Networks on FPGA hardware.

Line Graph Definition Let us start with a theoretical introduction. Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges connecting pairs of distinct vertices. The line graph of G is a graph, denoted as $L(G) = (V_L, E_L)$, constructed as follows:

1. For each edge in G create a vertex in $L(G)$:

$$V_L = E$$

.

2. Two vertices in $L(G)$ are connected by an edge if and only if their corresponding edges in G share a common vertex:

$$E_L = \{\{e_i, e_j\} \mid e_i, e_j \in E \wedge e_i \cap e_j \neq \emptyset\}$$

.

Architectural Motivation This architecture is fundamentally inspired by the efficiency of **Graph Convolutional Networks (GCNs)**. GCNs are a specialized subset of MPNN models designed for input graph representation that does not contain explicit edge features. In such cases, a single message-passing step can be performed via a highly effective matrix multiplication of the graph adjacency matrix (A) with the node feature matrix (X).

This reformulation greatly reduces the complexity of the required operations at each message-passing step and completely eliminates the need for dynamic node feature gathering during the message-generation phase. However, this efficiency comes at the cost of fully discarding the edge features from the graph representation. Because previous evaluations demonstrated significant benefits of edge features for the BCID estimation task, this work implements a novel transformation approach that allows the network to retain these information-rich graph edges while preserving the highly simplified and efficient matrix multiplication-based message-passing operations of the standard GCN.

To enable an efficient message-passing process without discarding the crucial edge feature representation, this architecture introduces two modifications to the general MPNN model: a paired graph representation and a custom GCN-style message-passing layer.

Paired Graph Representation

The original graph building pipeline, detailed in Section 2.4.1, is extended to compute a secondary graph. The process begins by constructing the primary graph $G = (V, E)$ to represent the raw input event group. Subsequently, the pipeline constructs the corresponding line graph $L(G) = (V_L, E_L)$, encoding the edge features computed during the construction of the primary graph. Let $|V|$ and $|E|$ denote the number of active nodes and edges of the primary graph. The construction process yields the following matrices:

- **Node Feature Matrix (X_V):** A real matrix $\mathbb{R}^{|V| \times D_V}$ of node features, where D_V is the dimension of the node feature vector.
- **Edge Feature Matrix (X_E):** A real matrix $\mathbb{R}^{|E| \times D_E}$ of edge features, where D_E is the dimension of the edge feature vector.
- **Primary Adjacency Matrix (A_V):** A square matrix $\{0,1\}^{|V| \times |V|}$ defining the spatial connections between the vertices of the primary graph G.
- **Line Graph Adjacency Matrix (A_L):** A square matrix $\{0,1\}^{|E| \times |E|}$ defining the spatial connections between the vertices of the line graph $L(G)$
- **Incidence Matrices ($B_{\text{snd}}, B_{\text{rcv}}$):** Rectangular matrices $\{0,1\}^{|V| \times |E|}$ acting as a bridge between the two graph representations, allowing the flow of information between the primary graph and the line graph. An entry $B_{\star,i,j} = 1$ if the primary node v_i is a *sender* or a *receiver* of the original edge e_j, and 0 otherwise.

Graph Construction Complexity As established, no additional feature computations are necessary during the construction of the line graph. Because the graph building phase itself is a crucial latency bottleneck for GNN hardware deployment, a careful analysis of the additional computational complexity, introduced by the construction of a second graph representation, was performed. The line graph construction process is divided into three sequential phases:

1. **Initialization:** Allocation of the line graph adjacency matrix (A_L) and incidence matrix (B). Because the deterministic graph construction pipeline established maximum representation sizes ahead of time, these matrices are assumed to be statically preallocated, resulting in a runtime allocation complexity of $O(1)$.

2. **Mapping Edges:** Iteration through the set of edges E to map edge indices to their corresponding primary nodes and to fill out the incidence matrices. As the matrix and the array assignments are executed in $O(1)$, the complexity of this step is $O(|E|)$.

3. **Building Line Graph Adjacency:** Iteration through the set of edges E and checking the constructed edge mapping for common vertices. Because the search space is strictly bound by the maximum out-degree D of a node, the complexity of this step is $O(|E| \cdot D)$.

Overall, the runtime computational complexity of the line graph construction process is $\boldsymbol{O(|E| \cdot D)}$. As previously established, each input event group contains at most 16 MDT hits ($|V| \leq 16$) per event per detector station. Furthermore, with the maximum out-degree of each node of $D = 4$, defined by the connection rules in Section 2.4.1, the primary graph will contain a maximum of 64 edges. This theoretical upper bound limits the maximum number of operations required to build the line graph representation.

GCN-Style Message-Passing

With the paired graph representation defined, the message-passing operations are executed exclusively via highly effective matrix multiplications. Before the message-passing process begins, the raw input features (X_V, X_E) are projected into the model's internal representation (H_V, H_E) via dense linear layers. Each message-passing step l consists of a GCN-style update of the line graph representation, followed by an update of the primary graph, using the incidence matrices to allow for the flow of information between the two graphs. Both update MLPs utilize an identical asymmetric setup with ReLU6 activation functions in the hidden layer and linear activation in the last layer, a combination that has demonstrated better performance with the modern pre-normalization update scheme. Furthermore, they employ residual connections and layer normalizations to stabilize the training of the architecture, mirroring the configuration of MPNN models described in Section 2.4.2.

Edge Feature Update The message-passing pipeline first updates the line graph (edge features) by aggregating messages from adjacent edges through the line graph adjacency matrix. Furthermore, the context of each given edge is enhanced by the node features mapped into the edge space via transposition of the incidence matrices, a process resembling the message construction in Equation 2.13:

$$M_E^{(l)} = A_L H_E^{(l)}, \tag{2.18}$$

$$M_{V \to E}^{(l)} = B_{snd}^{\mathrm{T}} H_V^{(l)} + B_{rcv}^{\mathrm{T}} H_V^{(l)}. \tag{2.19}$$

These aggregated messages are concatenated with the current edge state $H_E^{(l)}$ to form the combined representation $C_E^{(l)}$. Multiple variations of Equation 2.19 were evaluated, with the summation yielding the optimal trade-off between the model's size and performance, compared to strictly concatenating the sender and receiver vectors.

To stabilize the training, the layer utilizes a pre-normalization residual block, inspired by many modern deep learning architectures:

$$C_E^{(l)} = \left[H_E^{(l)} \parallel M_E^{(l)} \parallel M_{V \to E}^{(l)} \right], \tag{2.20}$$

$$H_E^{(l+1)} = \text{Mask}_{\text{Edge}} \odot \left(H_E^{(l)} + \text{MLP}_{\text{Edge}}(\text{LayerNorm}(C_E^{(l)})) \right), \tag{2.21}$$

where $\parallel$ denotes feature-wise concatenation, and $\odot$ is the element-wise product. Multiplication by the $\text{Mask}_{\text{Edge}}$ vector strictly zeroes out the zero-padded regions of the edge feature vectors. The addition of this masking was implemented after preliminary evaluations revealed performance degradation caused by the MLPs polluting the empty regions of feature vectors with non-zero noise.

Node Feature Update Subsequently, the primary nodes are updated. Notably, explicit node-to-node message-passing (via A_V) is consciously left out of this operation, as the updated edge features $H_E^{(l+1)}$ already contain the node-level interactions, presented in edge embeddings via Equation 2.19. Consequently, computing redundant messages would introduce unnecessary computational overhead. The node update relies completely on gathering the previously updated incoming and outgoing edge features:

$$M_{E \to V}^{(l)} = B_{snd} H_E^{(l+1)} + B_{rcv} H_E^{(l+1)}. \tag{2.22}$$

Similarly to the edge update, these aggregated messages are concatenated, normalized, and passed through a dedicated node update MLP:

$$C_V^{(l)} = \left[H_V^{(l)} \parallel M_{E \to V}^{(l)} \right], \tag{2.23}$$

$$H_V^{(l+1)} = \text{Mask}_{\text{Node}} \odot \left(H_V^{(l)} + \text{MLP}_{\text{Node}}(\text{LayerNorm}(C_V^{(l)})) \right). \tag{2.24}$$

Graph Readout and Classification Following the graph readout strategy of the MPNN model, after K message-passing steps, the final node representations $H_V^{(K)}$ are aggregated to form a single graph-level embedding. This readout utilizes the concatenation of the mean and max aggregation operations introduced in Equations 2.16 and 2.17. The resulting embedding is processed by a dedicated readout MLP before being passed into the final classification head to yield the probability distribution over the 32 BCID windows.

Consequently, these modifications significantly simplify the message-passing pipeline, utilizing exclusively highly efficient matrix multiplications. Most importantly, by precisely defining the size of input matrices, the entire pipeline unifies and restricts the memory footprint required for each prediction. This deterministic memory footprint is a critical advantage for future hardware synthesis, enabling extensive latency optimizations through static memory preallocation.

Model Evaluation and Performance

Because the primary goal of the LGNN architecture is to implement an optimized, hardware- and quantization-friendly alternative to the standard MPNN model, the

configuration of the training environment, the dataset, and the baseline hyperparameters are kept identical to those established for the MPNN model in Section 2.4.2.

Consequently, this section evaluates the structural advantages introduced by the paired graph representation. The following experiments focus on an LGNN-specific message-passing depth analysis, comparing the model's predictive performance against the previously established models.

Baseline Comparison During the preliminary architectural assessment, multiple combinations of structural configurations were explored. The major architectural difference between the LGNN and the baseline MPNN models is in the use of a linear activation function at the last layer of the update MLPs.

Theoretically, the linear activation at the output of the MLP creates a linear bottleneck, analogous to the architecture of MobileNet blocks. Together with the pre-normalization approach utilized in the LGNN update blocks, the linear projection allows for longer feature retention as negative features are not immediately zeroed out before the addition of the residual connection. This update resulted in noticeable improvements in the predictive accuracy of the LGNN.

To provide a fair benchmark, the LGNN architecture was initially evaluated using a hyperparameter configuration identical to that of one of the best-performing MPNN models explored during the architectural search detailed in Section 2.4.2. The network optimization setup and the dataset remained identical to the MPNN training setup described in Section 2.4.2. The result of this evaluation, which compares the LGNN architecture against the top ten MPNN architectures across the full dataset, is demonstrated in Figure 2.30.

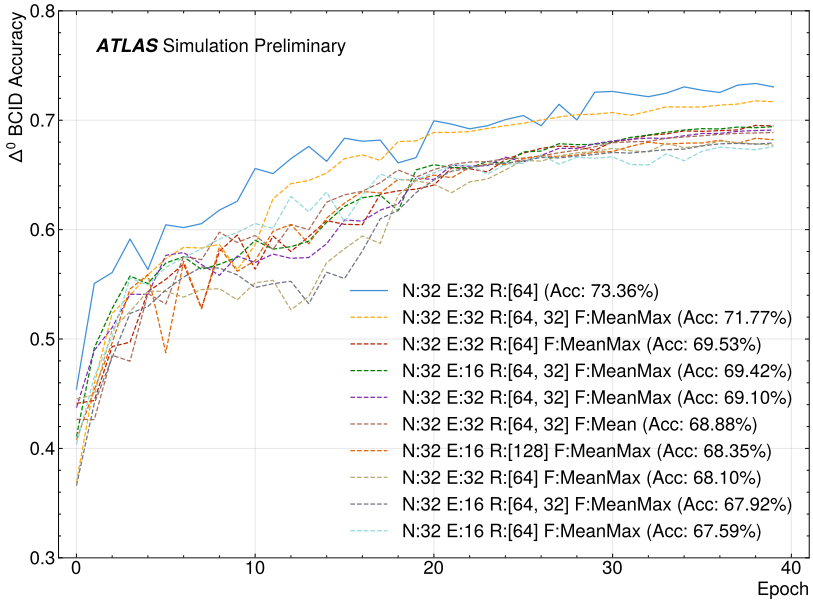


Figure 2.30 Visualization of the validation exact accuracy (Δ^0BCID) trajectories over 40 epochs of the training, comparing the initial LGNN model (solid line), with the already detailed top ten MPNN configurations from the architectural search (dashed lines).

As demonstrated in Figure 2.30, the initial LGNN configuration, utilizing node and edge feature dimensions of 32, internal linear bottleneck blocks, and $K = 5$ message-passing iterations, concluded with a readout MLP that has a single dense layer of 64 units

(a total parameter count of 21 344), was able to surpass the previous best-performing MPNN baseline in the exact accuracy metric. Specifically, the LGNN achieved a peak Δ^0BCID accuracy of 73.36%, compared to the top MPNN model with a peak accuracy of 71.77%. Furthermore, it improved the baseline off-by-one error (Δ^1BCID) from the previous best of 96.35% to 96.65.

It is important to note that, to perform a fair analysis, the dataset utilized for this evaluation was identical to the one employed during the MPNN architectural search. Consequently, the graph edge features were processed using the same hybrid inverse square root approximation algorithm during the graph-building phase.

Maximum Out-Degree Optimization Although successful in the initial evaluation, a careful analysis of the model's computational footprint raised concerns about the substantial memory requirements of the input matrices passed to the model for each prediction.

The largest matrix utilized during the model's prediction is the line graph adjacency matrix (A_L). The size of this matrix is dictated by the maximum number of MDT hits per event ($|V| \leq 16$) combined with the deterministic connection rules of the graph building pipeline, which yields a maximum node out-degree of $D = 4$. While artificially reducing the maximum number of MDT hits would reduce the size of the input matrices, this approach risks discarding important detector information. For this reason, the next experiment evaluates optimization by restricting the maximum node out-degree.

Consequently, this evaluation introduces a modified connection strategy into the graph building pipeline. Following the initial sorting of the MDT hits, detailed in Section 2.4.1, each node is connected only to its single immediate neighbor in the node sequence, rather than to its two neighbors. This results in a reduction of the maximum out-degree of each node to $D = 2$, corresponding to a 75% compression of the line graph adjacency matrix from the initial dimensions of 64×64 down to 32×32.

The following evaluation demonstrates the performance impact of this modification to the connection strategy. The evaluation was performed using an LGNN model configuration identical to the previous baseline comparison experiment. The results of this evaluation are visualized in the following Figure 2.31.

As demonstrated in Figure 2.31, the modification of the graph building pipeline did not introduce any performance degradation on the clean single-event dataset. Because this modification provides significant improvements in latency and memory footprint without an immediate sacrifice in predictive power, all subsequent LGNN evaluations will use graph representations with a maximum node out-degree of $D = 2$.

Impact on Message-Passing Depth (K) With the paired graph representation with the node maximum out-degrees constrained to $D = 2$, the subsequent evaluation investigates the impact of varying the number of message-passing iterations (K) for the LGNN architecture.

In a standard graph representation, such as the one used for the MPNN architecture, increasing K expands the receptive field of each node, enabling information to propagate to distant nodes of the graph. The paired representation, utilized by the LGNN, improves this by creating explicit connections not only between nodes, but also between edges. By enabling the flow of information through both of these representations simultaneously (via incidence matrices), the LGNN can capture a wider receptive field within a single message-passing iteration compared to the standard MPNN implementation.

To evaluate this theoretical advantage, the predictive performance of this architecture was analyzed across a varying number of message-passing steps ($K \in \{1, 2, 3, 4, 5\}$) in

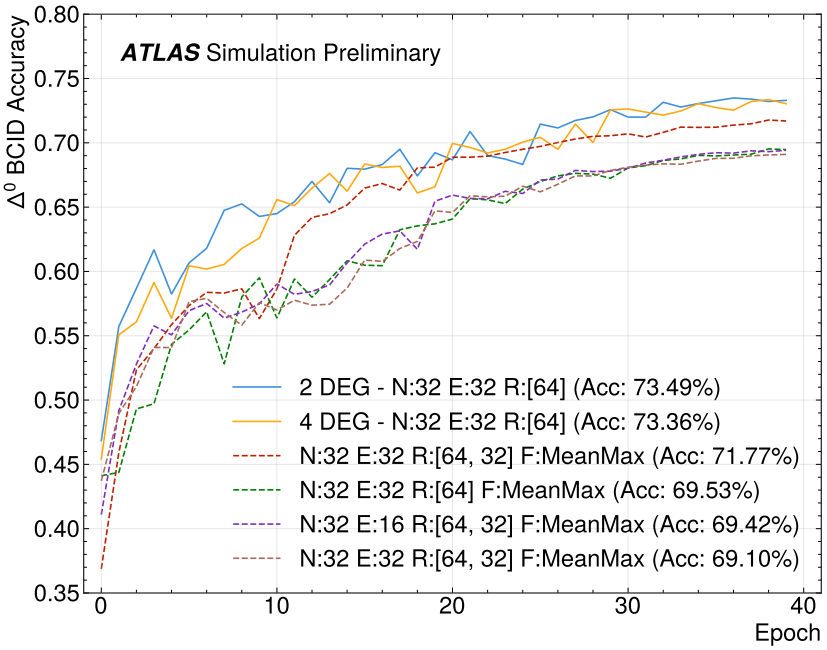


Figure 2.31 Visualization of the validation exact accuracy (Δ^0BCID) trajectories over 40 training epochs. The comparison displays the two identical LGNN configurations (solid lines) differentiated only by their maximum node out-degree ($D = 4$ vs. $D = 2$) within the internal graph representation, plotted alongside a subset of the best-performing MPNN models (dashed lines) selected through the architectural search.

multiple LGNNs with an otherwise identical structural configuration. The results of this analysis are visualized in Figure 2.32

As illustrated in Figure 2.32, the paired graph representation, combined with the LGNN message-passing pipeline, requires significantly fewer iterations to achieve optimal predictive performance. This confirms that routing information through both the node and edge graphs facilitates faster and more effective propagation of relevant features. Compared to the MPNN depth evaluation presented previously in Figure 2.27, the LGNN architecture reduces the required number of message-passing steps for optimal performance by roughly a factor of three.

Because the number of message-passing iterations (K) presents a primary latency bottleneck during FPGA hardware deployment, minimizing this parameter is a major objective. These results demonstrate that, thanks to the efficiency of the paired graph representation and the LGNN architecture, the model maintains high predictive performance even when the number of iterations is drastically reduced to as little as $K = 2$ or $K = 1$.

In the subsequent model optimization phase, introduced in Section 2.5, this message-passing depth will be systematically optimized alongside the rest of the architectural hyperparameters to identify appropriate configurations that balance model accuracy with strict hardware constraints.

Station Specialization Mirroring the final evaluation of the MPNN architecture, this experiment evaluates the performance of the LGNN architecture during station-specific fine-tuning. The architectural configuration remains identical to the configuration established in previous experiments, specifically using the updated graph building pipeline with a maximum node out-degree of $D = 2$ and a reduced number

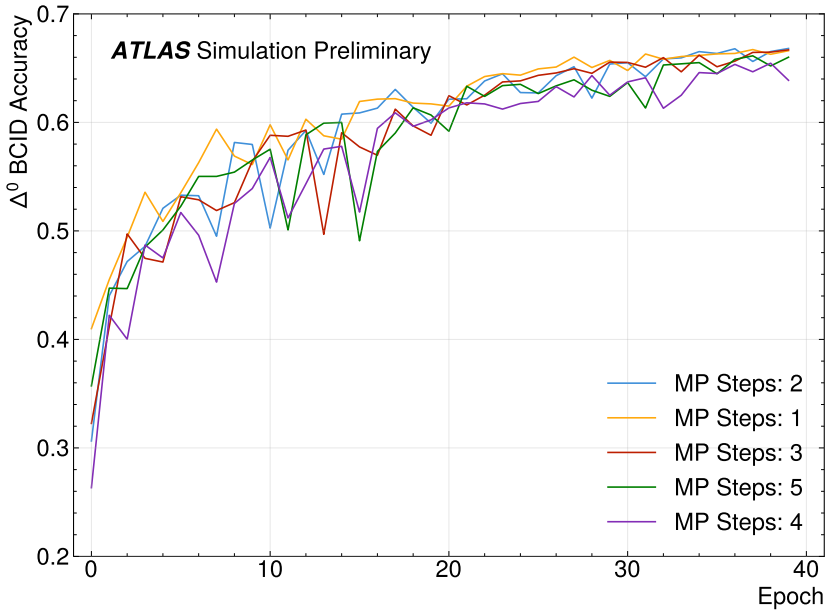


Figure 2.32 Visualization of the validation exact accuracy (Δ^0BCID) trajectories comparing the performance of the LGNN across varying numbers of message-passing steps (K). The experiment was performed on a reduced subset of 50 000 events, utilizing the previously established $D = 2$ maximum out-degree constraint.

of message-passing iterations of $K = 2$. It is important to note that while this specific hyperparameter configuration may not represent the absolute optimum, it provides a fair baseline demonstration of the LGNN architecture's capabilities.

To further improve the specialization of the network, this fine-tuning procedure introduces two modifications to theoretically enhance the performance of the fine-tuning pipeline, compared to the MPNN evaluation. Firstly, the Gaussian label smoothing process will be disabled during the fine-tuning phase, encouraging the model to converge towards the true target distribution. Secondly, the training batch size is reduced by 50% compared to the pre-training phase, to reduce the regularization effect of large batch sizes, theoretically providing the optimizer with more accurate gradients.

The results of this station-specific fine-tuning, compared with the predictive performance of models previously established in this work, are detailed in the following Table 2.9.

As demonstrated in Table 2.9, the LGNN consistently outperforms the MPNN model across all detector stations, while simultaneously being significantly better suited for hardware implementation.

During the pre-training phase, the LGNN achieved a peak Δ^0BCID accuracy of **75.17%**, improving the performance of the MPNN model by almost 2%. This performance difference was maintained throughout the station-specific fine-tuning phase, with the LGNN reaching a peak exact accuracy of **79.50%** in the Inner detector station.

Most importantly, these performance gains were achieved using the strictly constrained paired graph representation and a shorter, simpler message-passing pipeline ($D = 2$, $K = 2$), which establishes the LGNN architecture as the leading candidate for future hardware deployment.

Model	Detector Station	Δ^0BCID	Δ^1BCID
Exhaustive (16)	Global	87.6%	92.5%
Iterative (16)	Global	84.3%	88.5%
CNN (Large)	Global	60.8%	93.2%
CNN (Optimized)	Global	48.6%	88.7%
MPNN	Global	73.39%	96.59%
	Inner	78.05%	97.27%
	Middle	72.78%	96.32%
	Outer	71.04%	96.46%
LGNN	Global	**75.17%**	96.92%
	Inner	**79.50%**	97.25%
	Middle	**75.18%**	96.68%
	Outer	**73.37%**	96.89%

Table 2.9 Comprehensive comparison of peak validation exact (Δ^0BCID) and off-by-one (Δ^1BCID) accuracies. The table benchmarks the station-specific fine-tuning results of the LGNN architecture against the Geometrical and globally trained CNN methods, as well as the fine-tuned MPNN architecture.

2.5 Model Optimization

With the definition of the **Line Graph Neural Network (LGNN)** established and evaluated in the previous Section 2.4.3, the final phase of model development is hyperparameter tuning. This section details the optimization approach used to simultaneously tune all LGNN structural hyperparameters, transitioning from the theoretical architectural design to highly specific and practical model deployment constraints.

The primary goal of this optimization phase is to identify specific architectural configurations that maximize the predictive performance on the BCID estimation task, while at the same time conforming to the latency and memory constraints imposed by the objective of FPGA hardware deployment.

2.5.1 Multi-Objective Optimization via NSGA-II

Designing an optimal Machine Learning architecture for trigger-level FPGA deployment presents a complex task with a variety of strict computational constraints. In a standard neural network development process, hyperparameter optimization can typically be performed through independent evaluations and manual parameter selection based on heuristics.

However, within the LGNN architecture with the inherent coupling of multiple internal feature representations, manual hyperparameter selection becomes impractical. Furthermore, when the task requires an optimal architectural configuration under strict hardware constraints in an environment operating on the smallest performance margins, manual tuning becomes severely suboptimal and inappropriate.

As the current task specification does not present a specific set of computational constraints to which the model must adhere, the goal of this hyperparameter optimization pipeline is not to produce a single optimal model. Instead, the primary objective is to present a curated selection of multiple architectural configurations, each optimized

for a specific set of resource restrictions. This approach allows for postponing the final model selection until the precise hardware limits are finalized, enabling optimal resource utilization. In multi-objective optimization, this set of optimal trade-off configurations is formally defined as the *Pareto front* (or *Pareto optimal front*).

NSGA-II Configuration

To systematically map the Pareto front, this work implements the **Non-dominated Sorting Genetic Algorithm II (NSGA-II)**, previously outlined in Section 1.5.4. This algorithm belongs to the family of Multi-Objective Evolutionary Algorithms (MOEA), designed for simultaneous optimization of multiple objective functions. NSGA-II supports the definition of multiple independent performance metrics by which every candidate architecture configuration is evaluated.

Architectural Parameters For each candidate architecture, the LGNN hyperparameters are encoded into a genome defined by four variables: the node feature dimension (N), the edge feature dimension (E), the number of message-passing steps (K), and the topology of the readout MLP (R).

Resource Estimation Because it is infeasible to directly synthesize each candidate architecture to evaluate the true hardware resource utilization at each generation of the evolutionary search, cost proxy methods were developed to estimate the model's memory footprint and its computational latency, which represent the two primary hardware constraints.

The memory footprint of a model is largely dominated by the trainable parameters. Within the LGNN message-passing pipeline, two primary memory contributors are the two-layer node- and edge-update MLPs (the internal architecture of the LGNN model is detailed in Section 2.4.3). The memory footprint of each MLP is formalized as follows:

Edge Update MLP: The edge update network (MLP_{Edge}) processes the concatenated vector of the given edge feature (size E), the aggregated edge message (size E), and the summed features of its adjacent nodes (size N). This results in an input vector of size $2E + N$. The first MLP layer projects this into the hidden space of size $2E$, requiring $(2E + N) \cdot 2E = 4E^2 + 2EN$ weights. Subsequently, the output layer projects this representation back to the target edge dimension E, utilizing $2E \cdot E = 2E^2$ weights. As a result, the total parameter count for the edge update MLP is computed as follows:

$$M_{\text{Edge}} = (4E^2 + 2EN) + 2E^2 = 6E^2 + 2EN. \tag{2.25}$$

Node Update MLP: Similarly, the node update network (MLP_{Node}) processes the concatenated node feature vector (size N) with the summed messages from its connected edges (size E). This input is projected into a hidden space of size $2N$, utilizing $(N + E) \cdot 2N = 2N^2 + 2NE$ weights. The output layer then again projects this representation down to the target node dimension N, utilizing $2N \cdot N = 2N^2$ weights, resulting in:

$$M_{\text{Node}} = (2N^2 + 2NE) + 2N^2 = 4N^2 + 2NE. \tag{2.26}$$

The combined memory footprint for a single message-passing iteration (M_{MP}) is the sum of these MLPs:

$$M_{\text{MP}} = M_{\text{Edge}} + M_{\text{Node}} = 6E^2 + 4N^2 + 4EN. \tag{2.27}$$

Following the message-passing phase, the final graph-level embedding is constructed via two graph pooling operations that concatenate the element-wise mean and maximum of all node feature vectors. This creates a single graph-level embedding of size $2N$.

This vector is then processed by the readout MLP, the topology of which is defined by the hyperparameter R. The memory footprint of this layer is computed across all its dense layers. Let $S_0 = 2N$ represent the initial input size, and let S_l represent the number of units in the l-th layer of the readout MLP. The total parameter count for the readout MLP is formalized as:

$$M_{\text{Readout}} = \sum_{l=1}^{|R|} (S_{l-1} \cdot S_l). \tag{2.28}$$

With the parameter counts for all parts of the model computed, the **Memory Proxy** (or memory cost, denoted C_{Mem}) metric is constructed as:

$$C_{\text{Mem}} = M_{\text{MP}} + M_{\text{Readout}}. \tag{2.29}$$

Furthermore, the **Computational Latency Proxy** (denoted C_{Comp}) defines the total number of hardware operations executed during a single prediction and is computed as follows:

$$C_{\text{Comp}} = (K \cdot M_{\text{MP}}) + M_{\text{Readout}}, \tag{2.30}$$

as the total number of operations needs to reflect the computations performed by the multiple iterations of the message-passing process.

By minimizing these two hardware proxies simultaneously while maximizing the predictive accuracy, measured by the Δ^0BCID metric, the NSGA-II algorithm performs a systematic and thorough search over the different relationships among combinations of LGNN structural hyperparameters.

Evolutionary Search Space To actively guide the evolutionary search toward building hardware-efficient architectures, strict boundaries were designed into the search space. The node and edge feature dimensions were restricted to a predefined set of values ($N, E \in \{4, 8, 12, 16, 20, 24, 28, 32\}$). As established during the previous LGNN message-passing depth analysis, the number of message-passing steps was also tightly restricted to $K \in \{1, 2, 3\}$.

The readout funnel topology (R) was selected from a predefined set of single- and two-layer configurations with unit counts ranging from 4 to 64 (similar restrictions to node and edge feature dimensions). To enforce the funnel topology, the two-layer readout configurations were strictly constrained so that the size of the first layer was always greater than or equal to the size of the second layer.

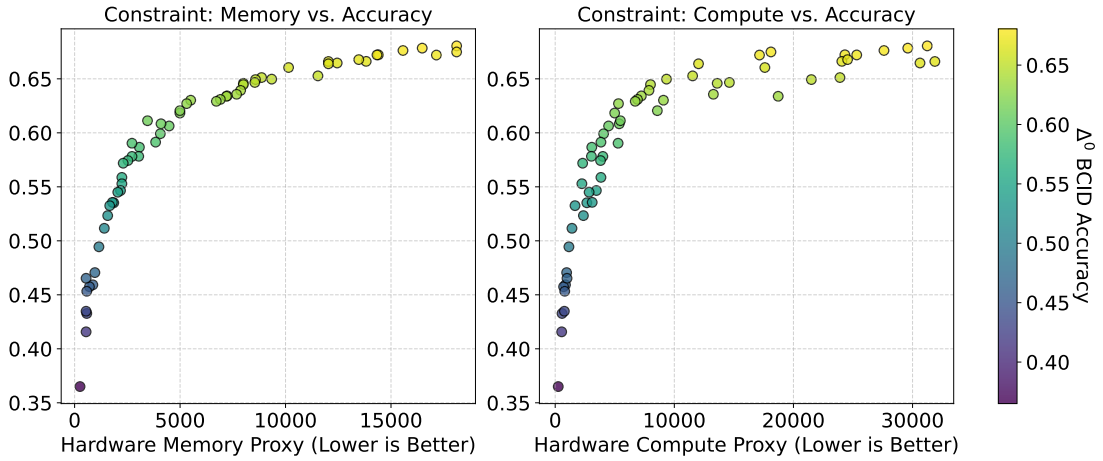
Search Execution To map the design space as much as possible, the NSGA-II algorithm was initialized with a population of 20 individuals and evolved for 50 generations.

Importantly, to evaluate the predictive performance (Δ^0BCID) of each candidate configuration, a shorter training cycle was implemented. During the evolutionary search, the LGNN model was dynamically constructed, based on the parameters specified by the given genome, and trained for 20 epochs on the complete 178 000-event dataset. This provided a shorter but robust training process that correctly indicated the predictive capacity of the given architecture.

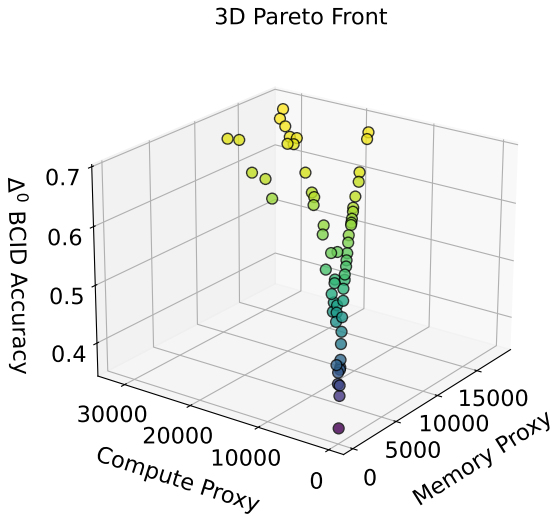
While the training pipeline was heavily optimized to distribute the fitness evaluation across multiple GPUs, the evolutionary process of mapping the full 3D Pareto front over numerous generations remains a highly computationally intensive task. Evaluation of hundreds of unique architectural combinations on the complete dataset requires substantial computational infrastructure, highlighting the cost of such powerful techniques, such as Neural Architecture Search.

Optimization Results

The resulting 3D Pareto optimal front, mapped over the course of 50 generations, is illustrated in Figure 2.33. Each point across these projections represents a unique non-dominated set of parameters of an LGNN architecture, for which no single objective can be further improved, without lowering at least one of the remaining metrics.



(a) 2D projections isolating the Memory vs. Accuracy axis (left) and the Compute vs. Accuracy axis (right).



(b) 3D trade-off space.

Figure 2.33 Pareto optimal front generated via NSGA-II hyperparameter optimization. (a) A 2D projection isolating the hardware memory proxy (left) and the compute latency proxy (right). (b) The full 3D search space mapping exact validation accuracy (Δ^0BCID) against both computational latency and memory proxies simultaneously.

For complete transparency, a detailed breakdown of all configurations mapped onto the Pareto-optimal front is provided in Table B in the Appendix.

The topology of the Pareto front reveals the clear non-linear trade-off between true predictive performance, hardware memory consumption, and processing latency. Notably, the configurations within the Pareto front visibly diverge into three distinct branches. This behavior is directly caused by the sequential hardware execution of the

message-passing pipeline, where increments of the message-passing depth (K) linearly scale the computational latency proxy.

Furthermore, the 3D search space successfully isolated a region of incompatible hyperparameter configurations, as visualized by the gap present across all three branches of the Pareto front. This further validates the utilization of the NSGA-II algorithm for this task. By mapping such incompatible structural regions, it proves that manual hyperparameter tuning within such a coupled architectural space is practically infeasible.

3 Experiments

The previous chapter established the methodological framework, introduced the datasets, and defined both the baseline and the machine learning algorithms evaluated for the muon time-of-passage (BCID) estimation task. All previously introduced models, including the analytical baseline strategy (Section 2.2), the Convolutional Neural Networks (Section 2.3), and the multiple variations of Message-Passing Neural Networks (Section 2.4), have demonstrated their predictive performance through experiments performed on clean single-muon event datasets.

The experimental evaluation, detailed in the following Section 3.1, focuses on a comparative analysis of the predictive robustness of these distinct architectures when exposed to events containing artificially injected background noise.

3.1 GNN Noise Robustness Evaluation

Although predictive performance on clean detector events suggests the inherent model capabilities, under real-world operational conditions, the events always encounter some degree of background noise. Consequently, evaluation and optimization of model robustness against these random noise hits is a crucial part of the development process, especially with algorithms targeted for deployment in an active detector environment.

Dataset Because the primary dataset prepared for this task contains predominantly clean single-muon events, a dataset augmentation pipeline was implemented. This pipeline artificially introduces background noise hits into the events at rates tunable through an intuitive noise probability parameter. Both the underlying dataset and this augmentation pipeline are introduced in detail in Section 2.1, with the specific pseudocode of the noise injection algorithm displayed in Algorithm 1.

To evaluate the degradation of model performance under varying levels of background noise, the clean single-muon dataset was augmented into five distinct evaluation sets. Each set then corresponds to a specific noise injection probability ranging from 1 % to 5 % ($p_{\text{noise}} \in \{0.01, 0.02, 0.03, 0.04, 0.05\}$).

Model Evaluation Model performance across these augmented datasets was evaluated using both the exact (Δ^0BCID) and the off-by-one (Δ^1BCID) accuracy metrics, defined in Section 2.2.3.

Because the previous machine learning architectural search established Graph Neural Networks as the optimal candidate within this work for the BCID estimation task, this benchmark utilizes the general Message-Passing Neural Network (MPNN) configurations selected during the architectural evaluation (Section 2.4.2). The general MPNNs were selected to serve as a fair representative for assessing the noise robustness capabilities of the machine learning paradigm, independent of the hardware optimization steps implemented later for the LGNN design (Section 2.4.3).

Baseline Susceptibility of Clean Models

The first experiment compares the *zero-shot* noise robustness of the MPNN architectures, trained exclusively on clean events, against the analytical tangent-based baseline strategy. The degradation of the predictive performance of models across the noise range 0 % to 5 % is illustrated in Figure 3.1.

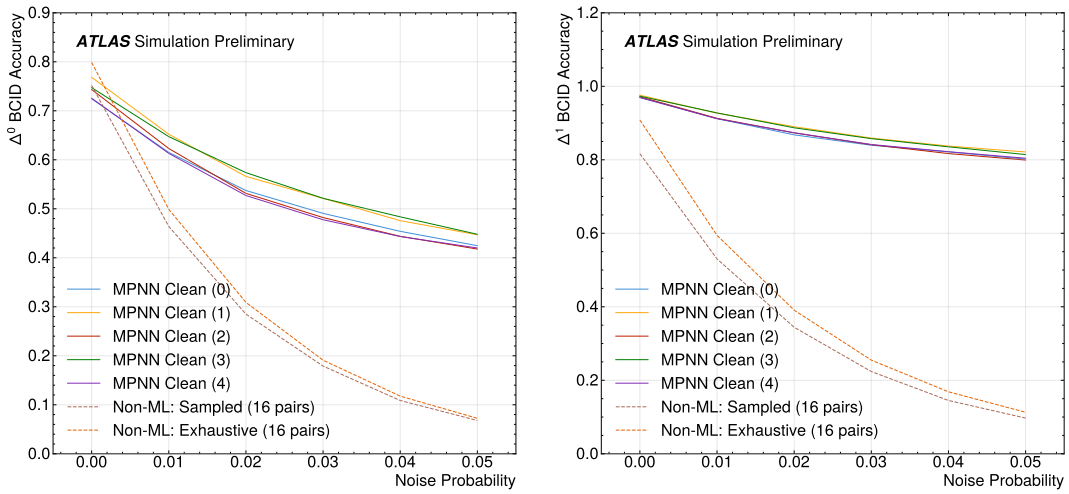


Figure 3.1 Illustration of performance degradation under increasing levels of injected background noise (p_{noise}). The figure compares the analytical baseline algorithms (dashed lines) against the top five MPNN architectures trained exclusively on clean events (solid lines, labeled by an identifier in parentheses). Performance is measured using both the exact (Δ^0BCID, left) and off-by-one (Δ^1BCID, right) accuracy metrics.

The evaluation reveals a major difference in the inherent noise robustness of the two paradigms. As established in previous machine learning evaluations, in a clean event environment ($p_{\text{noise}} = 0.00$), the analytical algorithms clearly outperform the MPNN architectures in the Δ^0BCID accuracy.

However, with the introduction of background noise, the predictive power of the analytical algorithms degrades extremely quickly, falling below 10% exact accuracy at the 5% noise level. The off-by-one accuracy closely mirrors this trend. This susceptibility is explained by the two fundamental ways in which the noise negatively affects the analytical tangent-based algorithm. Firstly, the algorithm relies heavily on the stochastic sampling of MDT pairs to construct candidate muon trajectories. As noise increases, the probability of successful sampling of a pair of true hits decreases significantly. Secondly, even if a pair of true hits is selected, a high number of noise hits can corrupt the trajectory approximation metric (Equation 2.4), so that the algorithm cannot recognize the true trajectory.

In contrast, clean MPNN models demonstrated inherent noise robustness. Although these models were not trained on noisy examples, the **generalization** ability of machine learning models enables them to retain significantly more of their original predictive power across the range of noise levels, compared to the baseline algorithms. This generalization effect is further observed in the Δ^1BCID metric, where machine learning models typically outperform geometrical baseline approaches, demonstrating that GNN models are capable of inferring the general BCID window, having a better understanding of the target distribution.

Model Noise Robustness Optimization

Although the inherent generalization of the MPNN architectures significantly outperforms the baseline, the predictive performance of these machine learning models still drops to roughly 45% exact accuracy under 5% noise levels. To mitigate this performance degradation, fine-tuning strategies for noisy inputs were evaluated.

In the first evaluation, a single-phase fine-tuning approach was utilized. The training dataset was augmented using a pipeline that generated two noise-injected events for every original clean event. The pre-trained MPNN models were subsequently subjected to fine-tuning for an additional 20 epochs on this dataset, augmented with $p_{\text{noise}} = 0.05$ noise probability. The objective was to expose the models to noisy events, forcing them to learn which hits contribute to the true event and which are background noise. The performance comparison between the clean and fine-tuned MPNN models is illustrated in Figure 3.2.

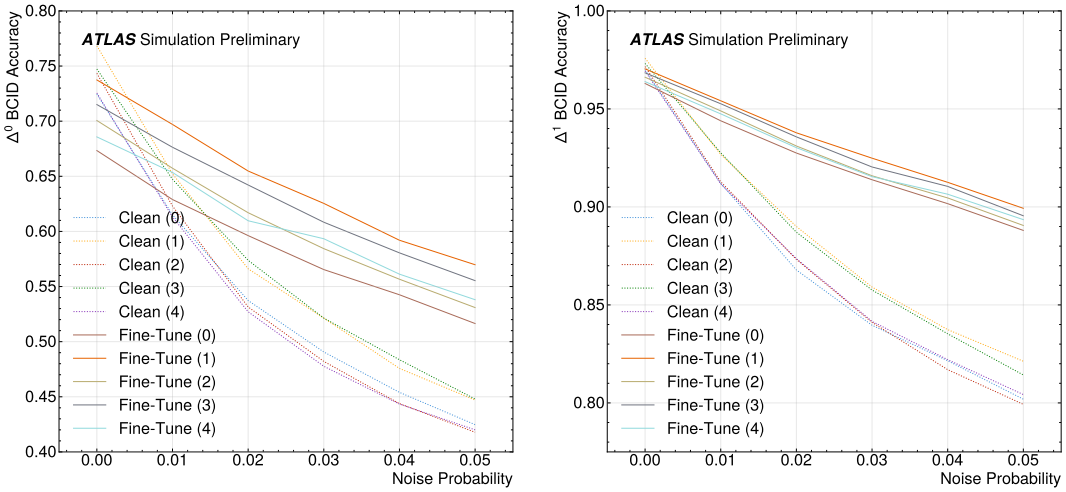


Figure 3.2 Performance comparison of five clean (dotted lines) and five single-phase fine-tuned MPNN models (solid lines, fine-tuned on the augmented dataset with $p_{\text{noise}} = 0.05$). Performance is measured using both the exact (Δ^0BCID, left) and off-by-one (Δ^1BCID, right) accuracy metrics.

The single-phase fine-tuning successfully improved the predictive robustness of the MPNN models by roughly 10% in both exact and off-by-one accuracies under the maximum 5% noise level. Although the pollution of the training data slightly reduced the fine-tuned model's accuracy on a completely clean event, this fine-tuning process proves its effectiveness in optimizing models for conditions more resembling the real detector environment.

Two-Phase Progressive Fine-Tuning Although the single-phase process was successful, a two-phase progressive fine-tuning approach was evaluated. The objective of this experiment was to introduce noisy examples to the model more gradually. Theoretically, this gradual exposure should allow the model to retain more of its original predictive performance.

For this evaluation, an intermediate augmented dataset was created using the same 2:1 ratio, with a milder noise injection probability of $p_{\text{noise}} = 0.025$. The clean pre-trained MPNN models were first fine-tuned on this milder dataset, before undergoing a secondary fine-tuning phase utilizing the 5% dataset. Performance comparisons between this progressive two-phase approach and the single-phase fine-tuning scheme are illustrated in Figure 3.3.

As demonstrated in Figure 3.3, the two-phase progressive fine-tuning did not drastically improve the predictive performance compared to the single-phase strategy, suggesting a point of diminishing returns of this optimization approach.

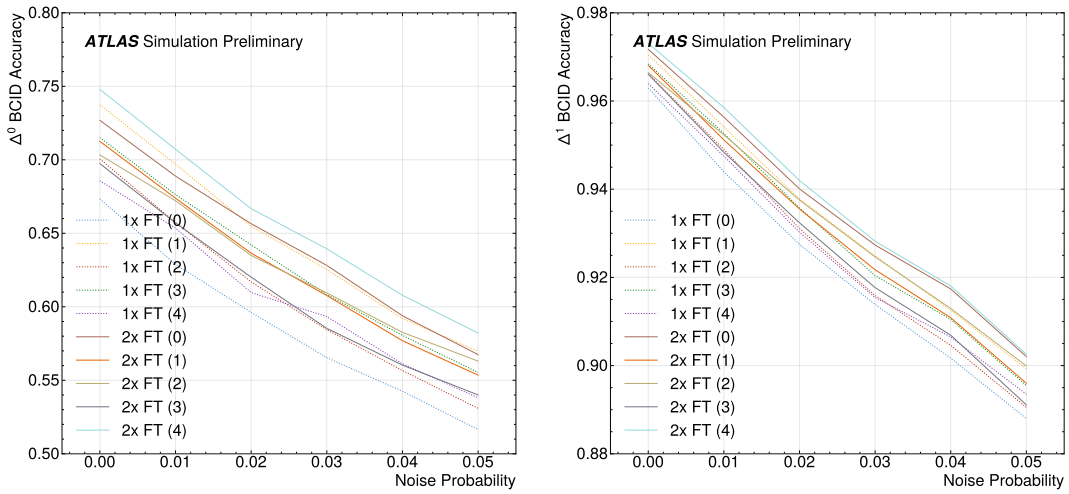


Figure 3.3 Performance comparison of the single-phase (dotted lines) and the progressive two-phase (solid lines) fine-tuning approaches for the five MPNN models. Performance is measured using both the exact (Δ^0BCID, left) and off-by-one (Δ^1BCID, right) accuracy metrics.

Experimental Summary

The objective of this experimental evaluation was to benchmark the robustness of graph-based machine learning models under conditions similar to those of the detector environment. The findings demonstrate the strong generalization capability of the MPNN architectures and their advantage compared to the analytical tangent-based baseline algorithm.

While the analytical approach suffered a drastic degradation in predictive performance in the presence of background noise, clean MPNN models exhibited inherent resilience to background noise, even without prior exposure to similar events. Furthermore, the application of single-phase and multi-phase progressive fine-tuning was highly successful at adapting the models, allowing them to retain their predictive performance even under severe noise levels.

Conclusion

Accurate muon time-of-passage (BCID) estimation is an essential step of a reliable muon trajectory reconstruction within the Monitored Drift Tube (MDT) chambers of the ATLAS experiment at CERN. However, developing an algorithm to perform this estimation presents a complex challenge with two major constraints: the system must maintain high predictive accuracy in the presence of background noise, while being able to operate within the strict sub-microsecond inference limits required by the increased number of collision events in the HL-LHC.

Conducted as part of the *Work Package 2.2* research team for the *Next Generation Trigger (NGT)* project at ATLAS, this research directly contributes to the enhancement of the Level-0 Muon trigger. The primary objective of this thesis was to propose, develop, and evaluate a time-of-passage estimation algorithm suitable for these highly restrictive constraints.

Contributions

To achieve this, the thesis first established an analytical tangent-based baseline algorithm to serve as a performance benchmark. The following research then examined various machine learning approaches.

Initially, the evaluation focused on Convolutional Neural Networks (CNNs). However, inherent inefficiencies related to sparse input representation resulted in an overall suboptimal model complexity for the targeted hardware environment. Subsequently, the architectural research pivoted to closely related Graph Neural Networks (GNNs). The possibility of a custom definition of the graph-structured input allowed for a highly descriptive and computationally efficient representation of the raw detector events. Within GNNs, the general Message-Passing Neural Network (MPNN) architecture was introduced. Through its weight-sharing mechanisms, the MPNN model demonstrated exceptional predictive capacity at a substantially reduced total parameter count.

Furthermore, this work introduced a custom Line Graph Neural Network (LGNN) architecture, defining and analyzing a specialized paired graph representation. This improved graph representation allowed the LGNN to significantly simplify operational complexity compared to the general MPNN, while considerably improving its predictive performance. These simplifications allow the LGNN model to be easily quantized via the introduced High Granularity Quantization (HGQ) framework and efficiently implemented on FPGA hardware. Building upon these results, a separate team member of the Work Package 2.2 research team is continuing this investigation, specifically targeting the hardware implementation and deployment phase for the introduced LGNN models.

Further performance optimization methods, such as detector station-specific fine-tuning, were introduced to maximize the predictive accuracy of the model. This approach reached the peak exact accuracy (Δ^0BCID) of 79.50% in the innermost station, selecting the precise time of particle collision at the interaction point. Moreover, the models demonstrated an exceptional off-by-one accuracy (Δ^1BCID) of 97.25%, successfully restricting the temporal estimation error to a maximum of a single 25 ns time window.

Additionally, this thesis introduced the Non-dominated Sorting Genetic Algorithm II (NSGA-II) and applied it to hyperparameter optimization of the LGNN architecture. Multi-objective optimization based on the model's predictive accuracy, inference latency, and memory footprint successfully mapped the full Pareto front, yielding a curated selection of architectural configurations. This flexibility allows to postpone the selection of a single final model configuration until the exact hardware resource budgets are

established and understood.

Finally, this work evaluated the robustness of the model against background noise. These results demonstrate that zero-shot message-passing machine learning models are inherently significantly more resilient compared to the strictly analytical baseline, which experienced quick performance degradation under increasing noise levels. Subsequent evaluations successfully experimented with noise robustness optimization through targeted noise fine-tuning phases, further improving the predictive performance of GNN models on events with background noise.

Future Steps

Although this thesis establishes a robust and optimized algorithm for machine learning-based time-of-passage estimation, several objectives remain outlined for future work.

The immediate next step is the physical synthesis and deployment of the LGNN architecture on the target FPGA hardware, which is currently undergoing in collaboration with the Work Package 2.2 team. More comprehensive evaluations are needed to benchmark the inference latency, power consumption, and resource utilization of quantized models under real-world deployment conditions, compared to the cost estimates provided by the NSGA-II Pareto front.

Additionally, while targeted noise fine-tuning demonstrated improved model robustness on events with artificial background noise, future efforts should validate these models against real-world collision data from the ATLAS detector to fully verify the ability of the architecture to remain robust under operational conditions.

Lastly, careful evaluation is necessary prior to the integration of this standalone time-of-passage estimation algorithm into the planned Level-0 Muon Trigger pipeline, connecting it to other event filtration and particle transverse momentum estimation algorithms.

Declaration of Artificial Intelligence Usage

I declare that the following AI tools have been used in preparation of this thesis in the ways indicated:

- **Gemini** in the period 10/4/2026 – 14/7/2026, usage description: Stylistic text revisions to improve readability. Outline brainstorming based on a prepared research journal.

Having used the above AI tools, I have thoroughly revised and edited the content as necessary and take full responsibility for the final form of the thesis.

Bibliography

1. Aad, G et al. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*. 2008, vol. 3, S08003. Available from DOI: 10.1088/1748-0221/3/08/S08003. Also published by CERN Geneva in 2010.
2. Bianchi, Riccardo Maria; Collaboration, ATLAS. *ATLAS experiment schematic or layout illustration*. 2022. Available also from: https://cds.cern.ch/record/2837191. General Photo.
3. Aad, G. et al. The ATLAS experiment at the CERN Large Hadron Collider: a description of the detector configuration for Run 3. *Journal of Instrumentation*. 2024, vol. 19, no. 05, P05063. ISSN 1748-0221. Available from DOI: 10.1088/1748-0221/19/05/p05063.
4. Dominguez, Daniel; Moles, Kevin Patrice; Mehlhase, Sascha. ATLAS detector schematics. 2021. Available also from: https://cds.cern.ch/record/2777214.
5. collaboration, The ATLAS. Resolution of the ATLAS muon spectrometer monitored drift tubes in LHC Run 2. *Journal of Instrumentation*. 2019, vol. 14, no. 09, P09011. Available from DOI: 10.1088/1748-0221/14/09/P09011.
6. Arai, Yasuo; Ball, B; Beretta, Mauro; Boterenbrood, H.; Brandenburg, G.W.; Ceradini, F; Chapman, J.W.; Dai, Tianhong; Ferretti, Ceres; Fries, T.; Gregory, J; Guimaraes da Costa, Joao; Harder, S; Hazen, Eric; Huth, John; Jansweijer, Peter; Kirsch, Lisa; Konig, A.; Lanza, Agostino; Wijnen, Thei. ATLAS muon drift tube electronics. *Journal of Instrumentation*. 2008, vol. 3. Available from DOI: 10.1088/1748-0221/3/09/P09001.
7. Bauer, F.; Bratzler, U.; Dietl, H.; Kroha, H.; Lagouri, Th.; Manz, A.; Ostapchuk, A.; Richter, Robert; Schael, S.; Chouridou, S.; Deile, M.; Kortner, O.; Staude, A.; Stroehmer, R.; Trefzger, T. Construction and Test of MDT Chambers for the ATLAS Muon Spectrometer. *Nucl. Instrum. Methods Phys. Res., A*. 2001, vol. 461, pp. 17–20. Available from DOI: 10.1016/S0168-9002(00)01156-6.
8. Aad, Georges et al. Standalone Vertex Finding in the ATLAS Muon Spectrometer. *JINST*. 2014, vol. 9, P02001. Available from DOI: 10.1088/1748-0221/9/02/P02001. Comments: 22 pages plus author list (43 pages total), 21 figures, 1 table, submitted to JINST. All figures including auxiliary figures are available at http://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PAPERS/PERF-2013-01/.
9. Aad, Georges et al. Performance of the ATLAS muon triggers in Run 2. *JINST*. 2020, vol. 15, P09015. Available from DOI: 10.1088/1748-0221/15/09/p09015. 60 pages in total, author list starting page 44, 34 figures, 5 tables, submitted to JINST. All figures including auxiliary figures are available at http://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PAPERS/TRIG-2018-01.
10. *Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System*. Geneva, 2017. Tech. rep. CERN. Available from DOI: 10.17181/CERN.2LBB.4IAL.
11. *Technical Design Report for the Phase-II Upgrade of the ATLAS Muon Spectrometer*. Geneva, 2017. Tech. rep. CERN. Available also from: https://cds.cern.ch/record/2285580.
12. Kuon, Ian; Tessier, Russell; Rose, Jonathan. FPGA architecture: Survey and challenges. *Foundatiosn and Trends in Electronic Design Automation*. 2008, vol. 2, no. 2, pp. 135–253.

13. Coussy, Philippe; Gajski, Daniel D.; Meredith, Michael; Takach, Andres. An Introduction to High-Level Synthesis. *IEEE Design Test of Computers*. 2009, vol. 26, no. 4, pp. 8–17. Available from DOI: 10.1109/MDT.2009.69.

14. Duarte, Javier; Han, Song; Harris, Philip; Jindariani, Sergo; Kreinar, Edward; Kreis, Benjamin; Ngadiuba, Jennifer; Pierini, Maurizio; Rivera, Ryan; Tran, Nhan, et al. Fast inference of deep neural networks in FPGAs for particle physics. *Journal of instrumentation*. 2018, vol. 13, no. 07, P07027–P07027.

15. Deiana, Allison McCarn; Tran, Nhan; Agar, Joshua; Blott, Michaela; Di Guglielmo, Giuseppe; Duarte, Javier; Harris, Philip; Hauck, Scott; Liu, Mia; Neubauer, Mark S.; Ngadiuba, Jennifer; Ogrenci-Memik, Seda; Pierini, Maurizio; Aarrestad, Thea; Bähr, Steffen; Becker, Jürgen; Berthold, Anne-Sophie; Bonventre, Richard J.; Müller Bravo, Tomás E.; Diefenthaler, Markus; Dong, Zhen; Fritzsche, Nick; Gholami, Amir; Govorkova, Ekaterina; Guo, Dongning; Hazelwood, Kyle J.; Herwig, Christian; Khan, Babar; Kim, Sehoon; Klijnsma, Thomas; Liu, Yaling; Lo, Kin Ho; Nguyen, Tri; Pezzullo, Gianantonio; Rasoulinezhad, Seyedramin; Rivera, Ryan A.; Scholberg, Kate; Selig, Justin; Sen, Sougata; Strukov, Dmitri; Tang, William; Thais, Savannah; Unger, Kai Lukas; Vilalta, Ricardo; Krosigk, Belina von; Wang, Shen; Warburton, Thomas K. Applications and Techniques for Fast Machine Learning in Science. *Frontiers in Big Data*. 2022, vol. Volume 5 - 2022. ISSN 2624-909X. Available from DOI: 10.3389/fdata.2022.787421.

16. Bishop, Christopher M.; Bishop, Hugh. Graph Neural Networks. In: *Deep Learning: Foundations and Concepts*. Cham: Springer International Publishing, 2024, pp. 407–427. ISBN 978-3-031-45468-4. Available from DOI: 10.1007/978-3-031-45468-4_13.

17. Mitchell, Tom. *Machine Learning*. McGraw-Hill Education, 1997.

18. Bertacchi, Valerio. *DeepCore: Convolutional Neural Network for high p_T jet tracking*. 2019. Tech. rep. Available from arXiv: 1910.08058. Proceeding of CTD/WIT 2019, 8 pages, 6 figures.

19. Abed Abud, Adam; DUNE collaboration, on behalf of the. Sparse Convolutional Neural Networks for particle classification in ProtoDUNE-SP events. *Journal of Physics: Conference Series*. 2023, vol. 2438, no. 1, p. 012125. Available from DOI: 10.1088/1742-6596/2438/1/012125.

20. Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

21. Commons, Wikimedia. *File:Typical cnn.png — Wikimedia Commons, the free media repository*. 2022. Available also from: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:Typical_cnn.png&oldid=671717156%7D. Online; accessed 26-April-2026.

22. Lazar, Alina; Ju, Xiangyang; Murnane, Daniel; Calafiura, Paolo; Farrell, Steven; Xu, Yaoyuan; Spiropulu, Maria; Vlimant, Jean-Roch; Cerati, Giuseppe; Gray, Lindsey; Klijnsma, Thomas; Kowalkowski, Jim; Atkinson, Markus; Neubauer, Mark; DeZoort, Gage; Thais, Savannah; Hsu, Shih-Chieh; Aurisano, Adam; Hewes, Jeremy; Ballow, Alexandra; Acharya, Nirajan; Wang, Chun-yi; Liu, Emma; Lucas, Alberto. Accelerating the Inference of the Exa.TrkX Pipeline. *Journal of Physics: Conference Series*. 2023, vol. 2438, no. 1, p. 012008. ISSN 1742-6596. Available from DOI: 10.1088/1742-6596/2438/1/012008.

23. Burleson, Jared Dynes; Caillou, Sylvain; Calafiura, Paolo; Chan, Jay; Collard, Christophe; Ju, Xiangyang; Murnane, Daniel Thomas; Neubauer, Mark; Pham, Minh Tuan; Rougier, Charline; Stark, Jan; Torres, Heberth; Vallier, Alexis. *Physics Performance of the ATLAS GNN4ITk Track Reconstruction Chain*. Geneva, 2023. Tech. rep. CERN. Available also from: `https://cds.cern.ch/record/2882507`.

24. Gholami, Amir; Kim, Sehoon; Dong, Zhen; Yao, Zhewei; Mahoney, Michael W; Keutzer, Kurt. A survey of quantization methods for efficient neural network inference. In: *Low-power computer vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

25. Sun, Chang; Que, Zhiqiang; Aarrestad, Thea; Loncar, Vladimir; Ngadiuba, Jennifer; Luk, Wayne; Spiropulu, Maria. HGQ: High Granularity Quantization for Real-time Neural Networks on FPGAs. In: *Proceedings of the 2026 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, 2026, pp. 79–91. Available from DOI: `10.1145/3748173.3779200`.

26. Benmeziane, Hadjer; El Maghraoui, Kaoutar; Ouarnoughi, Hamza; Niar, Smail; Wistuba, Martin; Wang, Naigang. Hardware-Aware Neural Architecture Search: Survey and Taxonomy. In: *IJCAI*. 2021, vol. 2021, pp. 4322–4329.

27. Fischler, Martin A; Bolles, Robert C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*. 1981, vol. 24, no. 6, pp. 381–395.

28. Derpanis, Konstantinos G. Overview of the RANSAC Algorithm. *Image Rochester NY*. 2010, vol. 4, no. 1, pp. 2–3.

29. Liu, Jianhua; Chang, Michael; Cheng, Chung-Kuan. An iterative division algorithm for FPGAs. In: *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*. 2006, pp. 83–89.

30. Suresh, Shashank; Beldianu, Spiridon F; Ziavras, Sotirios G. FPGA and ASIC square root designs for high performance and power efficiency. In: *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 2013, pp. 269–272.

31. He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian. *Deep Residual Learning for Image Recognition*. 2015. Available from arXiv: `1512.03385 [cs.CV]`.

32. Howard, Andrew G.; Zhu, Menglong; Chen, Bo; Kalenichenko, Dmitry; Wang, Weijun; Weyand, Tobias; Andreetto, Marco; Adam, Hartwig. *MobileNet: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. Available from arXiv: `1704.04861 [cs.CV]`.

33. Sandler, Mark; Howard, Andrew; Zhu, Menglong; Zhmoginov, Andrey; Chen, Liang-Chieh. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. Available from arXiv: `1801.04381 [cs.CV]`.

34. Howard, Andrew; Sandler, Mark; Chu, Grace; Chen, Liang-Chieh; Chen, Bo; Tan, Mingxing; Wang, Weijun; Zhu, Yukun; Pang, Ruoming; Vasudevan, Vijay; Le, Quoc V.; Adam, Hartwig. *Searching for MobileNetV3*. 2019. Available from arXiv: `1905.02244 [cs.CV]`.

35. Chollet, François et al. *Keras* [`https://keras.io`]. 2015.

36. Martín Abadi; Ashish Agarwal; Paul Barham; Eugene Brevdo; Zhifeng Chen; Craig Citro; Greg S. Corrado; Andy Davis; Jeffrey Dean; Matthieu Devin; Sanjay Ghemawat; Ian Goodfellow; Andrew Harp; Geoffrey Irving; Michael Isard; Jia, Yangqing; Rafal Jozefowicz; Lukasz Kaiser; Manjunath Kudlur; Josh Levenberg; Dandelion Mané; Rajat Monga; Sherry Moore; Derek Murray; Chris Olah; Mike Schuster; Jonathon Shlens; Benoit Steiner; Ilya Sutskever; Kunal Talwar; Paul Tucker; Vincent Vanhoucke; Vijay Vasudevan; Fernanda Viégas; Oriol Vinyals; Pete Warden; Martin Wattenberg; Martin Wicke; Yuan Yu; Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Available also from: `https://www.tensorflow.org/`. Software available from tensorflow.org.

37. Lomont, Chris. Fast inverse square root. *Tech-315 nical Report*. 2003, vol. 32, pp. 44–46.

List of Figures

1 Illustration of the machine learning pipeline for processing the MDT hit data. 7

1.1 Full ATLAS detector schematic of the Run 3 configuration [2]. 9
1.2 ATLAS schematics with highlighted magnet system (left) and Muon Spectrometer systems (right) [4]. 10
1.3 MDT tube cross section with an ionizing track [6]. 12
1.4 MDT $R(t)$ function for Ar/CO_2 gas mixture [5]. 12
1.5 Structure of an MDT chamber [7]. 13
1.6 ATLAS barrel cross section perpendicular to the beam axis with MDT stations highlighted, small sectors shown in blue and large in orange, the RPC layers are shown as red outlines of MDT chambers [8]. 14
1.7 ATLAS muon system cross section in the beam axis plane, MDT chambers in barrel highlighted in green, end-cap chambers in blue, RPC and TGC trigger chambers are highlighted alongside MDTs, pseudorapidity lines are present with η values [9]. 15
1.8 Illustration of basic FPGA architecture [12]. 19
1.9 HLS design steps [13], RTL (or *register-transfer level*) corresponds to a level of design abstraction for synchronous digital circuits used in HDL. 20
1.10 Illustration of a typical CNN architecture [21]. 22
1.11 Example of structured data visualized as a graph. The graph presents the distribution and interconnection of different programming languages as topics on *Stack Overflow*, with labels for major hubs. 24
1.12 Visualization of the GNN particle tracking pipeline [23]. 25
1.13 Illustration of the similarity between convolution operation and graph message-passing (red arrows symbolizing messages) [16]. 26
1.14 Illustration of the growth of the effective *receptive field* (nodes shown in red), with a growing number of message-passing steps (adapted from [16]). 27

2.1 Visualizations of representative event groups from the inner (left) and middle (right) stations. The blue circles represent the wall of active MDT tubes, and the red circles represent the true drift radii. 33
2.2 Visualization of a subset of the entire muon event dataset, with colors distinguishing the three MDT stations. 34
2.3 Visualization of a subset of the entire muon event dataset, with colors distinguishing individual MDT chambers. 34
2.4 Visualizations of the normalized local coordinate space of tube numbers (left) and tube layers (right). The color gradient indicates the final range of values after the normalization. 35
2.5 Visualization of the synthetic MDT noise injection process. On the left, the defined local MDT neighborhood (green tubes) around the active event group. The original event group is also included in this neighborhood set. On the right, the resulting event group after an augmentation with 5% ($p_{\text{noise}} = 0.05$) noise injection probability. The noise hits (orange) show the potential to activate, deactivate or perturb the tubes in the original event. 36
2.6 Showcase of the trajectory approximation as a straight line common tangent to each drift circle (red radii in each blue tube). 38

2.7 Visualization of two independent RANSAC resampling iterations. Both candidate models correctly classify all true MDT hits from the presented event group with injected noise. 39
2.8 Simplified illustration of the baseline algorithm pipeline. 41
2.9 Comparison of the trajectory approximation error (E) across discrete BCID samples from a single event group. The brute-force exhaustive search strategy (blue line) shows the full error landscape. The iterative search strategy minimizes the number of evaluations by first evaluating a small subset of BCID values (Phase 1, orange dots), followed by a discrete gradient approximation that takes a limited number of steps towards the local minimum (Phase 2, green crosses). 43
2.10 Histogram of the predicted BCID deviation (ΔBCID) relative to the true BCID of the event for the two baseline search strategies: Exhaustive (blue) and Iterative (orange). The evaluation using the BCID-conditioned RANSAC algorithm was performed using 16 stochastically sampled MDT pairs (left) and 8 sampled pairs (right) per each iteration. 45
2.11 Illustration of the spatial distribution of BCID estimation accuracy in the ATLAS detector. The MDT tubes are color-coded based on the baseline algorithm's prediction accuracy on their respective event groups, with green indicating correct predictions ($\Delta\text{BCID} = 0$), yellow indicating close predictions ($|\Delta\text{BCID}| = 1$), and red indicating incorrect predictions with a BCID deviation greater than one. 46
2.12 Visualization of the fraction of incorrect BCID assignments as a function of pseudorapidity (η). The histogram indicates degraded predictive performance of the baseline algorithms in the central $0.4 < \eta < 0.7$ region (approximately from 23° to 40° off the vertical axis). The results correspond to the experiment from the color-coded visualization in Figure 2.11. 46
2.13 Visualization of a **pathological** event group on which the geometric baseline algorithms fail. Due to the alignment of the muon path with the arrangement of tubes in the MDT chamber, an incorrect BCID hypothesis produces estimated drift circles (green) that perfectly align with the estimated trajectory (magenta). This configuration returns a near zero trajectory approximation error, which is why the algorithm selects it as the optimal solution. 47
2.14 Demonstration of mapping of an even group to a discrete 2D rectangular grid, where a single channel stores the signal detection information of the corresponding MDT hit. 49
2.15 Application of two different blurring kernels on the event group previously shown in Figure 2.14. The left input utilizes the standard 3×3 Gaussian blur kernel, the right input utilizes a custom 3×3 *directional* kernel (both of which are defined in Equation 2.10). 49
2.16 Diagram of the building block of *residual learning* [31]. 50
2.17 Comparison of the standard residual block (left) and the inverted residual block with linear bottleneck (right). The thickness of each block indicates the relative number of channels, and hatched blocks highlight layers utilizing only a linear activation function [33]. 51
2.18 Comparison of the original one-hot target vector (blue) and the Gaussian label smoothing transformed target vector (orange) with the standard deviation parameter $\sigma = 1.25$. 53

2.19 Visualization of the progression of the Δ^0BCID (left) and Δ^1BCID (right) validation accuracies during the training process of the Large and Optimized ResNet architectures defined in Table 2.4. The three solid lines distinguish between the three different input image configurations. The dashed green line shows the validation accuracy trajectory of the Optimized ResNet model, trained on the selected highest-performing clean input representation. 54
2.20 Histogram of the predicted BCID deviation (ΔBCID) relative to the true BCID for the Large (blue) and Optimized (orange) ResNet architectures. The distribution demonstrates that the models are not guessing the true class randomly and are able to roughly estimate the true BCID value, suggesting that stronger predictive models have the potential to further improve their predictive accuracy. 56
2.21 Visualization of the trajectories of validation accuracy during the training process of ResNet baselines (solid line) and their MobileNet equivalents (dashed lines). The left figure shows the exact (Δ^0BCID) accuracy of models, while the right figure shows the off-by-one (Δ^1BCID) accuracy. All models were trained using the clean input group representation (with no spatial blurring effects), as established in the initial experiment. . . . 57
2.22 Illustration comparing the validation (Δ^0BCID) predictive performance of the Large MobileNet architecture (defined in Table 2.5, when systematically restricted by decreasing all internal expansion factors to 2 and 1. The legend displays the resulting total parameter count for each configuration. For the baseline, the performance of the Optimized model is also plotted (dashed green line). 58
2.23 Visualizations of an original input event group in global coordinates (left) and its graph node representation in the local coordinate system (right). The node indices (0–6) represent the chronologically ordered sequence of MDT signal detection times. The specific node feature values corresponding to this event are detailed in Table 2.6. 61
2.24 Demonstration of the sorted node representation of the input event group previously displayed in Figure 2.23. The nodes are ordered primarily from bottom to top based on their layer index, and secondarily from left to right within the same layer based on the tube number. 62
2.25 Demonstration of the resulting interconnected graph representation of the input event group previously displayed in Figure 2.23. 63
2.26 Visualization of the validation accuracy trajectories during training of four general message-passing models, differentiated only by the normalization approach to the z and ρ directional edge features. The four share identical architectures and were trained on an otherwise identical dataset consisting of 25 000 distinct events (a subset of the 100 000 event dataset utilized in previous experiments). 64
2.27 Visualization of the validation Δ^0BCID accuracy trajectories during the training process for multiple MPNN architectures differentiated by the number of message-passing steps performed. 68

2.28 Validation accuracy trajectories for the top 10 evaluated MPNN configurations. The top figure (a) displays the exact prediction accuracy, while the bottom figure (b) displays the off-by-one accuracy. The legend (sorted by the exact accuracy for both figures) details the model configuration parameters: Node dimension (N), Edge dimension (E), Readout size (R), and the readout pooling function (F). The solid and dashed lines then differentiate between the utilization of ReLU6 or Linear activation function in the second layer of update MLPs. The best validation accuracy achieved over the 40 training epochs is displayed in parentheses for each model. 70
2.29 Validation exact accuracy (Δ^0BCID) trajectories for the top ten MPNN architectures during the two-stage training process (pre-training, fine-tuning). The vertical red dashed line marks the transitional epoch from global pre-training to Inner station-specific fine-tuning. 72
2.30 Visualization of the validation exact accuracy (Δ^0BCID) trajectories over 40 epochs of the training, comparing the initial LGNN model (solid line), with the already detailed top ten MPNN configurations from the architectural search (dashed lines). 77
2.31 Visualization of the validation exact accuracy (Δ^0BCID) trajectories over 40 training epochs. The comparison displays the two identical LGNN configurations (solid lines) differentiated only by their maximum node out-degree ($D = 4$ vs. $D = 2$) within the internal graph representation, plotted alongside a subset of the best-performing MPNN models (dashed lines) selected through the architectural search. 79
2.32 Visualization of the validation exact accuracy (Δ^0BCID) trajectories comparing the performance of the LGNN across varying numbers of message-passing steps (K). The experiment was performed on a reduced subset of 50 000 events, utilizing the previously established $D = 2$ maximum out-degree constraint. 80
2.33 Pareto optimal front generated via NSGA-II hyperparameter optimization. (a) A 2D projection isolating the hardware memory proxy (left) and the compute latency proxy (right). (b) The full 3D search space mapping exact validation accuracy (Δ^0BCID) against both computational latency and memory proxies simultaneously. 84

3.1 Illustration of performance degradation under increasing levels of injected background noise (p_{noise}). The figure compares the analytical baseline algorithms (dashed lines) against the top five MPNN architectures trained exclusively on clean events (solid lines, labeled by an identifier in parentheses). Performance is measured using both the exact (Δ^0BCID, left) and off-by-one (Δ^1BCID, right) accuracy metrics. 87
3.2 Performance comparison of five clean (dotted lines) and five single-phase fine-tuned MPNN models (solid lines, fine-tuned on the augmented dataset with
$p_{\text{noise}} = 0.05$). Performance is measured using both the exact (Δ^0BCID, left) and off-by-one (Δ^1BCID, right) accuracy metrics. 88
3.3 Performance comparison of the single-phase (dotted lines) and the progressive two-phase (solid lines) fine-tuning approaches for the five MPNN models. Performance is measured using both the exact (Δ^0BCID, left) and off-by-one (Δ^1BCID, right) accuracy metrics. 89

List of Tables

2.1 Summary of test vector features. Features marked with an asterisk (*) denote golden simulation variables not available under real-world conditions. 32

2.2 Summary of the single-muon 100 000-event dataset utilized for benchmarks and comparison of the baseline algorithms. 43

2.3 Comparison of the BCID estimation accuracy between the exhaustive and iterative search strategies. The evaluation was performed on a test set of $|X| = 46\,700$ event groups introduced in Table 2.2. 45

2.4 Definition of the two ResNet configurations evaluated in the initial experiments. The bracketed notation denotes the kernel dimensions and output channels of the internal block of convolutional layers, followed by the number of block repetitions. Both architectures apply one full-resolution residual block before the initial spatial subsampling (max pooling) to maximize the feature extraction from the full input representation. Following this initial step, the architecture follows the standard ResNet design paradigm. *The first convolution in Residual Block 3 utilizes a stride of 2 to perform the final spatial downsampling. 55

2.5 Definition of the MobileNet configurations mirroring the two selected ResNet-style architectures. Each **IRB** denotes an Inverted Residual Block, parametrized by five values: [*input channels*, *expansion factor*, *output channels*, *kernel size*, *stride*]. IRB blocks with strides greater than 1 are utilized instead of spatial downsampling layers, such as max pooling. Furthermore, each IRB block implements a *Squeeze-and-Excitation* (SE) attention block to improve channel-wise feature extraction. 57

2.6 Table of processed node features corresponding to the raw input event group visualized in Figure 2.23, demonstrating the processing of spatial and temporal features of MDTs. 61

2.7 Performance comparison of the three methodologies evaluated so far for the BCID estimation task. While the geometrical Exhaustive approach retains the highest exact accuracy (Δ^0BCID), the evaluated MPNN architecture demonstrates improved performance in the off-by-one accuracy (Δ^1BCID), significantly outperforming both classical algorithms and all established CNN configurations. 71

2.8 Performance summary of the specialized station-specific MPNN models across the three distinct MDT detector stations. The global model represents the accuracy baseline achieved during pre-training on the entire dataset before station-specific fine-tuning. The Inner station was the only one to exhibit significant performance improvements; the other two stations showed stale predictive performance, with a small degradation of model's accuracy. 73

2.9 Comprehensive comparison of peak validation exact (Δ^0BCID) and off-by-one (Δ^1BCID) accuracies. The table benchmarks the station-specific fine-tuning results of the LGNN architecture against the Geometrical and globally trained CNN methods, as well as the fine-tuned MPNN architecture. 81

B.1 Complete set of LGNN configurations from the NSGA-II Pareto front. . 102

A Code repository

The complete repository containing the source code and results is available in a GitLab repository at `https://gitlab.cern.ch/atlas-nextgen/work-package-2.2/ngt2.2-t0-estimation` with a detailed description of the repository provided in the `README.md` file. The project is also available as an attachment to this thesis in a `zip` archive, representing the state of the project at the time of submission.

B NSGA-II Pareto Front Results

The following table details the resulting architectural configurations generated during the LGNN model optimization. Using the NSGA-II algorithm, these results map the full 3D pareto optimal front, outlining the four structural hyperparameters (N, E, K, R) and their corresponding predictive performance and hardware cost proxies.

Table B.1 Complete set of LGNN configurations from the NSGA-II Pareto front.

	Hyperparameters				**Evaluated Proxies**		
Config ID	**N**	**E**	**K**	**R**	$\mathbf{\Delta^0}$BCID	$\mathbf{C}_{\text{mem}}$	$\mathbf{C}_{\text{comp}}$
0	28	32	2	[52, 40]	0.680506	18112	31232
3	28	32	1	[52, 40]	0.674901	18112	18112
5	32	32	1	[44]	0.672024	17152	17152
45	28	32	2	[60]	0.678435	16480	29600
46	24	32	2	[52, 20]	0.676240	15568	27600
47	24	28	2	[56, 32]	0.672259	14400	24320
48	28	28	2	[60]	0.672123	14336	25312
51	16	32	2	[56, 32]	0.666171	13824	24064
50	20	32	2	[60]	0.667820	13472	24544
53	28	24	3	[60]	0.664670	12448	30624
52	24	28	3	[44]	0.666071	12032	31872
10	28	20	1	[52, 32]	0.663926	12032	12032
14	28	20	1	[60, 12]	0.652728	11536	11536
57	28	20	2	[48]	0.660466	10144	17600
16	20	20	1	[52, 32]	0.649727	9344	9344
15	12	28	3	[48, 4]	0.651190	8864	23904
58	24	20	3	[44]	0.649318	8576	21504
61	28	16	2	[44]	0.646627	8544	14624
62	20	20	2	[60]	0.645846	8000	13600
63	24	16	1	[60]	0.644692	8000	8000
21	24	12	1	[60, 16]	0.639323	7872	7872
22	20	20	2	[52]	0.635689	7680	13280
64	24	16	1	[44]	0.634189	7232	7232
65	12	24	3	[60]	0.633842	7200	18720
66	24	12	1	[60]	0.630965	6912	6912
25	16	16	1	[56, 24]	0.629179	6720	6720
67	16	16	2	[60]	0.630196	5504	9088
26	12	20	1	[44]	0.627046	5312	5312
27	16	16	1	[44]	0.618316	4992	4992
69	16	16	2	[44]	0.620536	4992	8576
28	12	12	1	[56, 20]	0.606349	4480	4480

Continued on next page...

Table B.1 – continued from previous page

	Hyperparameters				Evaluated Proxies		
ID	**N**	**E**	**K**	**R**	$\mathbf{\Delta^0}$BCID	$\mathbf{C_{mem}}$	$\mathbf{C_{comp}}$
74	12	8	2	[64, 20]	0.608371	4096	5376
29	12	16	1	[44]	0.599169	4064	4064
77	16	12	1	[40]	0.591381	3840	3840
72	12	12	2	[60]	0.611223	3456	5472
31	12	12	1	[44]	0.586657	3072	3072
80	16	4	1	[40, 12]	0.578261	3040	3040
78	12	8	3	[60]	0.590377	2720	5280
81	12	8	2	[60]	0.578162	2720	4000
32	12	8	2	[52]	0.574355	2528	3808
33	8	12	1	[44]	0.571875	2304	2304
82	8	12	2	[40]	0.558730	2240	3840
84	12	4	1	[60]	0.552889	2240	2240
34	12	8	2	[32, 4]	0.546577	2176	3456
35	12	4	2	[52]	0.545027	2048	2848
36	12	4	2	[44]	0.535231	1856	2656
85	4	12	2	[60]	0.535615	1792	3104
86	16	4	1	[12]	0.532564	1664	1664
87	12	4	2	[24, 8]	0.523338	1568	2368
88	8	4	1	[60]	0.511595	1408	1408
40	8	4	1	[44]	0.494444	1152	1152
90	4	8	1	[40]	0.470573	960	960
42	4	8	1	[28]	0.459226	864	864
92	4	4	1	[60]	0.457490	704	704
43	4	4	2	[44]	0.453274	576	800
44	4	4	1	[44]	0.432713	576	576
91	4	4	3	[40]	0.465315	544	992
94	4	4	2	[40]	0.434772	544	768
96	4	4	1	[40]	0.415712	544	544
97	4	4	1	[4]	0.365030	256	256