Politecnico di Bari

Repository Istituzionale dei Prodotti della Ricerca del Politecnico di Bari

Hyperparameter optimization of deep learning models using high-performance computing resources

This is a PhD Thesis

Original Citation:
Hyperparameter optimization of deep learning models using high-performance computing resources / Anwar, M.N.. - ELETTRONICO. - (2026).

Availability:
This version is available at http://hdl.handle.net/11589/304460 since: 2026-07-02

Published version
DOI:

Publisher: Politecnico di Bari

Terms of use:

(Article begins on next page)

16 July 2026

Politecnico di Bari

Department of Mechanics, Mathematics and Management

MECHANICAL AND MANAGEMENT ENGINEERING

Engineering and Aerospace Science Ph.D. Program

SSD: FIS/01 - Experimental Physics
ING-IND/14 – Advanced Mechanical Design

Final Dissertation

Hyperparameter Optimization of Deep Learning Models Using High-Performance Computing Resources

---

by

Muhammad Numan Anwar

Supervisors:

Prof. Nicola De Filippis

Prof. Marcello Abbrescia

Dr. Domenico Diacono

Coordinator of Ph.D. Program:
Ch.ma Prof.ssa Caterina Ciminelli

CATERINA CIMINELLI
21.06.2026 06:54:11
GMT+02:00

Course n° 38, 01/04/2023–31/03/2026

Contents

Abstract

This thesis focuses on the hyperparameter optimization of deep-learning models using high-performance computing (HPC) resources, in order to study cluster counting in drift chambers. Cluster counting is a promising particle-identification technique in particle-physics experiments, where the goal is to reconstruct the number of primary ionization clusters from digitized waveforms. To support this study, realistic simulated samples are first generated using a Garfield based simulation package under conditions as close as possible to the real test-beam data recorded in 2022 (180 GeV) and 2023 (2–10 GeV).

The next part of this work focuses on the training, optimization, and selection of best machine-learning models using HPC resources on simulated samples for the cluster counting study in two main steps. In the first step, several jobs with different sets of hyperparameters were submitted simultaneously on the local ReCaS Bari HPC system to train Long Short-Term Memory (LSTM) models on simulated samples matched to the 2022 and 2023 real test-beam data for the peak-finding algorithm. This stage was treated as a classification problem, in which signal peaks were separated from the noise background in the waveform. Different hyperparameter settings, including activation functions, optimizers, number of epochs, batch size, patience, and dropout rate etc, were systematically explored, while computational factors such as CPU usage, memory consumption, and job duration were also taken into account. The best LSTM model was selected according to the highest area under the curve (AUC) among all tested configurations. In the second step, the same model-selection logic was applied to the Convolutional Neural Network (CNN) model, which was used for clusterization. This stage was treated as a regression problem, in which the number of primary ionization clusters was estimated from the peaks identified in the first step. Among all tested hyperparameter configurations, the best CNN model was selected according to the lowest mean squared error (MSE). The optimized LSTM and CNN models were then further evaluated on simulated samples of kaons and pions at 2, 4, 6, 8, and 10 GeV for particle-identification studies. In addition, the performance of the selected models was evaluated on national HPC resources and compared with that on the local ReCaS Bari HPC system in terms of CPU usage, memory consumption, and job duration.

The final and most important task is related to the application of optimized machine-learning models to real test-beam data. For this purpose, the simulated samples corresponding to the 2022 test-beam data are further tuned in terms of noise and maximum amplitude. After this tuning procedure, the LSTM and CNN models are trained on the tuned simulated samples using HPC resources. Among all tested hyperparameter configurations, the best LSTM model for peak finding is selected according to the highest area under the curve (AUC), while the best CNN model for clusterization is selected

according to the lowest mean squared error (MSE). Finally, the best selected CNN model is applied to the real digitized waveforms to reconstruct the number of primary clusters, and its performance is compared with that of the Running Template Algorithm (RTA).

Introduction

The main goal of this thesis is related "Hyperparameter Optimization of Deep Learning Models Using High-Performance Computing Resources" for the cluster counting study in drift chamber using machine learning algorithm. Its main idea is to train and test neural network models, such as the Long Short-Term Memory (LSTM) Model for the classification and Convolutional Neural Network (CNN) Model for regression on simulated samples matches with the real test beam 2022 and 2023 data using high performance computing resources such as CPU's, memory usage, and training time.

To train the neural-network models within the HPC-based optimization framework, realistic simulated samples are first generated using Garfield^{++} under detector conditions matched to the 2022 and 2023 real test-beam data. The simulation parameters include the muon momentum, cell size, angle between the muon track and the sense wire, and gas mixture etc. For this purpose, a detailed first-principles simulation package is developed, consisting of two main components: simulation and digitization. In the simulation stage, the geometry of the drift-chamber cells is constructed, and the ionization produced by charged particles is modeled using the Heed package. The analog waveforms of the drift-chamber cells are then generated by modeling the drift and diffusion times, together with the amplitude and pulse shape of the ionized electrons. In the digitization stage, noise is extracted from experimental data using the Fast Fourier Transform and added to the signal through the Inverse Fast Fourier Transform. This digitization procedure produces realistic digitized waveforms that reproduce the main features of the experimental data, particularly the peak rise time and noise level.

The second task of this thesis focuses on the training and hyperparameter optimization of machine-learning models using HPC resources for cluster-counting studies in drift chambers, and is divided into two main steps. In the first step, several jobs are submitted simultaneously on the local ReCaS Bari HPC system to train Long Short-Term Memory (LSTM) models for peak finding. This stage is treated as a classification problem, in which signal peaks are separated from the noise background in the waveform. Different sets of hyperparameters, including activation functions, optimizers, number of epochs, batch size, patience, and dropout rate, are systematically explored, while computational factors such as CPU usage, memory consumption, and job duration are also taken into account. The best LSTM model is selected according to the highest area under the curve (AUC) among all tested configurations.

In the second step, the same model-selection logic is applied to the Convolutional Neural Network (CNN) model, which is used for clusterization. This stage is treated as a regression problem, where the number of primary ionization clusters is estimated from the peaks identified in the first step. Among all tested hyperparameter configurations, the best CNN model is selected according to the lowest mean squared error (MSE). The optimized

LSTM and CNN models are then further evaluated on simulated samples of kaons and pions at 2, 4, 6, 8, and 10 GeV for particle-identification studies. In addition, the performance of the selected models is evaluated on national HPC resources and compared with that on the local ReCaS Bari HPC system in terms of CPU usage, memory consumption, and job duration.

The final task of this thesis focuses on the application of the optimized machine-learning framework to real test-beam data. In this stage, the local ReCaS Bari HPC resources are used to train and optimize the neural-network models on tuned simulated samples, while the real test-beam data serve as the target application for validation. The main objective is to reconstruct the number of primary ionization clusters from digitized waveforms using the best-selected machine-learning model and to compare the results with those obtained from the Running Template Algorithm (RTA). For this purpose, simulated samples are generated under the same experimental conditions as the real test-beam data, including the cell size, muon momentum, angle between the particle track and the sense wire, and gas mixture etc. These samples are then tuned in terms of noise and maximum signal amplitude to improve the agreement between simulation and data. After the tuning procedure, the Long Short-Term Memory (LSTM) model for peak finding and the Convolutional Neural Network (CNN) model for clusterization are trained on the tuned simulated samples using the local ReCaS Bari HPC resources. Among all tested hyperparameter configurations, the best LSTM model is selected according to the highest area under the curve (AUC), while the best CNN model is selected according to the lowest mean squared error (MSE). Finally, the selected LSTM and CNN models are applied to the real test-beam data for cluster counting, and the final number of primary clusters reconstructed by the CNN is directly compared with the result obtained from the Running Template Algorithm (RTA).

The thesis is structured as follows:

- Chapter 1 introduces the ICSC National Centre and its organizational structure, with particular emphasis on Spoke 2, which focuses on Fundamental Research and Space Economy, and on WP2, dedicated to the development of science-driven tools and innovative algorithms for Experimental High Energy Physics. Within this framework, the chapter highlights the Quasi-Interactive Analysis project, and motivates the use of high-performance computing (HPC) resources, including CPUs, memory usage, and training-time estimation, for the hyperparameter optimization of deep learning models such as LSTM and CNN in cluster-counting and particle-identification studies for the Future Circular Collider (FCC).

- Chapter 2 summarizes the basic concepts of different detectors, such as the Vertex Detector, Preshower Detector, and Muon Detector System, with particular focus on drift chambers, which is the central tracking part of the IDEA detector for the Future Circular Collider (FCC).

- Chapter 3 describes the Garfield++-based simulation for the generation of ionization using HEED, as well as the production of analog and digitized waveforms for the cluster-counting study of drift tubes. The simulation uses the same parameters as those adopted in the 2022 and 2023 real test-beam data, such as the incident-particle momentum, cell size, gas mixture, and the angle between the incident particle and the sense wire etc
- Chapter 4 provides an overview of different neural network models, such as deep neural network (DNN), Long Short-Term Memory (LSTM) model, and convolutional neural network (CNN), with emphasis on their basic building blocks and architectures. These models are commonly used for the two-step reconstruction algorithm in cluster-counting techniques over different momentum ranges.
- Chapter 5 presents the machine-learning-based algorithm for cluster-counting techniques in comparison with the traditional algorithm. It describes the two main steps of the ML approach: in the first step, an LSTM model is used for peak finding in digitized waveforms as a classification task, while in the second step, a CNN model is used for clusterization to estimate the number of primary ionization clusters, N_{cls}, as a regression task.
- Chapter 6 describes the hyperparameter optimization of deep neural network models, such as the Long Short-Term Memory (LSTM) peak-finding model and the Convolutional Neural Network (CNN) clusterization model, for cluster-counting techniques across different momentum ranges of muons using high-performance computing (HPC) resources, such as CPUs, memory, and training time. In addition, it discusses the selection of the best LSTM and CNN models among all tested hyperparameter configurations, as well as the separation power of pion–kaon and muon–kaon for particle-identification performance.
- Chapter 7 presents the analysis of real test-beam data by reconstructing the number of primary clusters from digitized waveforms using machine-learning models and comparing the results with the Running Template Algorithm (RTA). To achieve this goal, the chapter describes each step in detail, including the real test-beam data setup, the tuning of the simulation in comparison with the real data, the training of the neural network models on tuned simulated samples, and their application to real data for the reconstruction of the number of primary clusters in comparison with the RTA result.

Chapter 1

ICSC: The Italian National Research Centre on HPC, Big Data and Quantum computing

This chapter focuses on the high-performance computing (HPC) resources used in this thesis, including CPUs, memory usage, and training-time estimation. These resources were used to train multiple deep learning models, such as a Long Short-Term Memory (LSTM) network and a Convolutional Neural Network (CNN), in order to study cluster-counting techniques for the Future Circular Collider (FCC).

The HPC activity presented in this chapter represents a core component of quasi-interactive, high-throughput big-data analysis. Activities within this framework include hyperparameter optimization of deep learning models using HPC resources, which is the main focus of this thesis, vector boson scattering analysis with hadronic τ leptons and light leptons, di-Higgs analysis, and differential cross-section measurements for inclusive $t\bar{t}$ production. Overall, quasi-interactive high-throughput big-data analysis is one of the main projects in Work Package 2 (WP2).

ICSC, the Italian National Research Centre on High-Performance Computing, Big Data, and Quantum Computing, is one of the five National Center established under the National Recovery and Resilience Plan (NRRP), covering designated strategic sectors for the development of the country, including simulations, computing, high-performance data analysis, Agritech, gene therapy and drug development through RNA technology, sustainable mobility, and biodiversity.

The activities of the National Supercomputing Centre will focus on maintenance and upgrade of the Italian HPC and Big Data infrastructure, as well as on the development of advanced methods and numerical applications and software tools to integrate computing, simulation, collection, and analysis of data of interest for research, manufacturing, and society, also through cloud and distributed approaches.

Soon the Centre will be implementing solutions that will lead to a network speed of 1 Terabit/second, making a cloud infrastructure available to users, enabling management of ground-breaking activities, in scientific research and industrial development.

It will involve and promote the best interdisciplinary skills of science of engineering, in favour of substantial and sustainable innovations in areas spanning from basic research to computing sciences for climate, the environment, space, from the study of matter and life to medicine, from technologies of materials to information systems and devices. It will support advanced training and promote the development of policies for responsible management of data from an open data and open science perspective, combining regulatory, standardization and compliance profiles.

ICSC will be a shared and open cloud/HPC infrastructure, representing a unique strategic asset for Italy, but for the international community as well.

1.1 Scenario and motivations

In the coming years, science, industry, and public institutions will generate an extremely large amount of data. The main challenge will not only be storing this data, but also extracting real social and economic value from it. In this context, *supercomputing*, *numerical simulation*, *artificial intelligence*, *high-performance data analytics*, and *Big Data management* become essential tools. They are needed to address major societal problems and to support sustainable, people-centred growth, enabling universities, companies, and institutions to develop new services and discoveries.

For many years, countries such as the United States, China, and Japan have advanced rapidly in these areas. Europe started later, but in the last decade it has built a clear strategy through major initiatives, including the *EuroHPC Joint Undertaking*[1], the *European Open Science Cloud (EOSC)*[2], the *European Processor Initiative (EPI)*[3], and the *Quantum Flagship*[4], in line with the European Data Strategy and the European approach to AI. EU Member States have therefore invested heavily in petascale and pre-exascale computing infrastructures, while EuroHPC has also placed exascale systems (and quantum computing) on its roadmap. In this framework, Italy co-funds the EuroHPC *Leonardo* system, a world-class pre-exascale Tier-0 machine designed to provide competitive HPC services to both Italian and European public and private users. After the COVID-19 pandemic, the European Union started the *NextGenerationEU* programme to support recovery and, at the same time, modernise European economies and society. Out of a total budget of about 806.9 billion euros, around 191.5 billion euros were assigned to Italy. Within Italy, about 30.88 billion euros were allocated to research and education, including funding lines such as “From Research to Business” (about 11.44 billion euros) and “Research Infrastructures” (about 1.58 billion euros).

To build on these investments, in 2022 a project was prepared to implement the Italian

national strategy for HPC and Big Data, focusing on the following actions:

- build a world-class **supercomputing cloud** to store, manage, and analyse data, together with general-purpose cloud infrastructure;
- create **centres of excellence** with expert teams to develop domain-focused applications;
- strengthen links between **research and industry**, supporting both large organisations that require HPC and SMEs that need easy and scalable services;
- train **young scientists** and also managers, so they can become skilled in these technologies;
- introduce measures that increase **innovation and Technology Readiness Level (TRL)**, including support for SMEs without dedicated HPC skills;
- build national capacity by **sharing expertise** across the wider scientific community and reinforcing weaker areas;
- assess and maximise the **social and economic impact** at local, national, and EU level, with attention to ethical aspects;
- establish a dedicated group of experts to **monitor ethical issues** related to Big Data use and management.

Within this framework, the *Italian National Research Centre on HPC, Big Data and Quantum Computing (ICSC)* [5] is funded under the "From Research to Business" initiative with about 320 million euros, and it is also supported by synergic actions such as the *Terabit Network for Research and Academic Big Data in Italy.*

In addition, the *Terabit Network for Research and Academic Big Data in Italy (TeRABIT)*[6] has received about 41.0 million euros under the "Research Infrastructures" initiative. Together with these complementary investments, ICSC aims to build a national digital infrastructure that supports research and innovation across Italy.

The idea is to start from the existing national resources in *HPC*, *high-throughput computing (HTC)*, and *Big Data*, and then evolve them toward a more integrated model based on a **cloud data-lake**. In simple terms, this means providing a unified environment where data can be stored, accessed, and analysed more easily by both scientific and industrial users.

To achieve this, ICSC plans to offer flexible and consistent cloud-based interfaces (for example, web portals and common access tools), supported by a dedicated team of experts. At the same time, the project aims to create an attractive national ecosystem through strategic **public–private partnerships**, so that advanced infrastructure can be fully used for scientific and technological computing and can also stimulate the development of new computing technologies.

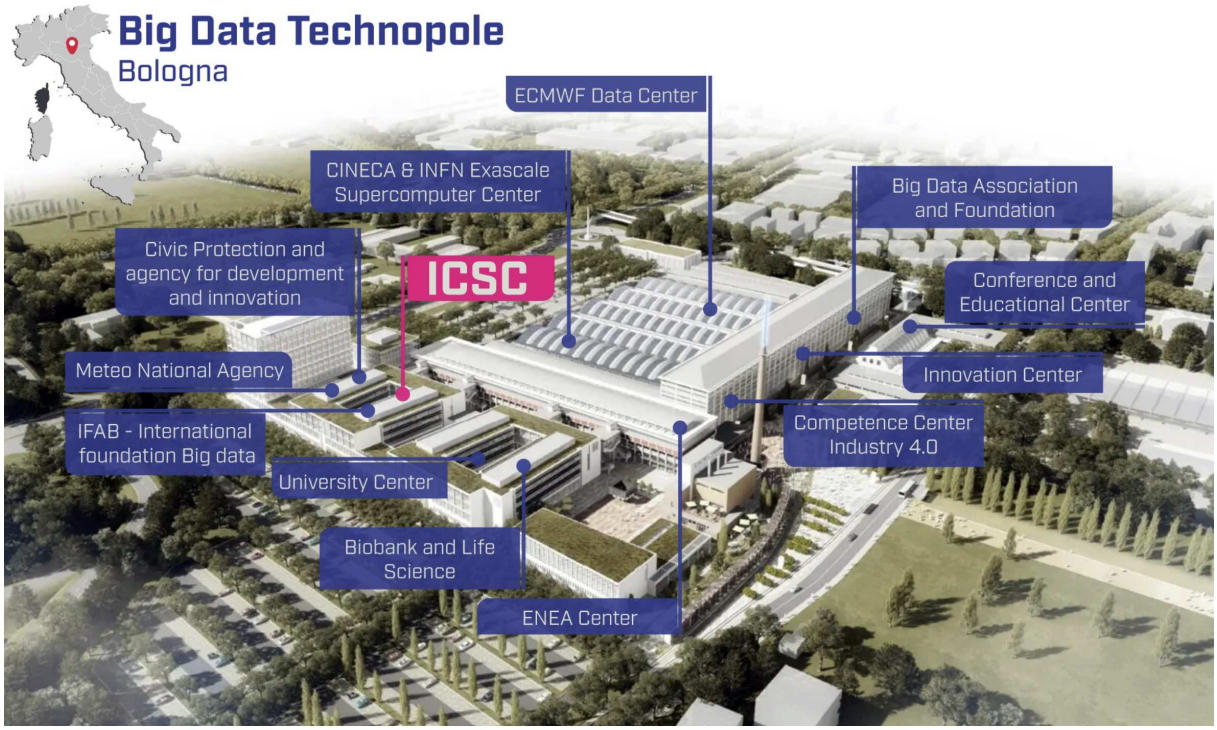


Figure 1.1: Big Data Technopole (Bologna, Italy), hosting key infrastructures and stakeholders connected to the ICSC activities.

1.2 Organization of the National Centre

The activities of the National Centre follow a *Hub-and-Spoke* model. The **Hub** mainly takes care of management and coordination, while the scientific and technological work is carried out by **11 Spokes** covering different research domains and application areas. In particular, **Spoke 0** is dedicated to deploying, optimising, and improving the national infrastructure (both hardware and middleware). Its core activities are located at the *Big Data Technopole* in Bologna, Italy (see Fig 1.1).

The other **10 thematic Spokes** focus on specific research topics. and the full set of Spokes is summarised in Fig.1.2. In the following, each Spoke is introduced briefly, while **Spoke 2** is discussed in more detail, especially the work packages 2 in the next sections because it is very close to my thesis goals.

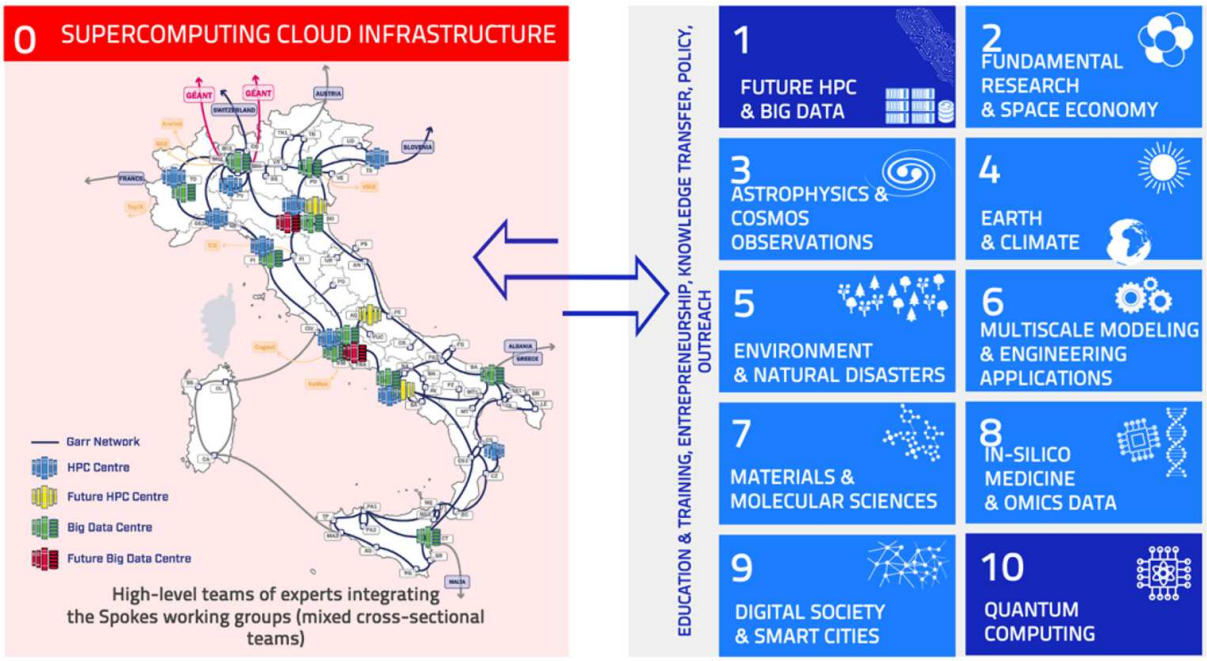


Figure 1.2: The eleven spokes of the Italian National Research Centre on HPC, Big Data and Quantum Computing (ICSC).

1.2.1 Spoke 0: "Supercomputing cloud infrastructure"

Spoke 0 coordinates and delivers the key actions needed to build a competitive, federated national infrastructure for High-Performance Computing (HPC) and Big Data. A central task is strengthening the GARR-T research network[7] to provide dedicated, very high-speed connectivity for research and education, reaching multi-terabit-per-second performance. In parallel, Spoke 0 supports the upgrade of computing and data facilities, with strong attention to energy efficiency, and promotes the use of open-source middleware so users can access distributed resources through a unified and smooth interface. An additional priority is to reduce gaps between regions by improving services in underserved areas and helping to bridge the digital divide.

These activities are aligned with similar initiatives at the European level. In particular, HPC and Quantum Computing (QC) resources are co-funded within the EuroHPC Joint Undertaking[8], while distributed computing services and data-management approaches are connected with the European Open Science Cloud (EOSC) framework[9]. The national research network is also part of the wider European GÉANT ecosystem.

A major synergic effort is provided by the TeRABIT project[6], which reinforces the digital infrastructures of GARR[10], CINECA[11], and INFN[12] in coordination with Spoke 0. Under GARR leadership, the national network is being upgraded toward terabit-scale connectivity, and new technologies are introduced to make connections more flexible and easier to manage. At the same time, computing centres are modernised to increase capacity, improve energy efficiency, and expand computing and storage resources. For

example, CINECA plans to increase the capacity of the Leonardo supercomputer by about 30% and to integrate it with a quantum computer, while INFN is strengthening its distributed computing system to serve a broad range of scientific communities. To make access simpler for both Italian and European users, a national cloud federation is being developed, following international standards promoted by initiatives such as EuroHPC, EOSC, and GAIA-X[13].

Finally, Spoke 0 also provides user services and consultancy to help researchers use these resources effectively. A dedicated support team is being set up to transfer know-how and improve skills in distributed and high-performance computing, so that the community can make the best use of the available infrastructure.

1.2.2 Spoke 1: "Future HPC and Big Data"

Spoke 1 is organised around five scientific–technological *flagship projects* that act as the main operational engines of the research programme. These flagships bring together contributions from 15 universities (and research institutions) and 9 major industrial partners. In addition, Spoke 1 includes two *living labs*, i.e., physical facilities hosted at leading universities (the University of Bologna and the University of Torino).

The flagship projects are designed to combine expertise and results from different centres, encouraging integration and cross-fertilisation, while also responding to the needs of industrial partners. The living labs collect and showcase contributions from all flagships by prototyping and demonstrating the most promising technologies. They also support technology transfer and aim to keep Spoke 1 activities sustainable beyond the funding period supported by the NRRP. More broadly, the living labs target the creation of a nationally federated centre with internationally recognised expertise in hardware–software co-design, strengthening Italy's role within the EuroHPC Joint Undertaking [14] and within the wider data-infrastructure ecosystem for both science and industry. Spoke 1 also supports local SMEs through open *cascade funding* calls, helping build a stronger national HPC supply chain.

The research themes of Spoke 1 cover: new HPC technologies; cloud computing and applications for Big Data and Artificial Intelligence (AI); prototypes of innovative computing systems; performance analysis with strong attention to energy efficiency; key non-functional properties (energy, power, reliability, portability); heterogeneous acceleration (architecture, tools, and software); workflows and high-performance I/O; cloud–HPC convergence and digital twins; confidential computing, trusted execution environments, and federated learning; as well as mini-applications and benchmarking.

A key methodological choice is to promote open licences (open hardware, open-source software, and open standards) as drivers of innovation. This approach can improve product quality and long-term sustainability of industrial solutions. One example is the adoption

of the RISC-V instruction set architecture, an open-standard ISA whose specifications are freely available and which supports diverse implementations [**?**, **?**].

1.2.3 Spoke 2: "Fundamental Research and Space Economy"'

Spoke 2, is one of the main scientific components of the National Centre and focuses on advanced computing solutions for data-intensive research domains. Since the HPC-based work presented in this thesis is developed within the framework of Spoke 2, its objectives, scope, and internal organization are discussed in more detail in Section 1.3.

1.2.4 Spoke 3: "Astrophysics and Cosmos Observations"

Spoke 3 supports modern astrophysics and cosmology, where research is increasingly driven by very large datasets and complex simulations. Its main goal is to provide efficient *High-Performance Computing (HPC)* and *Big Data* solutions for a wide range of topics, including cosmology, stars and galaxies, space physics (Earth, solar, and planetary), radio and time-domain astronomy, high-energy astrophysics, cosmic microwave background studies, large-scale structure, multi-messenger observations, and simulation-based modelling. In addition to technology, Spoke 3 invests in outreach and training to grow a skilled community and to encourage broad use of high-performance and high-throughput cloud solutions across universities, research institutes, and industry.

The activities follow a user-driven approach, so developments are aligned with real scientific needs. When building selected applications, a co-design method is used: scientists and community code developers work together with HPC and cloud experts to connect domain requirements with improved software and hardware solutions. The programme also promotes greener computing, aiming for coordinated progress in both applications and enabling technologies.

The work is organised into the following Work Packages (WPs). WP1 improves and optimises HPC codes by redesigning, re-implementing, and tuning software to fully exploit modern systems, including accelerators and alternative architectures such as GPUs and ARM. WP2 develops and upgrades algorithms and methods that scale toward exascale and post-exascale platforms, and then integrates these advances into real codes, workflows, and pipelines. WP3 focuses on big-data analysis, machine learning, and visualisation at scale, supporting the combined use of HPC, *High-Throughput Computing (HTC)*, *High-Performance Data Analytics (HPDA)*, and cloud tools. WP4 addresses data management, storage, and long-term archiving, following *FAIR* principles (Findable, Accessible, Interoperable, Reusable) and Open Science practices. WP5 provides the services and access layer by operating an integrated environment and enabling efficient development across all WPs, including national Data Lake support for managing and distributing large datasets and for high-rate analyses. Finally, WP0 provides project

management support, and all WPs are designed to work together and with interested private stakeholders connected to Spoke 3.

1.2.5 Spoke 4: "Earth & Climate"

Spoke 4 builds an interdisciplinary network that brings together modern Earth system modelling and provides the community with reliable and easy-to-use tools. Its main goal is to create a digital infrastructure that is fully integrated with the National Centre's HPC and cloud resources. This infrastructure will support three key activities: (i) developing and sharing Earth System Model (ESM) components (for example, models of the atmosphere, ocean, sea ice, land surface, vegetation, critical zone, and human–activity interactions), (ii) managing and producing large numerical simulations, and (iii) preparing a sustainable environment so the infrastructure can grow into a national resource. In this way, the platform can be used by the wider Italian research and education community, including groups working on climate forecasting and climate-change studies. Overall, Spoke 4 aims to strengthen Italy's position in advanced climate research and to support the NRRP goals for the digital and green transitions.

1.2.6 Spoke 5: "Environmental & Natural Disasters"

Recent advances in sensors deployed over wide areas (e.g., Internet of Things, IoT) and the ability to process large datasets make it possible to study environmental change and natural hazards in new ways. In this context, Spoke 5 focuses on understanding and reducing risks related to environmental degradation and the increasing frequency and severity of natural disasters, which are often connected to climate change.

Spoke 5 works closely with industrial partners and develops methods to monitor both built infrastructure (such as buildings and roads) and natural environments (such as rivers and slopes). Using the computing resources of the National Centre, it aims to build *digital twins*—virtual models of environments and infrastructures—to simulate and predict how these systems may behave during disasters or under changing environmental conditions. The spoke brings together expertise from mathematics and physics, advanced computing, Earth sciences, and civil and geo-environmental engineering. Overall, its goal is to turn research results into practical monitoring tools and measurable indicators of socio-economic impact that can support decision-making and benefit society.

1.2.7 Spoke 6: "Multiscale Modelling and Engineering Applications"

Spoke 6 uses advanced large-scale computing (including exascale systems and industrial-grade clusters) to address engineering and scientific problems that span multiple scales and

are difficult to solve with standard approaches. By combining high-performance computing (HPC) with modern machine learning (ML) and artificial intelligence (AI), this Spoke supports the development of digital twins and model-based design tools that can describe, predict, and improve the behaviour of complex systems. The overall goal is to speed up innovation, improve efficiency, and enhance safety in key engineering sectors, contributing to a cleaner and more sustainable future.

A central objective of Spoke 6 is to sustain innovation across different Technology Readiness Levels (TRLs), from early-stage concepts to solutions closer to real-world deployment. The targeted application areas have strong societal, industrial, and economic relevance for Italy and include sustainability, life sciences, digital technologies for Industry 4.0, advanced materials, and other complex systems.

Spoke 6 is organised around seven flagship projects. Through the National Centre, these projects can deliver proof-of-concept demonstrators and address concrete industrial challenges together with major companies in the Italian ecosystem (e.g., Thales Alenia Space, Leonardo, Terna, Fincantieri, Autostrade per l'Italia, Ferrovie dello Stato, ENI, and Engineering). SMEs and associations are also involved through the IFAB foundation, helping to connect research, computing resources, and real applications, such as the adoption of new materials in emerging technologies.

1.2.8 Spoke 7: "Materials and Molecular Sciences"

Over the years, progress in society has often been driven by new materials and by finding new uses for existing ones. Today, the energy, environmental, and climate challenges require a shift in how materials are developed. Instead of relying only on discovering and extracting natural materials, the focus is increasingly on designing artificial materials with properties tailored for specific applications, and on developing new ways to produce them. This change is supported by the digital revolution: modern computing can handle very large datasets and enables realistic simulations based on the fundamental laws that govern material behaviour. In this context, AI is becoming an important tool to help predict complex material properties and performance.

Spoke 7 aims to strengthen Italy's role in developing, improving, and sharing high-performance scientific software for simulating complex materials and molecular systems. It also works to broaden the range of applications of these software tools, so they can help address scientific, technological, and societal needs. In addition, Spoke 7 helps both public and private users access large-scale computing resources, supporting the adoption of new materials in emerging technologies.

1.2.9 Spoke 8: "In-Silico Medicine and Omics Data"

Spoke 8 focuses on biomedical research by using supercomputing to support in-silico medicine and to analyse very large datasets produced by omics studies (for example genomics, lipidomics, and transcriptomics). A key goal is to develop new clinical-trial approaches based on simulations, so that the time and cost of traditional trials can be reduced. In parallel, Spoke 8 aims to build a technology platform that can process and interpret big medical data using AI and machine-learning methods.

These activities are expected to improve our understanding of many diseases, support better diagnosis and prognosis, and accelerate drug discovery as well as the development of personalised treatments. By strengthening collaboration across different stakeholders, the spoke also aims to speed up experimental work in biomedical laboratories, which can help reduce the need for animal testing. Overall, integrating precision and personalised medicine into healthcare is expected to provide direct and measurable benefits for patients.

1.2.10 Spoke 9: "Digital Society and Smart Cities"

Spoke 9 addresses key applications for smart cities and digital society. Its work is organised into five main domains.

In *Health and Lifestyle*, the spoke uses HPC and Internet-of-Things (IoT) technologies to support e-health services. This includes tools such as digital twins for smart hospitals, better management of personal health records, and big-data platforms for advanced territorial healthcare management.

In *Mobility*, the spoke studies and models how people move in cities, including large-scale crowd dynamics.

In *Socio-economic analysis*, the spoke develops models of human behaviour, learning, and adaptation, and it analyses economic flows. It also works on methods to identify and follow disinformation and hate speech on social media, and to monitor urban crime.

In *Infrastructure and Services*, the spoke focuses on smarter and more resilient infrastructure. Examples include next-generation smart-grid solutions, drone-assisted maintenance, improved radio-coverage planning for wireless networks, real-time monitoring, and risk analysis and forecasting for interconnected systems.

Finally, in *Environment*, the spoke monitors and predicts environmental conditions. It supports activities such as biodiversity restoration, management of urban ecosystems, and improved water-cycle management, together with other environmental applications.

1.2.11 Spoke 10: “Quantum Computing”

Quantum computing is a promising technology because it can potentially solve some problems much faster than today’s classical computers, and it may also change how we

store and process data. In particular, quantum computers could tackle certain complex tasks that would take an impractically long time on classical machines—a concept often referred to as "quantum supremacy". However, despite the strong interest and the many potential applications, quantum computing is still at an early stage. Reaching full technological maturity remains challenging, mainly because quantum devices are sensitive, their components must be very reliable, and building and operating large quantum systems requires careful planning.

For these reasons, Spoke 10 focuses on enabling the practical use of quantum computing through three main research directions: (i) developing applications where quantum processors act as accelerators for problems that are otherwise very difficult to solve; (ii) creating the hardware and software tools needed to plan, control, and integrate quantum computers with conventional computing systems; and (iii) defining a roadmap toward large-scale and scalable quantum computers.

1.3 The focus on fundamental research and Space Economy in Spoke 2

Modern fundamental research increasingly depends on large computing resources. This is especially true for physics, where new generations of experiments push the limits of what we can measure and simulate. In both theoretical and experimental work, communities such as particle physics (using accelerators and detectors) and gravitational-wave experiments produce extremely large datasets. Although these fields evolve on different timescales, they share the same core challenges: computing systems must scale to much larger workloads, run efficiently, and reduce energy consumption. Spoke 2 addresses these needs by designing and testing computing solutions that can support both current experiments and the next ones.

A key goal of Spoke 2 is to make advanced computing resources easier to access, whether they are provided by the National Centre or by external infrastructures. At the same time, Spoke 2 promotes the transfer of knowledge and technologies from basic research to industry, so that methods developed for demanding scientific problems can also support applications with national economic and industrial impact. In this context, Spoke 2 also connects to the emerging Space Economy, including services built around Mirror Copernicus data and, more broadly, the European Copernicus Earth-observation programme, which monitors the planet and its environment.

The algorithms and computing solutions developed within Spoke 2 are intended to be shared across the National Centre, encouraging interdisciplinary progress. These results will be made available to the wider Italian ecosystem through dissemination activities and technology-transfer initiatives.

Within Spoke 2, the work is organised into two groups of Work Packages (WPs). The *scientific* WPs (WP1, WP2, WP3) study the needs of each research sub-domain and identify open problems where improved computing approaches are required. The *technological* WPs (WP4, WP5, WP6) instead focus on investigating and deploying technical solutions in computing, both within the National Centre infrastructure and beyond. These technological WPs also provide targeted support and training, and they promote the developed solutions to a broader community, including interested industrial partners.

All activities follow a common four-phase plan over the three-year NRRP funding period. The first phase is *planning and identification*: the first project year is dedicated to surveying the state of the art and selecting the most relevant use cases, leading to a work plan that defines the main activities and the algorithms/services to be delivered. The second phase is *realization*, where the concrete development is carried out by staff and hired personnel, producing usable and documented algorithms/services (at alpha/beta level) that are ready for larger-scale testing. The third phase is *validation*, where the realized outputs are tested in dedicated testbeds and proofs of concept, and benchmarked to check their performance against the required specifications. The fourth phase is *wrap-up*, where results are analysed and consolidated into reports and white papers that can serve as guidelines for similar future use cases.

From the technology point of view, Spoke 2 explores several complementary directions. On the software side, machine learning is studied as a possible alternative to fully hand-written algorithms in selected tasks. In parallel, major efforts focus on porting and optimising codes for Graphics Processing Unit (GPU)-based systems, including heterogeneous frameworks, and for Field-Programmable Gate Array (FPGA)-based solutions, using either High-Level Synthesis (HLS) or lower-level programming. Low-power computing architectures are also considered, with ARM-based systems at present and potentially RISC-V systems in the near future. In addition, scientific codebases are being adapted to distributed-computing approaches, together with scalable data-access methods, containerisation, and efficient execution from single-node to multi-node environments, for example using the Message Passing Interface (MPI). On the hardware side, Spoke 2 aims to deploy accessible pilot platforms based on FPGA, ARM, and RISC-V technologies, and to provide cloud-accessible clusters, all integrated with standard cloud-based management systems.

1.3.1 Flagship projects:

Within each Work Package (WP), Spoke 2 defines a set of "flagship projects" (about 20 in total). Each flagship is a focused demonstrator: it tests a specific technology, algorithm, or computing approach on a realistic scientific workflow. The main projects of each WP

are listed below and also shown in figure 1.3, with particular emphasis on **WP2**, which is the core focus of this thesis.

- **WP1(Theoretical Physics):** Hybrid Monte Carlo for lattice QCD", "QCD under extreme conditions", "Advanced Calculus for Precision Physics", "Large Scale Simulations of Complex Systems"
- **WP2 (tools and algorithms for Experimental Collider Physics):** WP2 concentrates on techniques that can make simulation and reconstruction faster, and that can scale to future data volumes. Key flagships include:
  - "Advanced ML - Flash Simulation and bleeding edge applications";
  - "Porting Algorithms on GPU;
  - "Ultra-fast algorithms running on FPGAs;
  - "Validation of HEP reconstruction code on ARM;
  - "Quasi interactive analysis of big data with high throughput.
- **WP3 (tools and algorithms for Experimental Astroparticle Physics and Gravitational waves).** Flagships in WP3 are: "Frequency Hough (FH) Transform analysis on Gravitational Waves (GW) continuous sources", "Efficient use of ML and GPUs in cosmological data analysis: from theory to likelihood to statistical inference", "Inference of cosmological and astrophysical population properties from GW observations with and without electromagnetic counterparts", "Development of innovative analysis techniques using realistic simulations of the upgraded Auger Observatory within the context of a ML environment", "Detection and classification of SSO in Euclid Simulated data", "Pipeline optimization for space and ground based experiments", "Hydrodynamical simulations to test the nature of dark matter".
- **WP4 and WP5 (Cross-cutting technologies and services).** WP4 and WP5 do not have specific flagships, but participate to most of the other flagships, with themes like GPU /FPGA utilization, low power computing with ARM, high rate analysis solutions on the data lake; ML at large scale; fast simulations via generative networks.
- **WP6 (cross domain initiatives, +space economy).** Representative flagships include: ""Enhancing Geant4 Monte Carlo Simulations through ML Integration", "Fast Extended Computer Vision", "AI algorithms for (satellite) imaging reconstruction".

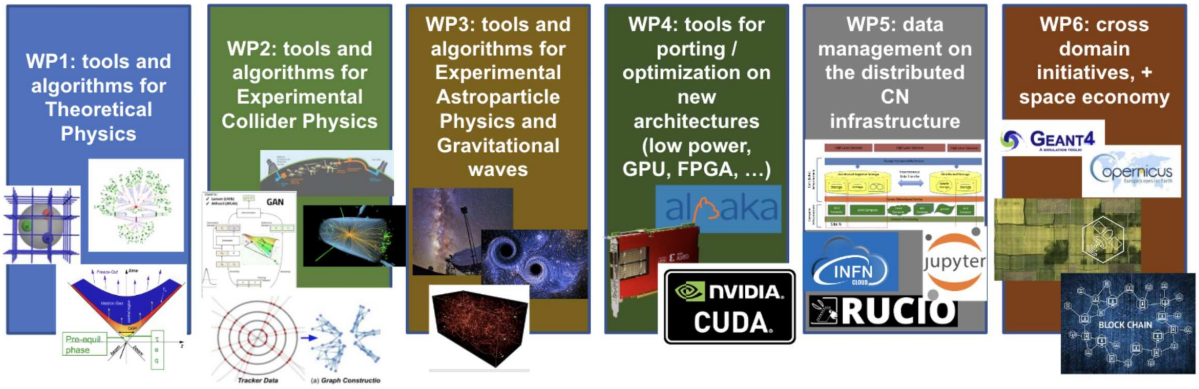


Figure 1.3: WP structure of Spoke 2 in the National Centre.

1.4 WP2: Design and development of science-driven tools and innovative algorithms for Experimental High Energy Physics

Experimental High Energy Physics (HEP) experiments produce extremely large amounts of data, so they need efficient computing methods at every step of the workflow. In WP2, the focus is on building and improving the key tools used to *select* interesting events, *reduce* the data to a manageable size, and *simulate* and *reconstruct* what happened in the detector. These methods can be developed with traditional programming, but also increasingly with large-scale Machine Learning models that can speed up complex tasks and improve performance. WP2 targets a wide range of experiments, including the LHC, future collider projects, and major laboratories such as KEK and IHEP, as well as neutrino experiments. The final aim is to make the full analysis chain faster, more accurate, and more scalable, enabling advanced applications such as new trigger ideas and distributed analysis techniques that can run efficiently on modern computing infrastructures.

1.4.1 WP2 flagship Use Cases (UC)

Within WP2, activities are organized into five main research lines, called *flagship use cases*. Each flagship use case (UC) project in Experimental High Energy Physics are briefly described below.

UC2.2.1 Advanced ML: flash simulation and other bleeding-edge applications

In modern LHC experiments, the most detailed detector simulation is responsible for most of the total CPU computing cost. For this reason, WP2 studies faster alternatives that can produce simulated events more quickly, where one of the most promising directions is the use of deep generative neural networks. The main idea is to speed up the expensive part of the

event-processing chain (from event generation to simulation, digitization, reconstruction, and analysis formats as shown in figure 1.4) by introducing "flash" approaches, and this work is already being pioneered in experiments such as LHCb and CMS. The goal is to develop and test these fast-simulation methods in at least two LHC experiments, and to explore other cutting-edge machine-learning applications that benefit from the same high-performance computing effort.

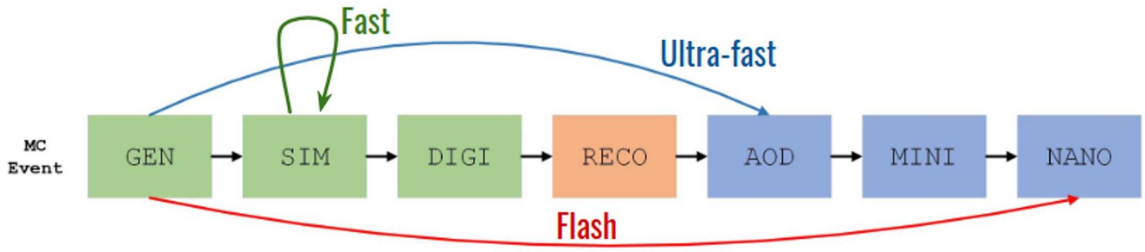


Figure 1.4: Event-processing chain (generation, simulation, digitization, reconstruction, and analysis formats) and the idea of "flash" approaches to accelerate the most expensive simulation steps.

As a concrete case study, the project considers flash simulation in LHCb and focuses on making complex analysis workflows easier to run across different types of computing resources. The key idea is to describe the workflow as a directed acyclic graph (DAG), so that different parts can be executed and "offloaded" smoothly to heterogeneous resources, such as HTC resources and HPC systems (see figure 1.5). In practice, the project uses tools such as Snakemake (with ongoing developments) to manage the workflow, and InterLink to connect HTC environments with HPC payload execution, in collaboration with the AI_INFN initiative.

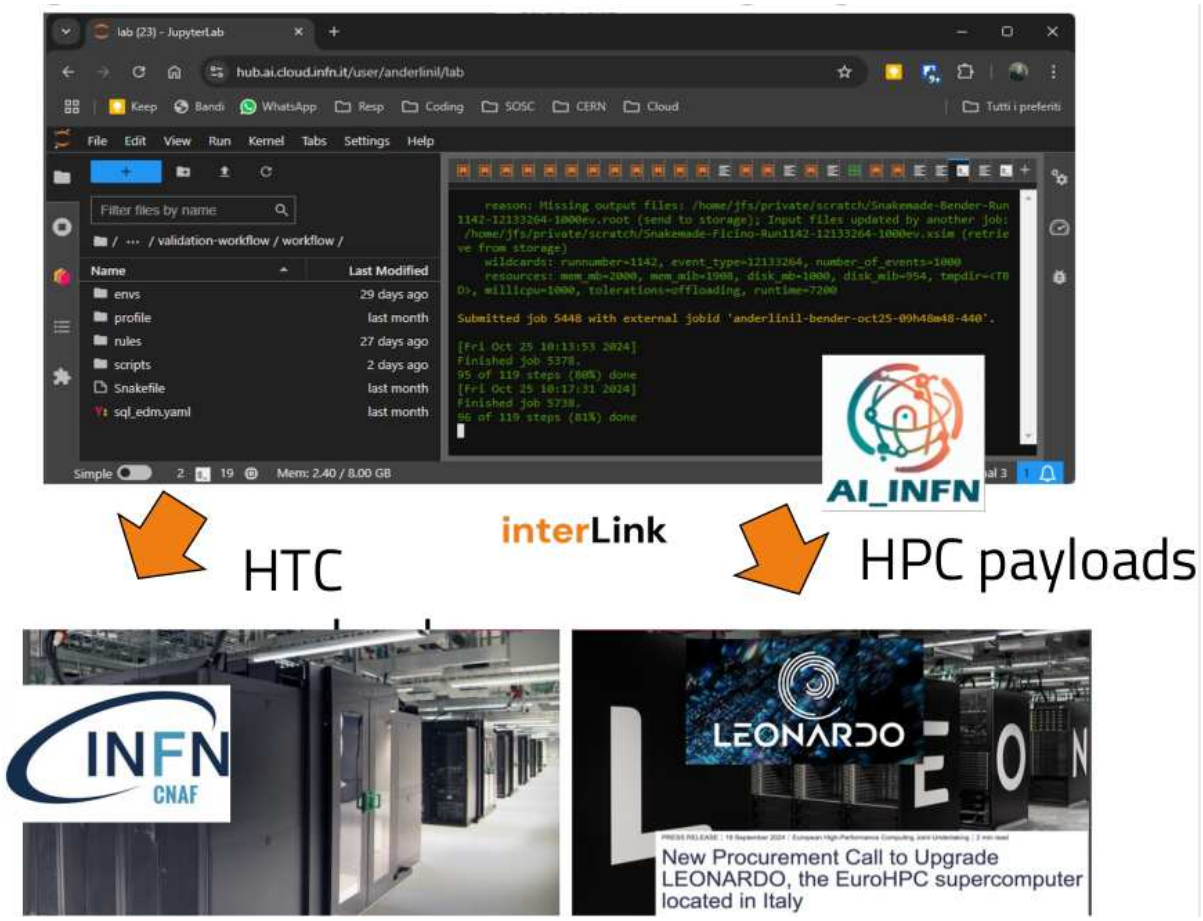


Figure 1.5: Flash-simulation workflow example showing how DAG-based execution can offload tasks between HTC resources and HPC payloads using tools such as Snakemake and InterLink.

UC2.2.3 Development of ultra-fast algorithms running on FPGAs

UC2.2.3 focuses on building very fast algorithms that run directly on FPGAs, so that key steps can be done immediately while data are being collected. The main goal is to develop FPGA-based solutions for **Trigger**, **DAQ** (data acquisition), and **online processing**. This includes developing trigger algorithms for several experiments (such as ATLAS and CMS), developing reconstruction algorithms (for example in LHCb), and enabling **scouting of data at 40 MHz** (for example in CMS). In addition, the project develops **FPGA toolboxes**, including tools for **algorithm synthesis** on clusters of FPGAs (e.g., *BondMachine*) and tools for **Direct Memory Access (DMA)** via Ethernet from FPGAs located on the detector front-ends as shown in figure 1.6.

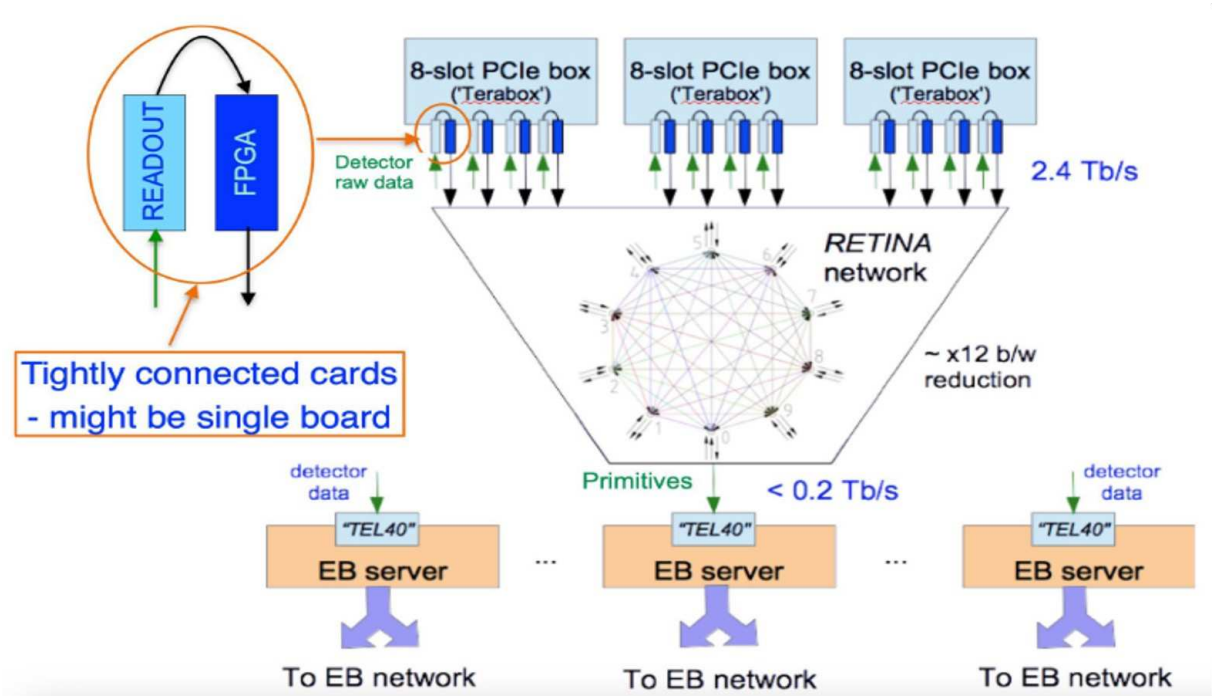


Figure 1.6: Overview of an ultra-fast FPGA-based algorithms for trigger, DAQ, and online processing, with real-time data handling and bandwidth reduction before transfer to the event-builder network.

Example: 40 MHz scouting at CMS. A key example discussed in this project is **L1 scouting at CMS**, where the idea is to acquire and analyze the **Level-1 Trigger (L1)** information for **all collisions** happening at a **40 MHz** rate. The target is to search for physics signatures that can be identified using only L1 trigger information, which could be missed by the standard CMS chain $L1T \rightarrow HLT \rightarrow Offline$(see figure 1.7). In addition to that it highlights why this is challenging: the backgrounds can be very large (including cases like narrow resonances with unknown mass), some signal-identification algorithms are too complex to fit within the L1 trigger constraints (for example due to combinatorics or complex neural networks), and some signals require **time-correlation across several bunch crossings** (for example long-lived BSM scenarios). This example is actively studied using an FPGA cluster in Milan procured within the project.

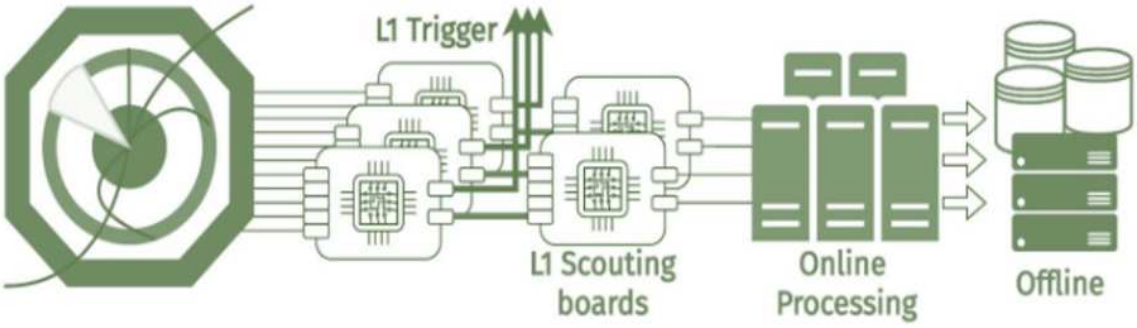


Figure 1.7: Concept of 40 MHz scouting: using L1 trigger information for all collisions to enable online selection and analysis beyond the standard processing chain.

UC2.2.4 Porting of algorithms to GPUs

Porting reconstruction and analysis algorithms to GPUs is a key step to reduce computing time in modern high-energy physics workflows. This activity builds on the experience already developed in LHCb, CMS, and ATLAS, and it also aims to grow a strong Italian community of people who can work with heterogeneous computing resources. A large part of the effort is focused on training developers, students, and collaborators to write algorithms that can run in parallel and can benefit from accelerators. In addition, the work supports experiments in extending GPU usage to new parts of their software, and it helps define practical strategies for integrating GPUs and other accelerators into full software frameworks. A specific focus is placed on portability solutions, in particular the Alpaka programming model as shown in figure 1.8.

Figure 1.8: Portability approach for heterogeneous accelerator programming, with a focus on Alpaka.

A major part of the activity is training and enabling developers, students, and collaborators to write algorithms that can run efficiently on different hardware. In particular, the project studies portability solutions and emphasizes `alpaka`, an abstraction library that allows the same parallel code to target different backends. This approach supports long-term software sustainability, because it reduces the need to rewrite algorithms separately for each GPU platform.

In practice, the project contributes through demonstrators and milestones in concrete reconstruction tasks. Example activities include porting the ALICE TPC (Time Projection Chamber) data decompression to GPUs, where the decoding algorithm has been shown to run on both AMD and NVIDIA devices. On the CMS side, the work includes developing a parallel algorithm for electron seeding on GPUs, improving primary-vertex reconstruction with a GPU-friendly implementation, and porting pixel and strip reconstruction components (including early layers relevant for Phase-2) to an `alpaka`-based implementation.

To ensure physics correctness and reliability, a dedicated validation infrastructure is developed and exercised on available pledged resources. The validation checks the `alpaka`-based GPU reconstruction against CPU workflows using pixel-track performance as key metrics. One validation approach runs different setups on the same RAW events

(including pre-`alpaka` and `alpaka` configurations) and requires a perfect match. A second approach runs GPU and CPU reconstruction in the same job and compares results event-by-event, including comparisons between `alpaka` and native CUDA behavior. The validation program is also extended to additional architectures by testing on AMD resources at CNAF, including local studies comparing AMD and CUDA and connecting these tests to the CMS submission infrastructure as given in figure 1.9.

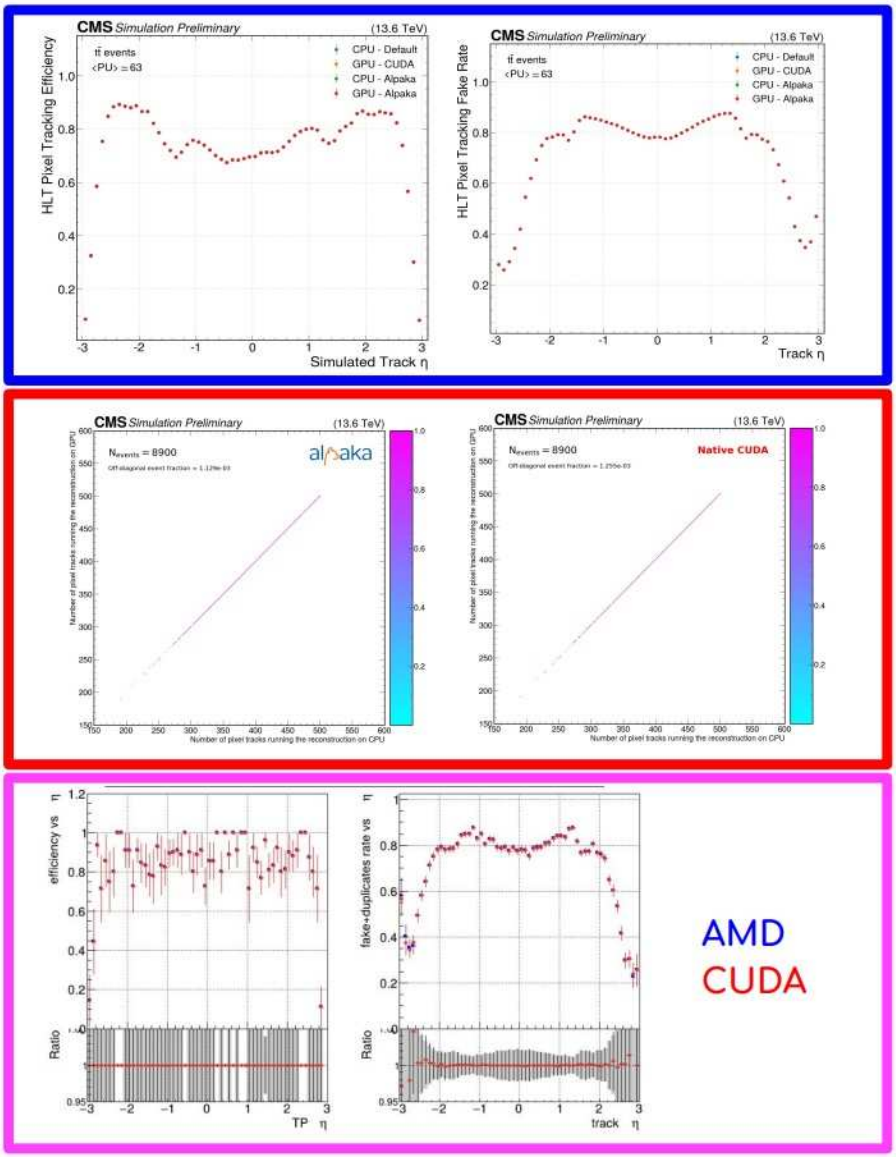


Figure 1.9: Validation strategy comparing CPU and GPU reconstruction, including Alpaka and native CUDA, and cross-platform tests.

UC2.2.5 Physics validation of reconstruction code on ARM

This activity focuses on the physics validation of reconstruction software on ARM processors for at least two LHC experiments. ARM builds already exist for the software of the four main LHC experiments, but large-scale physics validation is still missing, meaning

comparisons that run on the order of a million events. The work therefore uses four ARM machines at CNAF and sets up a validation strategy that can run representative workflows and compare the results to standard x86 executions, including performance and resource-related metrics as shown in figure 1.10.

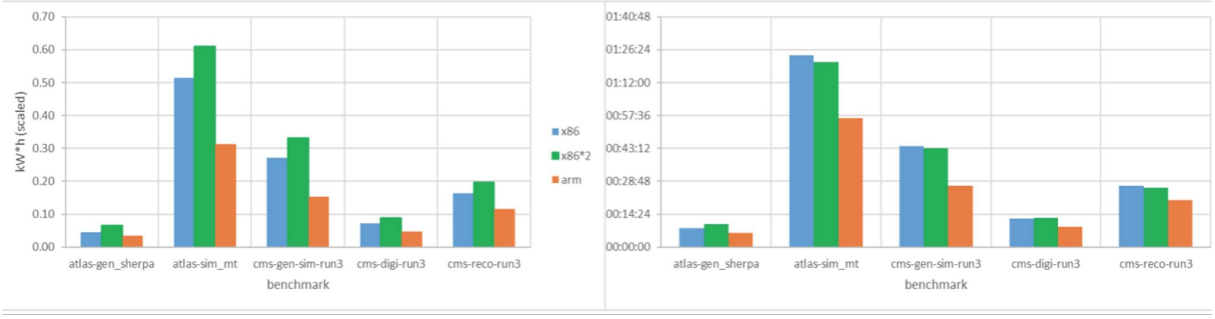


Figure 1.10: Benchmark-style comparisons across representative reconstruction workflows on x86 and ARM, used to support large-scale validation.

The main use cases come from the ALICE, ATLAS and CMS experiments, which are already running tests and configuring the ARM nodes to execute different workflows such as data processing, Monte Carlo production, and machine-learning related tasks. Current tests show that these workflows can run successfully on the ARM nodes, and the next step is to carry out systematic measurements at CNAF, including power-consumption monitoring. Early studies also indicate a promising reduction in memory usage on ARM compared to x86, as observed in preliminary tests for Pb–Pb Monte Carlo simulations in ALICE (see figure 1.11).

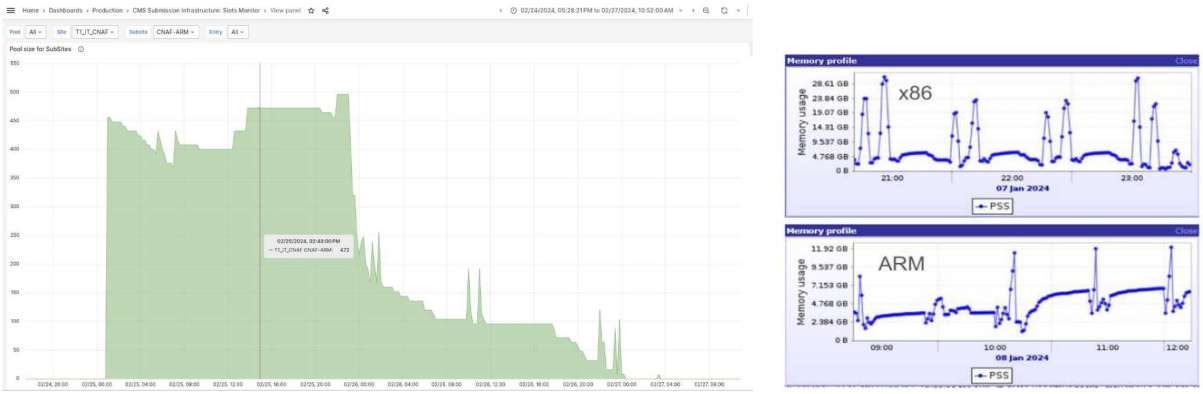


Figure 1.11: Operational monitoring on ARM nodes at CNAF (left), and memory-usage profiles comparing x86 and ARM for Pb–Pb Monte Carlo simulation workloads in ALICE (right).

1.4.2 UC2.2.2 Quasi interactive analysis of big data with high throughput

"Quasi interactive analysis of big data with high throughput" is one of the main project of the Work package (WP2) which is closely relevant to my thesis. The following sections describe the motivation, activties and especially Neural Network optimization using HPC resources such as CPU's, memory usage and time taken for each job.

Motivation

Many modern applications need to ingest, process, and analyse very large datasets with the shortest possible turnaround time. This is especially true in High Energy Physics, where the next phase of the LHC will produce very large data volumes and analysts will need to iterate quickly on selections, calibrations, and validation studies. For this reason, UC2.2.2 focuses on enabling flexible access to big data using distributed computing resources, including the storage and compute services planned within the ICSC DataLake.

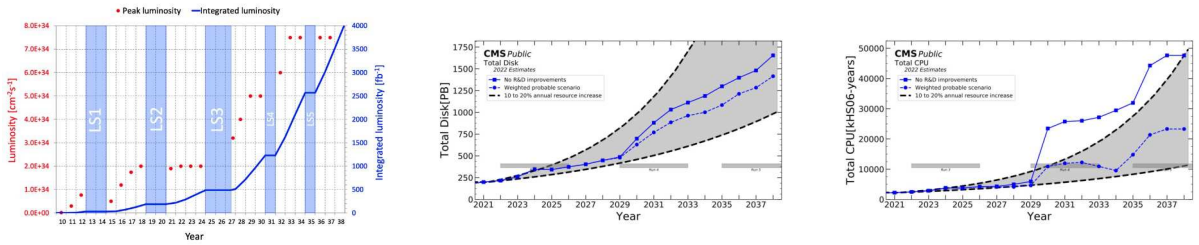


Figure 1.12: The upcoming high-luminosity phase at the CERN Large Hadron Collider (LHC) will require an increasing amount of computing and storage resources.

High-throughput analysis workflow within ICSC

A high-throughput analysis workflow can be viewed as a pipeline that starts from detector data and ends with user-level analysis results (see figure 1.13). First, the data stream is reduced by trigger systems so that only the most useful events are kept for further processing. After this reduction, the remaining data is analysed using scalable computing resources, where many tasks can run in parallel to keep the turnaround time short.

In UC2.2.2, the key point is that the user interacts with the analysis through a simple interface, while the heavy work is executed on distributed resources in the background. In practice, analysis code is developed and tested in an interactive environment such as Jupyter, and then scaled out when higher throughput is needed. At the same time, resource monitoring is essential, because it shows whether CPU, memory, and workers are being used efficiently and helps identify bottlenecks during large runs.

This workflow is closely releted to neual network models training on HPC resources, because model development requires repeated experiments, fast feedback, and stable

execution at scale. Using the same facility idea, training jobs and evaluation tasks can be launched across distributed resources while keeping the development loop interactive and reproducible.

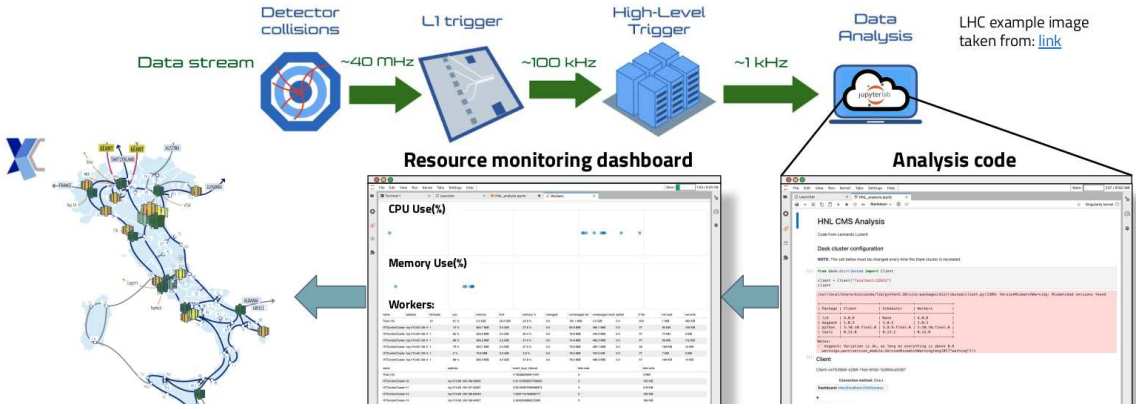


Figure 1.13: Overview of the high-throughput analysis workflow within ICSC, from detector data and trigger selection to interactive analysis, distributed computing, and resource monitoring.

Tools and software stack

The facility is based on widely used open-source tools:

- **Jupyter Notebooks:** the interactive front-end where analysis code is written and tested[15].
- **Dask:** a framework that can scale the same analysis to distributed resources when higher throughput is needed [16].
- **HTCondor:** a workload manager used to schedule jobs, integrate heterogeneous resources, and support fair-share usage [17].

Together, these tools allow users to run interactive development locally and then scale to large computing capacity without changing the overall analysis logic.

Local deployments and reproducibility

In addition to the central facility, local deployments support easier setup and more reproducible workflows. One example is a local testbed based on **OpenStack IaaS** [18] combined with container technologies such as **Rancher Kubernetes Engine (RKE)**[19] and **Docker**[20]. This setup makes services easier to update and helps keep software environments consistent. Similar R&D efforts exist at different sites, including the **Bari ReCaS** centre, with the long-term goal of integration into a wider ICSC ecosystem.

My contribution: Neural Network Model Training on HPC Resources

Use case UC2.2.2 includes several high-throughput analysis activities, such as Vector Boson Scattering analysis, heavy neutral lepton searches, differential cross-section measurements for inclusive $t\bar{t}$ production, di-Higgs studies, muon detector performance analysis, and benchmark interactive analysis for FCC-ee. Within this framework, **my contribution is the optimization of neural-network models using HPC resources for Future Circular Collider (FCC) studies**. In particular, I used both the local Bari ReCaS HPC infrastructure and national computing resources to train and evaluate neural-network models for cluster-counting studies. This work required many repeated training runs with different hyperparameter settings, so HPC resources were essential to make the workflow fast, efficient, and reproducible. Since this activity is motivated by future collider studies, the next chapter introduces the Future Circular Collider and the physics context relevant to this thesis.

1.4.3 Neural Network (NN) Model Optimization Using HPC Resources

In this section, I briefly introduce the neural-network optimization work carried out using High-Performance Computing (HPC) resources for studies related to the Future Circular Collider (FCC). The FCC, which is introduced in Chapter 2, provides the scientific context and motivation for this work.

The main goal of this study was to train and test neural-network models with different hyperparameter settings, such as the activation function, optimizer, number of epochs, batch size, early-stopping patience, and dropout rate. In this work, two neural-network models were optimized.

Because this work required many repeated training cycles, HPC resources were important for carrying out the study in a systematic and efficient way. During this process, practical computing aspects such as execution time, memory usage, CPU usage, and job duration were also monitored on both local Bari ReCaS resources and national computing resources. The detailed optimization procedure, the selection of the final models, and the corresponding performance results are presented in Chapter 6 and Chapter 7.

Chapter 2

Future Circular Collider (FCC) and the IDEA Detector

The optimzation of deep learning models, studies related to Future Circular Collider (FCC) experiments was mentioned in the previous chapter. Building on that context, this chapter, introduces the FCC by summarizing its accelerator design, expected performance, and key detector requirements. It then describes the overall layout of the IDEA detector and briefly explains the role of its main subsystems, such as the vertex detector, dual-readout calorimeter, and preshower detector. Finally, this chapter focuses on the Drift Chamber (DCH), including its mechanical design, structure, cell size, tracking performance, particle-identification capability, and wire configuration.

2.1 Introduction to the Future Circular Collider (FCC)

The Future Circular Collider (FCC) study explores the feasibility of building next-generation particle colliders in a circular tunnel with a circumference of about 100 km [21]. The detection of the Higgs boson at the Large Hadron Collider (LHC) completed the particle content of the Standard Model (SM) and marked an important milestone in our knowledge of fundamental interactions. Over many decades, Standard Model has proven to be a very successful and reliable theory, accurately describing all phenomena that have been observed in collider experiments so far. However, several experimental observations cannot be explained within this framework, indicating that the Standard Model is incomplete. Addressing these open questions requires new collider facilities with higher energy and higher collision rates. In this context, the FCC study provides the basis for a new research infrastructure that could succeed High-Luminosity LHC (HL-LHC) and support global particle-physics program throughout the 21st century, with ambition of exploring physics ahead from the Standard Model.

Within FCC program, the FCC-ee is planned as a high-precision electron–positron

collider made to study natures fundamental laws at very small distance scales. It will enable extremely accurate measurement of the Z and W bosons, the Higgs boson, and the top quark by collecting very large data samples, including trillions of Z bosons and millions of Higgs bosons and top-quark pairs. These large statistics allow precise comparisons between experimental results and Standard Model predictions, making it possible to detect small deviations, rare decay processes, or signals undiscovered particles which interact only weakly with known matter. To achieve the potential of this physics, FCC-ee will operate at several collision energies in a staged manner, starting at the Z-boson energy, followed by the W-boson and Higgs production energies, and finally reaching the energy required for top-quark pair production. This approach ensures that each particle can be studied under optimal experimental conditions

2.1.1 Accelerator Design and Layout

The accelerator of the FCC-ee is designed to deliver very high luminosity across a broad area of center-of-mass energies, from 88 GeV upto 365 GeV, while remaining within strict technical and cost constraints[22]. To reach the highest energy needed for top–anti-top ($t\bar{t}$) production at 365 GeV, it was planned to builtin the collider in a circular tunnel which was about 100 km, which provides a practical compromise between performance and cost. The FCC-ee adopts a double-ring collider configuration, similar to earlier electron–positron machines such as KEKB and PEP-II, where electrons and positrons circulate in separate rings; this allows a large number of bunches to be stored simultaneously and therefore increases the collision rate as shown in Figure 2.1. At four points of interaction the two beams collide, where detectors are located, with a 30 mrad small horizontal crossing angle. Since required beam current changes with collision energy, the machine is operated flexibly by mainly adjusting the number of bunches, enabling efficient running across all planned energy stages.

In conventional electron storage rings, the properties of the circulating beam are mainly determined by synchrotron radiation emitted when particles are bent by magnets in the arcs of the accelerator. In the FCC-ee, however, an additional effect becomes important due to the very high beam intensity. This is know as bremsstrahlung effect and is a form of emitted radiation from collisions when electromagnetic field strongly affect particles in one bunch of opposing bunch. Because of this, the beam properties in FCC-ee are influenced not only by the accelerator magnets but also by the collision process itself.

To keep the beam current and luminosity at their maximum values during data taking, the FCC-ee will use a top-up injection scheme, in which new particles are continuously injected into the collider. Without this method, the total collected luminosity would be more than ten times smaller. For this reason, a booster synchrotron installed inside the collider tunnel is an essential part of the machine. Particles with 20 GeV energy enter

the booster following a scheme similar to that used at LEP. The injector chain consists of a 6 GeV linear accelerator, a pre-booster that accelerates electrons and positrons from 6 up to 20 GeV, and a positron source where target is struck by electrons to produce positrons, and a positron damping ring to improve beam quality. The linac delivers one to two bunches per pulse with repetition rates of 100 or up to 200 Hz. Operation at Z-boson energy places the highest demands on the injector system, requiring a very large number of bunches with high intensity, corresponding to about 2×10^{10} particles per bunch for both electrons and positrons. Alternative injector designs, such as a longer linac reaching 20 GeV directly or a recirculating superconducting linac, are also being considered.

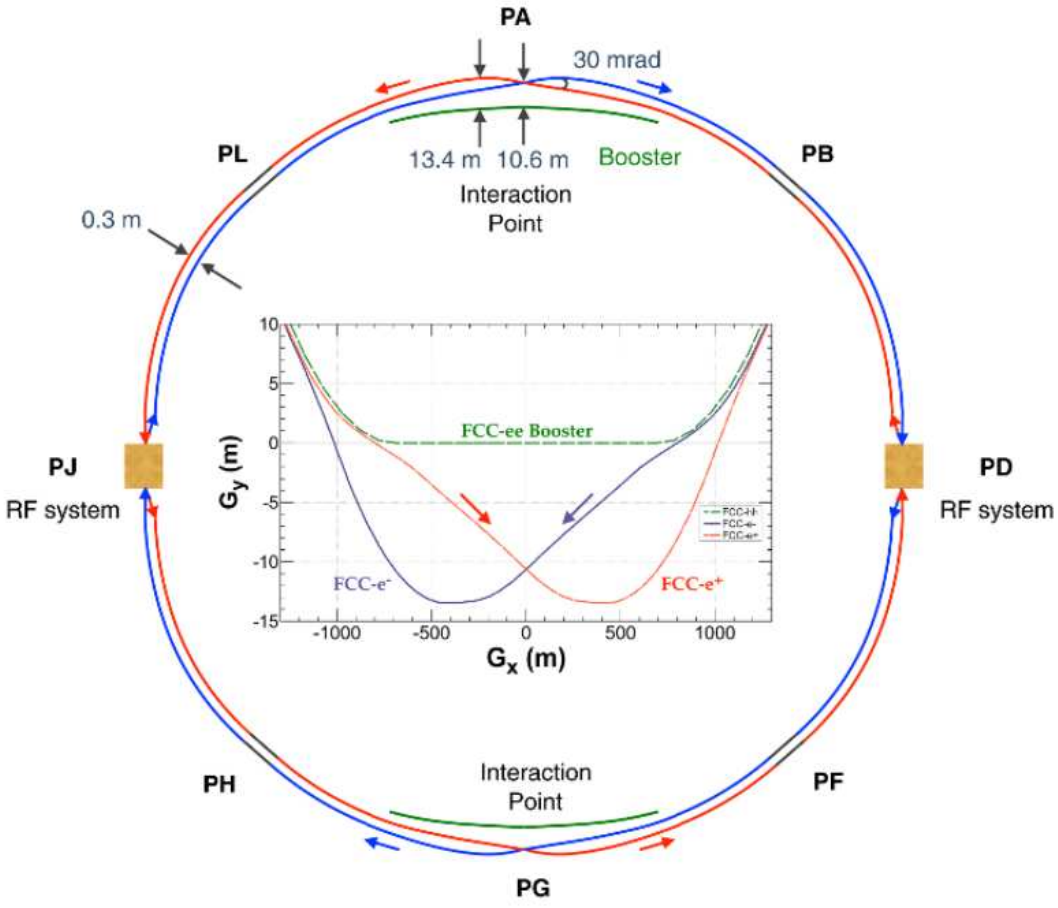


Figure 2.1: The overall layout of the FCC-ee, with a detailed view of particle trajectories near interaction point G, is illustrated. The FCC-ee rings are strategically positioned, 1 meter outside the footprint of the FCC-hh collider, which is denoted in green within the figure. Within the arc section, the e+ and e rings maintain a horizontal separation of 30 cm. The primary booster ring closely follows the trajectory of the FCC-hh collider ring. Notably, the interaction points are displaced by 10.6 meters outward from the FCC-hh layout. Additionally, to mitigate synchrotron radiation effects at the interaction point, the incoming beam trajectories are adjusted to be more linear compared to the outgoing trajectories, as shown on the (Gx , Gy) plot in the middle of the figure.

2.1.2 Performance

The worldwide recent interest in electron–positron (e^+e^-) colliders, especially after LHC discovered higgs boson, has led several proposals aimed for very highly precise study of Higgs boson with other Standard Model particles . Future Circular Collider, and in particular its electron–positron phase FCC-ee, is one of these proposals, but it is not the only one currently under consideration.

At present, four major e^+e^- collider designs are being studied worldwide. Two of them are circular colliders: the FCC-ee, a 100km new tunnel was built in CERN , with Circular Electron Positron Collider (CEPC), planned to be situated in China with a similar tunnel size and operating energies between 90 and 250 GeV[23]. In addition, two linear collider projects are under development: the International Linear Collider (ILC), designed to operate with 250 GeV[24] center of mass energy , and the Compact Linear Collider (CLIC), whose lowest planned energy stage has been revised to 380 GeV[25]. Together, these projects reflect the strong international effort to achieve precision studies of Higgs boson with other fundamental particles.

Expected performance of the main electron–positron collider projects, namely ILC, CLIC, CEPC, and FCC-ee, can be compared in terms of the luminosity they can achieve at different collision energies as shown in Figure 2.2. Among these options, FCC-ee clearly provides the highest data rates, while operating in a very clean and well-controlled experimental environment. It covers all the most important energy points for precision physics: the Z boson energy (91 GeV),The (161GeV) threshold of the w pair production, the Higgs factory energy (240 GeV), and the top-quark pair production region (340–365 GeV), with collisions taking place at four interaction points. The particular advantage of FCC-ee is its ability to provide transversely polarized beams at energies above 80 GeV, which determines the collision energy very accurately, about 100 keV of precision is reached at the W and Z energies. This property , specific to circular colliders, makes FCC-ee particularly powerful to precisely measures properties of Standard Model particle and for searching for rare processes or small violations of well-established symmetries

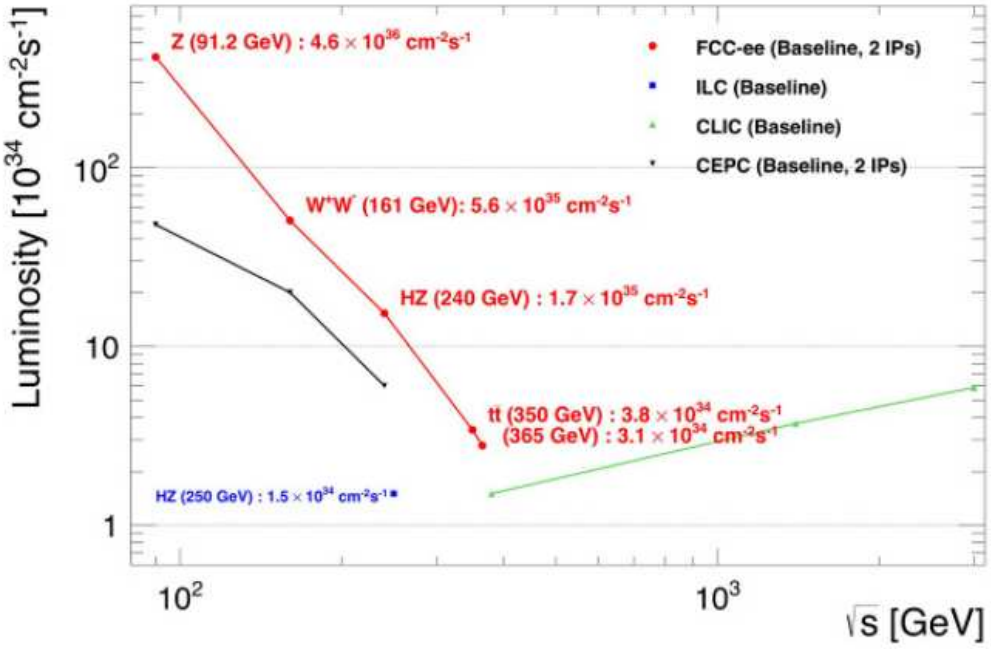


Figure 2.2: Baseline luminosities, defined as the sum of the luminosities in all points of interaction, shown as of the center-of-mass energy function $\sqrt{s}$ mainly proposed ee collider projects. The comparison includes ILC (blue squares), CLIC (green upward triangles), CEPC (black downward triangles), and FCC-ee (red dots). All luminosity values safety margin of 10% was included. The FCC-ee luminosities are taken , updated parameters of CEPC are based on Ref. [26], and the luminosity estimates for the linear collider projects are obtained from Refs.[27], [28]].

The accelerator must operate under very different conditions depending on the chosen energy. On Z pole, FCC-ee behaves like high-current storage ring, running with large beam current but requires relatively small radiofrequency (RF) which is voltage of almost 0.1 GV (see Fig 2.3). In contrast, at the top-quark energy, with very low beam current, similar to that of the former LEP2 collider, while more than 10 GeV of very high RF voltage is needed. all operating modes, a continuous RF power of around 100 MW is required to sustain the circulating beams. To meet these diverse requirements, different types of RF cavities are foreseen: 400 MHz cavities for high-intensity operation at lower energies, more powerful 400 MHz multi-cell cavities for higher-energy running, and additional 800 MHz cavities for top-quark operation.

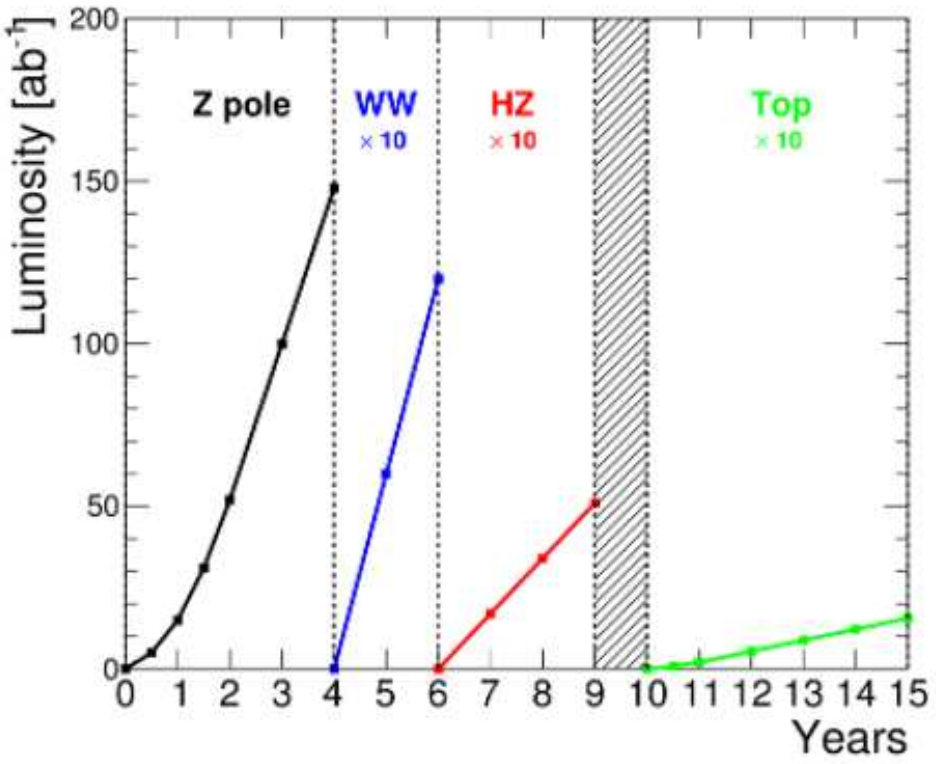


Figure 2.3: FCC-ee based conceptual design of five years was planned to study the evolution of cumulative integrated luminosity as time function. The various running phases include operation at the Z pole (black), the WW threshold (blue), the Higgs factory energy (red), and the top-pair production threshold (green). Hatched regions indicate shutdown periods required for machine preparation and upgrades necessary to reach the highest center-of-mass energies.

The installation strategy follows an approach similar to that used at LEP, and high overall energy efficiency is achieved by combining efficient RF systems, optimized magnet designs, and continuous top-up injection as shown in figure 2.4.

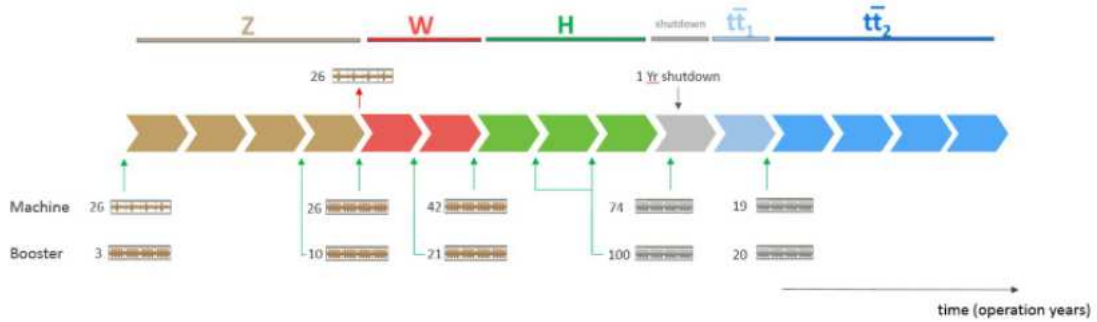


Figure 2.4: Operational timeline of the FCC-ee, illustrating the staged installation strategy of accelerator components. During successive winter shutdown number of installed cryomodules in the rings of booster and collider by the lower panel. Additional details on the installation plan are provided in Ref. [29].

2.1.3 Detector considerations

Circular colliders such as FCC-ee provide collisions at many points of interaction, that allows several detector concepts to be developed and optimized in parallel. The detectors designed for FCC-ee must be able to fully exploit the rich physics programm with very high statistical precision delivered by the machine. This requires excellent performance in key areas such as tracking, particle identification, heavy-flavour tagging, the reconstruction of particle flow, accurate lepton measurements, jets, missing energy, momentum and angular distributions. The detector performance must therefore be closely matched to the physics goals, that the new physics Standard Model processes and potential signals measurement achievements are ensured

In the same time, detector designs must respect important machine-related constraints. These include minimizing backgrounds produced by the beams, coping with very high rates of interaction (up to 100 kHz at the Z pole), and ensuring that data sizes and readout speeds remain manageable. Additional constraints arise from the interaction region layout, such as limiting the detector solenoid magnetic field to 2 T to prevent reducing luminosity. Precise knowledge of the center-of-mass energy spread, that can reach 90 MeV at the Z pole and the highest energies at 500 MeV, needs excellent angular resolution, especially for muons. Furthermore, the luminosity detector should be put very near to the interaction point, at about 1 m, while still achieving a luminosity measurement precision better than 10^{-4}. Within this framework, two detectors for general purpose concepts have been studied in detail: CLD, based on a silicon tracker and highly granular calorimetry, and IDEA, an alternative design using a short-drift wire chamber and dual-readout calorimetry with lightweight solenoid. These examples demonstrate that detectors capable of meeting the demanding FCC-ee requirements are feasible, while further concepts may still be explored to best match the full physics programme.

2.2 The IDEA layout

The Innovative Detector for Electron–positron Accelerators (IDEA) has been proposed for the FCC-ee and CEPC projects [30][31]. The IDEA detector is designed to study electron–positron annihilations over a vast area of center-of-mass energies, providing performances required to fully exploit the precision physics potential of future e^+e^- colliders.

The conceptual design of the IDEA experiment is based on the use of innovative detector technologies developed in recent years and optimized for precision measurements at electron–positron colliders. The central tracking system consists of a Silicon Inner Tracker located close to the interaction point, followed by a very light Drift Chamber serving as the main tracking detector, and an external Silicon wrapper providing additional high-precision tracking measurements.

The inner detector is operated inside a uniform magnetic field of 2 T, generated by a thin superconducting solenoid, which enables precise momentum determination of charged particles while minimizing material effects. Out of the solenoid, a dual-readout calorimeter is installed, positioned inside the magnet return yoke, to provide accurate energy measurements.

The return yoke is instrumented with a muon tracking system based on μ-RWELL technology, which is also employed to implement a preshower detector located in front of the calorimeter. IDEA detector layout is shown in a schematic way in Figure 2.5, while the summary of parameters of main detectors are showed in Table 2.1.

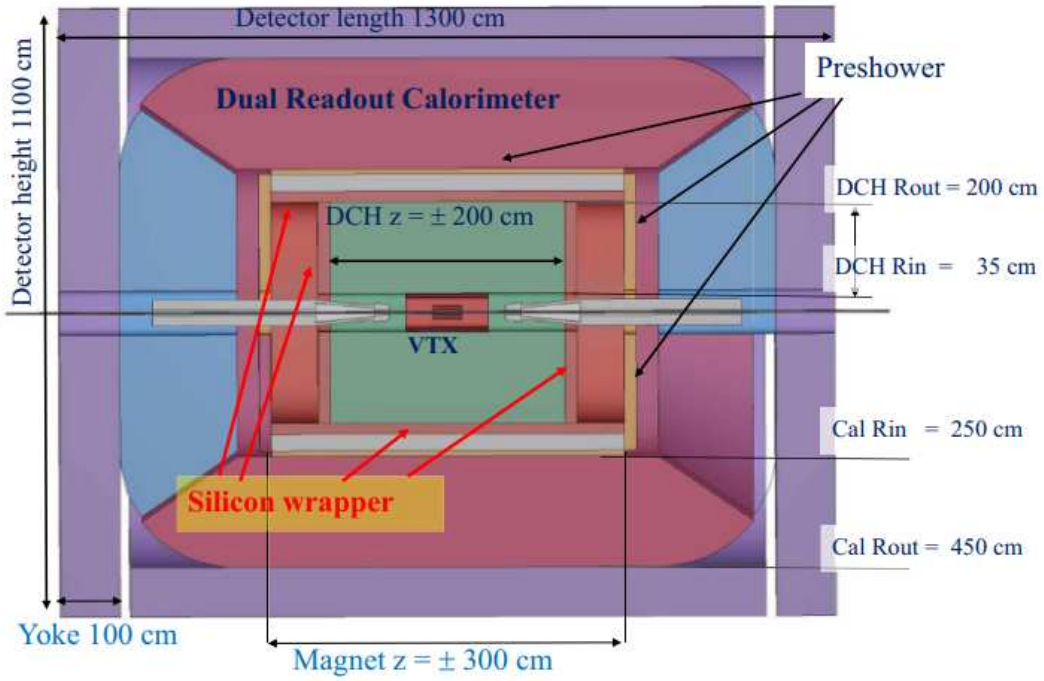


Figure 2.5: Schematic layout of the IDEA detector [32].

Table 2.1: Main parameters of the IDEA detector.

Parameter	**Value**
Vertex technology	Silicon
Vertex inner / outer radius	1.7 cm / 34 cm
Tracker technology	Drift chamber + silicon wrapper
Tracker half length / outer radius	2.0 m / 2.0 m
Solenoid bore radius / half length	2.1 m / 3.0 m
Preshower / calorimeter absorber	Lead / lead
Preshower inner / outer radius	2.4 m / 2.5 m
Dual-readout calorimeter inner / outer radius	2.5 m / 4.5 m
Overall height / length	11 m / 13 m

2.2.1 The Vertex Detector

In the IDEA tracking system the vertex detector is the innermost component and surrounds the beam pipe with a radius of approximately 1.5 cm. It is based on a silicon pixel detector employing monolithic active pixel sensor technology and is designed for giving precise measurements of impact parameter of charged particle tracks.

Significant effort is devoted to achieving an excellent spatial resolution at the level of a few micrometers, while a very small material budget of about 0.15–0.30 % of radiation length per layer is maintained and limiting the power dissipation to below 20 mW/cm^2 These requirements are essential in order to minimize multiple scattering effects and to ensure optimal vertex reconstruction performance.

2.2.2 The Preshower Detector

Micro-Resistive WELL (μ-RWELL) technology is the basis of preshower detector in the IDEA experiment , a compact Micro-Pattern Gaseous Detector featuring a mono amplification stage with intrinsic spark protection. This technology offers stable operation and is well suited for large-area detector applications [33].

In area of the barrel, the superconducting magnet coil is used for absorption corresponding to approximately 0.7 radiation lengths. A layer of μ-RWELL chambers is placed immediately downstream of the coil, allowing the detection of early electromagnetic shower development. In the forward region, a lead absorber with a thickness of 1 radiation length is installed in front of a layer of μ-RWELL chambers, which are positioned directly upstream of the endcap calorimeter.

This detector configuration enables the identification of neutral pion decays by detecting both photons from $\pi^0 \to \gamma\gamma$ processes, resulting in an estimated tagging efficiency of about 30%. Owing to the high mechanical stability of the μ-RWELL chambers, the preshower system provides a reliable determination of photon acceptance. Furthermore, the combined information from the μ-RWELL chambers and the silicon detectors allows for a precise determination of the acceptance for charged particles.

2.2.3 The Magnet System

The IDEA detector is equipped with a solenoidal magnet the entire tracking system is encircled. The magnet has a length of approximately 5 m and an inner diameter of about 4.2 m, providing sufficient volume to accommodate the inner detectors. It generates a uniform magnetic field of 2 T, which is adequate for precise momentum measurements of charged particles.

Owing to strength of moderate magnetic field and compact detector dimensions, overall thickness of the magnet package can be limited to about 30 cm. With these parameters, an iron return yoke with smaller than 100cm of width is enough to fully comprise the magnetic flux, while also ensuring effective magnetic shielding and providing the necessary mechanical support for the muon chambers.

2.2.4 Dual-Readout Calorimeter

Copper-based dual-readout fiber calorimeter encircles the preshower detector in the IDEA experiment. The total depth of calorimeter is approximately 2 m, which is equal to almost 8 interaction lengths. Based on GEANT4 simulations, the expected energy resolution is around $11\%/\sqrt{E}$ for electrons and about $33\%/\sqrt{E}$ for pions that are isolated, with constant terms that are very minute .

Technique of dual readout gives excellent intrinsic variation among muons, photons and electrons , and hadrons for particles that are isolated. besides its abilities for the detection of particles , the fine transverse granularity of the calorimeter enables close showers to each other to be separated efficiently and enables accurate matching with tracks reconstructed in the inner detector, signals from the preshower system, and muon tracks. This makes the dual-readout calorimeter a suitable choice for efficient particle-flow reconstruction.

A longitudinal segmentation of the calorimeter is required in order to disentangle signals originating from overlapping electromagnetic and hadronic showers. Different segmentation solutions are currently under investigation through detailed simulations and dedicated beam tests[34].

2.2.5 The Muon Detector System

(μ-RWELL) technology is the basis muon detector system of the IDEA experiment, provides high tracking efficiency together with precise spatial resolution on the muon track coordinates, of the order of 200–300 μm, and good time resolution. These characteristics make μ-RWELL well suited for efficient muon identification and tracking outside the calorimeter volume.

In addition to its performance, the use of μ-RWELL technology allows a significant reduction in cost, which is a crucial aspect for instrumenting the extremely large surfaces required by the muon system. The proposed layout foresees the use of μ-RWELL detector tiles with dimensions of 50×50 cm^2, assembled into three muon detector stations. every station is equipped with a layer of μ-RWELL detectors featuring bi-dimensional readout, enabling accurate position measurements of traversing muons[35].

2.3 The Drift Chamber of the IDEA Detector

The IDEA Central Drift Chamber (DCH) is the main tracking detector of the IDEA experiment. Its design builds on the experience gained from the KLOE and MEG–II drift chambers. It is a single-volume, low-mass cylindrical detector with high granularity and a fully stereo wire configuration. The chamber is placed coaxially inside a 2 T solenoidal magnetic field and operates with a light gas mixture of He 90% and iC_4H_{10} 10%. This design is intended to provide precise tracking and particle identification while keeping the material budget low.

The active volume extends from an inner radius of $R_{\text{in}} = 0.35$ m to an outer radius of $R_{\text{out}} = 2.0$ m, with a total length of $L = 4.0$ m. As shown in Fig. 2.6, the chamber contains 112 coaxial layers, grouped into 14 superlayers of 8 layers each, and is segmented into 24 identical azimuthal sectors. The wires are arranged with alternating positive and negative stereo angles, ranging from 43 mrad to 223 mrad. This stereo geometry gives the chamber true three-dimensional tracking capability.

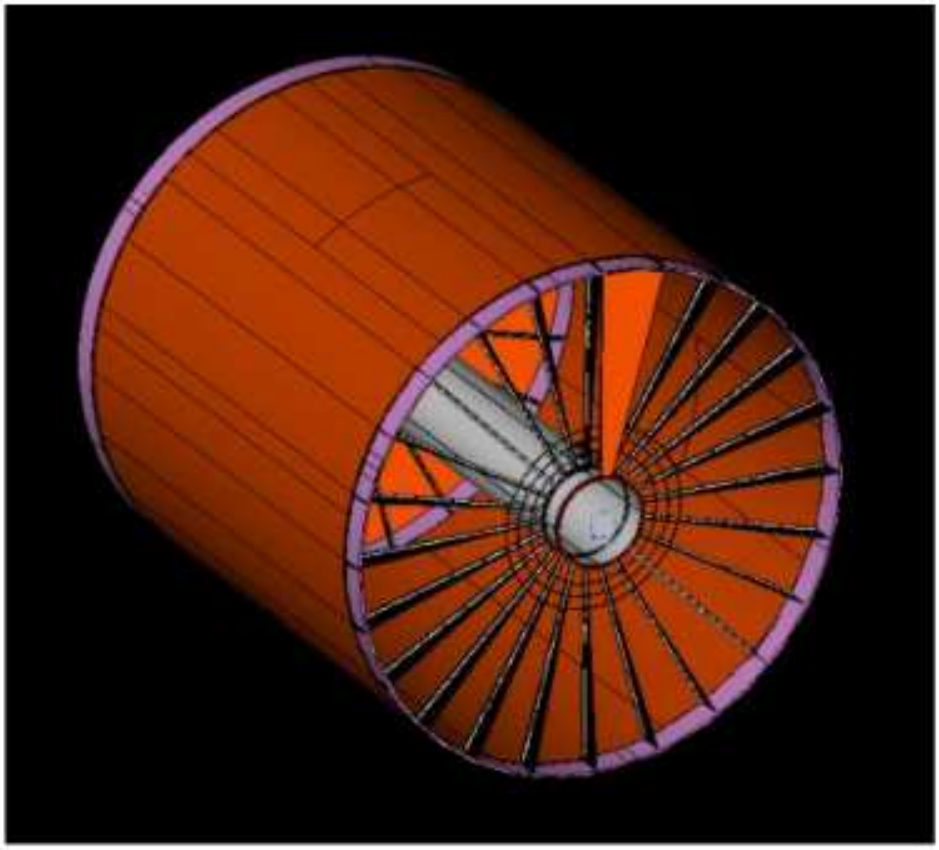


Figure 2.6: Schematic layout of the IDEA drift chamber (DCH), showing its cylindrical geometry and the arrangement of the tracking volume.

An important feature of the DCH is its stereo wire geometry. As illustrated in Fig. 2.7, the wires are arranged in alternating positive and negative stereo layers. Neighboring layers are combined in a "zipping" pattern, and the chamber is organized into sectors and

superlayers. In this configuration, a sector on one endcap is connected to the matching sector on the opposite endcap, which gives the wires a hyperbolic profile along the chamber length. This arrangement is essential for reconstructing the longitudinal position of the tracks and therefore for full three-dimensional tracking [36].

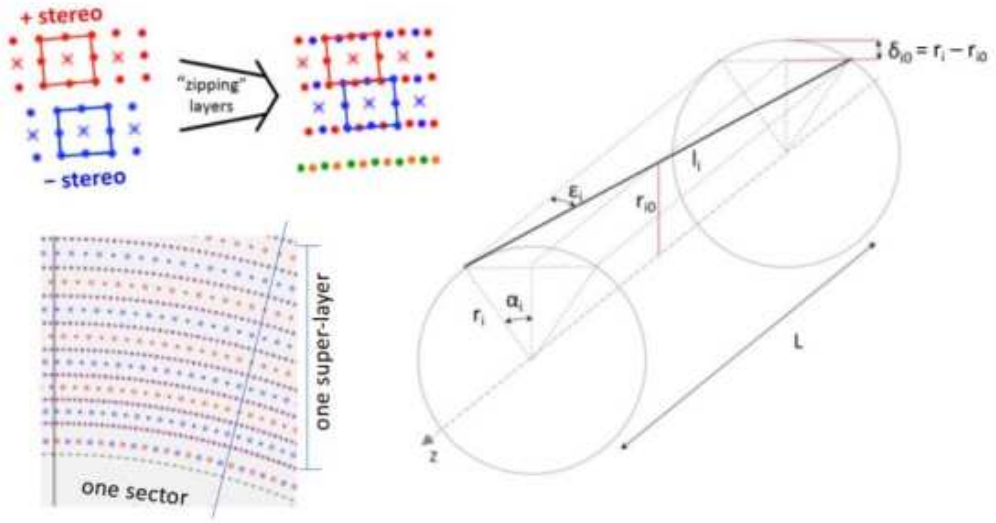


Figure 2.7: Stereo wire configuration of the IDEA drift chamber. The figure illustrates alternating positive and negative stereo layers, their arrangement within a sector and superlayer, and the resulting hyperbolic wire profile along the chamber length.

The drift cells are designed to be approximately square. Their size varies from about 11.8 mm in the innermost layers to about 14.9 mm in the outermost layers. The number of cells per layer increases from 192 in the inner region to 816 in the outer region, corresponding to 16 cells per sector. In total, the chamber contains 56,448 sense cells. The design targets a drift length of about 1 cm and a drift time of about 150 ns, with an expected transverse resolution of $\sigma_{xy} < 100$ μm and a longitudinal resolution of $\sigma_z < 1$ mm.

Because the chamber is 4 m long, the cell dimensions must satisfy both electrostatic and mechanical constraints. The electrostatic stability condition is written as

$$T_c \geq \frac{C^2 V_0^2}{4\pi\epsilon w^2} L^2,$$

where T_c is the wire tension, C is the capacitance per unit length, V_0 is the anode–cathode voltage, ϵ is the permittivity of the medium, w is the cell width, and L is the wire length. In addition, the wire tension must remain below the elastic limit,

$$T_c < \mathrm{YTS}\, \pi r_w^2,$$

where YTS is the yield tensile strength of the wire material and r_w is the wire radius. These conditions show that the long chamber length leaves only a small design margin. For this reason, the field-to-sense wire ratio is chosen to be 5:1, and different wire types

are used in the final design.

Each drift cell contains one anode wire and two cathode sub-layers, as shown in Fig. 2.8. The anode wires are made of 20 μm tungsten, while the cathode wires are made of aluminum alloy with diameters of 40 μm and 50 μm. The two alternating-sign stereo layers are often referred to as the U and V views, and they improve the measurement of the longitudinal coordinate.

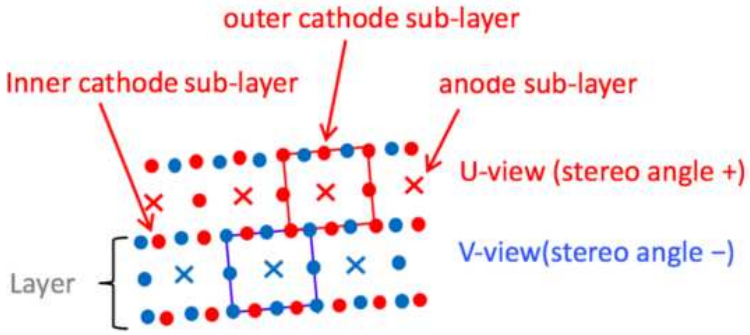


Figure 2.8: Structure of the drift cells with two alternating-sign stereo layers [36].

Overall, the IDEA Central Drift Chamber contains 56,448 sense wires together with more than 2.8×10^5 field and guard wires, for a total of about 3.4×10^5 wires. The guard wires are used to equalize the gas gain between the innermost and outermost layers. Because of the very large number of wires, the chamber requires a dedicated wiring procedure. Following the approach successfully adopted for the MEG–II drift chamber, a feed-through-less wiring concept is employed in the IDEA design [36].

2.3.1 Tracking performance and fast simulations studies

A Geant4-based simulation has been done in order to determine the tracking ability of the IDEA detector. The study assumes an average single-cell spatial resolution of 100 μm for the drift chamber, consistent with the expected chamber performance, and a conservative spatial resolution for the silicon detectors given by the pitch divided by $\sqrt{12}$. Under these assumptions, the IDEA tracking system achieves the expected excellent performance, as shown in the left side of Figure 2.9.

Very low material budget of the drift chamber enables the tracking system to significantly reduce the effects of multiple scattering. As a consequence, the momentum resolution of the IDEA tracker improves by almost a factor of three with respect to a full silicon tracking system for particle momenta up to approximately 50 GeV/c, as illustrated in the right side of Figure 2.9 [37].

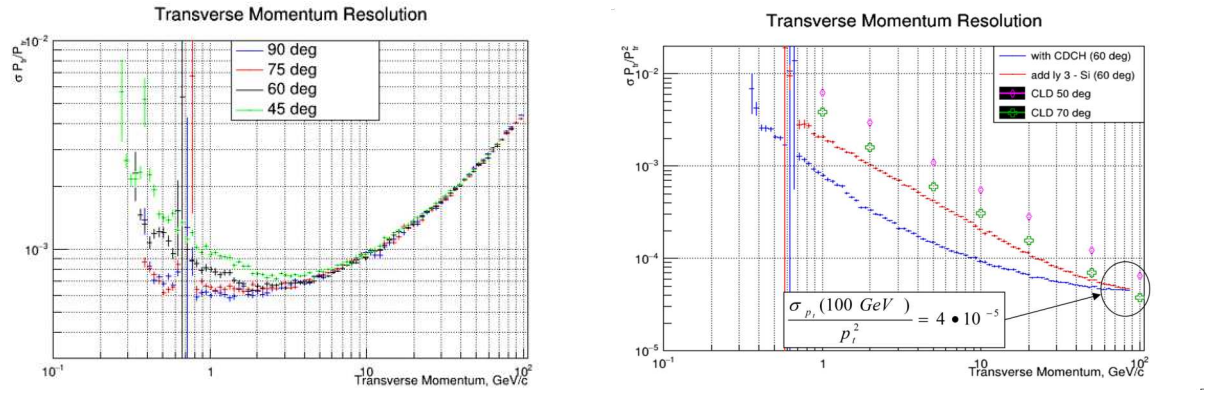


Figure 2.9: Transverse momentum resolution of the IDEA tracking system. Left: transverse-momentum resolution evaluated at different polar angles. Right: comparison of the transverse-momentum resolution at a fixed polar angle between the IDEA tracking system and a full silicon-based tracking system similar to that of the CLD detector.

In addition, the IDEA Central Drift Chamber exploits the cluster timing technique to further enhance tracking performance. This approach makes use of the information provided by the ionization clusters produced along the particle trajectory. The positions estimated from the different clusters are combined through a weighted average, leading to an improved determination of the impact parameter. The resulting improvement obtained with cluster timing and cluster counting techniques, compared to other tracking methods, that we have showed you in Figure 2.10

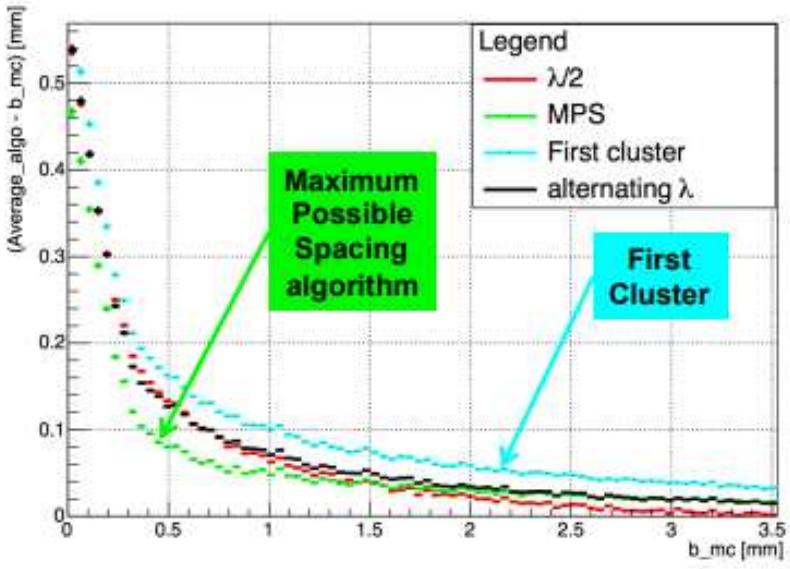


Figure 2.10: Cluster timing technique effect on the spatial resolution of a drift chamber cell. In an 8 mm cell, the average spatial resolution improves to 85 μm which was 100 μm when cluster timing is applied. For a given first-cluster drift time t_1, the technique exploits the drift time distribution of all subsequent clusters to determine, on a hit-by-hit basis, the most probable impact parameter. This approach reduces the bias associated with the first-cluster method and leads to an improved average spatial resolution[38].

2.3.2 Particle Identification with the Cluster Counting Technique

The use of the cluster counting technique, as an alternative to the traditional dE/dx method, allows a significant improvement in particle separation capabilities, by approximately a factor of two [39]. This improvement arises from the different statistical nature of the two approaches.

In the case of dE/dx measurements, the relative resolution can be described by the Walenta parameterization:

$$\frac{\sigma_{dE/dx}}{dE/dx} = 0.41\, n^{-0.43}\ (L_{\text{track}}\,[\text{m}])\ P\,[\text{atm}]^{-0.32}, \tag{2.1}$$

where n is the number of samples, L_{track} is the track length, and P is the gas pressure.

In contrast, when using cluster counting, the resolution is governed by Poisson statistics and can be expressed as:

$$\frac{\sigma_{dN_{\text{cl}}/dx}}{dN_{\text{cl}}/dx} = (\delta_{\text{cl}}\, L_{\text{track}})^{-1/2}\,, \tag{2.2}$$

where δ_{cl} represents the primary cluster density.

The cluster counting technique therefore takes advantage of the Poissonian nature of primary ionization, leading to reduced fluctuations with respect to the traditional dE/dx approach. As a result, it provides a more statistically robust method for particle identification and mass inference. A more detailed discussion of this technique is presented in Chapter 6.

Chapter 3

Simulation of the IDEA Drift Tubes

This chapter presents the full simulation framework developed for the IDEA drift tubes at the Future Circular Collider (FCC) using Garfield^{++}. Since cluster counting is based on the ionization signals produced in a gaseous detector, a realistic simulation of the drift tube is required. For this purpose, the chapter first introduces the main Garfield^{++} classes used to define the detector geometry, gas properties, and electric-field configuration. It then describes how primary ionization clusters are generated using Heed, how analogue waveforms are produced from the incident particles in the drift tube, and how these analogue signals are converted into realistic digitized waveforms by including a data-driven electronics-noise model.

My main contribution in this chapter is the development of the simulation and digitization procedure used to produce realistic waveforms for the later machine-learning studies. In particular, I generate the number of primary ionization clusters and construct the analogue waveforms by combining the drift time, diffusion time, gas amplification, and single-electron pulse response. In the digitization stage, I extract the noise from experimental measurements using the Fast Fourier Transform (FFT) and add it to the analogue waveforms in order to obtain realistic digitized waveforms. The simulation is performed using detector and parameters matched to the 2022 and 2023 real test-beam data, including the cell size, particle momentum, angle between the muon track and the sense wire, and gas mixture etc. These realistic digitized waveforms are then used as input to the deep-learning models described in Chapter 6, where the number of primary clusters is reconstructed and compared with the Monte Carlo truth.

3.1 Introduction to Garfield++

Garfield++ is an object-oriented toolkit that helps us simulate particle detectors that work by measuring **ionisation signals** in **gases** or **semiconductors**. In simple words, it lets us model what happens inside a detector when a charged particle passes through it and creates electrons and signals.

To simulate a detector in a clear and realistic way, Garfield++ follows the same chain of steps we expect in the real detector. First, we describe the detector material (for example the gas mixture), and we describe the detector shape. Then we calculate the electric field, because this field is what guides the electrons. After that, we simulate how ionisation is produced, and finally we simulate how the charges move and how a signal is created on the readout.

Garfield++ provides several practical options for each step. The most important options used in detector simulations are listed below.

- **Material properties (gas or semiconductor):**
  - For gases, Garfield++ can connect to Magboltz to obtain electron transport properties (for example drift velocity, drift time and diffusion).
  - It also supports semiconductor materials in the same general framework.
- **Electric-field calculation (needed before drifting electrons):**
  - Fast analytic solutions in the **thin-wire limit** for detectors made of wires and planes.
  - Interfaces to **finite-element** tools, which can compute approximate fields for almost any 2D or 3D geometry, including conductors and insulating materials.
  - An interface to the Synopsys Sentaurus device simulation program.
  - An interface to the neBEM field solver.
- **Primary ionisation (how electrons are created):**
  - Heed can simulate the ionisation pattern from relativistic charged particles.
  - For low-energy ions, results from SRIM can be imported.
  - Degrade can simulate ionisation produced by electrons.
- **Visual checks (to understand and verify the setup):**
  - Garfield++ can plot drift lines, draw contour maps of the electric potential, and show the detector layout.
  - These visual tools rely strongly on the ROOT framework.

Figure 3.1 shows the main idea of the class structure such as Medium, Track, Component, Sensor etc will be discussed briefly in the next subsection of "Main Classes Used in HEP Garefield++"

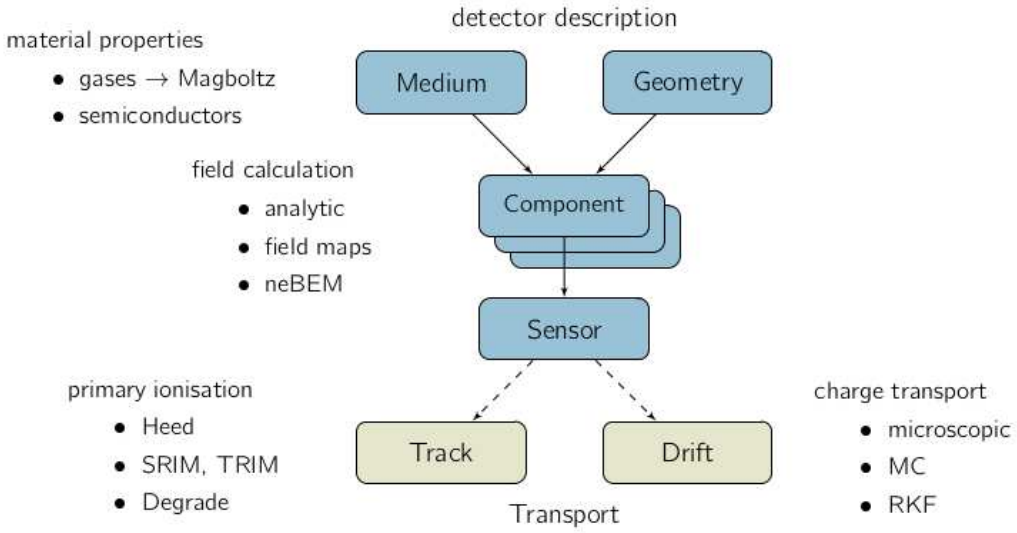


Figure 3.1: Overview of the main Garfield++ class structure and how the parts connect to each other.

3.2 Main Classes of Garfield++ in HEP

In Garfield++, a gaseous-detector simulation is organised as a chain from the particle to the readout signal. First, the primary ionisation (clusters of electron–ion pairs created along a charged-particle track in the gas) is generated with `TrackHeed` (Heed)[40]. Second, the transport properties of the electrons and ions in the chosen gas mixture—such as drift velocity and diffusion—are provided by the gas-medium description `MediumMagboltz` (Magboltz). Third, the detector *electric field* (and, when needed, the weighting field/potential for signal formation) is defined by a `Component` object, for example an analytic field description (`ComponentAnalyticField`). Finally, the induced current/charge on the readout electrode is collected by the `Sensor` class using the Shockley–Ramo concept[41][42]. This ordering matches the physical picture of a drift tube: ionisation in the gas → charge transport in the electric field → induced signal on the wire/electrode. Now let us briefly discuss each classes one by one.

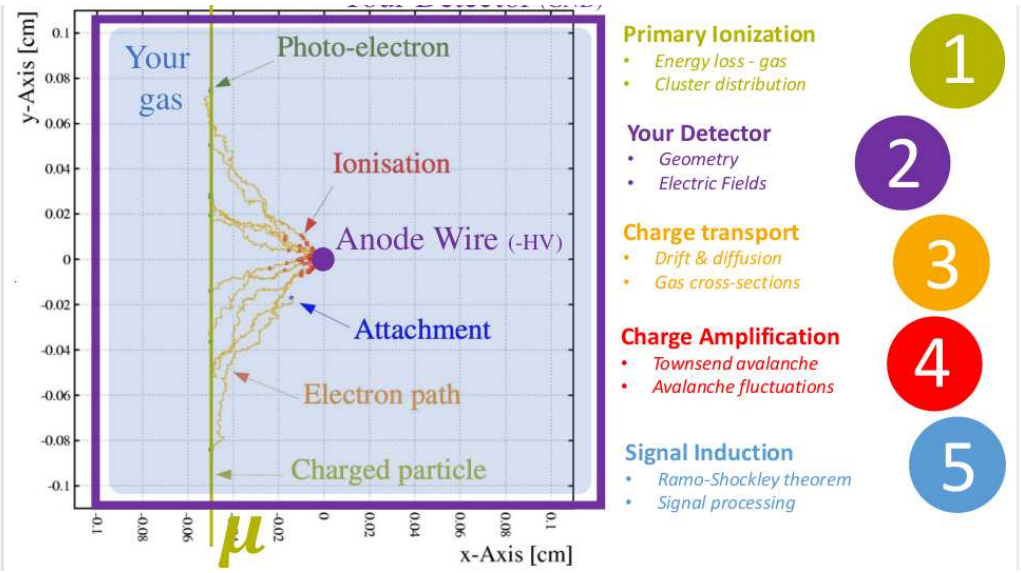


Figure 3.2: All steps from a charged particle to the induced signal in a gaseous detector.

3.2.1 Track: from a charged particle to primary ionisation in a drift tube

Usually in a detector, the measured signal begins when a charged particle crosses the gas and creates small, discrete ionisation acts along its path. The `Track` classes in Garfield++ are designed to model exactly this first physical step: they convert the passage of a particle into a sequence of ionisation "clusters" in space and time, which later become the input for electron drift, diffusion, amplification, and signal formation in the drift tube.

Practically, a track is defined by (i) the particle species (e.g. $\mu^{\pm}$, $\pi^{\pm}$, $K^{\pm}$), (ii) its kinematics (how energetic/fast it is), and (iii) its initial position, time, and direction inside the gas volume. After this setup, the simulation returns a list of ionisation clusters along the trajectory. In this context, a *cluster* corresponds to one ionising interaction of the primary particle in the gas: it represents a local energy transfer and the creation of secondary electrons. For drift-tube studies, these cluster positions are important because they set the initial distances to the anode wire and therefore control the drift/diffusion-time pattern that you finally measure.

Table 3.1: Available charged particles that can be used as the primary particle in the Track simulation.

Symbol	**Name(s)**	**Mass [MeV/c^2]**	**Charge**
e^-	electron, e^-	0.510998910	-1
e^+	positron, e^+	0.510998910	$+1$
μ^-	muon, μ^-	105.658367	-1
μ^+	muon, μ^+	105.658367	$+1$
π^-	pion, π, π^-	139.57018	-1
π^+	π^+	139.57018	$+1$
K^-	kaon, K, K^-	493.677	-1
K^+	K^+	493.677	$+1$
p	proton, p	938.272013	$+1$
$\bar{p}$	anti-proton, antiproton, p-bar	938.272013	-1
d	deuteron, d	1875.612793	$+1$

HEED Package (TrackHeed): a realistic microscopic ionisation model

For many HEP gas-detector applications, we want the ionisation pattern to be realistic at the microscopic level rather than a simplified average. The program `Heed` [40] implements the photo-absorption ionisation (PAI) approach and is accessed in Garfield++ through the class `TrackHeed`. In simple words, the ionization pattern is generated by the package "HEED" from the incoming particle into gaseous detector. The number of primary clusters can be described by a Poisson distribution:

$$P(\bar{N}_p, k) = \frac{\bar{N}_p^{\,k}}{k!} e^{-\bar{N}_p} \tag{3.1}$$

where $P(\bar{N}_p, k)$ is the probability of observing exactly k primary clusters in one event, $\bar{N}_p$ is the mean number of primary clusters, and k is the actual number of primary clusters produced in that event. In the term $\bar{N}_p^{\,k}$, the mean number of clusters $\bar{N}_p$ is raised to the power k. The quantity $k!$ denotes the factorial of k, and e is the base of the natural logarithm. Therefore, this expression gives the probability of obtaining k primary clusters when the average expected number is $\bar{N}_p$.

The output of `TrackHeed` is again a list of clusters along the track, and each cluster carries the key information needed for a drift tube: where the ionisation happened, when it happened, and how many electrons were produced. These electrons then become the starting points for the next simulation stage (drift and diffusion toward the anode wire, followed by amplification and signal induction).

Two additional physical options are commonly relevant: (i) **delta-electron transport**,

where energetic secondary electrons can create extra ionisation away from the main trajectory (this can broaden the effective ionisation region in a drift tube), and (ii) **photon transport**, where Heed can simulate X-ray photoabsorption and the resulting primary electrons, which is useful for calibration or photon-background studies. If a magnetic field is present, `TrackHeed` can also account for the curved charged-particle trajectory, which matters because the track shape changes the cluster positions and therefore the drift distances in the tube.

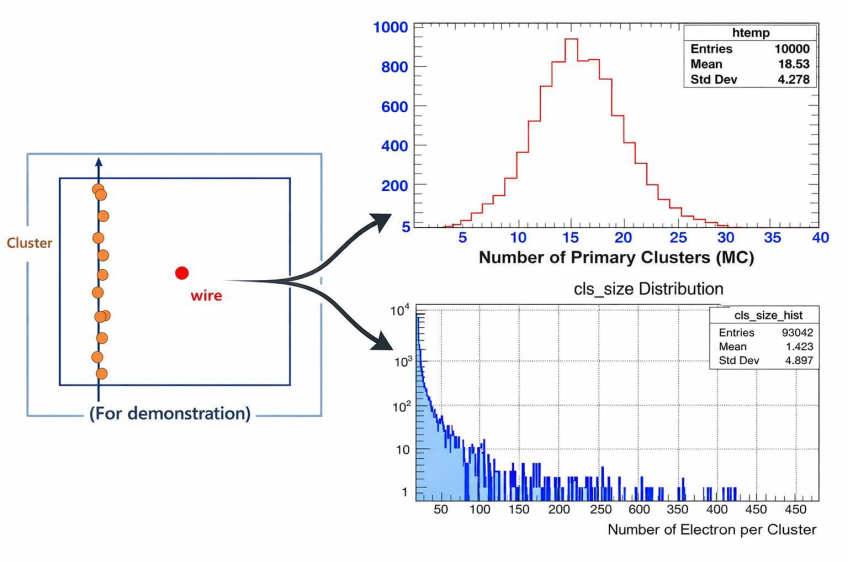


Figure 3.3: Illustration of primary ionisation clusters produced along a charged-particle track in a drift-tube gas as well as the distrubution of the number of primary clusters and cluster size. Each cluster is a local ionising interaction that creates electrons, which later drift toward the anode wire and contribute to the measured signal.

3.2.2 Magboltz (MediumMagboltz): electron transport in gas mixtures

After ionisation is created by the primary particle (for example with HEED), the next question in a drift tube is simple: *how do the free electrons move through the gas toward the anode wire?* In Garfield++, this transport step is handled by Magboltz. It is a well-known program that uses a semi-classical Monte Carlo approach to compute electron transport properties in gas mixtures, based on large databases of electron–atom/molecule cross-sections [41].

In high-energy physics detectors, these transport properties directly control what we finally measure. For a drift tube, Magboltz determines the **drift velocity** (so it sets the drift time), the **diffusion** (so it sets how much the electron cloud spreads), and also **attachment** (loss of electrons) and **ionisation rates** (important for gain and avalanches). Therefore, Magboltz is the key link between "ionisation points in the gas" and the realistic arrival-time distribution of electrons at the wire.

Table 3.2: Classification of electron collision processes used in Magboltz transport modelling.

Collision type	Index
elastic collision	0
ionisation	1
attachment	2
inelastic collision	3
excitation	4
superelastic collision	5
virtual ("null") collision	6

Transport tables and why they are used

Magboltz results are typically stored as **gas tables**. A gas table is simply a precomputed map of transport parameters as a function of the electric field E (and, when relevant, magnetic field B and the relative angle between them). This is useful because Garfield++ can then look up the transport values quickly during tracking and avalanche simulation, instead of re-running the full Monte Carlo every time. In practice, these tables also store rates for excitation and ionisation processes, which can later be used when studying gas amplification and related effects.

Electron collisions: the physical meaning

Inside a drift tube, an electron does not move in a straight line. It repeatedly collides with gas atoms/molecules, and each collision changes its direction and energy. Magboltz organises these collisions into standard categories, which helps us understand what is happening physically in the gas. Table 3.2 summarises the main collision classes.

This classification is helpful for drift-tube studies because it separates effects that **keep** electrons (elastic, excitation) from effects that **remove** electrons (attachment) or **create more** electrons (ionisation), and each of these influences timing and gain.

Energy range and thermal motion

Transport parameters depend on the electron energy range that is included in the cross-section tables. For many gases, reasonable accuracy can be reached with a moderate number of collisions, while higher precision requires more collisions to be simulated [7]. Recent versions of Magboltz can also include the **thermal motion** of gas atoms/molecules, instead of assuming an idealised 0 K gas. This becomes important when one wants transport properties that are closer to real operating conditions (temperature and pressure).

Penning transfer and its impact on gain

In mixtures, excited atoms can transfer their energy to another component and cause extra ionisation. This effect is called **Penning transfer**. Physically, it increases the number of ionisation electrons produced during amplification, and it can effectively increase the Townsend coefficient (and therefore the gas gain). Magboltz can include Penning transfer using literature-based parameterisations when available [43] [44], and this is important when the detector performance depends strongly on the achievable gain and its fluctuations.

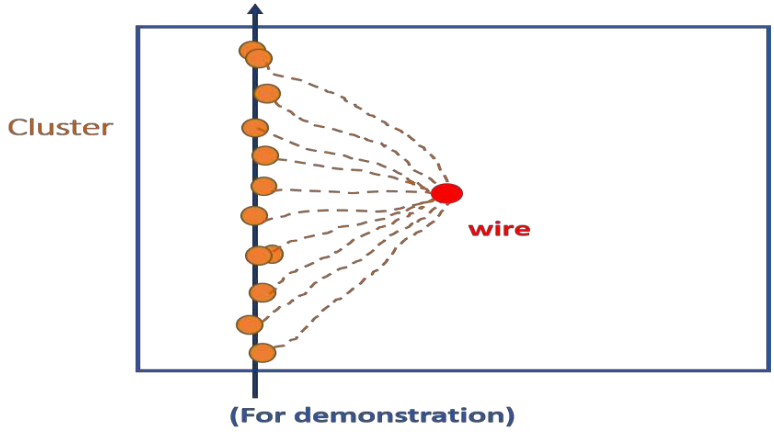


Figure 3.4: Schematic illustration of the drift-tube chain: ionisation clusters are created along the charged-particle path (Track/HEED), and the released electrons drift through the gas toward the anode wire (Magboltz can handle such an information).

3.2.3 Sensor: from drifting charges to a Signal

After ionisation (TrackHeed/Heed) and transport (Magboltz + drift/avalanche), Garfield++ still needs one last step: **turn the motion of charges into the signal that a readout electrode would see**. This job is done by the **Sensor** object.

The Sensor does not "create" electrons or drift them by itself. Instead, it **records** the induced signal on a chosen electrode in a clear and organised way. The physics rule behind this recording is the Shockley–Ramo theorem [36]. It states that when a charge q moves with velocity $\mathbf{v}$, it induces a current on the readout electrode given by

$$i(t) = -\,q\,\mathbf{v}\cdot\mathbf{E}_w(\mathbf{r}), \tag{3.2}$$

where $i(t)$ is the induced current, q is the moving charge, $\mathbf{v}$ is its velocity, and $\mathbf{E}_w(\mathbf{r})$ is the weighting field at position $\mathbf{r}$. The weighting field is a mathematical field used to calculate how strongly the motion of a charge contributes to the signal seen by a given electrode. It depends only on the detector geometry and on the chosen readout electrode, and it is not the same as the real electric drift field in the detector. In simple words, moving charges can produce a signal even before they arrive at the electrode.

Sometimes it is easier to think in terms of **total induced charge** rather than instant current. Using the **weighting potential** ϕ_w, the Shockley–Ramo relation can also be written as[42]

$$\int_{t_1}^{t_2} i(t)\,dt = q\,[\,\phi_w(\mathbf{r}_2) - \phi_w(\mathbf{r}_1)\,]. \tag{3.3}$$

This form makes the message simple: the induced signal depends on **where the charge starts and ends**, and on the electrode geometry through ϕ_w.

With these rules, the Sensor follows a very practical workflow. First, we tell the Sensor **which electrode is the readout**. Second, we define **how we want to store time**, by

choosing a time range and dividing it into small bins. This binning is important because the output of the Sensor is a **waveform**: a list of current values $i(t)$ at discrete times. Third, when we simulate drift lines and avalanches, the Sensor **adds up** the current contributions from all moving electrons and ions into the correct time bins. In the end, we can read the waveform as the **total signal**, or separately as the **electron-induced** and **ion-induced** parts. Finally, after finishing one track/event, we reset the Sensor signal so the next track starts from zero [1]. so in this way the Sensor object converts "charge motion" into a stored current-vs. time waveform on the readout electrode, using Shockley–Ramo, and ready for electronics and digitisation[45].

3.2.4 Drift Tubes and Full Simulation Chain

In the IDEA detector concept, a drift chamber is proposed as part of the central tracking system. This detector improves the performance in two important ways. First, it provides particle identification through the ionization pattern produced in the gas, in particular through the number of primary ionization clusters along the particle track. Since different particles produce different ionization patterns, this information can be used for particle identification. Second, the drift chamber improves the tracking performance by providing precise space–time information, which helps the track fit and improves the momentum resolution of charged particles.

For our drift-tube full-simulation study, the detector and operating conditions used in Garfield++ are summarized in Table 3.3. These simulation parameters were chosen to match the real test-beam setups as closely as possible, namely the 2022 real-data conditions for 180 GeV muons and the 2023 real-data conditions for lower-momentum muons in the range of 2–10 GeV. In this way, the simulated waveforms and detector response can be compared with data under consistent conditions.

The table includes the gas mixture (He–isobutane) and the operating conditions (temperature and pressure), which determine the ionization yield and the electron-transport properties. Each cell is designed with a 5:1 field-to-sense wire ratio to ensure a suitable electrostatic configuration. In this configuration, the field wires are made of aluminum alloy, while the sense wire is made of tungsten coated gold. The table enlist different simulation parameters such as the cell size, wire diameter, and applied voltages, which define the electric-field configuration and therefore the drift behavior near the sense wire. Finally, the incident particle type (muons), the set of muon momenta, and the track angle with respect to the sense-wire axis are also specified in Table 3.3.

Parameter	2022 Real Data	2023 Data
Sampling rate	1.5 GHz	1.5 GHz
Temperature	293 K	293 K
Gas mixture	He (90%) & C_4H_{10} (10%)	He (90%) & C_4H_{10} (10%)
Pressure	725 Torr	725 Torr
Cell size	0.8 cm	0.8 cm
High voltage	1350 V	**1450 V**
Track angle	45°	45°
Sense wire diameter	20 µm	20 µm
Particle	Muon	Muon
Momentum	180 GeV	**2, 4, 6, 8, and 10 GeV**

Table 3.3: Simulation and operating parameters are chosen to match the real test-beam conditions (2022–2023) for the cluster counting study.

3.2.5 Analogue waveform generation

Our simulation package for the cluster-counting study consists of two main parts: simulation and digitization. This section describes the simulation part and explains how the analogue waveform is produced. The overall workflow of the simulation chain, from the drift chamber response and HEED-based ionization to the final analogue waveform generation, is summarized in Figure 3.5. When a charged particle traverses the drift tube, it creates ionization clusters along its trajectory, as described in the HEED-based ionization step. Each cluster releases primary electrons at different positions in the gas.

These electrons drift and diffuse toward the anode wire. Near the wire, where the electric field is very strong, gas amplification takes place and produces an avalanche. The motion of the charges generated in this process induces a signal on the readout electrode.

In practice, the full signal formation is represented by a pulse template together with two event-dependent parameters: the pulse start time, mainly determined by drift and diffusion, and the pulse amplitude, mainly determined by the avalanche multiplication.

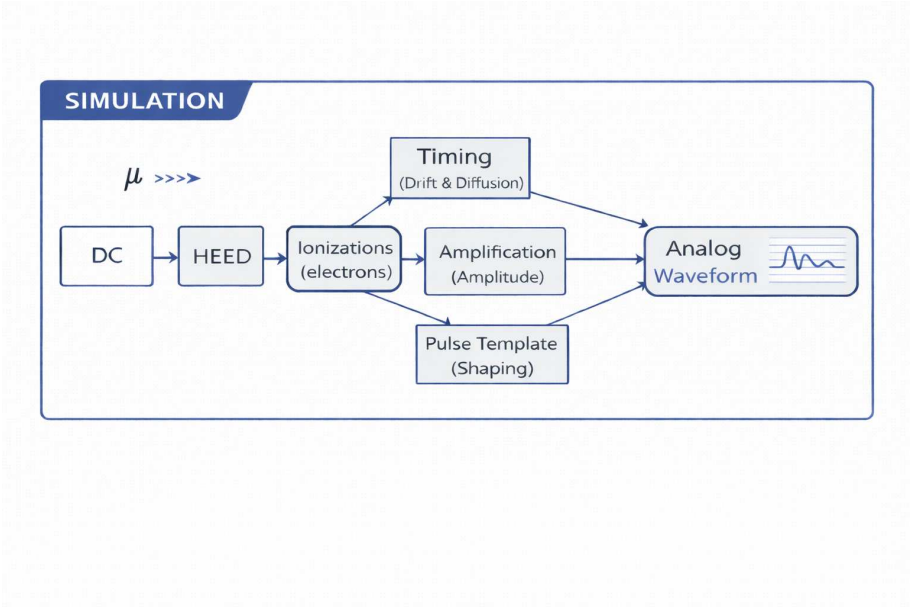


Figure 3.5: The simulation package creates analog induced current waveforms from ionizations. This ionization pattern is generated by the package HEED.

Drift and diffusion time of electrons in gases

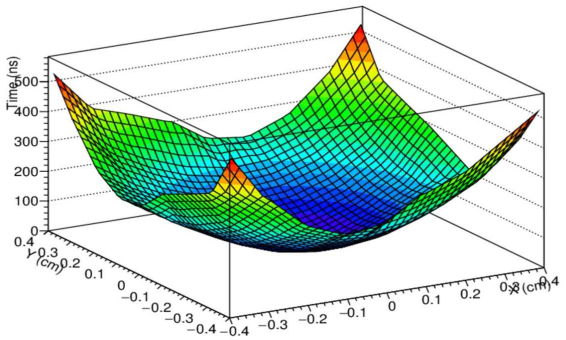
The goal of this part is to describe how drift and diffusion time is measured in drift tube comprises of one cell. To extract timing information, we calculate the drift time—this is the time difference between when the ionization happens and when the electrons reach the wire. This calculation is done for many positions inside the cell to fully understand how the drift time varies across the cell. The cell has a size of 0.8 cm × 0.8 cm. We performed a systematic scan of the cell by dividing it into a grid, with steps of 0.1 cm along both the X and Y directions, from -0.4 cm to +0.4 cm. This gave us a complete map of the drift behavior. Finally, the statistical spread (error) in the average drift time of the electrons gives us an estimate of the diffusion time. The distribution is described by a Gaussian model, and the resulting two-dimensional maps of the mean drift time and diffusion-driven time spread are shown in Figure 3.6.

$$t(x, y) \sim \mathcal{N}(\mu(x, y), \sigma(x, y)), \tag{3.4}$$

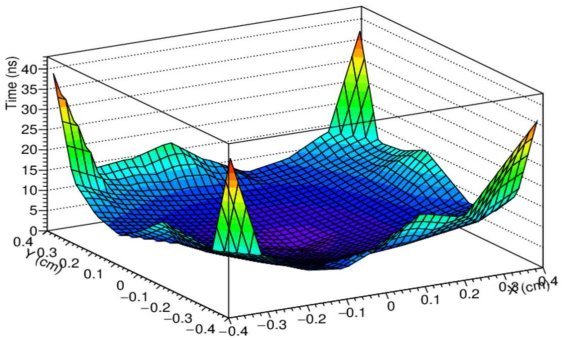
where $\mu(x, y)$ is the mean drift time and $\sigma(x, y)$ represents the time spread dominated by diffusion.

In this drift-tube cell, for each ionisation electron, we record the **begin time** t_{begin} (when the electron is created in the gas) and the **arrival time** t_{arrive} (when it reaches the amplification region near the sense wire). The drift time is then defined event-by-event as

$$t_{\text{drift}} = t_{\text{arrive}} - t_{\text{begin}}. \tag{3.5}$$



Mean drift time map, $\mu(x, y)$.



Drift-time spread (diffusion time) map, $\sigma(x, y)$.

Figure 3.6: Two-dimensional timing information for the drift-tube cell obtained from Garfield++ simulations. The mean arrival time of avalanche electrons provides the drift-time surface $\mu(x, y)$ (left), while the corresponding spread around the mean gives the diffusion-driven time broadening $\sigma(x, y)$ (right).

For the detector configuration used here, a representative mean value is about $t_{\text{drift}} \approx 198$ ns. and the standard deviation of the time give us diffusion time as mentioned in equation 3.4. This timing sets the start of the analogue pulse in the waveform construction by shifting the single-electron pulse according to the drift-time delay.

Amplification of ionisation

This part usually focuses on the pulse height produced when electrons reach the high-field region near the anode wire. Gas amplification is inherently stochastic: even for identical starting conditions, the avalanche size fluctuates. Therefore, the pulse height must be generated from a distribution rather than a single value.

Near a thin anode wire, the strong and highly non-uniform electric field leads to Townsend multiplication. The number of electrons in the avalanche, N_{aval}, fluctuates event by event and is commonly described by a Polya-like behaviour in wire detectors. The N_{aval} distribution is obtained from full Garfield++ simulations as shown in figure 3.7. To translate avalanche size into pulse height, the peak amplitude A is taken to scale with the avalanche size,

$$A \;\propto\; N_{\text{aval}}, \tag{3.6}$$

so, higher will be the amplitude of the pulse if there are more number of avalanches electrons

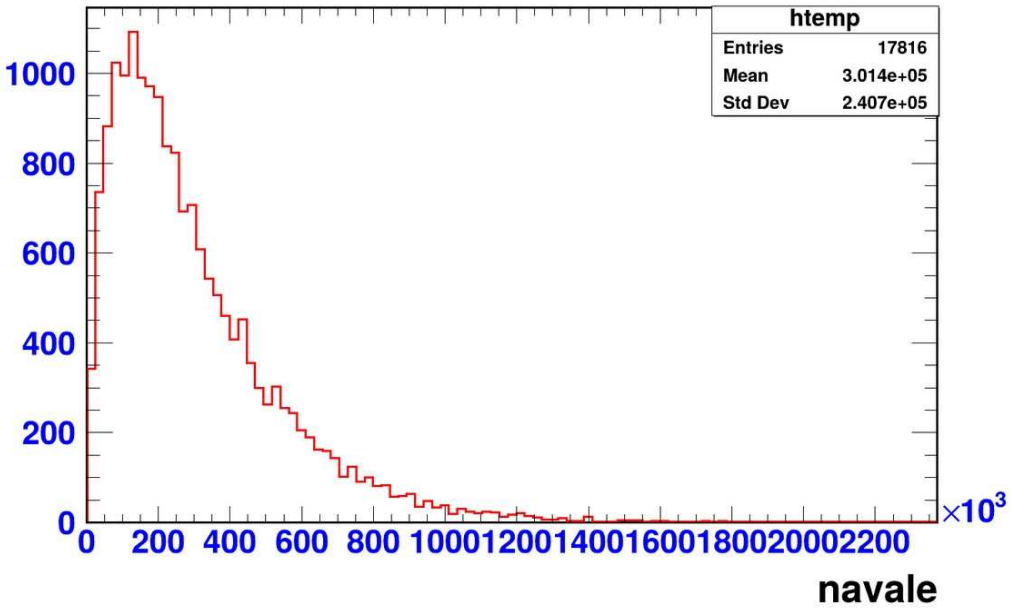


Figure 3.7: Avalanche-size distribution (N_{aval}) obtained from the Garfield++ full simulation, used to model the event-by-event amplitude fluctuations.

Single pulse

The third ingredient used to construct the analogue waveform is the *single-electron pulse*, which is taken as the basic template for the signal shape. In the drift-tube model, each electron arriving in the anode region produces an individual pulse. The full analogue waveform is then obtained by summing many such pulses with different start times, given by the timing model, and different amplitudes, given by the amplification model.

To describe the pulse shape in a simple and efficient way, a parameterized rise–fall model is used. In this model, the pulse rises rapidly near its starting time and then decreases more slowly, producing the characteristic signal tail. The single-pulse model is written as

$$f(x|A,t) = \begin{cases} p_0 \times \dfrac{e^{-p_1(x-p_2)}}{1+e^{-(t-p_3)/p_4}}, & x < t, \\ A \times \dfrac{p_5^{p_6}}{(x-t)^{p_6}+p_5^{p_6}}, & x \geq t, \end{cases} \tag{3.7}$$

where x is the time, t is the pulse start time, and A is the pulse amplitude. The first part of the function describes the rising edge of the pulse, while the second part describes its decay.

Figure 3.8 shows the single-electron pulse obtained together with the fitted pulse-shape model used in this work. During waveform construction, this template is shifted according to the electron arrival time and scaled according to the corresponding pulse amplitude. In this way, the timing model determines when the signal appears, the amplification model determines its size, and the single-pulse model determines its shape in time.

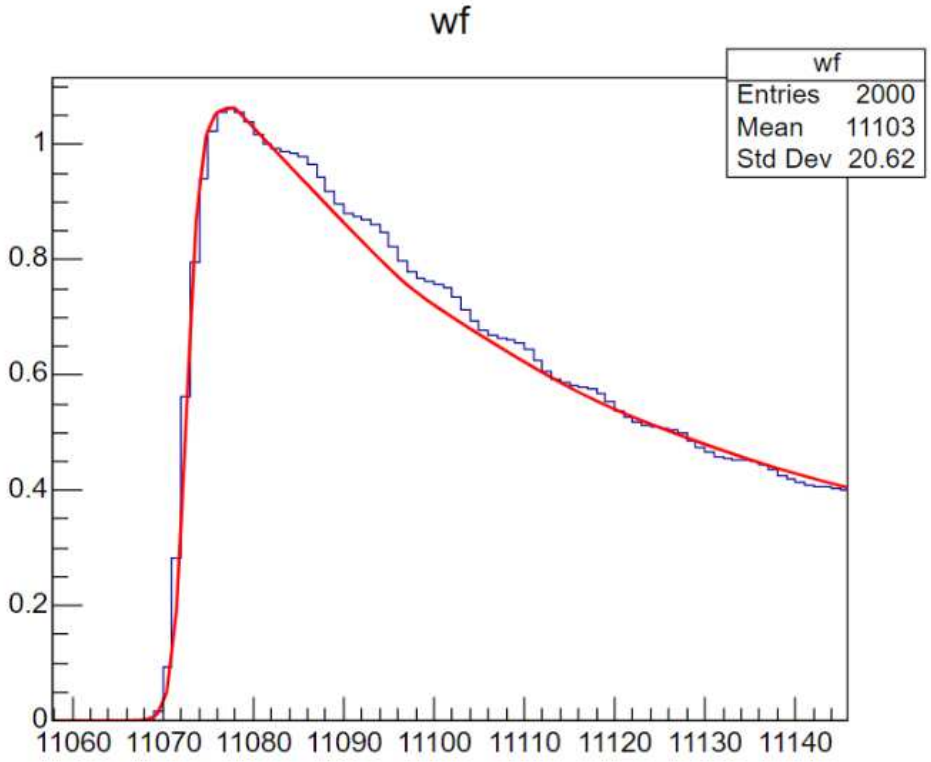


Figure 3.8: Single-electron pulse together with the fitted pulse-shape model used as the template for analogue waveform construction. The histogram shows the pulse shape, while the curve represents the analytical fit used to describe its rise and fall behavior.

Representative examples of analogue waveforms generated with Garfield++ for the cluster-counting study are shown in Figure 3.9.

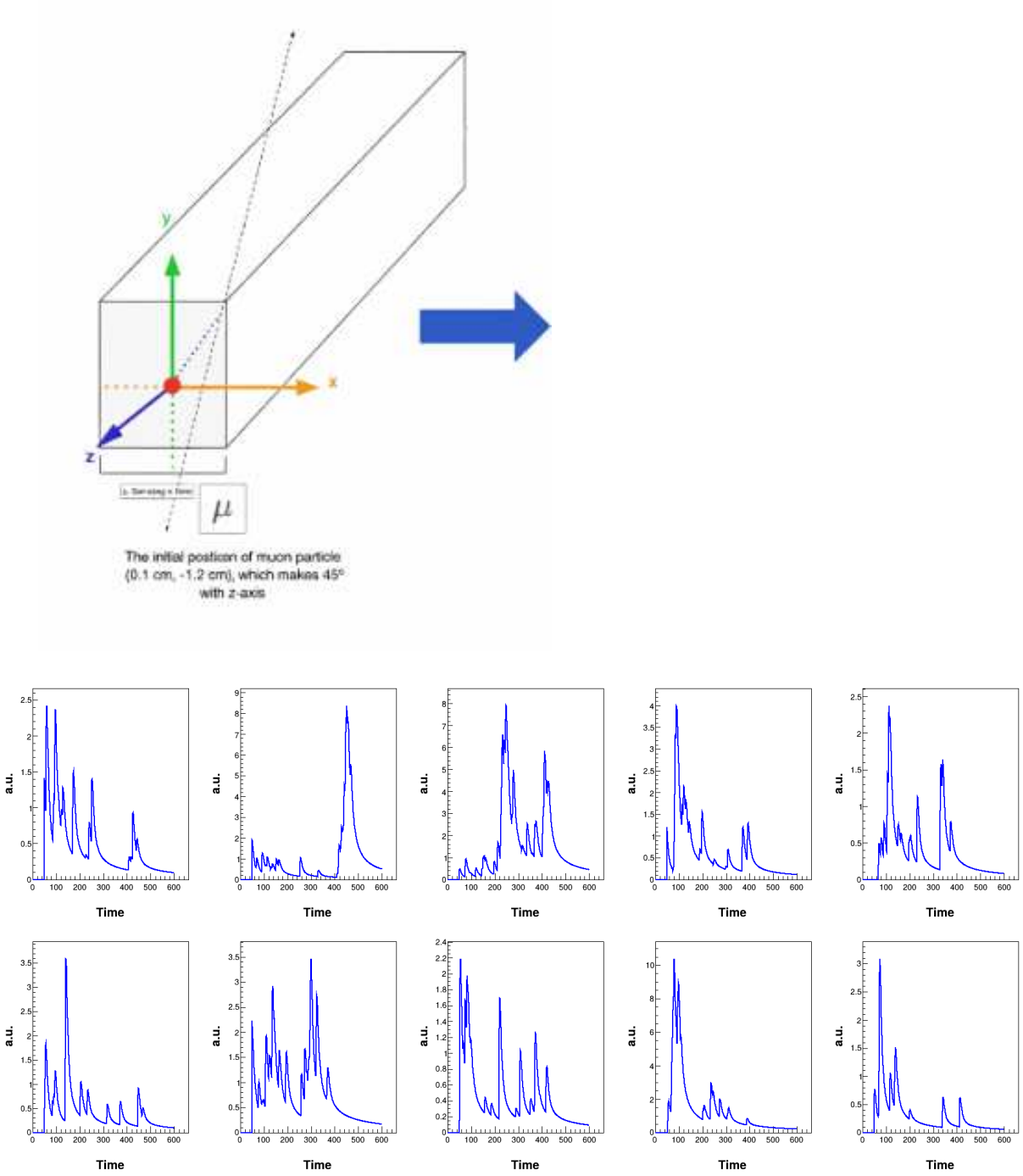


Figure 3.9: Example analogue waveforms simulated in the drift tube cell with Garfield++ for a muon passing through the detector. The upper panel illustrates the initial muon position and trajectory, and the lower panels present representative induced-current waveforms versus time obtained from the simulation.

3.2.6 Digitization

Digitization is the second stage of our Garfield++-based simulation chain. In the previous stage, the analogue waveform is built from the drift and diffusion timing, the pulse amplitude, and the single-pulse template. In this stage, the analogue waveform is converted into

a more realistic digitized waveform by adding electronic noise taken from the experimental data. The overall digitization procedure is summarized in Figure 3.10 .

To model the noise, we start from noise-only waveforms recorded with the same readout electronics used in the experimental measurements. These measured waveforms are first studied in the frequency domain using the Fast Fourier Transform (FFT). For a time-domain signal $f(t)$, the Fourier transform is written as

$$\mathcal{F}[f(t)] = \int_{-\infty}^{\infty} f(t)\, e^{-i\omega t}\, dt = F(i\omega), \qquad \omega = 2\pi f, \tag{3.8}$$

where f is the ordinary frequency and ω is the angular frequency. The corresponding inverse transform reconstructs the signal in the time domain:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(i\omega)\, e^{i\omega t}\, d\omega. \tag{3.9}$$

In practice, the FFT of the measured noise waveforms provides the noise amplitude spectrum, as illustrated in Figure 3.11. This spectrum shows how strong the noise is at each frequency. To generate a noise waveform in the time domain, we keep this measured amplitude spectrum and assign a random phase to each frequency component. In simple words, the amplitude tells us *how much* noise is present at a given frequency, while the phase determines *how these frequency components combine in time*. In this way, the generated waveform preserves the same frequency content as the measured noise, but it is not simply a copy of one recorded waveform.

After that, the noise waveform is reconstructed in the time domain by inverse Fourier transformation. The reconstructed noise is then added to the simulated analogue waveform. In this way, the analogue waveform is converted into a realistic digitized waveform for the cluster-counting study.

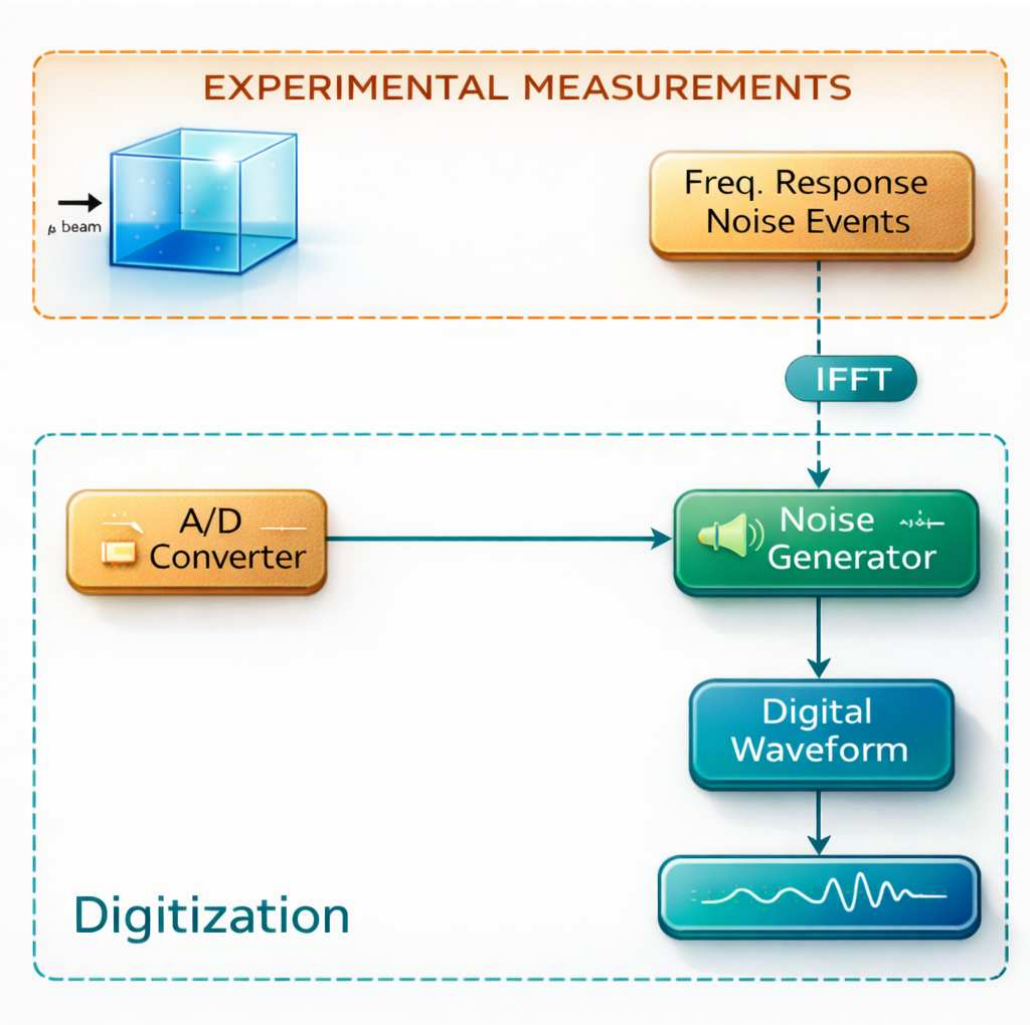


Figure 3.10: Schematic illustration of the digitization step. The measured noise information from experimental data is used to generate a time-domain noise waveform, which is then added to the simulated analogue waveform to obtain the final digitized waveform.

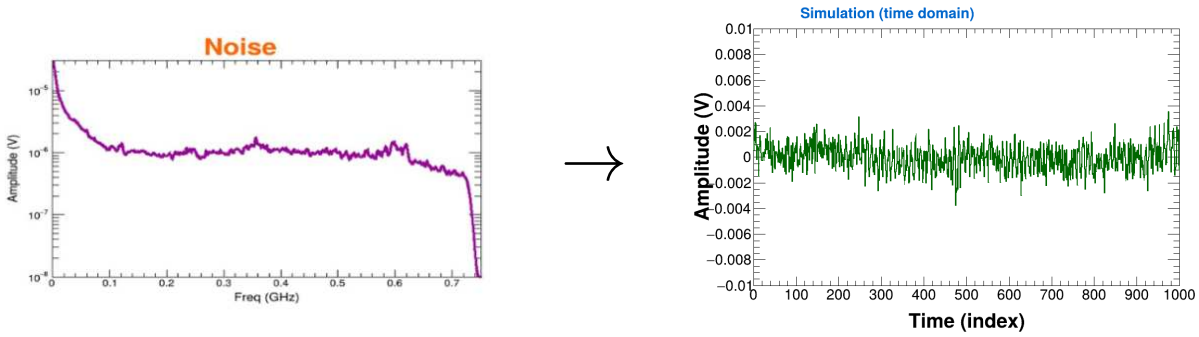


Figure 3.11: Data-driven noise model used in the digitization step. The left panel shows the measured noise spectrum in the frequency domain obtained from noise-only experimental waveforms using the Fast Fourier Transform (FFT), while the right panel shows a generated noise waveform reconstructed in the time domain from the same spectral information by inverse Fourier transformation.

In this way, the analogue waveform is converted into a realistic digitized waveform for the cluster-counting study. Representative examples of the final digitized waveforms obtained after adding the generated electronic noise are shown in Figure 3.12.

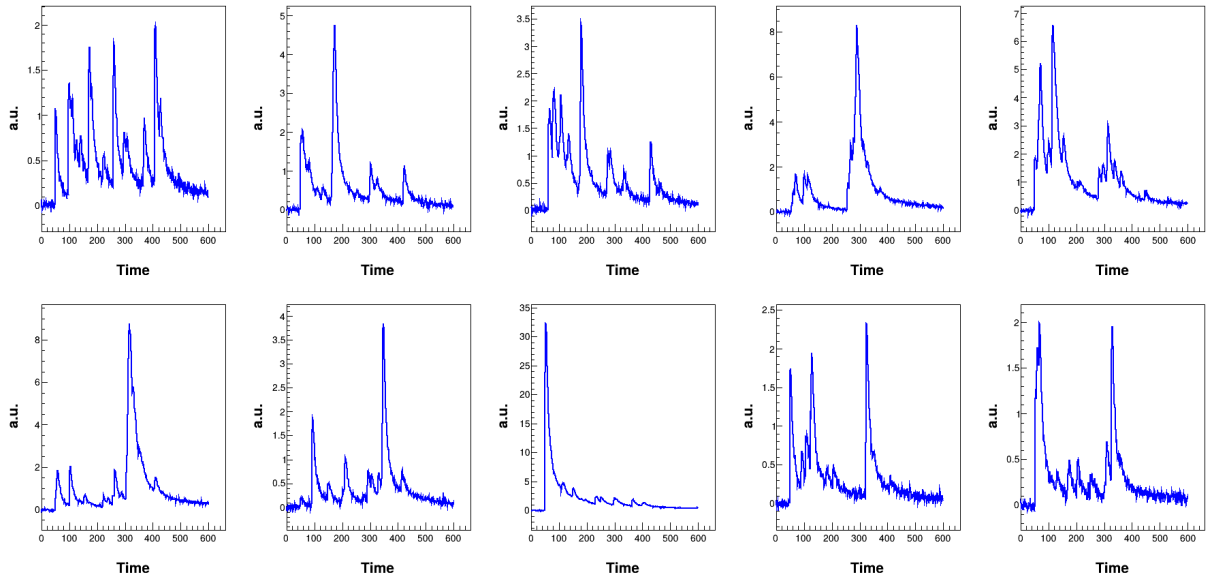


Figure 3.12: Examples of digitized waveforms obtained after adding the generated electronic noise to the simulated analogue waveforms.

Chapter 4

Deep Neural Network Models in Machine Learning

The main goal of this chapter is to define the scope of the neural-network models used in this thesis and to provide the theoretical background necessary for the later methodological developments. In particular, this chapter introduces the basic concepts of deep neural networks, Long Short-Term Memory (LSTM) networks, and Convolutional Neural Networks (CNNs), with emphasis on their main building blocks, architectures, and learning principles relevant to this work. It also introduces the main hyperparameters that control the training procedure, since the hyperparameter optimization of deep-learning models is a central component of this thesis and is later used to identify the best-performing models.

This chapter further discusses the main strategies used to improve model generalization and training stability, including methods to reduce overfitting and underfitting. In this sense, the scope of the chapter is methodological: it provides the conceptual basis needed to understand the training, optimization, and selection of neural-network models for the cluster-counting study in the following chapters.

4.1 Building Blocks of Neural Networks

This section introduces the main components of deep neural networks that are used to build models for data analysis. In experimental high energy physics, we often start from measured quantities (for example, kinematic variables, detector-level features, or reconstructed objects) and we want a model that can learn a useful relationship between these inputs and a target output. A neural network provides a flexible way to learn this relationship directly from data.

A neural network is made of connected *nodes* (also called neurons) arranged in *layers*. The first layer receives the input variables, one or more middle layers process the information, and the final layer produces the output. At each node, the computation follows

the same two-step pattern. First, the node forms a *linear mapping with a displacement* (an affine transformation). Second, the result is passed through a *nonlinear activation function*. The parameters of the linear mapping and the displacement are adjustable, so they can be learned during training. The activation function is important because it allows the network to represent nonlinear behavior, which is needed for most realistic physics problems.

The output of a network can be a single value or a vector of values. When the output is a continuous quantity (for example, an energy scale factor or a calibrated response), the task is called *regression*. When the output represents a category (for example, signal vs. background, or jet flavor classes), the task is called *classification*. In the next sections, we explain the core mathematical operations that appear in almost every neural network. After that, we discuss why these building blocks are powerful enough to approximate complicated functions (universal approximation).

4.1.1 Linear mapping and displacement

We start with the simplest operation inside a neural network: a linear mapping plus a displacement. Figure 4.1 shows a small example network with two input values,

$$\vec{x} = (x_1, x_2)^T, \tag{4.1}$$

two nodes in a hidden layer, and one output value z. The role of the hidden layer is to combine the inputs into new intermediate quantities that can later be used to form the final output.

Hidden layer as an affine transformation. Let the hidden layer produce the vector

$$\vec{y} = (y_1, y_2)^T. \tag{4.2}$$

The hidden layer computes $\vec{y}$ from the input vector $\vec{x}$ using a weight matrix $\mathbf{W}$ and a bias vector $\vec{b}$:

$$\vec{y} = \mathbf{W}\,\vec{x} + \vec{b}. \tag{4.3}$$

For the two-input, two-node example, this becomes

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \tag{4.4}$$

so each hidden node is

$$y_1 = W_{11}x_1 + W_{12}x_2 + b_1, \tag{4.5}$$
$$y_2 = W_{21}x_1 + W_{22}x_2 + b_2. \tag{4.6}$$

The weights W_{ij} control how strongly each input contributes to each node. The bias terms b_i shift the result up or down, which helps the network fit data even when the best mapping does not pass through the origin.

Output layer as another affine transformation. After the hidden layer, the network forms the output z using another set of weights and a bias. If we write the output weights as

$$\mathbf{W}' = (c_1 \;\; c_2), \tag{4.7}$$

and the output bias as d, then the output is

$$z = \mathbf{W}'\,\vec{y} + d = c_1 y_1 + c_2 y_2 + d. \tag{4.8}$$

By inserting the values of y_1 and y_2 into Eq. (4.8), we obtain

$$\begin{aligned} z &= c_1(W_{11}x_1 + W_{12}x_2 + b_1) + c_2(W_{21}x_1 + W_{22}x_2 + b_2) + d \\ &= (c_1W_{11} + c_2W_{21})x_1 + (c_1W_{12} + c_2W_{22})x_2 + (c_1b_1 + c_2b_2 + d). \end{aligned} \tag{4.9}$$

This result highlights an important point, if we use only linear layers and biases, the network is still just a *linear-like* formula of the inputs. Adding more such layers does not make the model smarter, because all of them can be combined into one single layer (one new set of weights and one new bias).

4.1.2 Nonlinear Mapping: Activation Function

In the previous subsection, we explained that each node in a neural network first performs a linear mapping with a displacement (bias). If we stack only these affine transformations, the full network still behaves like a single linear function of the input variables. Such a model can only describe simple linear relations. However, the data observed in experimental high energy physics often contain complex and nonlinear patterns. For example, detector response, particle interactions, and reconstruction effects introduce nonlinear correlations among variables. Therefore, an additional operation is required to allow the network to learn such complex behavior.

To introduce nonlinearity, a function denoted by σ is applied after the affine transformation at each node. This function is called an **activation function**. If the hidden layer

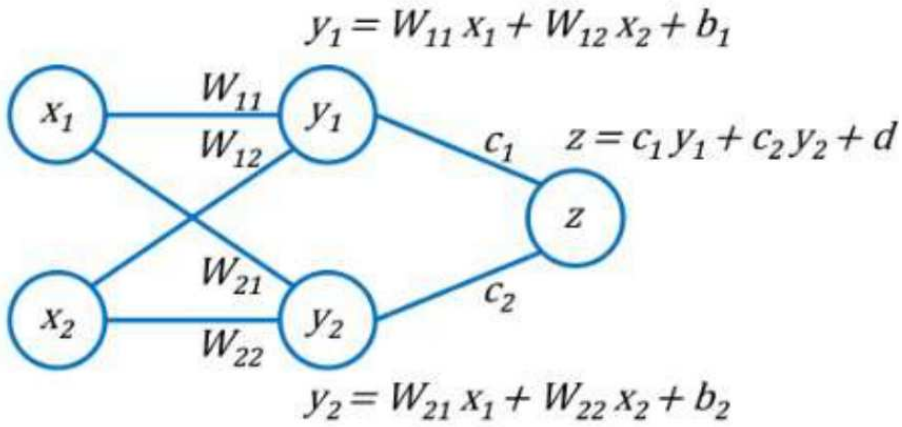


Figure 4.1: Example of a small network where each connection represents a weight and each node includes a bias term. The hidden layer first computes $\vec{y} = \mathbf{W}\vec{x} + \vec{b}$, and the output layer then computes $z = \mathbf{W}'\vec{y} + d$.[46]

output vector after linear mapping is

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \tag{4.10}$$

then the activation function is applied element-wise as

$$\sigma(\vec{y}) = \begin{pmatrix} \sigma(y_1) \\ \sigma(y_2) \end{pmatrix}. \tag{4.11}$$

This means each node output is passed through the nonlinear function independently. The transformed values are then used as inputs to the next layer.

Figure 4.2 illustrates this operation. Compared to the purely linear network shown earlier, the hidden nodes now include an activation step. The final output z is therefore written as

$$z = \mathbf{W}'\,\sigma(\vec{y}) + d, \tag{4.12}$$

or explicitly,

$$z = c_1\,\sigma(y_1) + c_2\,\sigma(y_2) + d. \tag{4.13}$$

If we substitute the linear expressions of y_1 and y_2, the output becomes

$$z = c_1\,\sigma(W_{11}x_1 + W_{12}x_2 + b_1) + c_2\,\sigma(W_{21}x_1 + W_{22}x_2 + b_2) + d. \tag{4.14}$$

At this stage, the network output can no longer be simplified into a single affine function of the inputs. The activation function breaks the linear structure and allows the model to represent curved boundaries and nonlinear responses. This property is essential

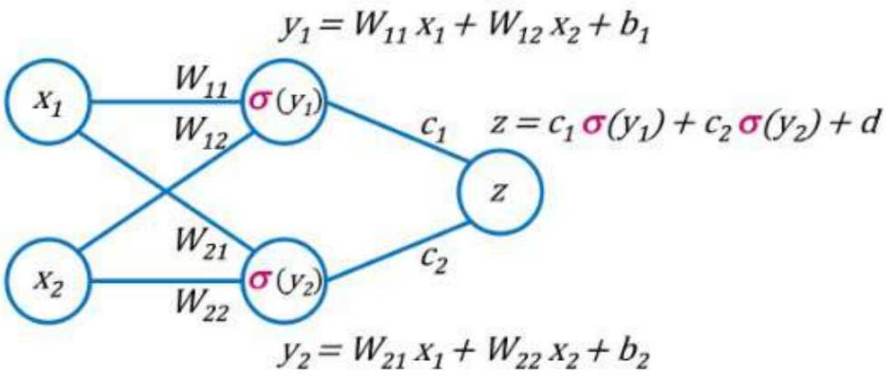


Figure 4.2: Neural network structure including nonlinear activation functions applied at hidden nodes. The activation step introduces nonlinearity, allowing the network to model complex relationships between input variables and the final output.

for classification and regression tasks in physics analyses.

Why we need nonlinearity. Because a purely affine network stays linear in its behavior, it cannot describe many patterns that appear in real data. In high energy physics, distributions and correlations are often nonlinear due to detector response, selection requirements, and the underlying event dynamics. For this reason, neural networks include *activation functions* between affine transformations. The activation functions introduce nonlinearity, and this is what makes deep networks useful as general modeling tools.

Example: Rectified Linear Unit (ReLU). One of the most widely used activation functions is the Rectified Linear Unit (ReLU). It is defined as

$$\mathrm{ReLU}(y) = \max(0, y). \tag{4.15}$$

This function passes positive values unchanged, while negative values are set to zero. Because of this behavior, the bias parameters b_1 and b_2 act like threshold controls. Only signals above a certain level are propagated to the next layer. ReLU is commonly used in deep learning applications because it is simple to compute and performs well in practice.

Other activation functions. Several other activation functions are also used. The **sigmoid** function was one of the earliest choices. It maps input values into the range between 0 and 1, which makes it useful for probability-like outputs. Another example is the **hyperbolic tangent (tanh)** function, which maps inputs into the range between -1 and $+1$. Different activation functions can be selected depending on the problem and the network design.

4.1.3 Network Prediction (Forward Pass)

After introducing the linear mapping and the activation function, we can now describe how a neural network produces an output for a given input event. At each node, the same two operations are performed in a fixed order. First, the node computes an affine transformation using weights and a bias. Second, the node applies a nonlinear activation function to the result. Because this procedure is repeated layer by layer, the network can build a complex mapping from the input variables to the final output.

For one layer, we write the affine transformation as

$$\vec{y} = \mathbf{W}\,\vec{x} + \vec{b}, \tag{4.16}$$

where $\vec{x}$ is the input vector to the layer, $\mathbf{W}$ is the weight matrix, and $\vec{b}$ is the bias vector. The activation function $\sigma(\cdot)$ is then applied element-by-element to obtain the layer output

$$\vec{z} = \sigma(\vec{y}) = \sigma(\mathbf{W}\,\vec{x} + \vec{b}). \tag{4.17}$$

This compact expression summarizes what a single network layer does. The output vector $\vec{z}$ becomes the input to the next layer, and the same steps are repeated until the final layer produces the network output.

Neural networks can contain many layers, and each layer can contain many nodes. In every layer, the network takes the information coming from the previous layer and transforms it into a new representation. The complete procedure of computing the output from the input by passing through all intermediate layers is called the **forward pass**. The value produced at the end of this forward pass is the **network prediction**.

In experimental high energy physics, the input vector usually contains event-level or object-level quantities, such as momenta, angular variables, isolation information, impact parameters, or detector response features. The network prediction then represents the quantity we want to estimate from these measurements. For example, the output can represent a classifier score used to separate signal from background, or it can represent a continuous value used for calibration or regression.

Common activation functions. The role of the activation function is to introduce nonlinearity. Figure 4.3 shows three common activation functions that are often used in practice. The Rectified Linear Unit (ReLU) keeps positive inputs and sets negative inputs to zero, which makes it simple and efficient. The sigmoid function maps values into the range between 0 and 1, which is useful when the output should behave like a probability. The hyperbolic tangent function maps values into the range between -1 and $+1$, which can be helpful when we want outputs centered around zero. These choices affect how information flows through the network, but the forward-pass idea remains the same: affine

mapping followed by a nonlinear activation.

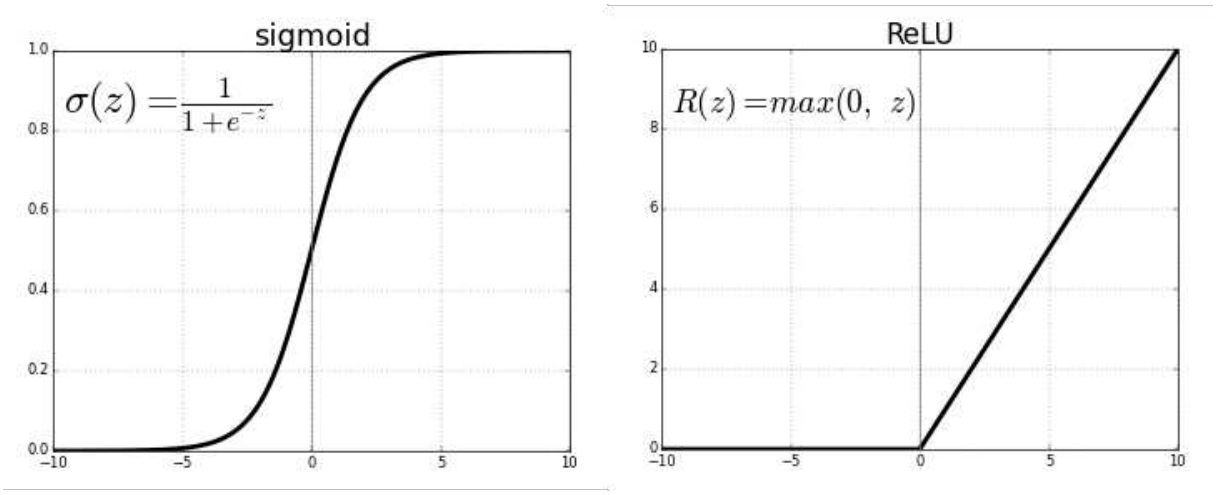


Figure 4.3: Examples of commonly used activation functions: sigmoid(left side) and ReLU (right side),. These nonlinear functions are applied after the affine transformation in each node, allowing the network to learn nonlinear relationships.

Regression

After the forward pass is defined, the meaning of the network output depends on the physics task. In many cases, we want the network to return a *continuous* value. This situation is called **regression**. A simple example is when the network receives one real input $x \in R$ and returns one real output $z \in R$. In that case, the network acts like a function

$$z = f(x), \tag{4.18}$$

where $f(x)$ is learned from data.

In experimental high energy physics, regression is widely used because many important quantities are continuous. For example, a network can be trained to estimate an energy correction, a momentum calibration factor, number of primary clusters, or a resolution-related quantity. The goal is that the predicted value follows the trend seen in the training data and can also generalize to new events.

To make the network produce the desired regression output, we adjust its weights and biases using a process called **network training**. In a simple way, this training behaves like a fit. The network parameters are tuned such that the predicted outputs are as close as possible to the target values. When training is finished, the network can be applied to new data that were not used during training. You will find more details in chapter 6

Figure 4.4 (left) illustrates the idea of regression. The blue points represent data, and the curve represents the learned function $f_\theta(x)$, where θ denotes the set of trainable parameters. Although the example uses only one input and one output, the same idea extends directly to multiple input variables. In practice, physics problems often use many

input features, and the network learns a mapping from a high-dimensional input space to one continuous output.

Classification

Another common task is to assign each input event to a category. This task is called **classification**. In high energy physics, classification is central because we often want to separate signal events from background events. For example, a network can be trained to distinguish Higgs boson candidates from Standard Model background processes, or to identify a certain particle type based on detector signatures.

In the simplest case, we have two categories: *signal* and *background*. The network still produces an output value, but now we interpret that output as a score that tells how signal-like the event is. A common approach is to apply a sigmoid function to the last network output:

$$z' = \sigma(z), \tag{4.19}$$

so that the final output satisfies $z' \in [0, 1]$. This range is useful because it can be interpreted as a probability-like value. During training, the network parameters are adjusted such that signal events tend to give large values (for example $z' > 0.5$), while background events tend to give small values (for example $z' < 0.5$). In this way, the network learns a decision boundary that separates the two classes.

Figure 4.4 (right) shows an example where two input variables (x_1, x_2) are needed. In this situation, using only x_1 or only x_2 is not enough to separate the classes. The network combines the variables and learns a separation curve in the two-dimensional space. This is exactly the type of behavior that is useful in physics analyses, where discrimination power often comes from combining multiple weak features into a stronger overall decision.

The classification concept also extends to many input variables ($n \gg 1$) and to more than two output categories. In that case, the network assigns each event to one of several classes, such as different background types or different physics objects. After training, the network can be evaluated on independent data to test how well it generalizes beyond the training sample.

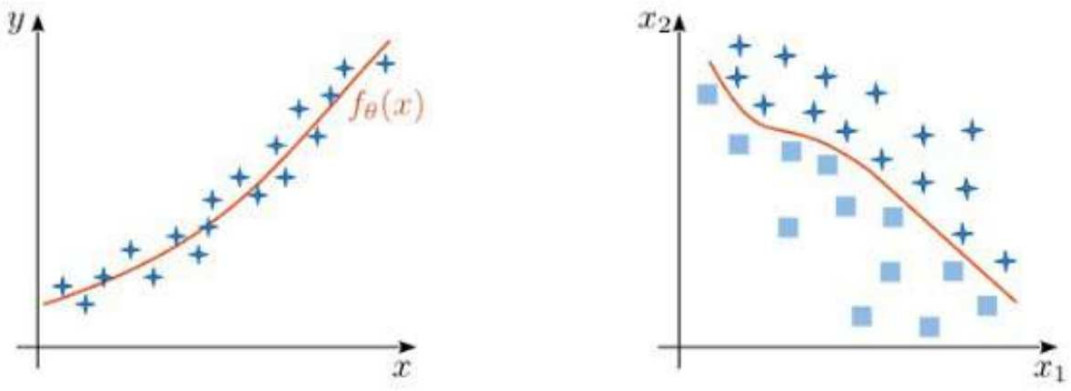


Figure 4.4: Illustration of two common neural network tasks. Left: regression, where the network learns a continuous function that follows the trend of the data points. Right: classification, where the network learns a decision boundary in the input feature space (here shown for two variables x_1 and x_2) to separate two categories.[46]

4.2 Overview of Deep Learning Models

Deep learning models has developed rapidly over the past years and has become one of the most powerful tools in modern data analysis. Its strength comes from the ability to learn complex patterns directly from data without requiring manual feature design. Because of this capability, deep learning methods are now widely used in many scientific and technological fields, including experimental high energy physics.

Traditional deep learning architectures, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have played an important role in this progress. CNNs were originally developed for image processing tasks. They use convolution operations to learn spatial structures in data, which makes them highly effective for recognizing patterns in images and videos[19]. Early studies, such as digit recognition and large-scale image classification, demonstrated that deep convolutional networks can achieve much higher accuracy than earlier machine learning approaches.

On the other hand, RNNs were designed to handle sequential data. Their structure allows information to flow from one step to the next, making them suitable for time-dependent problems such as speech recognition and language modeling. However, early RNN models faced difficulties in learning long-range dependencies. The introduction of Long Short-Term Memory (LSTM) units solved many of these issues by controlling how information is stored, updated, and forgotten over time.

From a theoretical point of view, neural networks are supported by the universal approximation property. This principle states that, given a nonlinear activation function, even a neural network with a single hidden layer can approximate any continuous function if it contains a sufficient number of neurons. This result explains why neural networks are flexible modeling tools and why they can represent complicated relationships present in

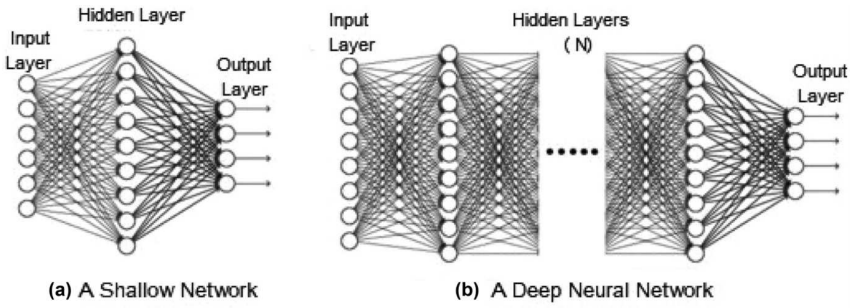


Figure 4.5: Comparison between shallow (simple neural network) and deep neural network architectures. A shallow network contains a small number of hidden layers, while a deep neural network includes multiple hidden layers, enabling the learning of more complex and hierarchical data representations[48].

physics data.

Neural networks are commonly divided into shallow (simple neural network) and deep models. The main difference lies in the number of hidden layer. Shallow networks usually contain one or two hidden layers and therefore have limited ability to learn complex structures in contrast, deep neural networks contain multiple hidden layers[47]. This increased depth allows them to learn hierarchical representations, where simple features are learned in early layers and more abstract features are built in later layers.

The depth of a network plays a key role in its performance. With more layers, the network can capture more detailed structures in the data. This capability forms the foundation of many modern artificial intelligence systems and motivates the use of deep learning models in physics analyses.

4.2.1 Components of Deep Learning Models

Deep learning models are built from several fundamental components that work together to learn from data and produce predictions. Each component has a specific role, and the performance of the full model depends on how these parts interact. The main components include layers, activation functions, loss functions, and optimization algorithms.

Layers define how information flows through the network. Activation functions introduce nonlinearity, allowing the model to learn complex mappings. Loss functions measure how close the network predictions are to the target values. Optimization algorithms then adjust the trainable parameters to minimize this loss during training. Together, these elements allow deep learning models to learn meaningful representations from large datasets.

Layers

Layers form the core structure of any deep learning model. They determine how input data are processed and transformed as they move through the network. A typical deep

learning architecture consists of three main types of layers: the input layer, one or more hidden layers, and the output layer.

The input layer receives the raw data/features of the object. In physics applications, these inputs may include reconstructed kinematic variables, detector observables, total events(signal + background) or high-level analysis features. The information is then passed to the hidden layers, where most of the computation takes place. Each hidden layer applies mathematical transformations that allow the network to extract relevant patterns from the data.

As the data propagate through multiple hidden layers, the network gradually builds more abstract representations. Early layers may learn simple structures, while deeper layers combine them into more complex features. Finally, the processed information reaches the output layer, which produces the final prediction. This prediction can correspond to a continuous value in regression tasks or a category label in classification problems.

In addition to standard dense layers, deep learning models often include specialized layer types designed for particular data structures. Convolutional layers, used in CNNs, apply learnable filters to detect spatial patterns such as edges or textures in images. Fully connected layers, typically placed near the network output, combine all extracted features to perform high-level reasoning[49]. Recurrent layers, used in RNNs and LSTMs, are designed to capture temporal dependencies in sequential data.

By combining these different layer types, deep learning models become capable of learning rich and complex structures in data. This flexibility makes them powerful tools for solving challenging problems across many domains, including experimental high energy physics.

Activation Functions

Activation functions are also essential parts of deep learning models because they introduce nonlinearity into the network. If activation functions are not used, the network would only perform linear transformations, and this would strongly limit its ability to model complex relationships in the data [50]. For this reason, activation functions are applied inside the network so that the model can learn more detailed patterns and produce better predictions.

A commonly used activation function is the sigmoid function. It maps the input to the range $(0, 1)$, which is useful in binary classification because the output can be interpreted as a probability [51]. It is written as

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{1}$$

However, sigmoid can lead to vanishing gradients in deep networks, which can slow down training and make learning more difficult.

To reduce this issue, the Rectified Linear Unit (ReLU) is often used in hidden layers. ReLU keeps positive values and sets negative values to zero:

$$\text{ReLU}(x) = \max(0, x). \tag{4.20}$$

ReLU is widely used because it is simple, fast, and often helps training converge more effectively [52].

For multi-class classification, the softmax function is usually applied in the output layer. Softmax converts the raw outputs of the network (logits) into a probability distribution across all classes. It is defined as

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{2}$$

Here, z_i is the logit for class i, and the denominator normalizes the outputs so that the probabilities sum to one.

Other activation functions are also used depending on the problem. For example, tanh maps values to $(-1, 1)$ and is often useful because it is centered around zero. Another option is leaky ReLU, which keeps a small slope for negative inputs. This helps reduce the "dying ReLU" problem where some neurons can stop updating during training [39].

Objective/Loss Functions

Loss functions, also called cost or objective functions, are used to measure how far the model prediction is from the true target value. During training, the model parameters are updated to reduce this loss, so the loss function directly guides the learning process [53]. The best loss function depends on the task. Regression and classification have different goals, so they often require different loss definitions.

For regression, one of the most common choices is the Mean Squared Error (MSE). It computes the average of the squared difference between the true values y_i and the predicted values $\hat{y}_i$:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{3}$$

In this equation, n is the number of samples. Because the errors are squared, large errors are penalized strongly. This is helpful in many cases, but it also makes MSE sensitive to outliers[54].

For classification, a widely used loss is cross-entropy. Cross-entropy measures the difference between the true label distribution p and the predicted distribution q. It can be written as

$$\text{Cross Entropy} = -\sum_i p(y_i)\log q(y_i). \tag{4}$$

In binary classification, this simplifies to binary cross-entropy, where the target is either 0

or 1. In multi-class classification, categorical cross-entropy is used, where $p(y_i)$ is often encoded as a one-hot vector and $q(y_i)$ is the predicted probability distribution over all classes. Cross-entropy is often preferred because it strongly penalizes confident but wrong predictions, which pushes the model to produce better probability estimates.

In some situations, we want a regression loss that is less sensitive to outliers. A common solution is the Huber loss, which combines the behavior of MSE and MAE. It is quadratic for small errors and linear for large errors. It is defined as

$$\text{Huber Loss} = \begin{cases} \frac{1}{2}\left(y_i - \hat{y}_i\right)^2, & \text{for } |y_i - \hat{y}_i| \leq \delta, \\ \delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases} \tag{5}$$

Here, δ controls where the loss changes from quadratic to linear. Because large errors are treated linearly, the Huber loss is usually more robust to outliers than MSE [43].

Optimization Algorithms

Optimization algorithms are used to update the model parameters during training so that the loss function becomes smaller. In simple words, training is a repeated process: we compute how wrong the model is (the loss), then we slightly change the parameters to reduce this error. A good optimization method is important because it affects how fast the model learns and how stable the training becomes, especially for deep networks.

- **Stochastic Gradient Descent (SGD):** Stochastic Gradient Descent is one of the basic and most widely used optimization methods in deep learning [55]. It updates the parameters θ by moving them in the opposite direction of the gradient of the loss function $\nabla_\theta \mathcal{L}(\theta)$. The update rule is

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta). \tag{6}$$

  Here, η is the learning rate, which controls the step size of each update. Unlike full-batch gradient descent, SGD often uses a randomly selected subset of the training data (a mini-batch). This makes each update cheaper and usually faster in practice. However, because the updates are noisy, SGD can be sensitive to the choice of learning rate and may fluctuate around the optimum instead of smoothly converging.

- **AdaGrad:** AdaGrad (Adaptive Gradient Algorithm) changes the learning rate automatically for each parameter based on the past gradients [56]. The main idea is that parameters that receive large gradients many times will get smaller future steps, while parameters with rare updates can keep larger steps. The update rule is

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_{t,t}} + \epsilon} \nabla_\theta \mathcal{L}(\theta). \tag{7}$$

In this expression, G_t is a diagonal matrix, and each diagonal element $G_{t,t}$ is the sum of squared gradients up to time step t [45]. The small constant ϵ avoids division by zero. AdaGrad can work well for sparse features, but a common drawback is that the effective learning rate can keep decreasing over time and may become too small, which can slow down learning.

- **RMSProp:** RMSProp (Root Mean Square Propagation) was introduced to avoid the continuously shrinking learning rate problem of AdaGrad [57]. Instead of accumulating all past squared gradients, RMSProp keeps a moving average of recent squared gradients. This helps the learning rate stay at a reasonable scale, so training can continue effectively. The update rule is

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla_\theta \mathcal{L}(\theta). \tag{8}$$

  Here, $E[g^2]_t$ is the exponentially weighted moving average of the squared gradients. RMSProp is often useful when the loss function changes during training, and it is commonly applied in training recurrent models [47].

- **Adam Optimizer:** Adam (Adaptive Moment Estimation) is a very popular optimizer because it combines ideas from AdaGrad and RMSProp [58]. Adam keeps track of (i) the moving average of the gradients (first moment, mean) and (ii) the moving average of the squared gradients (second moment, uncentered variance). Using these two estimates, Adam adapts the step size for each parameter. A common form of the Adam update is

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \tag{9}$$

  The terms $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected estimates, computed as

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \qquad \text{with } m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta \mathcal{L}(\theta). \tag{10}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \qquad \text{with } v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\nabla_\theta \mathcal{L}(\theta)\right)^2. \tag{11}$$

  Here, β_1 and β_2 control how strongly the moving averages remember the past, and ϵ is a small constant for numerical stability. In practice, Adam often converges faster than basic SGD, especially when gradients are noisy or sparse, which is why it is widely used in deep learning applications [48,49].

4.2.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks are designed to process sequential data. In many real problems, the input data are ordered in time or sequence. Examples include time series measurements, speech signals, text, and detector readout streams. In experimental high energy physics,

collider experiments also generate sequential data where the order of signals carries physical meaning.

Sequential data differ from standard data because each element depends on previous elements. Traditional feedforward neural networks process all inputs at once and ignore this order. Therefore, they cannot naturally learn temporal relations.

To overcome this limitation, recurrent neural networks introduce an internal memory. This memory allows the network to store past information and use it while processing new inputs.

Basic RNN Architecture

The basic structure of an RNN at a single time step is shown in Figure 4.6. At each time step t, the network receives an input vector $\mathbf{x}_t$. It also keeps a hidden state vector $\mathbf{h}_t$ that stores past information. The hidden state is computed using the current input and the previous hidden state:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{4.21}$$

The output is obtained from the hidden state:

$$\mathbf{y}_t = g(\mathbf{h}_t) \tag{4.22}$$

The following figure shows the basic structure of an RNN at a single time step.

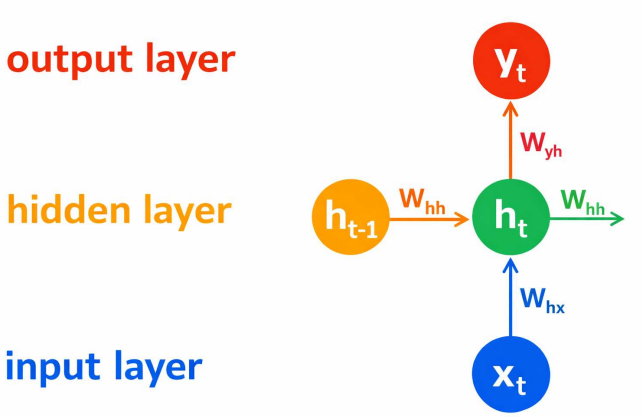


Figure 4.6: Basic architecture of a recurrent neural network showing input, hidden, and output connections at one time step.

In this structure, the hidden state acts as a memory that carries information forward.

Mathematical Formulation

In practice, the hidden state is computed using matrix operations and nonlinear activation functions:

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{4.23}$$

$$\mathbf{y}_t = \sigma(\mathbf{W}_{yh}\mathbf{h}_t + \mathbf{b}_y) \tag{4.24}$$

Because the hidden state depends on earlier states, it contains information from all previous inputs:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \dots) \tag{4.25}$$

Unrolling Through Time

For a finite sequence, the recurrent loop can be unfolded into multiple layers. This process is called unrolling through time and is illustrated in Figure 4.7. Each layer represents one time step of the sequence.

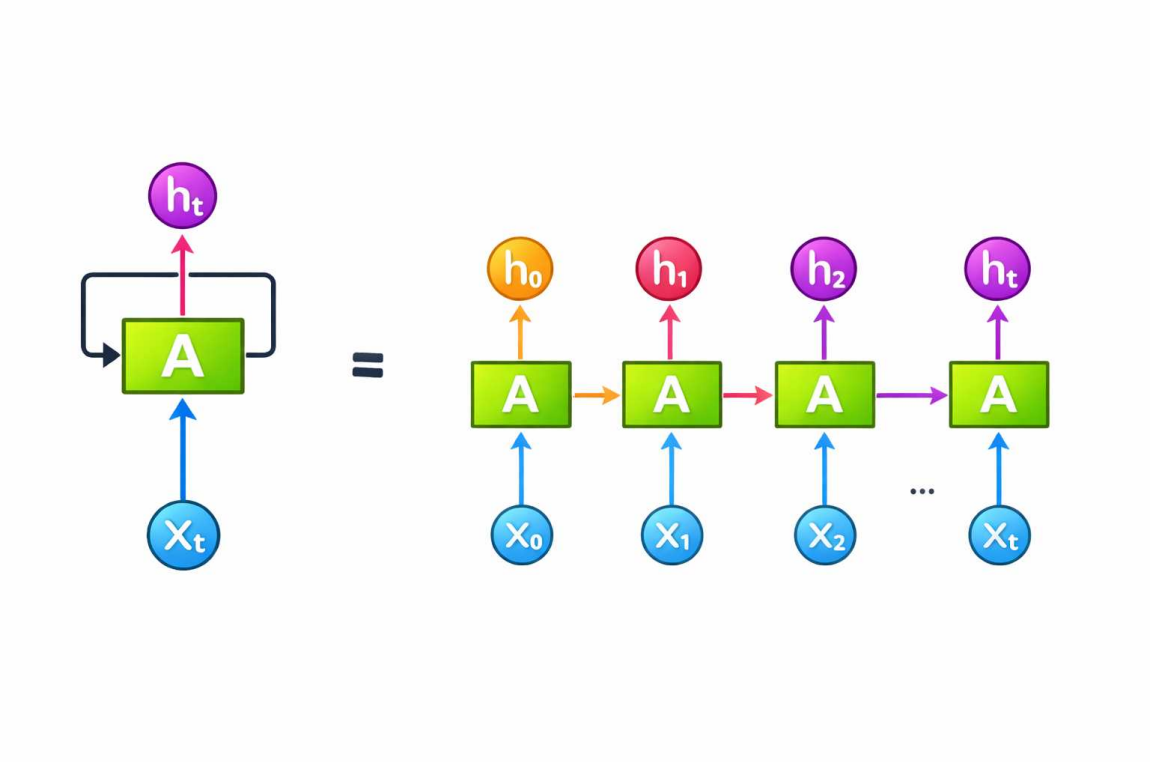


Figure 4.7: Unrolled representation of a recurrent neural network across multiple time steps. Each unit shares the same parameters while processing sequential inputs.

At the first step, the input $\mathbf{x}_0$ produces $\mathbf{h}_0$. At the next step, the network uses $\mathbf{x}_1$ together with $\mathbf{h}_0$. This process continues across the full sequence.

Because information flows across time, RNNs can learn temporal patterns effectively.

However, training standard RNNs becomes difficult for long sequences due to gradient instability. This limitation led to improved models such as Long Short-Term Memory

(LSTM) networks, which will be discussed next.

4.2.3 Long Short-Term Memory Models

Recurrent Neural Networks (RNNs) are a class of neural network models that are specifically designed to process sequential data. Unlike feed-forward neural networks, RNNs contain internal feedback connections that allow information to persist over time. This makes them particularly suitable for time-dependent or ordered data, such as detector signals, particle tracks, or event sequences in experimental high energy physics.

Although RNNs are powerful for sequence modeling, they suffer from the well-known **vanishing gradient problem**, which makes it difficult to learn long-term dependencies. To address this limitation, the Long Short-Term Memory (LSTM) architecture was developed. The component of LSTM model are:

LSTM Gates and Cells

The Long Short-Term Memory (LSTM) network was introduced to overcome the difficulty of learning long-range dependencies in sequential data[59]. In particular, it mitigates the vanishing gradient problem that commonly appears in standard RNNs. Since its introduction, LSTM has significantly improved the performance of recurrent models and has enabled their practical success in many applications.

Unlike standard RNN units, LSTM units—often called cells—are capable of remembering information over arbitrarily long time intervals. This capability is achieved through two key components, namely an internal memory known as the cell state and a set of gates that regulate the flow of information, as illustrated in Figure 4.8.

- An internal memory known as the **cell state**.
- A set of **gates** that regulate the flow of information.

The cell state acts as a memory pathway that can preserve, update, or discard information. A typical LSTM cell contains three gates that control this process:

- **Forget gate:** decides what information should be removed from the cell state.
- **Input gate:** determines what new information should be stored.
- **Output gate:** controls how much information is passed to the hidden state.

These gates do not make binary decisions. Instead, they use an activation function σ (commonly sigmoid) that outputs values between 0 and 1, allowing partial information flow.

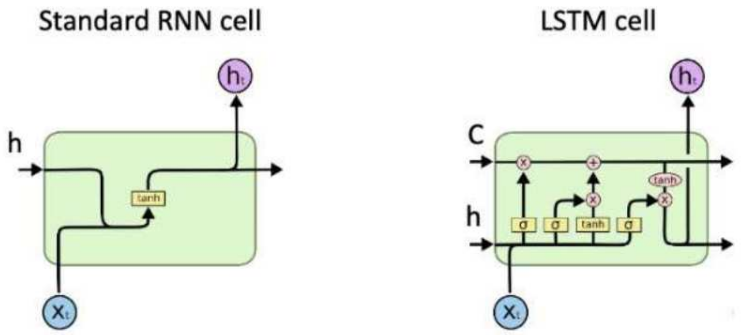


Figure 4.8: Comparison between a standard RNN cell and an LSTM cell. The LSTM introduces a cell state pathway and gating mechanisms that enable long-term information retention[53].

Intermediate Vectors

Instead of directly updating the hidden state, LSTM introduces intermediate control vectors that regulate memory updates. For simplicity, considering a one-dimensional case, the intermediate variables are defined as:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{4.26}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{4.27}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{4.28}$$

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c). \tag{4.29}$$

Here:

- $f_t \rightarrow$ Forget gate
- $i_t \rightarrow$ Input gate
- $o_t \rightarrow$ Output gate
- $\tilde{C}_t \rightarrow$ Candidate cell input

These variables determine how information flows through the memory cell.

Cell State Update

The updated cell state is computed as

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t. \tag{4.30}$$

The first term controls how much past memory is retained, while the second term determines how much new information is added. If $f_t = 0$, the previous memory is

completely forgotten; if $i_t = 1$, the new information is fully stored. The interaction between consecutive LSTM cells and the roles of the forget, input, and output gates in controlling the memory flow are illustrated in Figure 4.9.

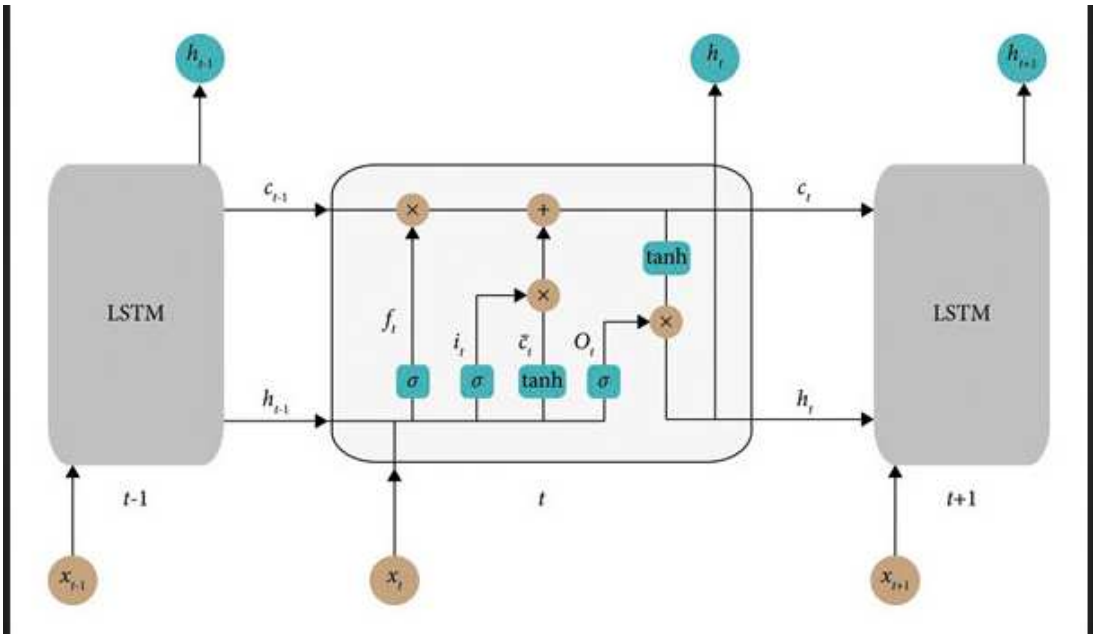


Figure 4.9: Illustration of three LSTM cells showing forget, input and output gates.

Hidden State Update

The hidden state is obtained from the updated cell state as

$$h_t = o_t \cdot \tanh(C_t). \tag{4.31}$$

Here, the output gate regulates how much of the internal memory contributes to the hidden representation. A step-by-step view of the internal LSTM operations, including the forget gate, input gate, cell-state update, and output gate, is shown in Figure 4.10.

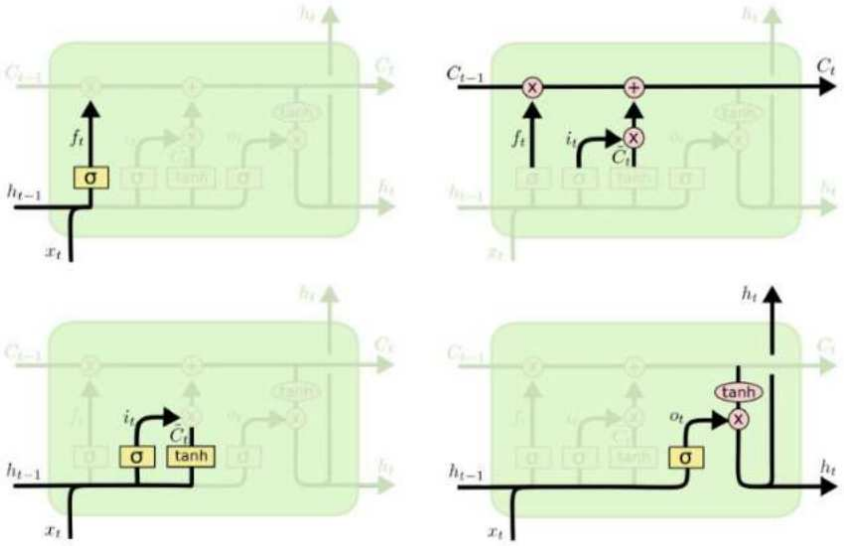


Figure 4.10: Step-by-step internal operations of an LSTM cell showing forget, input, cell update, and output gates.

Workflow of the Stacked LSTM–Flatten–Dense Architecture

Figure 4.11 shows the complete workflow of the LSTM-based model used to process temporal (sequential) inputs. The main idea is that the model reads the input step by step, learns useful patterns over time, and then converts the learned sequence information into a single output prediction.

First, the input is a time-ordered sequence, written as $\{X_0, X_1, X_2, \ldots, X_n\}$. Each X_t represents the information at time step t. The LSTM layer processes this sequence one step at a time and updates its internal memory so that important information can be carried forward to later time steps.

Second, the diagram shows multiple LSTM blocks placed one after another. This is called a **stacked LSTM**. The first LSTM layer learns basic temporal features from the input sequence. Then, the next LSTM layer receives these features and learns higher-level temporal patterns. This stacking helps the model capture more complex dependencies in the sequence compared to a single LSTM layer.

Third, after the last LSTM layer, the output is still arranged as a sequence across time steps. However, the next layers (fully connected layers) expect a single vector as input. For this reason, a **Flatten** operation is applied. The Flatten layer reshapes the sequence output into one long one-dimensional feature vector, without changing the values. In simple words, it just convert the learned features from multi dimensional i into a single dimensional list that can be used by dense layers.

Finally, this flattened vector is passed to the **fully connected (dense) layers**. These layers combine all extracted features and produce the final prediction at the output node. In classification tasks, this output can represent class probabilities, while in regression

tasks it can represent a continuous value. The structure of the LSTM model for the detection of primary and secondary electrons in the waveform are shown in chapter 5 with more details

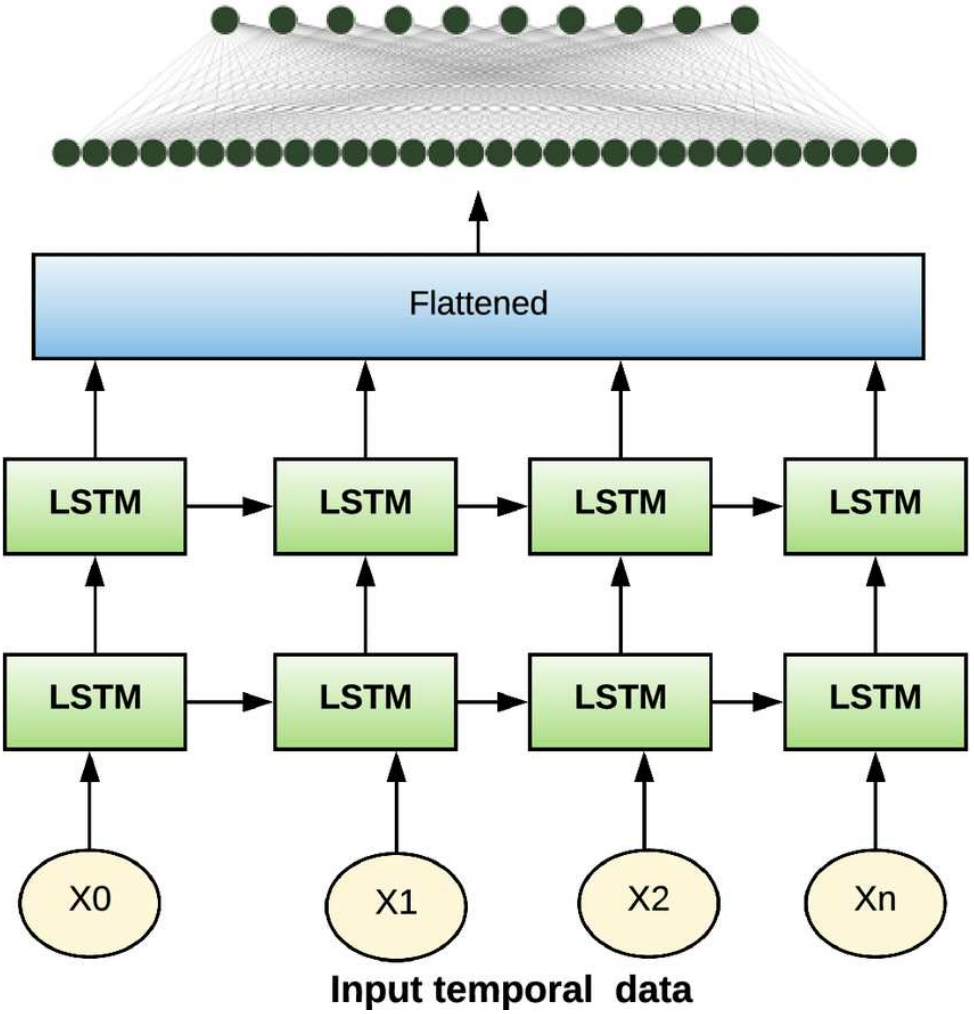


Figure 4.11: Workflow of a stacked LSTM model for temporal data. The input sequence $(X_0, X_1, \ldots, X_n)$ is processed through multiple LSTM layers, then converted into a single feature vector using a Flatten layer, and finally mapped to the output using fully connected layers [59].

4.2.4 Convolutional Neural Networks (CNNs)

Artificial neural networks (ANNs) are computing models inspired by how biological brains process information. A CNN is a special type of ANN that is designed for image-like data, where the input can be seen as a grid of values [60]. Because many detector and digitization outputs can also be organized as structured grids or image-like patterns, CNNs are widely used to learn useful spatial features automatically .

CNNs have shown strong performance in many visual tasks such as classification, clusterization, segmentation, retrieval, and object detection [1]. This success comes from using convolution operations inside the network instead of only using general matrix

multiplication. The key idea is similar to the human visual system: each unit focuses on a limited local region (a receptive field), and the network builds complex understanding by combining many local patterns. With this design, CNNs learn a hierarchy of features during training through backpropagation, using building blocks such as convolution layers, pooling layers, and fully connected layers, as illustrated in Figure 4.12. [2].

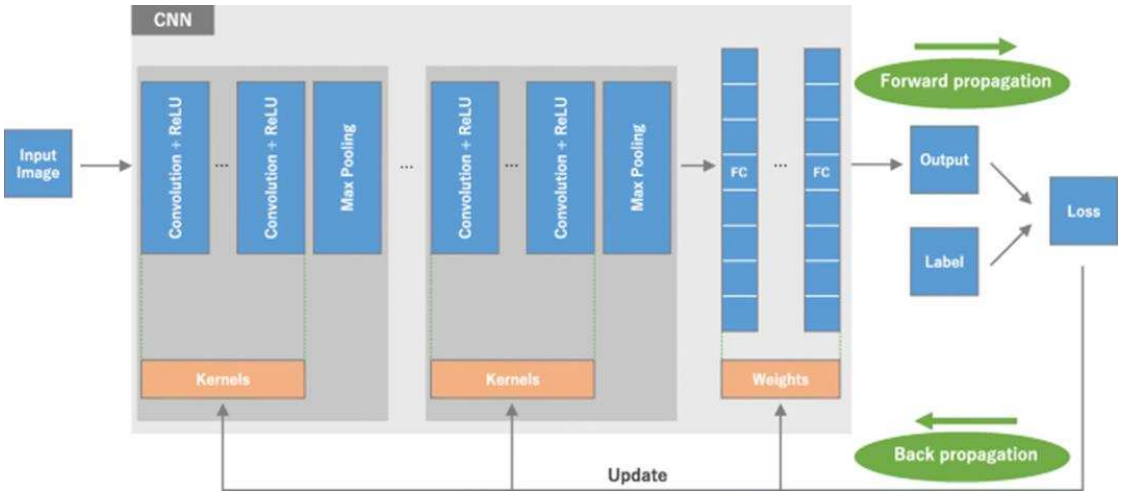


Figure 4.12: Conceptual overview of CNN motivation: CNNs extend standard ANNs by using local receptive fields and convolution operations that are well-suited for grid-like (image-like) inputs [1].

Convolutional layer

A CNN is commonly described using major layers such as the convolution layer, an activation function (for example ReLU), a pooling layer, and a fully connected layer. In practice, the network may contain many repeated convolution blocks, and each block learns different features at different resolutions. CNN layers are usually arranged so that early layers detect simple patterns first (like edges or short curves), while deeper layers combine these simple patterns to form more complex structures . Another important difference from regular ANNs is connectivity: in a standard ANN, neurons are often connected to all neurons in the previous layer, but in a CNN each neuron connects only to a small local region of the previous layer. This local connectivity reduces computation and helps the model focus on meaningful spatial features.

The convolution layer is the core part of a CNN because it performs feature extraction directly from the input. It applies a small filter (also called a kernel) to a local part of the input and produces one output value for that local region. The filter then shifts across the input step by step, repeating the same operation until the full input is processed, as illustrated in Figure 4.13. This scanning process means each output value corresponds to a specific receptive field, and the full set of outputs forms a feature map that represents the input in a learned way. Because the same filter weights are reused at every position, the convolution layer can learn patterns that appear at different locations while keeping

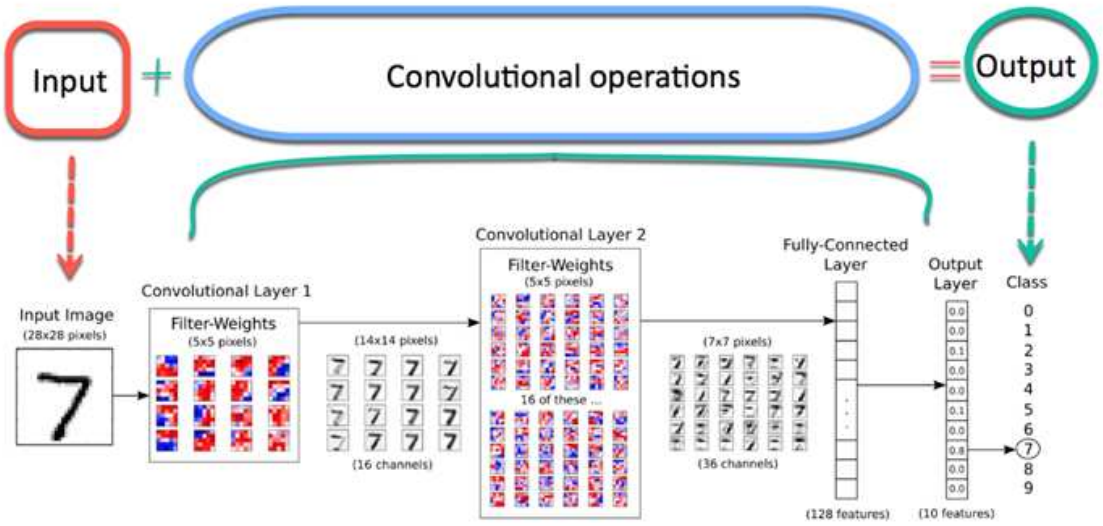


Figure 4.13: Illustration of the convolution concept: a small kernel scans the input using local receptive fields and produces feature maps that encode learned spatial patterns [1].

the number of parameters manageable.

In many cases, the output of the convolution layer is stored as a 3D representation (height, width, and depth), where the depth corresponds to the number of learned feature maps. For example, if the input has three channels (such as RGB), the filter is small in height and width but extends through all channels so that each output mixes information from all channels in that local region.

Rectified Linear Unit (ReLU) Layer

The Rectified Linear Unit (ReLU) layer is one of the most widely used activation layers in Convolutional Neural Networks (CNNs). After the convolution operation extracts features from the input data, the ReLU layer introduces non-linearity into the network. This non-linear transformation is essential because real detector signals and physical patterns are inherently non-linear in nature.

In simple terms, the ReLU activation function applies a threshold operation to the input signal. If the input value is positive, it is passed forward unchanged. If the input value is negative, it is converted to zero. Mathematically, this operation can be expressed as:

$$\phi(x) = \max(0, x)$$

This behaviour allows the network to retain meaningful detector-related information while suppressing non-physical or noise-like negative responses.

One important advantage of ReLU is that it does not saturate in the positive region. Traditional activation functions such as sigmoid and tanh compress the input into a limited

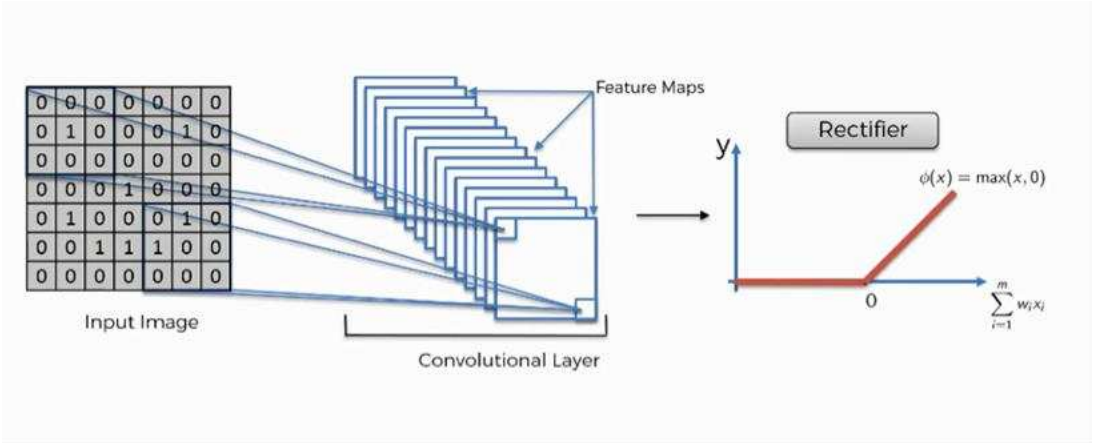


Figure 4.14: Illustration of convolution followed by ReLU activation. The convolutional layer produces feature maps which are then passed through the rectifier function $\phi(x) = \max(0, x)$ to introduce non-linearity.

range between 0 and 1 or between 1 and 1. This compression can slow down learning because gradients become very small. In contrast, ReLU maintains a gradient of unity for positive activations, which improves convergence speed and training stability.

Furthermore, ReLU does not change the dimensionality of the feature maps. It only modifies the activation values. This property is particularly useful in deep CNN architectures where preserving spatial information is critical for downstream processing such as clustering and feature localization.

As shown in Fig. 4.14, the convolutional layer first generates multiple feature maps from the input image. These feature maps are then passed through the rectifier, where negative activations are suppressed while positive responses are preserved.

Another important characteristic of ReLU is its computational simplicity. Since the function only involves a comparison with zero, it is computationally efficient and well suited for large-scale detector data processing [3]. This efficiency becomes crucial when processing high-granularity detector signals, where millions of activation values must be evaluated during training.

In practical CNN pipelines, ReLU activation is typically applied before pooling operations . This ensures that only the most relevant and activated features are propagated to the next stage of the network.

Despite its advantages, ReLU has certain limitations. Since all negative inputs are mapped to zero, some neurons may stop responding during training. This phenomenon is commonly referred to as the “dead neuron” problem. However, in most deep learning applications, including detector signal analysis, the benefits of faster convergence and sparse activation outweigh this limitation.

Fig. 4.15 compares ReLU with the logistic sigmoid activation function. The sigmoid curve saturates at higher values, which reduces gradient magnitude and slows learning. In contrast, ReLU maintains a strong gradient in the positive region, making it more suitable

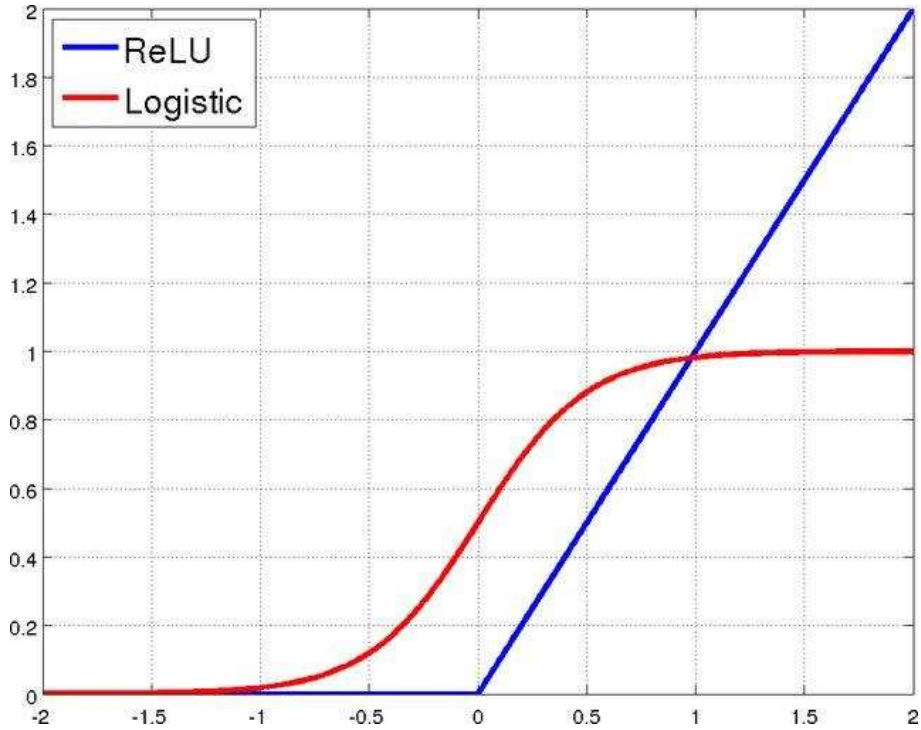


Figure 4.15: Comparison of ReLU and logistic sigmoid activation functions. ReLU maintains linear growth in the positive region, whereas sigmoid saturates, leading to slower gradient propagation.

for deep CNN training [2].

Pooling Layer

After the convolution and activation stages, the next important component of the Convolutional Neural Network (CNN) is the pooling layer. The pooling layer is mainly responsible for reducing the size of the activation maps, and therefore it is often referred to as a down-sampling layer. An example of this dimensional reduction through pooling is shown in Fig. 4.16. The reduction in spatial size helps to simplify the feature representation while preserving the most important information extracted from the input data.

In general, the pooling operation is performed by applying a small filter over the feature map using a defined stride. During this process, the feature map is divided into multiple small regions, and a statistical value is computed from each region. Commonly, either the maximum value or the average value is selected from each region to form the pooled output. Because of this operation, dominant features are preserved while less significant information is discarded, which makes the representation more compact and computationally efficient .

From a deep learning perspective, pooling also plays an important role in reducing overfitting. Since the dimensionality of the feature maps decreases, the number of trainable parameters in the subsequent layers is reduced. This leads to lower computational cost

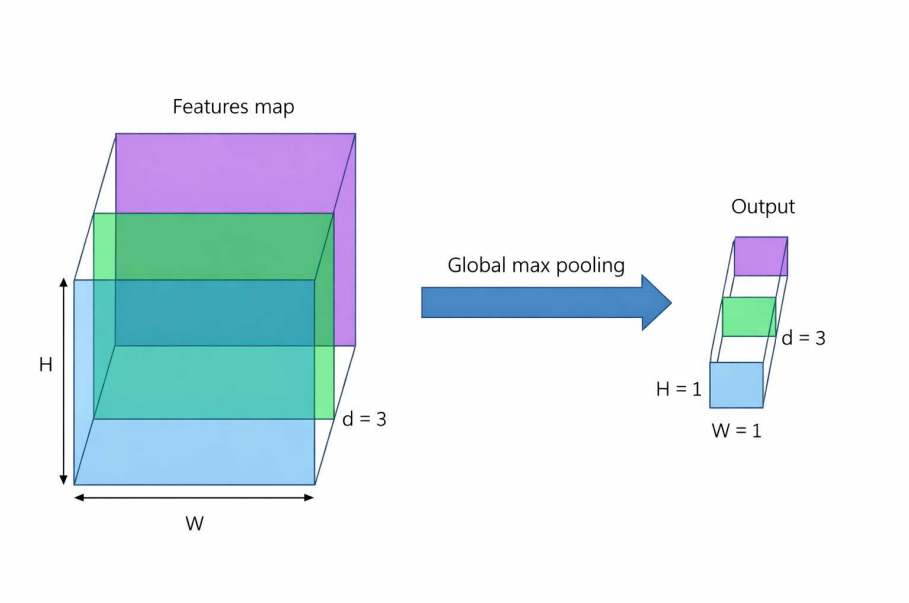


Figure 4.16: Example of feature maps after convolution and their dimensional reduction through pooling operations. ReLU activation is applied before pooling to enhance dominant spatial features

and improved generalization performance of the model.

Another important advantage of pooling is its ability to provide robustness against small spatial variations in the input. Brownlee (2019) explained that pooling introduces a property known as *local translation invariance*, meaning that small shifts in the position of features in the input image do not significantly affect the pooled output [3]. This property is particularly useful in image-based pattern recognition tasks where the exact position of features may vary slightly.

Technically, the pooling operation is applied independently to each feature map. A pooling filter, commonly of size 2×2, slides across the feature map with a stride of 2. This reduces the spatial resolution by approximately a factor of two, as illustrated in Figure 4.17. As a result, the output is a lower-resolution representation that still retains the most important structural information of the original input signal.

It is also important to note that pooling is a non-learnable operation. Unlike convolutional filters, pooling does not contain trainable parameters. Instead, it performs a predefined mathematical operation. The two most widely used pooling functions are:

- **Max Pooling** — selects the maximum value from each pooling region.
- **Average Pooling** — computes the average value of each pooling region.

Max pooling is the most commonly used technique because it preserves the strongest activations, which often correspond to the most relevant features for classification tasks.

Through this hierarchical process of convolution, activation, and pooling, CNN models progressively transform raw input data into compact and informative feature representa-

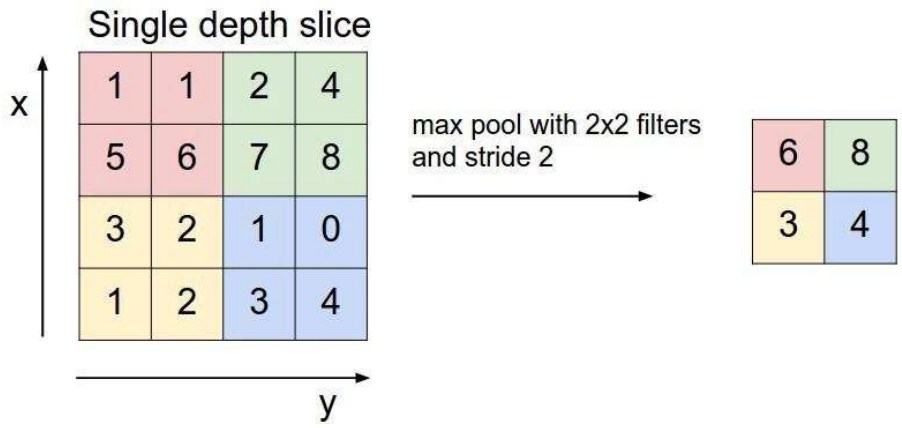


Figure 4.17: Illustration of the pooling operation. A feature map is downsampled using a pooling filter (e.g., 2×2) with a stride of 2, producing a reduced representation while preserving dominant features.

tions. These representations are later used by fully connected layers for final prediction or classification tasks.

Fully Connected Layer (FC)

After the convolution, activation, and pooling stages, the final component of the Convolutional Neural Network (CNN) architecture is the fully connected layer, commonly referred to as the dense layer. This layer plays the role of the final decision-making unit of the network.

A fully connected layer consists of neurons, weights, and biases, where each neuron in this layer is connected to every neuron in the previous layer. Because of this dense connectivity, the layer is capable of learning complex relationships between extracted features and the target output classes.

In CNN architectures, the input to the fully connected layer is typically the output of the last convolutional or pooling layer. At this stage, the multi-dimensional feature maps are first flattened into a one-dimensional vector and then passed to the fully connected layer, as illustrated in Figure 4.18. This transformation converts the spatial feature representation into a format suitable for classification or regression tasks[1].

Mathematically, this flattened vector represents high-level features learned from the input data. These features may correspond to shapes, edges, energy deposition patterns, or spatial charge distributions, depending on the application domain. The fully connected layer processes these features and maps them to an N-dimensional output vector, where N represents the total number of output classes [2].

For example, in a general image classification problem, the network may learn high-level features such as object parts (e.g., legs, head, or tail in animal images). The fully connected layer combines these learned features and produces the final classification output. In CNN models, this layer is therefore often referred to as the *classification layer* or *completion*

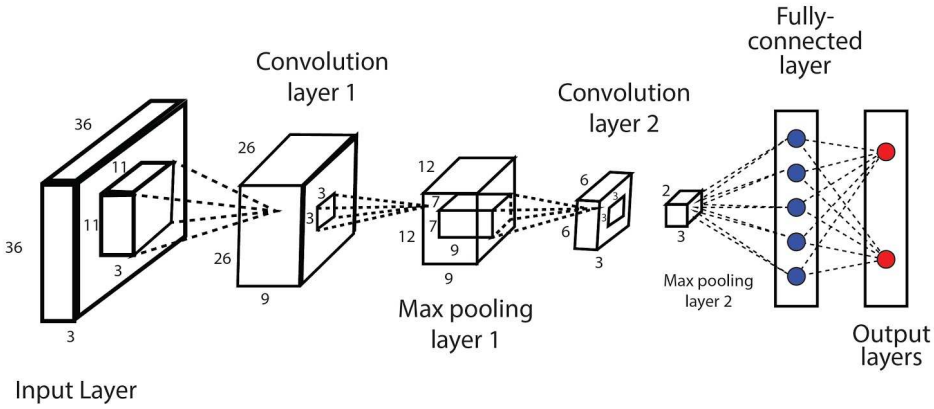


Figure 4.18: Illustration of the structure of the CNN model with an input layer, convolutional layers, pooling layers, and a fully connected layer, which is further connected to two dense neurons for final classification

layer [1].

Technically, the output feature maps from the final convolution or pooling stage are transformed into a one-dimensional array and then passed through one or more fully connected layers [3]. Each connection has a learnable weight, allowing the network to assign importance to different extracted features. The final output layer usually contains the same number of neurons as the number of prediction classes and is followed by a nonlinear activation function such as ReLU or Softmax for decision making [3].

It is important to emphasize that the complete sepecific structure of CNN model is showed in chapter 6 which we used to estimate the number of primary clusters based on the total number of electrons as regression task.

4.3 Hyper-parameters of Neural Network Model

Neural network models are controlled not only by their internal weights and biases, but also by a set of external configuration variables known as hyper-parameters. These hyper-parameters define how the model learns from the data, how fast it converges, and how well it generalizes to unseen samples. In deep learning applications used in experimental high-energy physics, including detector signal processing and cluster reconstruction, the careful selection of hyper-parameters plays a crucial role in achieving stable and reliable performance.

In this work, hyper-parameters are particularly important because the neural network model is trained on detector-related inputs for precise feature extraction. Since the training process is iterative and computationally demanding, proper tuning of these parameters ensures efficient learning, avoids overfitting, and improves prediction accuracy. The most relevant hyper-parameters considered in this study include the number of epochs, batch size, the choice of objective (loss) function and Regularization. Each of these is discussed in detail in the following subsections and other(objective and optimzer) are discussed in the previous section.

4.3.1 Epoch and Batch

Neural network training is performed in an iterative manner, where the available training dataset is used repeatedly and as efficiently as possible. Two key terms are used to describe this learning procedure: *epoch* and *batch*.

Epoch. One epoch represents one complete pass of the entire training dataset through the neural network. In other words, during one epoch, every training sample is used once for updating the model parameters. However, a single pass is usually not sufficient for the model to learn complex patterns. Therefore, training is carried out over multiple epochs, allowing the network to progressively refine its internal weights through repeated exposure to the data.

Batch and Mini-batch. Instead of processing the entire dataset at once, the training data are divided into smaller subsets known as batches or mini-batches. The network parameters are then updated iteratively using these smaller data groups. This approach significantly reduces computational cost and improves memory efficiency.

Using the full dataset in every update step would be highly time-consuming and inefficient. Therefore, randomly selected mini-batches are used as a practical compromise between computational speed and learning stability. Batch sizes are often selected in powers of two, such as $k = 32$, for optimized hardware performance.

The choice of batch size depends on the complexity of the problem and directly affects model performance. Larger batch sizes increase computational cost, while very small batches may reduce generalization capability. Training progress is typically evaluated in terms of epochs, while the total number of update iterations depends on the ratio between the dataset size d and the batch size k.

.

4.3.2 Learning Rate

The learning rate is one of the most important hyper-parameters in a neural network model. It controls how fast or how slow the network learns during the training process. In gradient-descent optimization, the model parameters such as weights W and biases b are updated by computing the partial derivative of the objective function $\partial\mathcal{L}/\partial W$. This derivative indicates the direction in which the parameter should move in order to reduce the loss.

To update the parameter from iteration t to $t+1$, the gradient is multiplied by a scaling factor α, which is called the learning rate. The update rule for the weight parameter is written as:

$$W_{t+1} = W_t - \alpha\, E\left[\frac{\partial\mathcal{L}}{\partial W}\right]_t \tag{4.32}$$

Similarly, the bias parameter is updated as:

$$b_{t+1} = b_t - \alpha\, E\left[\frac{\partial\mathcal{L}}{\partial b}\right]_t \tag{4.33}$$

These updates are applied iteratively so that the model gradually approaches an optimal solution where the loss function becomes minimal, as illustrated in Figure 4.19.

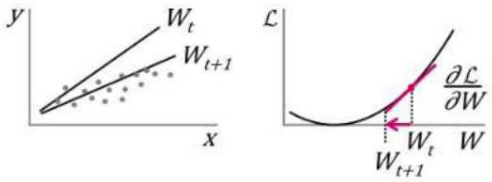


Figure 4.19: Illustration of parameter update using gradient descent, where model parameters move along the loss surface according to the gradient direction.

The choice of learning rate has a strong impact on the training behavior. If the learning rate is too small, the training process becomes very slow and may fail to reach the optimal minimum within a reasonable time. On the other hand, if the learning rate is too large, the optimization may overshoot the minimum and become unstable, as illustrated in Figure 4.20.

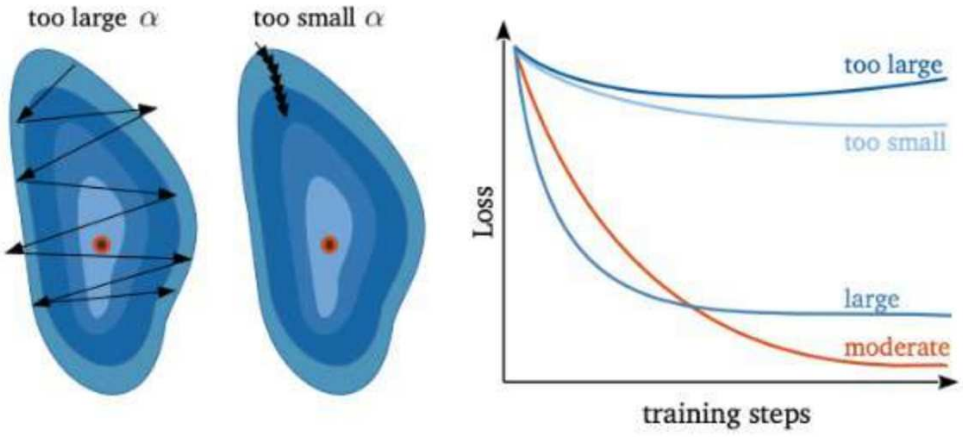


Figure 4.20: Effect of different learning rates on convergence. Very small learning rates lead to slow training, while very large values may prevent convergence.

Therefore, selecting an appropriate learning rate is essential. In practice, learning rates are often chosen within the range:

$$\alpha = 10^{-5} \text{ to } 10^{-2} \tag{4.34}$$

4.3.3 Regularization

During neural network training, the main goal is to reach a good local minimum in a very high-dimensional parameter space as efficiently as possible. However, this optimization process is not always straightforward. Sometimes the training procedure can get trapped in an undesirable local minimum, or the model may start learning patterns that are only specific to the training dataset and not generally valid. To address these issues, several regularization techniques are introduced during training.

According to Ian Goodfellow, regularization can be defined as:

> *Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.*

This definition highlights the main purpose of regularization. The objective is not to improve performance only on the training data, but rather to ensure that the trained model can generalize well to unseen data. Without regularization, the network may learn very specific features present only in the training dataset. Such behavior leads to **overtraining (overfitting)**, where the model performs well on training data but poorly on validation or test data. To monitor this effect, the objective function is evaluated not only on training data but also on validation data.

Early Stopping

One of the simplest and most effective regularization techniques is **early stopping**. During training, the loss (objective function) is monitored for both training and validation datasets. Initially, both losses decrease as the network learns meaningful patterns. However, a critical point is reached when the validation loss starts increasing while the training loss continues to decrease.

This behavior indicates the beginning of overfitting. At this stage, the model starts memorizing the training data instead of learning general features. Therefore, training is stopped at the point where the validation loss is minimal. In this way, early stopping prevents the model from overtraining while preserving its generalization capability.

Dropout

Another widely used regularization method is **dropout**. The idea of dropout is simple but very powerful. During training, a fraction of neurons in the network is randomly deactivated (dropped out). This means that these neurons do not participate in forward propagation or backpropagation for that training step.

Typical dropout rates range between 0.2 and 0.5. By randomly removing neurons, the network is forced to learn multiple independent feature representations instead of relying on specific neuron connections. This improves the robustness and stability of the model.

It is important to note that dropout is applied only during training. During inference (testing or prediction), all neurons are active. To maintain the magnitude of outputs, the weights of active neurons are scaled inversely proportional to the dropout rate during training.

The effect of dropout regularization is illustrated in Fig. 4.21. The left side shows a fully connected network, while the right side demonstrates randomly dropped neurons during training.

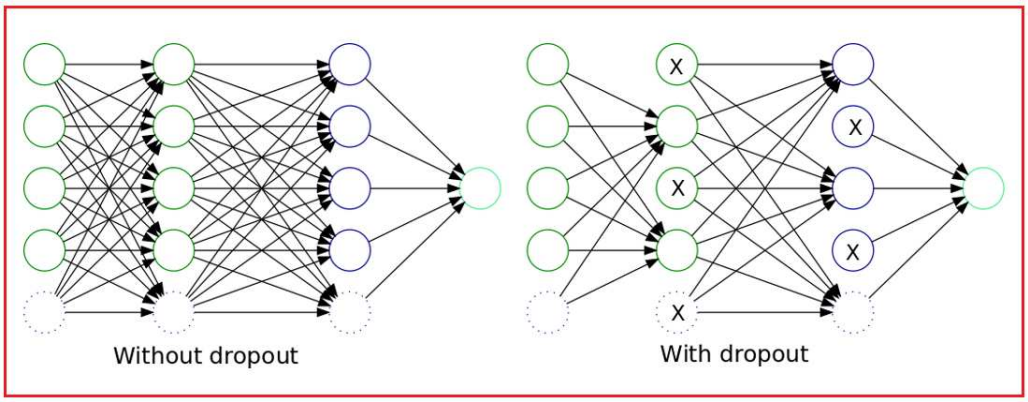


Figure 4.21: Regularization using dropout. During training, random neurons are deactivated to prevent overfitting and improve generalization.

In summary, regularization techniques such as early stopping and dropout play a crucial role in improving the generalization performance of neural networks. They ensure that the trained model captures meaningful physical patterns rather than noise or dataset-specific fluctuations, which is particularly important in experimental high-energy physics analyses.

4.3.4 Activation Functions

Another important hyper-parameter in a neural network model is the choice of the activation function. The activation function defines how the weighted input signal of a neuron is transformed before it is passed to the next layer. This transformation introduces non-linearity into the network, which is essential for learning complex patterns from detector data. The choice of activation function depends on the modeling objective and the network architecture.

Several commonly used activation functions are shown in Fig. 4.22. Among them, the Rectified Linear Unit (ReLU) is widely used and is often considered a good first choice for deep learning models [1]. The ReLU function outputs zero for negative input values and passes positive values directly without modification. This simple behaviour makes ReLU computationally efficient and helps the network learn faster.

An extension of ReLU is the Leaky ReLU function, where small negative values are also allowed to pass through the network instead of being set strictly to zero [26]. This helps to reduce the problem of inactive neurons during training.

Earlier neural network models frequently used the sigmoid activation function due to its similarity to biological neurons. The sigmoid transforms input values into a probability range between 0 and 1. However, it is rarely used in intermediate layers of deep networks today because its output is always positive, which can disturb gradient updates in deep architectures. In addition, sigmoid and hyperbolic tangent (tanh) functions suffer from saturation regions where gradients become very small, leading to the well-known vanishing gradient problem. This slows down or even stops the training process in deep networks.

Because of this limitation, sigmoid and tanh functions are now mainly used in the final output layers where the prediction range must be constrained, such as between [0,1] or [-1,1]. More advanced activation functions such as ELU and SELU have also been introduced to improve gradient flow and training stability in deeper models.

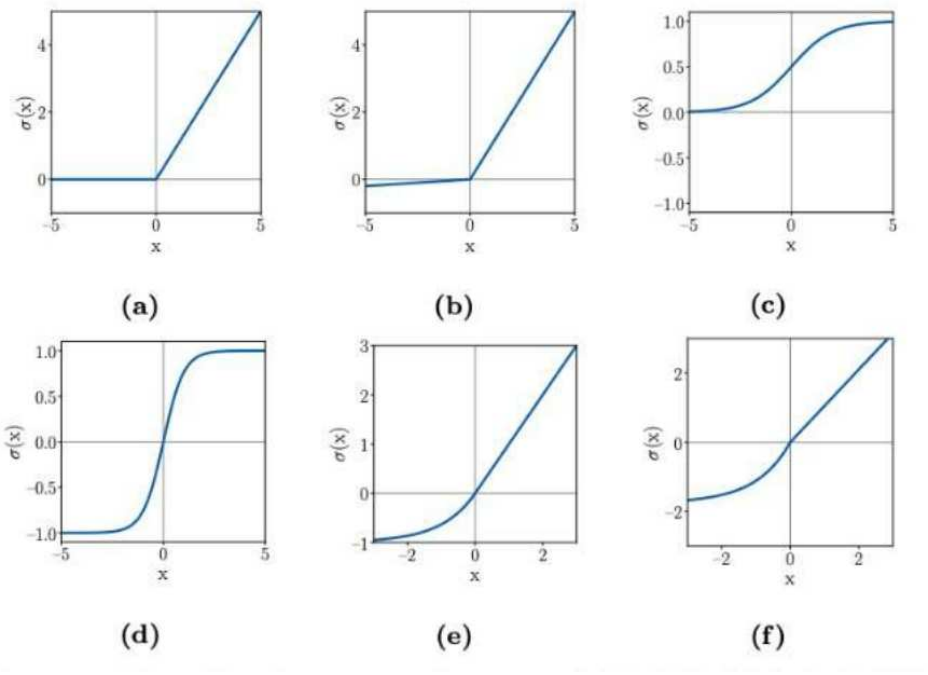


Figure 4.22: Examples of commonly used activation functions in neural networks: (a) ReLU, (b) Leaky ReLU, (c) Sigmoid, (d) Hyperbolic tangent (tanh), (e) ELU, and (f) SELU. These functions introduce non-linearity and control gradient propagation during training.

Chapter 5

ML Based Algorithm for Cluster Counting Technique

This chapter focuses mainly on a machine-learning-based cluster-counting algorithm for cluster-counting techniques, in comparison with the traditional algorithm. The ML algorithm has two main steps. In the first step, an LSTM model is used to detect peaks (primary and secondary electrons) in the digitized waveform against noise. This is a classification task, and this algorithm is referred to as the peak-finding algorithm. In the second step, a CNN model estimates the number of primary ionization clusters, N_{cls}, based on the peaks detected in the first step. This is a regression task, and this algorithm is referred to as clusterization.

Usually, the traditional dE/dx method measures the total energy deposited by primary and secondary electrons in the gaseous detector. This method has large fluctuations because the energy loss follows a Landau distribution with a long tail. In contrast, the cluster-counting technique, dN/dx, counts the number of primary clusters per unit length and follows a Poisson-like behavior with smaller fluctuations. These smaller fluctuations provide better separation between particle species, as explained in more detail in the next chapter.

5.1 Passage of Particles Through Matter

Charged particles interact with matter through two main mechanisms: electromagnetic and nuclear interactions. As a fast charged particle traverses a medium, it feels the electromagnetic field of atomic electrons and nuclei along its path. This produces three important effects [61]:

- **Multiple scattering**: This occurs when a particle repeatedly changes its direction by small angles due to interactions with other particles in the matter, causing its path to bend or deviate.

- **Bremsstrahlung**: This is the process where a particle emits photons due to the electromagnetic field of atoms (nuclei and electrons). The effect of this process becomes more significant when the particle's energy is much higher than its mass squared. For electrons, it can become dominant at energies around the MeV scale. For heavier particles like muons and pions, it only becomes important at much higher energies, typically hundreds of GeV.
- **Inelastic collisions with atomic electrons**: In this process, the particle transfers energy to the surrounding medium, leading to the excitation or ionization of atoms. This is the primary source of the measurable signal in gas detectors.

5.1.1 Energy Loss Due to Electromagnetic Interactions

When a charged particle enters a material, it interacts multiple times with atomic electrons. In each interaction, the atomic electron can either be excited to a higher energy level or completely removed from the atom (ionization). Ionization creates an ion pair, which consists of a free electron and a positive ion. In many detectors, an electric field is applied to prevent recombination of the ion pair, allowing the charges to be collected and converted into an electrical signal.

Some interactions transfer enough energy to produce energetic secondary electrons, known as *delta rays*. Delta rays have a much shorter range than the primary particle, so their ionization remains close to the main track. As a result, the ionization is not evenly distributed along the path but instead appears as clusters of ion pairs along the trajectory.

The mean energy loss per unit length, normalized to the material density, is given by the Bethe–Bloch formula [62]:

$$-\frac{dE}{dx} = K\, z^2\, \frac{Z}{A}\, \rho\, \frac{1}{\beta^2} \left[\frac{1}{2} \ln\left(\frac{2 m_e c^2 \beta^2 \gamma^2 T_{\max}}{I^2} \right) - \beta^2 - \frac{\delta}{2} - \frac{C}{Z} \right], \tag{5.1}$$

where Z and A are the atomic number and mass of the absorber, I is the mean excitation potential, z is the particle's charge in units of e, and βc and γ represent the particle's velocity and Lorentz factor. The terms δ and C account for density and shell effects. The quantity $T_{\max}$ represents the maximum kinetic energy that can be transferred to a free electron in a single collision:

$$T_{\max} = \frac{2 m_e c^2 \beta^2 \gamma^2}{1 + 2\gamma \frac{m_e}{M} + \left(\frac{m_e}{M}\right)^2}, \tag{5.2}$$

where M is the mass of the incident particle.

Equation (5.1) represents the **mass stopping power** (commonly expressed in units of $\mathrm{MeV\,g^{-1}\,cm^2}$). The **linear stopping power** is obtained by multiplying the mass stopping power by the material density, resulting in $\rho\, dE/dx$ (MeV/cm). The Bethe–Bloch formula

is valid over a wide kinematic range, approximately 0.1 $\beta\gamma$ 1000.

At low $\beta\gamma$, the stopping power decreases as the particle's velocity increases, reaching a broad minimum known as the **minimum ionization** region. At higher $\beta\gamma$, the logarithmic term increases, causing the stopping power to rise again.

At high energies, the density-effect correction, $\delta(\beta\gamma)$, reduces this rise because the medium becomes polarized, weakening the effective electric field at large distances. At extremely high energies, radiative processes become important, depending on both the particle type and the material.

Figure 5.1 shows the typical stopping-power curve for positive muons in copper as a function of $\beta\gamma = p/(Mc)$. The plot highlights the low-$\beta\gamma$ region (where nuclear effects can matter), the minimum-ionization region, and the high-energy region where radiative losses grow.

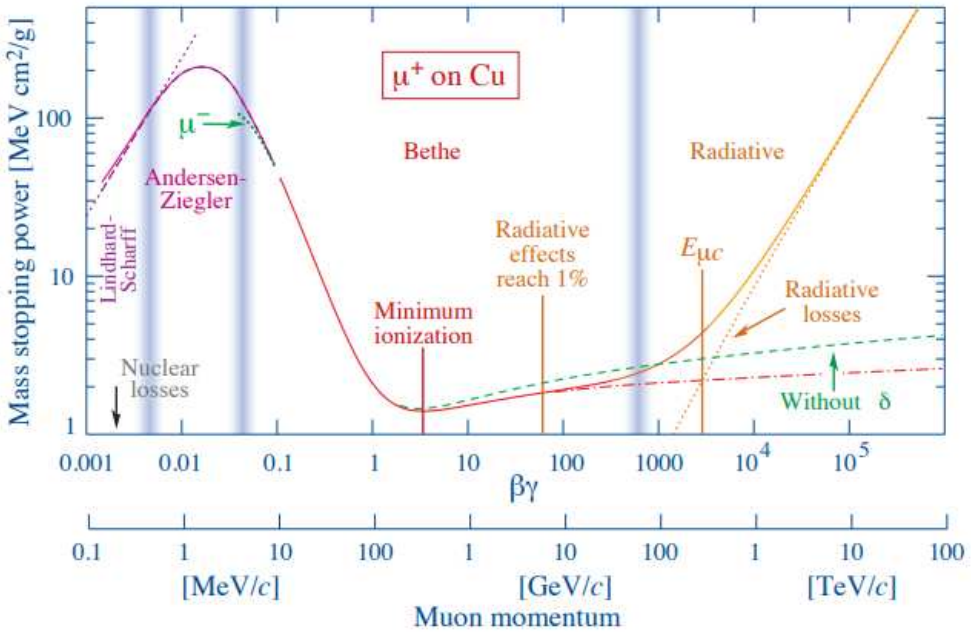


Figure 5.1: Mass stopping power for positive muons in copper as a function of $\beta\gamma$, spanning nine orders of magnitude in momentum. The short dotted lines labeled μ illustrate the *Barkas effect*, which shows the dependence of stopping power on the projectile charge at very low energies. [63].

5.2 Traditional Algorithm (dE/dx)

Particle identification is a key task in high-energy physics experiments. Particles can be identified in two general ways.

The first way uses the detector signature. A particle leaves a characteristic response depending on how it interacts with detector materials. For example, if a particle produces

signals only in an electromagnetic calorimeter, it is consistent with a photon.

The second way is based on particle properties such as mass and charge. The momentum is measured from the curvature of the track in a magnetic field, while the velocity can be inferred from dedicated measurements. One important method to access the velocity is ionization energy loss in matter. The charge sign is obtained from the direction of track curvature.

In gaseous detectors, a charged particle loses energy mainly through inelastic collisions with atomic electrons. In each collision, an atom can be excited or ionized, and the particle loses a small part of its kinetic energy. The observable used in the traditional method is the energy loss per unit length:

$$\frac{dE}{dx}. \tag{5.3}$$

Experimentally, gaseous counters provide a pulse height that is proportional to the number of electrons produced by ionization along the track length inside the detector. Therefore, the pulse height is proportional to the deposited energy, and it can be used to estimate dE/dx.

The limitation of this method is the significant uncertainty in the total deposited energy. Even in the most favorable momentum range (the relativistic rise), the difference between the mean energy-loss curves of different particles is often smaller than the spread of the measured distributions. This limits the ability to effectively distinguish between particles.

5.2.1 Landau Distribution

The deposited energy varies because the ionization process is random. Most collisions transfer only a small amount of energy, but some rare collisions transfer much larger amounts. These rare collisions cause a long tail in the distribution, extending toward higher energy values.

As a result, the deposited energy distribution follows a Landau function. The Landau distribution is asymmetric and has a long high-energy tail (the Landau tail). Due to this tail, the mean value is shifted upward and becomes much higher than the most probable value.

Figure 5.2 shows a typical example of the deposited energy distribution along a track. The long tail explains why the mean is not a reliable measure for energy deposition.

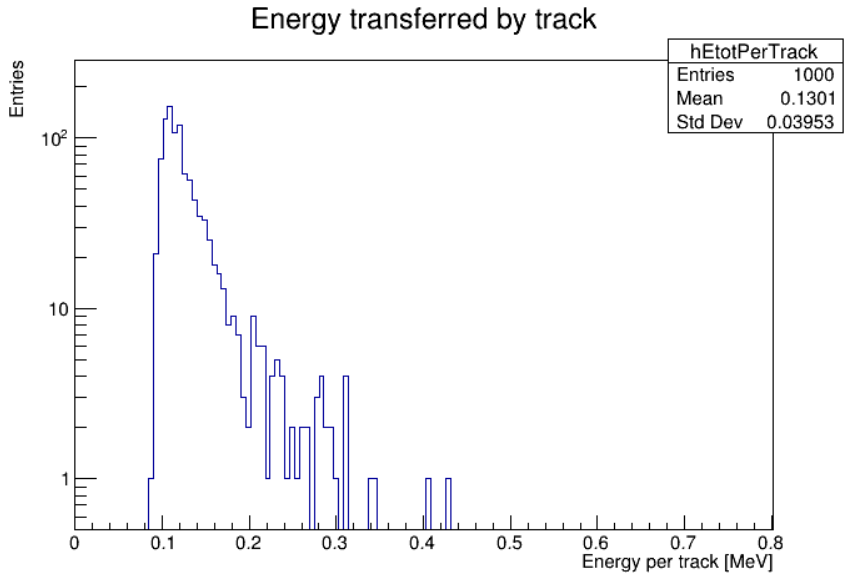


Figure 5.2: Example of the deposited energy distribution per track. The distribution is asymmetric with a long high-energy tail, consistent with Landau fluctuations.

Because the mean is heavily influenced by rare large energy transfers, it is not a good estimator for particle identification. A common solution is to use a *truncated mean*. In this method, the largest energy deposits are removed from the calculation. Typically, about 40% to 80% of the data is kept.

5.3 Overview of Cluster Counting Techniques

As we mentioned earlier, particle identification (PID) is important in high-energy physics, especially flavor physics and jet tagging[64]. PID helps reduce combinatorial background and helps separate final states that look similar. Future experiments need detector methods with better PID performance than the current generation [65].

A drift chamber can provide tracking and also PID with almost no extra detector cost. In a drift chamber, PID comes from how a charged particle ionizes the gas along its path. A common PID method is to measure the average energy loss per unit length, dE/dx [66]. In a drift chamber cell, the particle ionizes the gas and produces electrons that form the detector signal. The first ionizations are called *primary ionization*, and the number of these primary ionizations follows a Poisson process. Some electrons also create *secondary ionizations*. These secondary ionizations lead to a Landau-type dE/dx distribution with a long tail and large fluctuations [67]. These large fluctuations limit the dE/dx resolution and weaken PID .

Cluster counting uses a different idea. It measures the average number of primary ionization clusters per unit length (dN/dx) directly from the digitized waveform [68]. This method focuses on counting primary clusters instead of measuring the total deposited

energy. In this way, the impact of secondary ionizations becomes smaller. As a result, cluster counting can improve PID performance and can reach about a factor of two better resolution compared to the traditional approach. For this reason, cluster counting is proposed for future colliders such as CEPC and FCC . A study for the BESIII upgrade also reports that cluster counting gives better PID than dE/dx, with about 1.7× higher π/K separation power.

The main difficulty in cluster counting is reconstruction. The reconstruction algorithm must find the number of primary ionizations in a digitized waveform accurately and efficiently. The ionization process is random and many signals can overlap, so the waveform can be complex. Therefore, modern machine learning cluster counting algorithm consist of two main steps: peak finding by LSTM Model and clusterization by CNN Model.

5.3.1 Peak-Finding Algorithm (LSTM Model)

The task of peak finding can be framed as a classification problem in machine learning. In the first step of cluster counting techniques, Long short term memory model (LSTM) is used to discrimate signal (peaks arises from primary and secondary electrons) from the noise in the digitized waveform. This task is usually referred as classification task and this algorithm is known as peak finding algorithm.

Data Preprocessing

To reduce the complexity, the waveform are divided into segments, each comprising 15 bins. Each segment can represent either a signal or a noise. The list of the time depended amplitudes of a segment , is served as the input feature for the neural network. For simulated waveforms, full labels can be obtained from the first-principle MC truth information.

Why LSTM is Suitable?

The digitized waveform is a time-ordered signal, so it can be treated as a time series. In peak finding, we scan the waveform using sliding windows. Recurrent neural networks (RNNs) are designed for this type of sequential data because they pass information from one time step to the next [69]. However, a basic RNN often fails when the sequence becomes long. During training, the gradients can become very small or very large, which makes learning unstable. This is known as the vanishing-gradient and exploding-gradient problem [70]. Because of this, a basic RNN cannot reliably keep important information for a long time.

An LSTM is built to solve this issue. It adds a memory cell that can store useful information across many time steps, so important features from earlier parts of the waveform can remain available safe for long time. The flow of information is controlled by

gates. The input gate controls how much new information enters the memory. The forget gate controls how much old information is kept or removed. The output gate controls how much of the stored memory is sent to the next layer. With this controlled memory, the LSTM can keep long-range time information and handle long sequences better than a basic RNN [71].

LSTM Architecture for Peak Detection

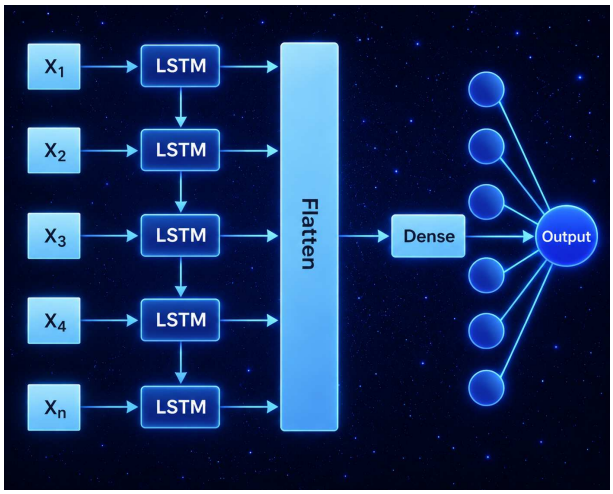
The LSTM-based model used in this study to detect peaks in the digitized waveform has a unique architecture with memory cells and gates that help the network remember important information over time. These memory cells keep track of past events, which is essential for analyzing sequences like waveforms.

The architecture of the LSTM network for detecting primary and secondary electron peaks is as follows:

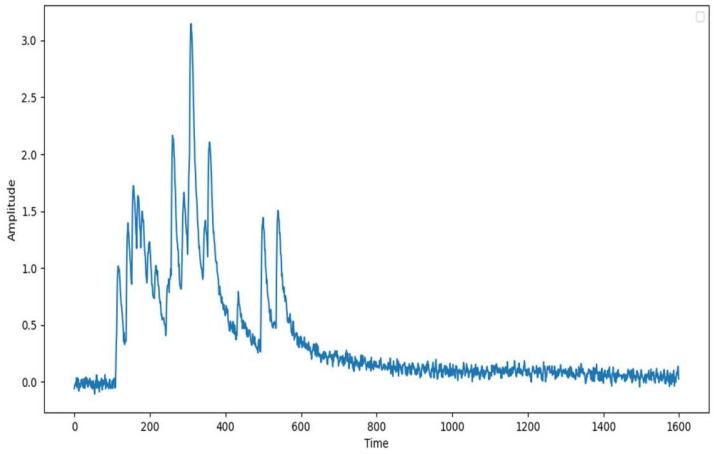
- **LSTM Layer:** The LSTM layer processes the sequential waveform data. It captures long-term dependencies between data points, helping the model learn patterns from previous time steps. This layer takes 15 time dependent data points as an input features at one time step. These 15 data point usually represent time depended amplitude point for one peaks in the digitized waveform.

- **Flatten and Two Linear Layers:** Then this information is forwarded to the Flatten layer which reshapes the sequence output into one long one-dimensional feature vector, without changing the values if there is two or three dimensional features. So, this flattened vector is passed to the fully connected (dense) layer. Finally,the output linear layer takes all the inputs from dense layer and produces a single output . A sigmoid activation function is applied to this output to get the final classification result [72].

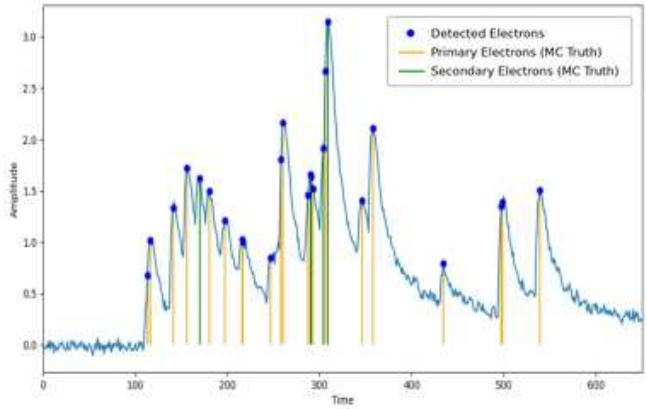
  In addition to that one important loss function was used for the classification task. The loss function is the binary cross-entropy loss function for binary classification problems, to quantifies the discrepancy between the true labels and the predicted probabilities, effectively guiding the model towards accurate predictions. This choice is driven by its effectiveness in handling cases where the output is a probability value between 0 and 1, making it particularly suited for our binary classification task. In figure 5.3 , you can see that LSTM model is applied to the digitized waveform to detect peaks (blue dots) arises from primary and secondary electrons rather than noise for 10 GeV muon momentum. In the next chapter, we will discuss the structure of the LSTM model based on its specific hyperparameters, which are typically used for the peak-finding task for different momenta (2–10 GeV and 180 GeV) of muons, kaons, and pions. We will also compare the distributions with the mean number of primary clusters reconstructed by the LSTM model (matched with the primary ionization truth in the peak-finding stage) to the mean number of primary clusters (Monte Carlo truth). This LSTM Model with



(a) LSTM model with input, flatten, dense and output layer



(b) Input digitized waveform for an LSTM model



(c) Detected peaks (blue) with primary (orange) and secondary (green) electron MC truth

Figure 5.3: ong Short-Term Memory model applied to the waveform to detect peaks (shown by blue dots) and distinguish them from noise as a classification task.

specific hyperparameter is usually selected as best model by method known as search grid hyper-parameter optimization method on the behalf of important metric(Area Under the Curve Value).

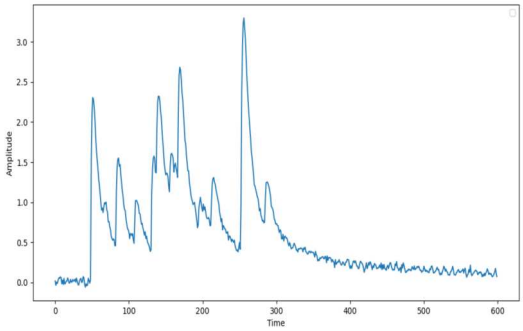


Figure 5.4: Waveform A: Digitized input waveform for an LSTM model.

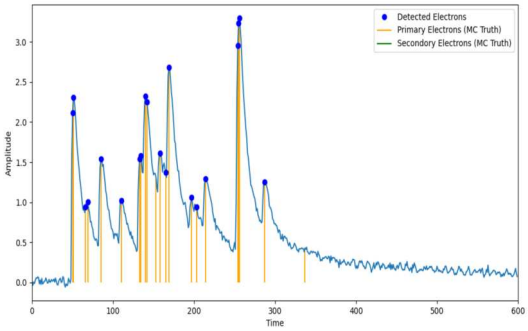


Figure 5.5: Waveform B: Detected peaks (Blue dot) by using an LSTM Model.

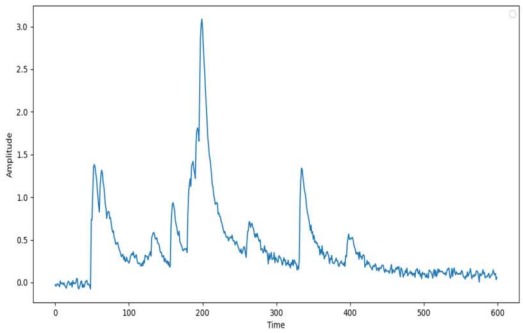


Figure 5.6: Waveform C: Another Digitized input waveform for an LSTM model.

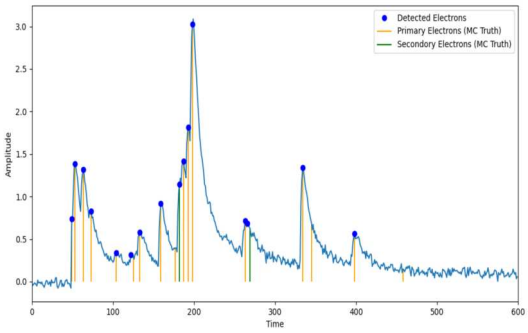


Figure 5.7: Waveform D: Detected peaks (Blue dots) by using an LSTM Model.

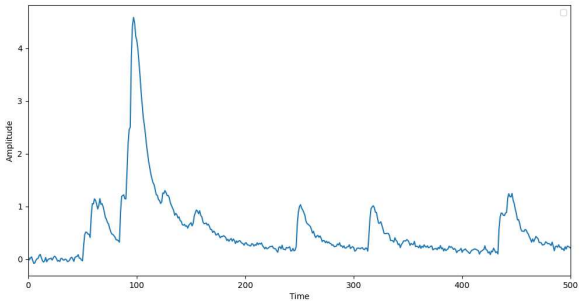


Figure 5.8: Waveform 1: Digitized input waveform for an LSTM model.

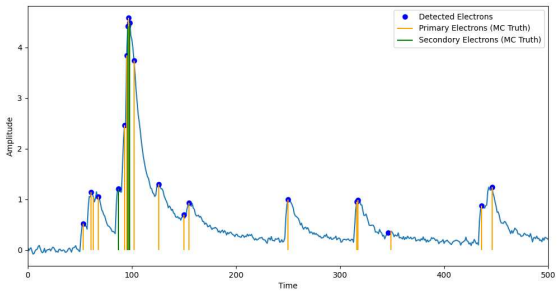


Figure 5.9: Waveform 1: Detected peaks (blue dots) by using an LSTM model.

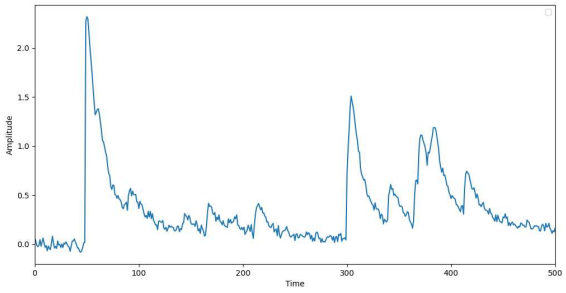


Figure 5.10: Waveform 2: Digitized input waveform for an LSTM model.

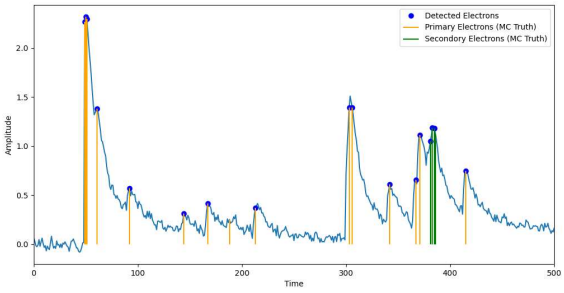


Figure 5.11: Waveform 2: Detected peaks (blue dots) by using an LSTM model.

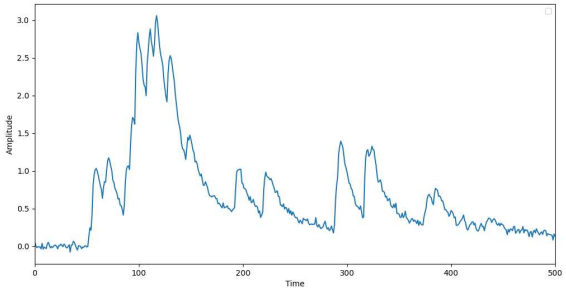


Figure 5.12: Waveform 3: Digitized input waveform for an LSTM model.

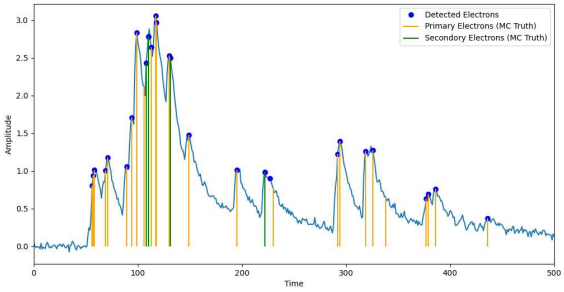


Figure 5.13: Waveform 3: Detected peaks (blue dots) by using an LSTM model.

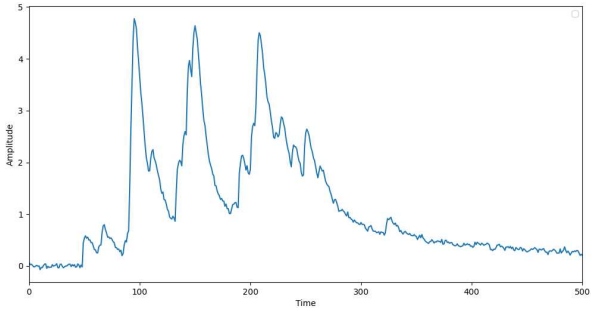


Figure 5.14: Waveform 4: Digitized input waveform for an LSTM model.

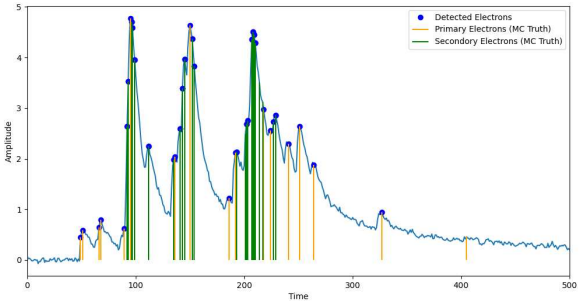


Figure 5.15: Waveform 4: Detected peaks (blue dots) by using an LSTM model.

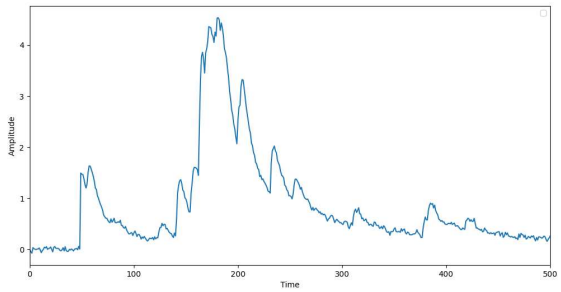


Figure 5.16: Waveform 5: Digitized input waveform for an LSTM model.

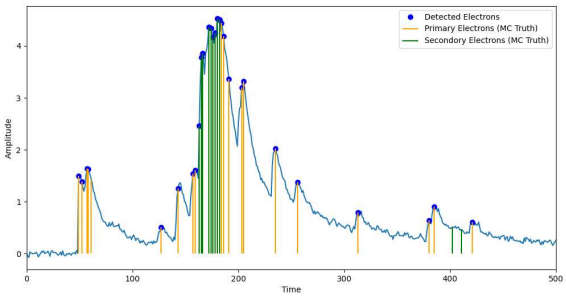


Figure 5.17: Waveform 5: Detected peaks (blue dots) by using an LSTM model.

Examples of different digitized input waveforms and the corresponding outputs of the Long Short-Term Memory (LSTM) peak-finding model are shown in Figures 5.4–5.17. The left panels, Figures 5.4, 5.6, 5.8, 5.10, 5.12, 5.14, and 5.16, present the original waveforms, whereas the right panels, Figures 5.5, 5.7, 5.9, 5.11, 5.13, 5.15, and 5.17, show the primary and secondary electron peaks detected by the LSTM model. Blue markers denote the detected peaks, while the vertical lines indicate the Monte Carlo truth positions for primary electrons (orange) and secondary electrons (green). These examples demonstrate the ability of the LSTM model to identify signal peaks and separate them from waveform noise.

5.3.2 CNN-Based Clusterization Algorithm

After the LSTM peak-finding step, the waveform contains a list of all detected ionization peaks. These peaks are related to the primary and secondary electrons. Because the peak finder does not separate these two types of detected peaks, a second step is needed. This second step is known as the clusterization algorithm.

In principle, secondary ionization occurs locally with respect to the primary electrons if the primary electrons have large enough energy. This leads to the electrons from a single cluster being located close to each other in the waveform. One can exploit this property to design an algorithm based on CNN clusterization. This clusterziation algorithm usually estimate the number of primary clusters based on the detected peaks found in the previous step.

The input to the CNN-based clustering model is the time sequence of the detected peaks obtained from the peak-finding step. This information is encoded into a one-dimensional feature array of size $(1024, 1)$. Each element in this array represents the detected peak at a given time bin. Therefore, the complete peak-time pattern is provided to the network as an input sequence.

First, the input features pass through a one-dimensional convolutional layer. This layer uses multiple convolution filters to scan the input locally along the time axis. These filters are typically learned parameters that capture patterns in the feature map, helping the model understand how peaks belonging to the same physical cluster appear close to each other. After convolution, a ReLU activation function is applied. This activation keeps

positive values and removes negative ones, allowing the network to learn in the best way

The extracted feature maps are then passed to a pooling layer. The pooling operation reduces the size of the feature maps while preserving the most important peak information.

After pooling, the feature maps are forwarded to the flatten layer. The flatten layer reshapes the sequence output into one long one-dimensional feature vector, without changing the values, so they can be processed by dense layers.

The flattened one dimensional vector is then passed through fully connected layers. These layers combine all extracted features.

The final fully connected layer maps the learned features to a single output node. This output node produces one regression value, which represents the predicted mean number of primary ionization clusters in comparison with truth monte carlo value. In addition to that one important loss function was used for the regression task. The loss function is the mean square error function for regression problems, to measure the difference between truth and predicted value, effectively guiding the model towards accurate predictions

Figures 5.18 and 5.19 show that the CNN model is applied to the digitized waveform to estimate the number of primary electrons (blue dots) for the 180 GeV muon sample, based on the total detected peaks found in the previous peak-finding step. In addition, the number of primary electrons estimated by the CNN is also shown in Figures 5.20, 5.22, 5.24, 5.26, and 5.28, based on the total detected peaks presented in Figures 5.21, 5.23, 5.25, 5.27, and 5.29.

It is important to note that the best CNN model on the behalf of its hyperparameters are showed in the next chapter. You will also see all the distributions related the predicted mean number of primary clusters by CNN for different momenta 2, 4, 6, 8, 10 and 180 GeV of muon, Pion and kaon. These reconstructed mean number of primary clusters will compare to the mean number of primary clusters (MC) generated based on garfled++.

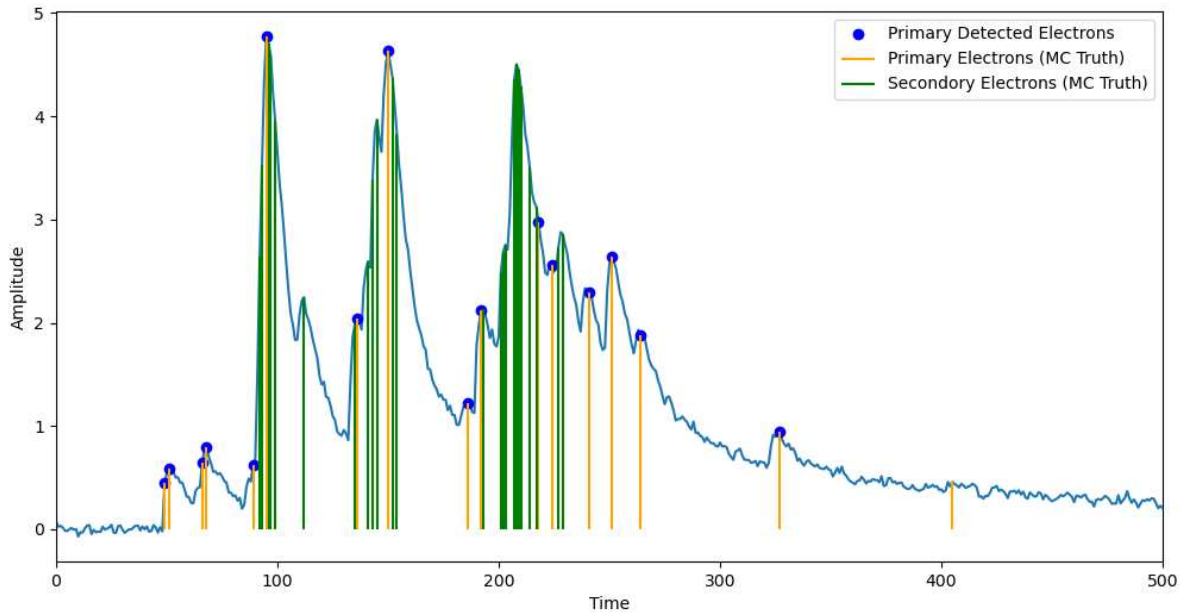


Figure 5.18: **(a) Peak pattern after peak-finding.** The waveform contains detected electrons. Blue points mark detected primary electrons. The vertical lines indicate the corresponding MC-truth positions for primary (orange) and secondary (green) electrons.

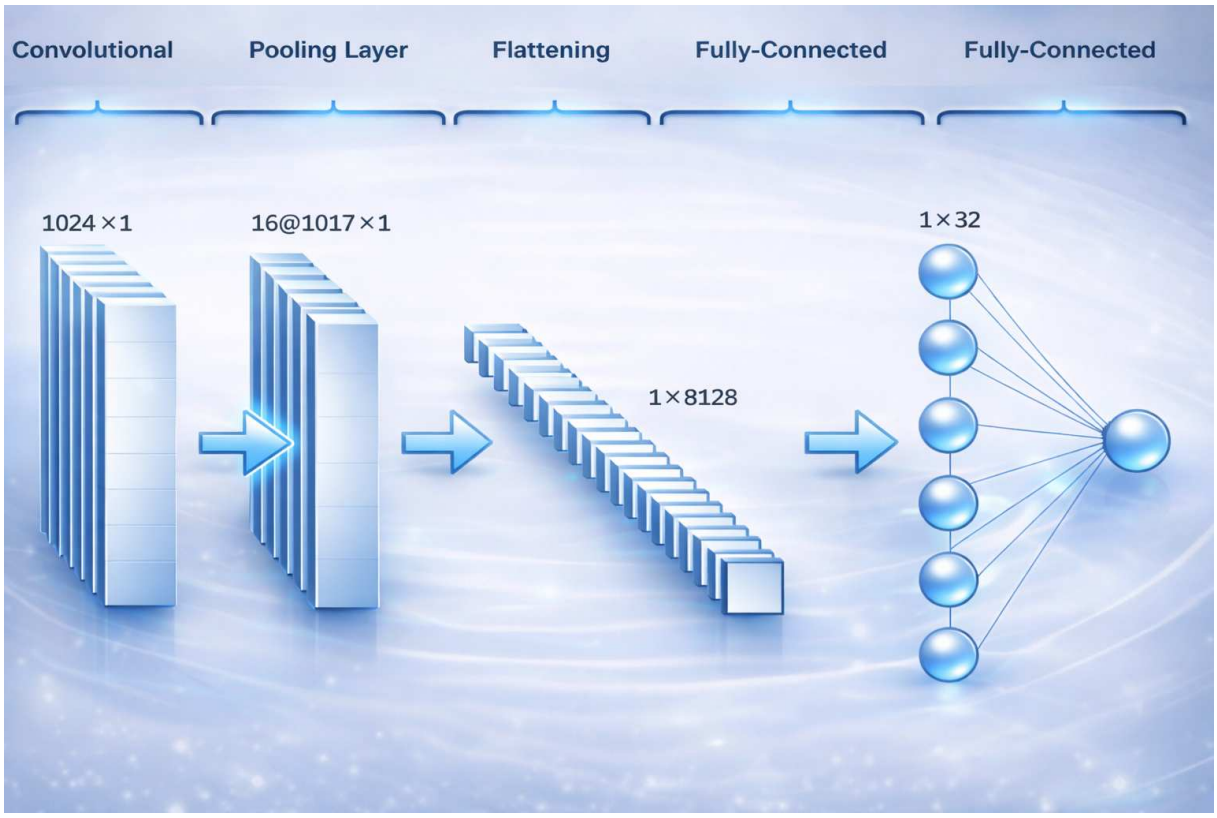


Figure 5.19: **(b) CNN-based clusterization model.** The CNN takes the peak information from step 1, encoded as a 1D input feature sequence, and predicts the number of primary clusters (N_{cls}).

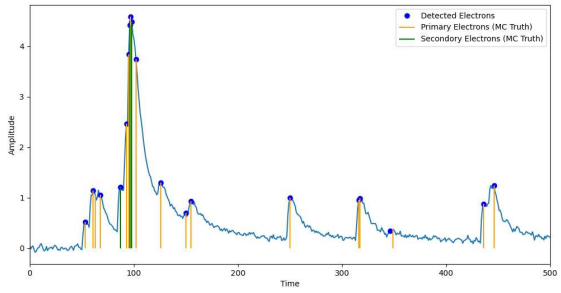


Figure 5.20: Waveform 1: Detected peaks in the waveform.

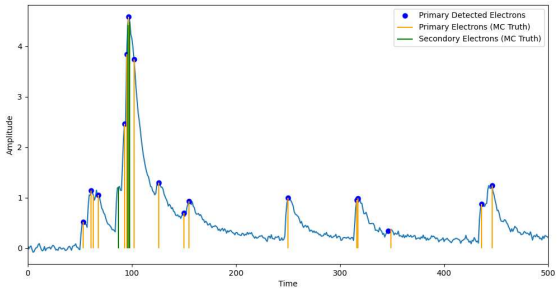


Figure 5.21: Waveform 1: Primary detected electrons in the waveform.

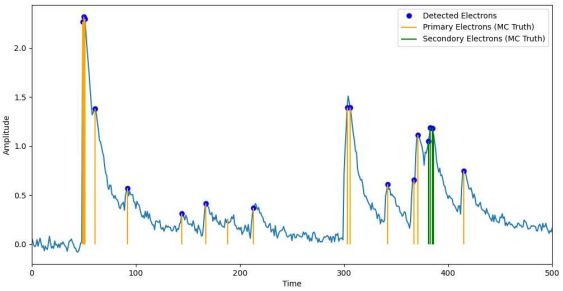


Figure 5.22: Waveform 2: Detected peaks in the waveform.

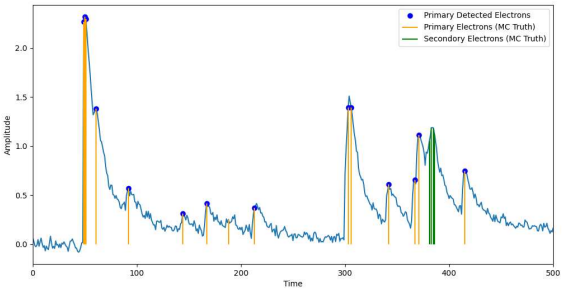


Figure 5.23: Waveform 2: Primary detected electrons in the waveform.

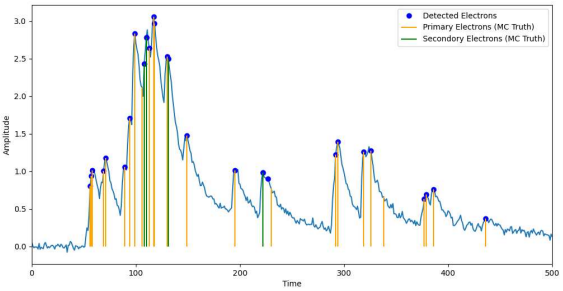


Figure 5.24: Waveform 3: Detected peaks in the waveform.

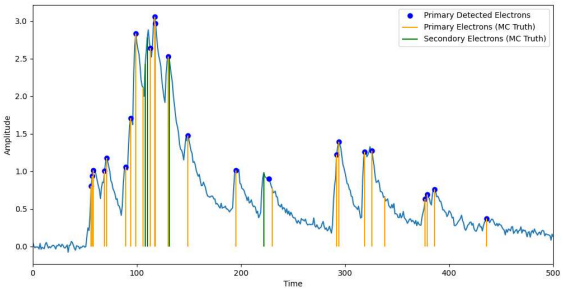


Figure 5.25: Waveform 3: Primary detected electrons in the waveform.

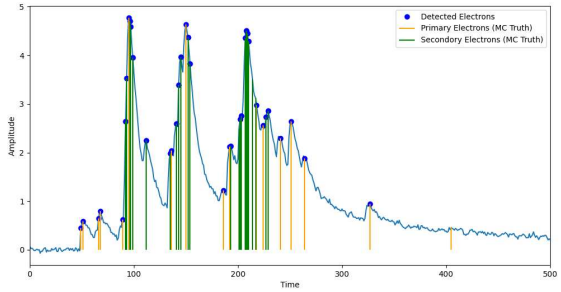


Figure 5.26: Waveform 4: Detected peaks in the waveform.

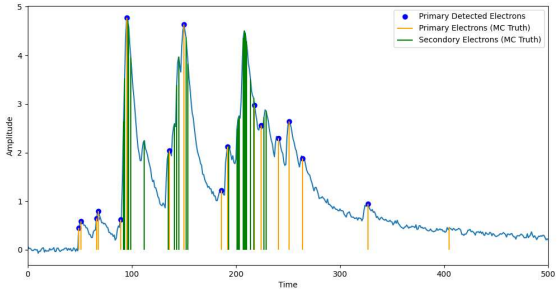


Figure 5.27: Waveform 4: Primary detected electrons in the waveform.

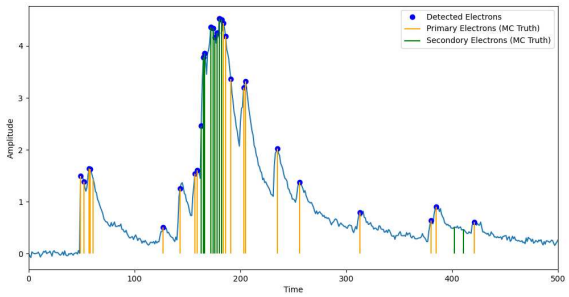


Figure 5.28: Waveform 5: Detected peaks in the waveform.

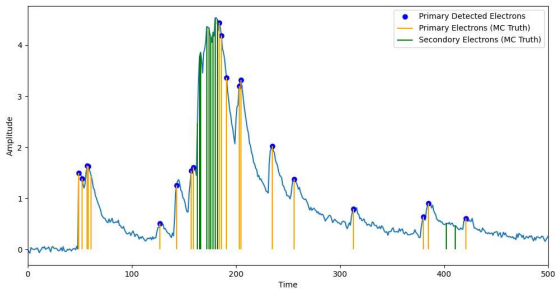


Figure 5.29: Waveform 5: Primary detected electrons in the waveform.

Comparison of different peak finding waveforms(left and right side) are showed above. The left panel shows all detected peaks in the waveform, including both primary and secondary electrons. The right panel shows only the electrons identified as primary by the model. In both panels, the vertical lines indicate the Monte Carlo truth positions of primary electrons (orange) and secondary electrons (green), while the blue markers represent the model predictions.

It is important to note that all the distributions related to the mean number of primary cluster of Monte Carlo Truth and reconstructed by LSTM and CNN model are showed in the next chapter.

Chapter 6

Hyperparameter Optimization of NN Using HPC Resources and PID Performance

The main goal of this chapter is to train neural network models, such as the Long Short-Term Memory (LSTM) Model and Convolutional Neural Network (CNN) Model, on momentum of muon particle ranges from 2 GeV to 20 GeV and 180 GeV. After training, we apply these models to simulated sample of 2, 4, 6, 8, 10, and 180 GeV muon, kaon and pion momenta as a test to assess the performance of the models. These models are usually trained and test for a two-step reconstruction algorithm, which involves peak finding and clusterization in cluster counting techniques. So therefore, I designed a task involving the simultaneous submission of several jobs using local HPC resources at ReCaS, including memory requests, job duration, and CPU usage. The best LSTM peak-finding and CNN clusterization models are selected based on the highest Area Under the Curve (AUC) value and the lowest Mean Squared Error (MSE) value, respectively, among all configurations. The best LSTM and CNN models are also tested using national computing HPC resource (CPU's usage, memory usage, and job duration) in comparison with local bari HPC ReCaS resources. In addition to that, the **final reconstructed results** using neural network models are also presented in comparison with **Monte Carlo (MC)** for different momenta of muon, pion and kaon. Finally, the separation power for muon–kaon and kaon–pion discrimination is presented for both the Monte Carlo (MC) truth and the Convolutional Neural Network (CNN) model.

6.1 Introduction

Every machine learning model has two types of values. The first type is the **model parameters**, which are learned automatically during training (for example, weights and

biases). The second type is the **hyperparameters**, which we must choose *before* training starts (for example, learning rate, batch size, number of layers, number of hidden units, dropout rate, and optimizer settings). These hyperparameters control **how the model learns** and **how complex the model becomes**. If the hyperparameters are not chosen well, the training can become slow, unstable, or the final accuracy can be poor, even if the dataset is good. This is especially true for modern deep neural networks, where small changes in hyperparameters can lead to large changes in performance.

Hyperparameter Optimization (HPO) means selecting hyperparameters in a systematic and automatic way so that the model performance becomes better. This idea is also an important part of **automated machine learning (AutoML)**, where the goal is to reduce manual effort and make model building more automatic. In practice, HPO is useful for three main reasons. First, it **reduces the human effort** required to tune a model, which is important when many experiments are needed. Second, it can **improve model performance** by matching the training settings to the specific dataset and task, and many studies show that tuned hyperparameters can outperform default settings. Third, it **improves reproducibility and fairness** in scientific work, because different models can be compared more fairly when they all receive a similar level of tuning [73].

HPO has a long history and has been studied since the 1990s, and early work already showed that **different datasets often need different hyperparameter choices**. Today, HPO is widely used not only in research but also in industry, because even small improvements can be valuable when models are deployed at scale . At the same time, HPO is still challenging in real applications because it is often expensive and complex. The main practical difficulties can be summarized as follows:

- **Training is expensive:** each trial may require full training of a deep model, which costs a lot of time and computing resources.
- **The search space is large:** hyperparameters can be continuous (like learning rate), discrete (like number of layers), or categorical (like optimizer type), and sometimes one choice controls which other choices are relevant.
- **No direct gradient information:** we usually cannot compute gradients of the validation score with respect to hyperparameters, so HPO behaves like a black-box optimization problem.
- **Generalization is indirect:** we want strong performance on unseen data, but we can only measure performance using limited training and validation datasets, so we must be careful to avoid overfitting the validation set.

Because each hyperparameter trial can be costly, **high-performance computing (HPC)** resources become very important for HPO. The key advantage of HPC is that it allows us to run **many training jobs in parallel**. This parallel execution turns

a slow sequential tuning process into a faster and more structured workflow. In these sections, we use this idea to optimize the hyperparameters of two neural network models for the two steps of cluster counting study: the **LSTM peak-finding model** and the **CNN clusterization model**. The main goal is simple: we want stable training, strong validation performance, and reliable reconstruction results across different momentum ranges.

6.2 Hyperparameter Optimzation Methods: Grid Search and Random Search

In hyperparameter optimization (HPO), we usually treat the training pipeline like a **black box**. This means we choose a set of hyperparameters, train the model, and then only observe the final validation score (for example accuracy, AUC, or loss). We cannot directly compute a gradient with respect to hyperparameters, and the performance surface is often non-convex, so we rely on search strategies that can test many configurations. In this section, we focus on two simple and widely used methods because they are easy to understand, easy to reproduce, and very suitable for running many independent jobs on high-performance computing (HPC) resources.

6.2.1 Grid Search

Grid search is one of the simplest methods for hyperparameter optimization. It is also called full factorial search[74]. In this method, a small set of candidate values is chosen for each hyperparameter before training starts. For example, one may define several values for learning rate, batch size, number of neurons, dropout rate, or number of epochs. Grid search then tests all possible combinations of these values.

The main advantage of grid search is that it is clear and systematic. Every configuration inside the chosen grid is evaluated, so no combination is skipped. This makes the comparison between hyperparameters easy to understand. It is also easy to reproduce, because running the same grid again gives the same set of experiments. Another important advantage is that each training job is independent, so many runs can be executed in parallel on HPC resources.

In this work, grid search was selected because it matched the goal of the study very well. Therefore, it was important to test every selected combination, compare the effect of each setting in a controlled way, and keep the full optimization process easy to reproduce. Since each run was independent, grid search was also practical for parallel execution on HPC resources.

6.2.2 Random Search

Random search is another method for hyperparameter optimization. Instead of testing every point in a predefined grid, it selects hyperparameter combinations randomly and continues until a fixed number of trials is reached.

However, random search does not test all selected combinations. Some useful regions may be missed by chance, and the final set of runs is less structured. As a result, it is less suitable when the goal is to make a complete and fair comparison among all predefined hyperparameter settings.

For this reason, random search is often useful for large search spaces, but grid search was more appropriate for the present study. Here, the search space was not extremely large, and the main goal was to perform a complete, systematic, and reproducible comparison of all selected hyperparameter combinations, as illustrated in Figure 6.1.

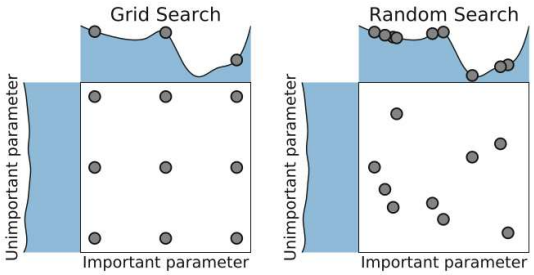


Figure 6.1: Comparison of grid search and random search in a search space with one important and one less important parameter. Grid search evaluates all possible predefined combinations in a precise and systematic way, while random search samples configurations randomly. In this work, grid search was chosen because it provides complete coverage of the selected search space and enables a clear and reproducible comparison of hyperparameter settings [75].

6.3 Hyperparameter Optimization of the LSTM Peak-Finding Model

In this section, the LSTM peak-finding model is trained for **classification** task. The goal is to discriminate signal peaks (associated with primary and secondary electrons) from noise in digitized waveforms. This step is usually a binary classifcation problem and represents the core of the peak finding algorithm used in cluster-counting techniques(see chapter 5)

Training on Simulated Samples: The training for an LSTM peak finding model was conducted on momentum ranges from 2 GeV to 20 GeV and 180 GeV. The trained model (LSTM) was then tested on simualted samples of muon, pion and kaon with momenta of 2

GeV, 4 GeV, 6 GeV, 8 GeV, 10 GeV, and 180 GeV. The simulation parameters included 10% Helium (He) and 90% Isobutane (C_4H_{10}), a cell size of 0.8 cm, a sampling rate of 1.5 GHz, a time window of 400 ns, 10,000 events, and a 45-degree angle between the muon particle track and the z-axis (sense wire) of the drift chamber. The testing aimed to evaluate the performance of the LSTM model for peak finding and the CNN model for clusterization. These simulation parameters are selected to match the real test muon beam data collected in 2022 and 2023. At some point, we will also apply the LSTM peak-finding model on the real 2022 data, which will be discussed in more detail in Chapter 7.

Parameter	2022 Real Data	2023 Data
Sampling rate	1.5 GHz	1.5 GHz
Temperature	293 K	293 K
Gas mixture	He (90%) & C_4H_{10} (10%)	He (90%) & C_4H_{10} (10%)
Pressure	725 Torr	725 Torr
Cell size	0.8 cm	0.8 cm
High voltage	1350 V	**1450 V**
Track angle	45°	45°
Sense wire diameter	20 μm	20 μm
Particle	Muon	Muon
Momentum	180 GeV	**2, 4, 6, 8, and 10 GeV**

Table 6.1: Simulation and operating parameters used to train and test LSTM and CNN Model for the cluster counting technique algorithm. These settings are chosen to match the real test-beam conditions (2022–2023).

Why HPC is needed for LSTM hyperparameter optimization: Hyperparameter optimization requires training the same model many times using different settings. If we do this on a single computer, the process becomes very slow. For this reason, we use local **HPC bari ReCas resources** and submit many jobs at the same time. Each job trains one LSTM configuration independently, which makes the workflow easy to parallelize.

Grid-search workflow using HPC: The workflow follows one clear loop that is repeated for every configuration:

1. Define a set of hyperparameters such as activation functions, optimzizers, diiferent number of neurons in different layers, dropout rate, early stop etc.

2. Create training job for each hyperparameter combination.
3. Submit all jobs simultaneously on the local HPC bari ReCas Resources.
4. For each job, store the trained model and evaluation metrics (Area under the curve value.)
5. Collect results from all jobs and compare them using the same selection rule(highest AUC value).

During submission, we also manage HPC resources such as **memory request**, **job duration (wall-time)**, and **CPU usage**. This is necessary to keep the jobs stable and to avoid failures when many runs are executed in parallel.

Hyperparameters scanned for the LSTM peak-finding model: The grid search includes different sets of hyperparameters, such as the optimizer, batch size, number of epochs, early-stopping patience, activation function, and dropout rate, as summarized in Table 6.2. For each configuration, the available dataset was divided into (70%) for training and (30%) for validation, and the validation subset was used only to evaluate the model performance without updating its parameters. These are the main factors that control training stability and overfitting.

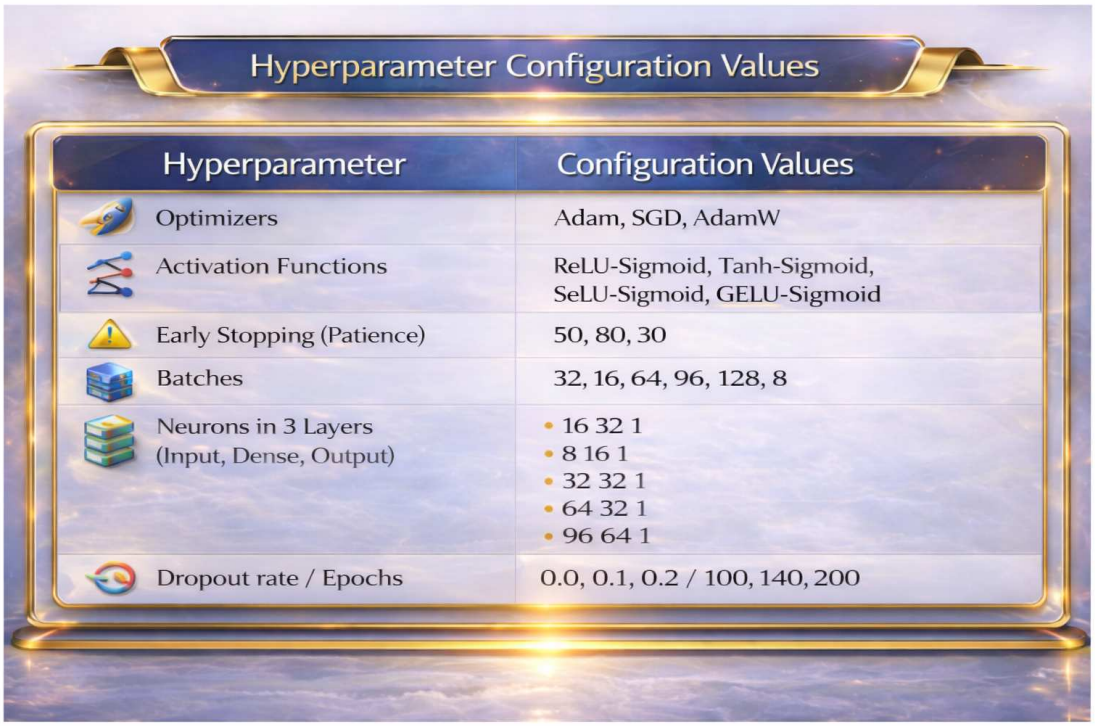


Table 6.2: Hyperparameters and configuration values scanned in the grid search for the LSTM peak-finding model.

6.3.1 Criteria for Selecting the Best LSTM Model

After finishing the grid search on the HPC, we must select one LSTM configuration as the final peak-finding model. In this section, the main selection rule is very simple: we choose the model with the **highest validation AUC** among all trained configurations.

Why we use AUC as the main criterion: AUC (Area Under the ROC Curve) is a strong metric for a classification task because in peak finding, we want a model that works well in different threshold; to seperate *signal peaks* and *background/noise*. Therefore, AUC is used as the main score to compare all sets of hyperparameters set for an LSTM model. Higher AUC value of LSTM model with specific hyperparameters means better classification of signal and background in the digitized waveform.

How the grid search was executed on HPC: The grid search means we trained LSTM model by trying different combinations of hyperparameters.Each configuration was trained as an independent job on HPC resources, so many runs could execute in parallel. For each run, we saved the validation AUC and then ranked all configurations to select best one.

Workflow used to select the best model: The selection procedure is consistent and easy to reproduce:

1. Define a grid of LSTM hyperparameters (optimizer, activations, batch size, epochs, dropout, early stopping patience, and neurons in different layers etc).
2. Submit all configurations on HPC resources (parallel training).
3. For each configuration, compute the **validation AUC**.
4. Rank all configurations by AUC.
5. Select the configuration with the **maximum validation AUC** as the final LSTM peak-finding model.

Different configurations for an LSTM Model: During the grid search, many configurations are trained, and each one produces a validation AUC value. Figure 6.2 shows the scatter plot for the muon momentum (2-20 GeV) that summarizes the LSTM performance for the tested configurations. The x-axis represents the model index (0, 1, 2, etc), where each index corresponds to a unique hyperparameter set, and the y-axis shows the corresponding AUC value obtained for that model. In Figure 6.3, the best LSTM peak finding model is highlighted by the red point, whose AUC value is 0.9784, approximately equal to 0.98.

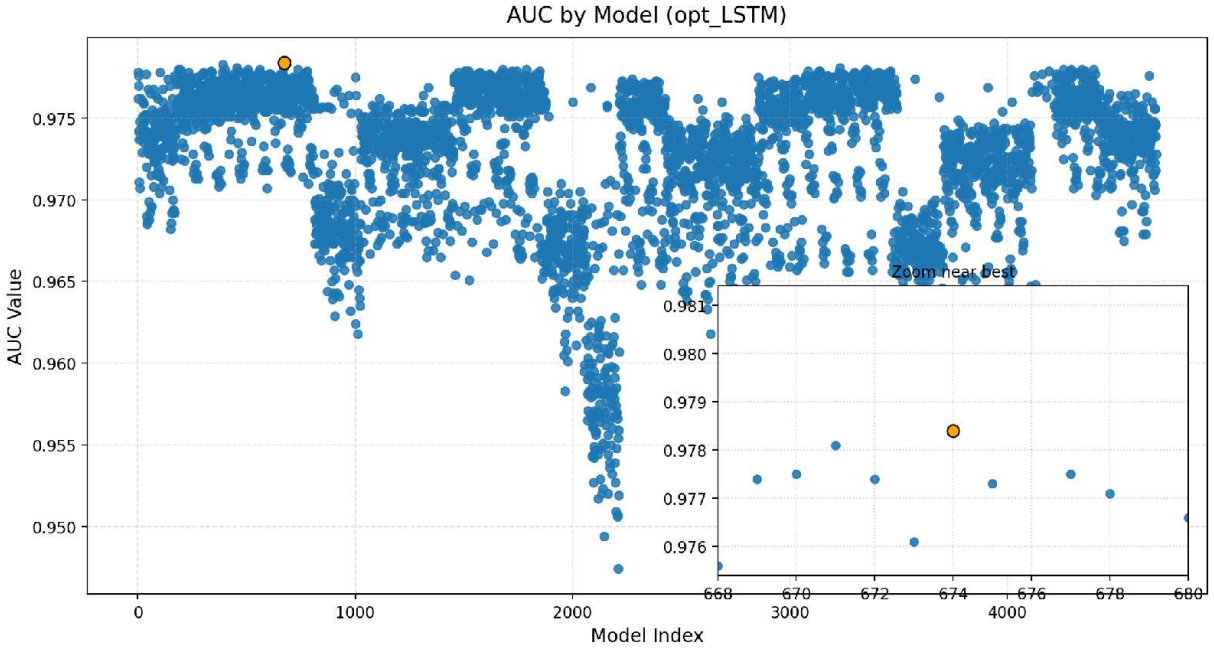


Figure 6.2: AUC values for the LSTM model using muon momenta in the range 2–20 GeV, for all grid-search configurations.

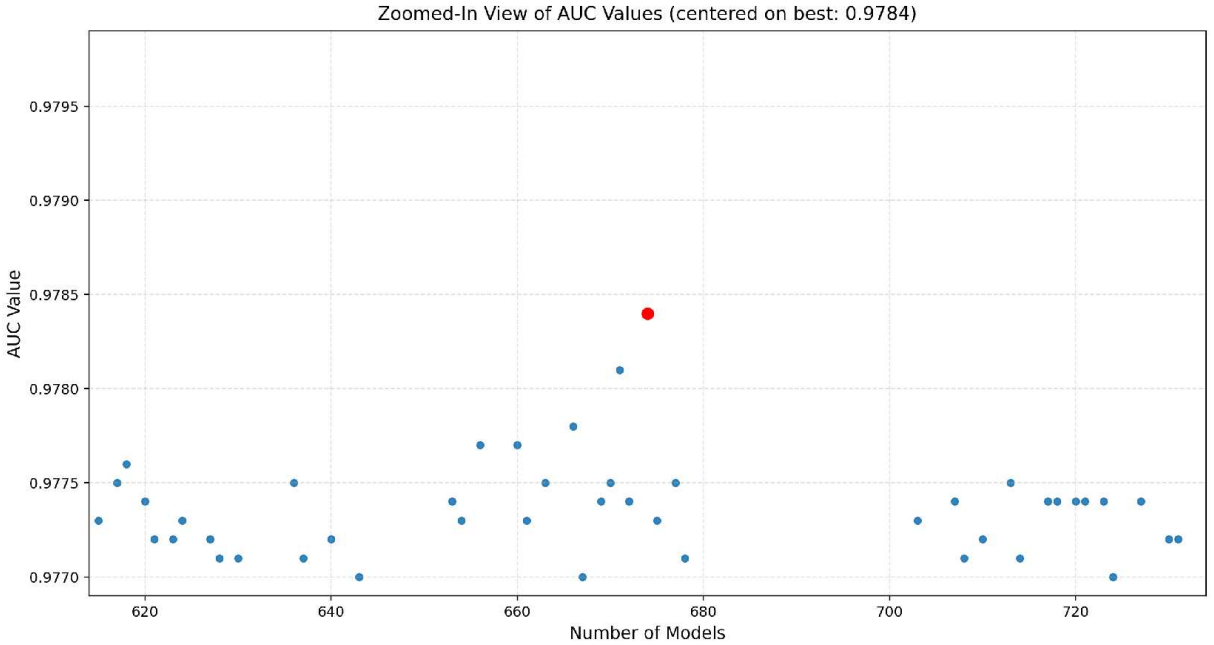
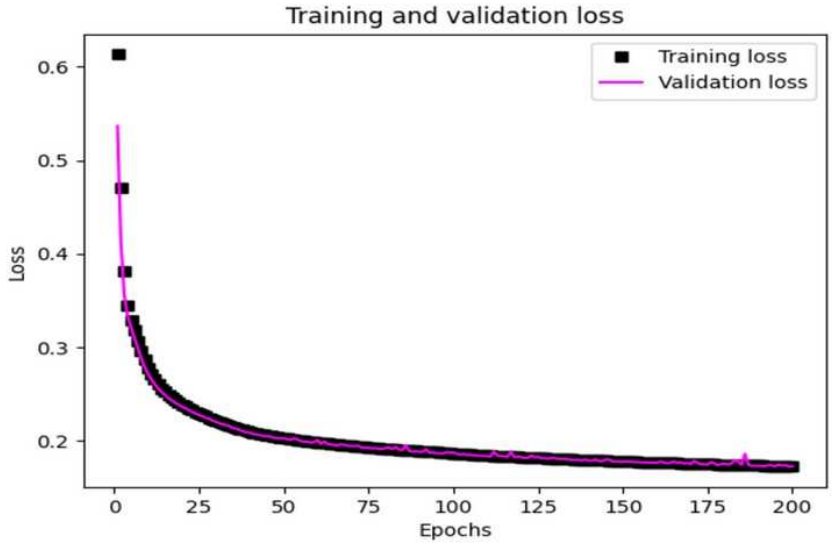
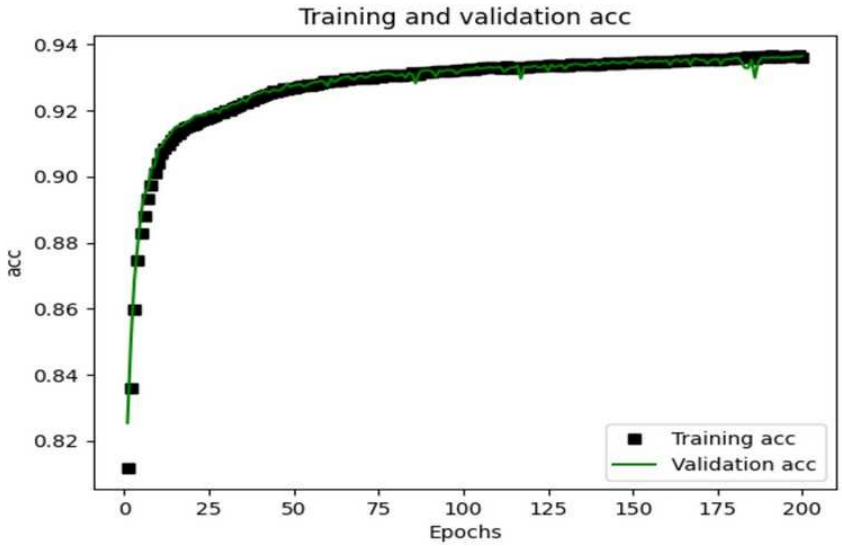


Figure 6.3: Zoomed-in scatter plot focusing on the region with the highest AUC values for muon momenta (2–20 GeV)

Accuracy and Loss of Best LSTM model. Using the above steps, the best configuration reached an AUC of about **0.98**, which indicates strong separation of signal from background in the digitized waveform. Figure 6.4 shows the training behavior of the selected model. The loss decreases with epochs and then becomes almost constant, while the accuracy increases and then stabilizes. This stable behavior is a good sign that the



(a) Training accuracy of the selected LSTM peak-finding model (muon momentum: 2–20 GeV)



(b) Training loss of the selected LSTM peak-finding model (muon momentum: 2–20 GeV)

Figure 6.4: Training history of the selected LSTM peak-finding model for the muon momentum range 2–20 GeV. The accuracy increases and stabilizes, while the loss decreases and becomes nearly constant, indicating good convergence.

model has learned well.

ROC curve. The figure 6.5 shows the Receiver Operating Characteristic (ROC) curve of the selected model. The x-axis represents the False Positive Rate (FPR), while the y-axis represents the True Positive Rate (TPR). The solid blue curve shows the performance of the classifier, and the dashed diagonal line represents the performance of a random classifier. Since the ROC curve lies close to the upper-left corner and the area under the curve is approximately AUC ≈ 0.98, the model has excellent discrimination ability between signal and background.

The quantities are defined as

$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{and} \qquad \text{FPR} = \frac{FP}{FP + TN}.$$

where TP is the number of correctly detected peaks and FP is the number of wrong detected peaks, (TP+FP) is the total number of detected peaks, and (TP+FN) is the total number of peaks in MC truth of the waveform

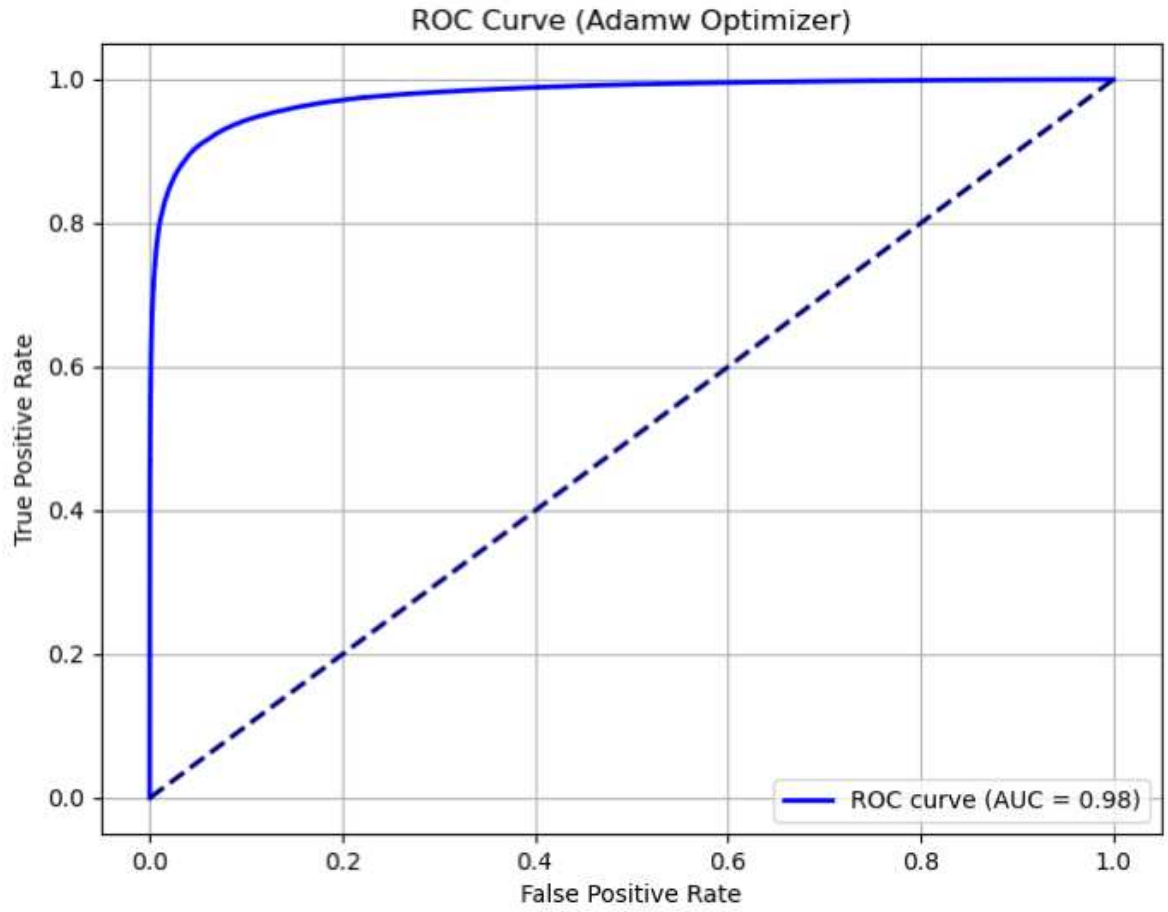


Figure 6.5: ROC curve of the selected LSTM model evaluated on muon momenta between 2 and 20 GeV (AUC $\approx$ 0.98).

In addition, the figures 6.6 -6.9 shows the scatter plots, accuracy, loss, and ROC curve for muons with momentum $p_\mu = 180$ GeV.

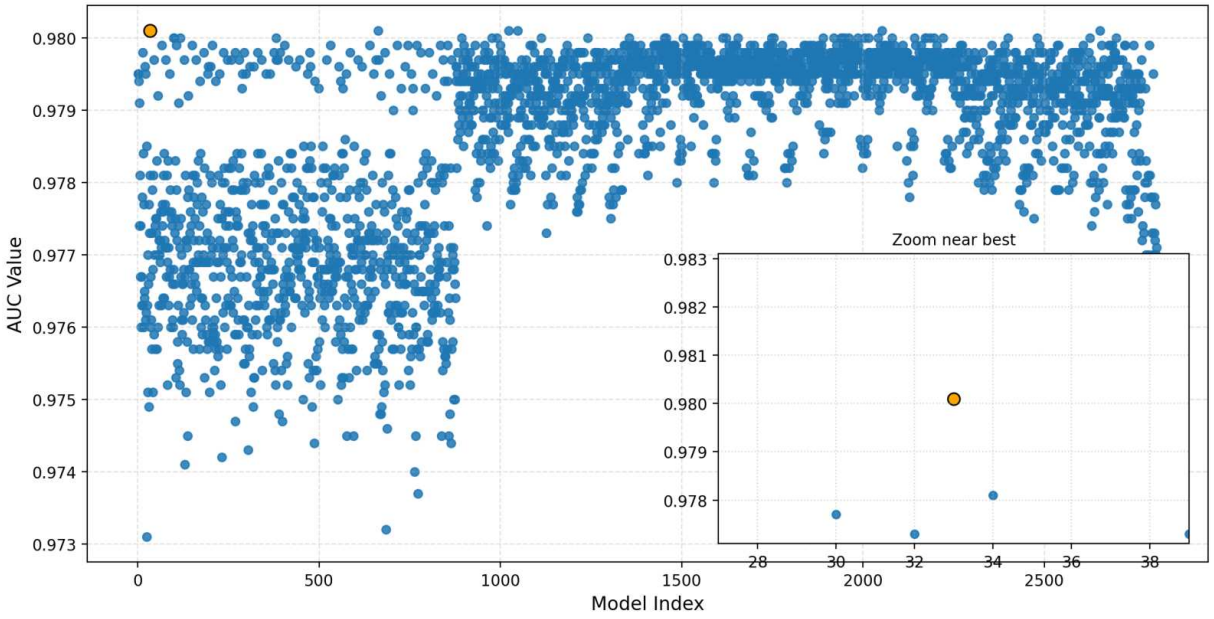


Figure 6.6: AUC values for the LSTM model using 180 GeV muon momenta, for all grid-search configurations.

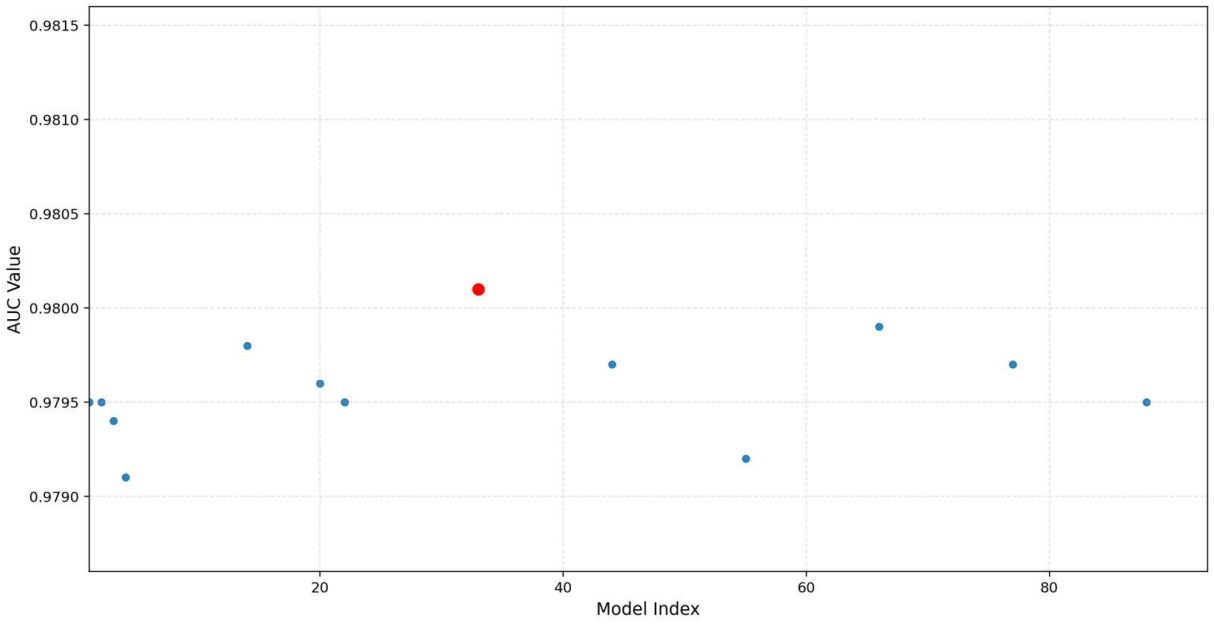


Figure 6.7: Zoomed-in scatter plot focusing on the region with the highest AUC value (0.98) for muon momenta (180 GeV)

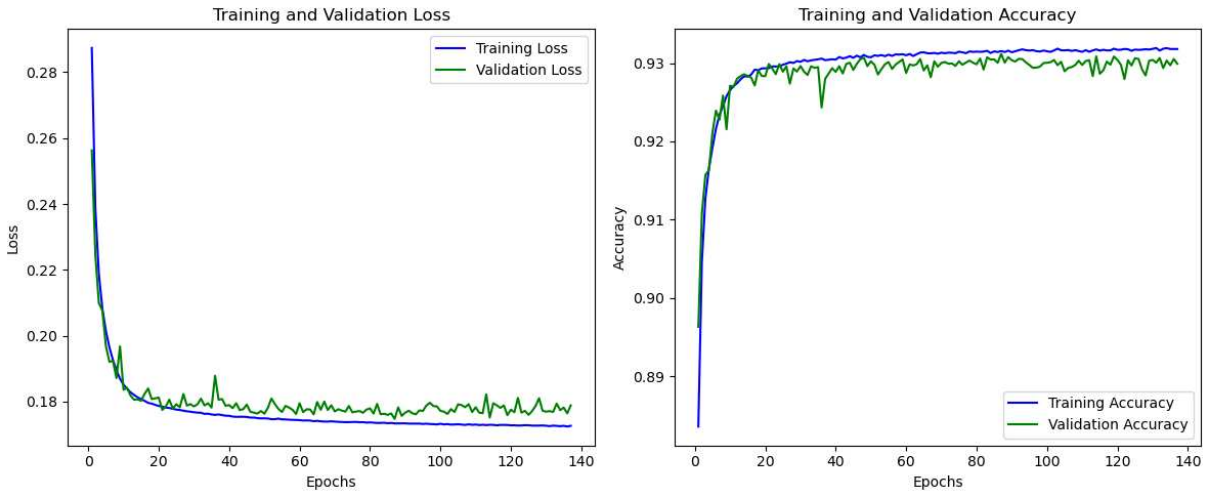


Figure 6.8: Above right side plot show the the loss vs epochs and right side plot shows accuracy vs epochs for 180 GeV muon momentum. The loss and accuracy is decreasing and increasing respectively with epochs initially. The two curve become almost stable after some epochs .

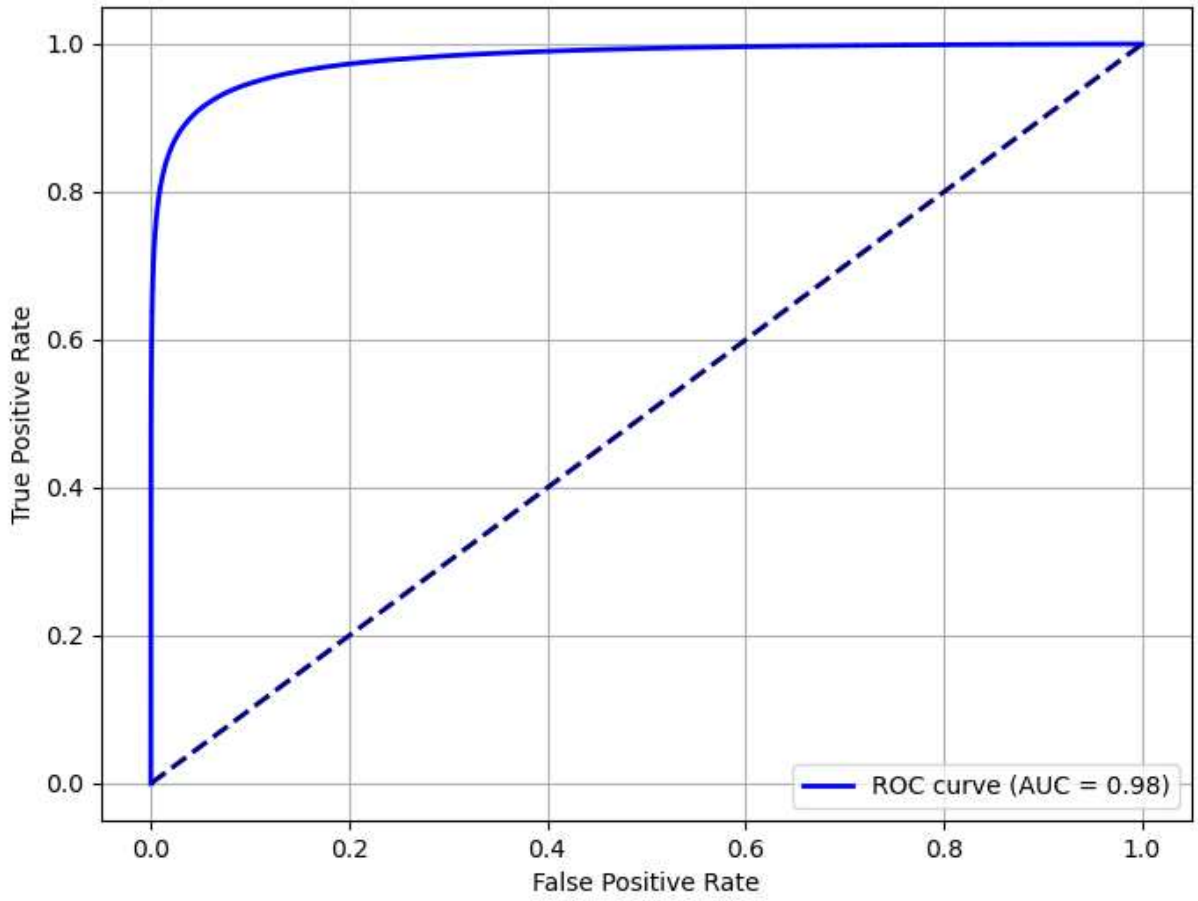
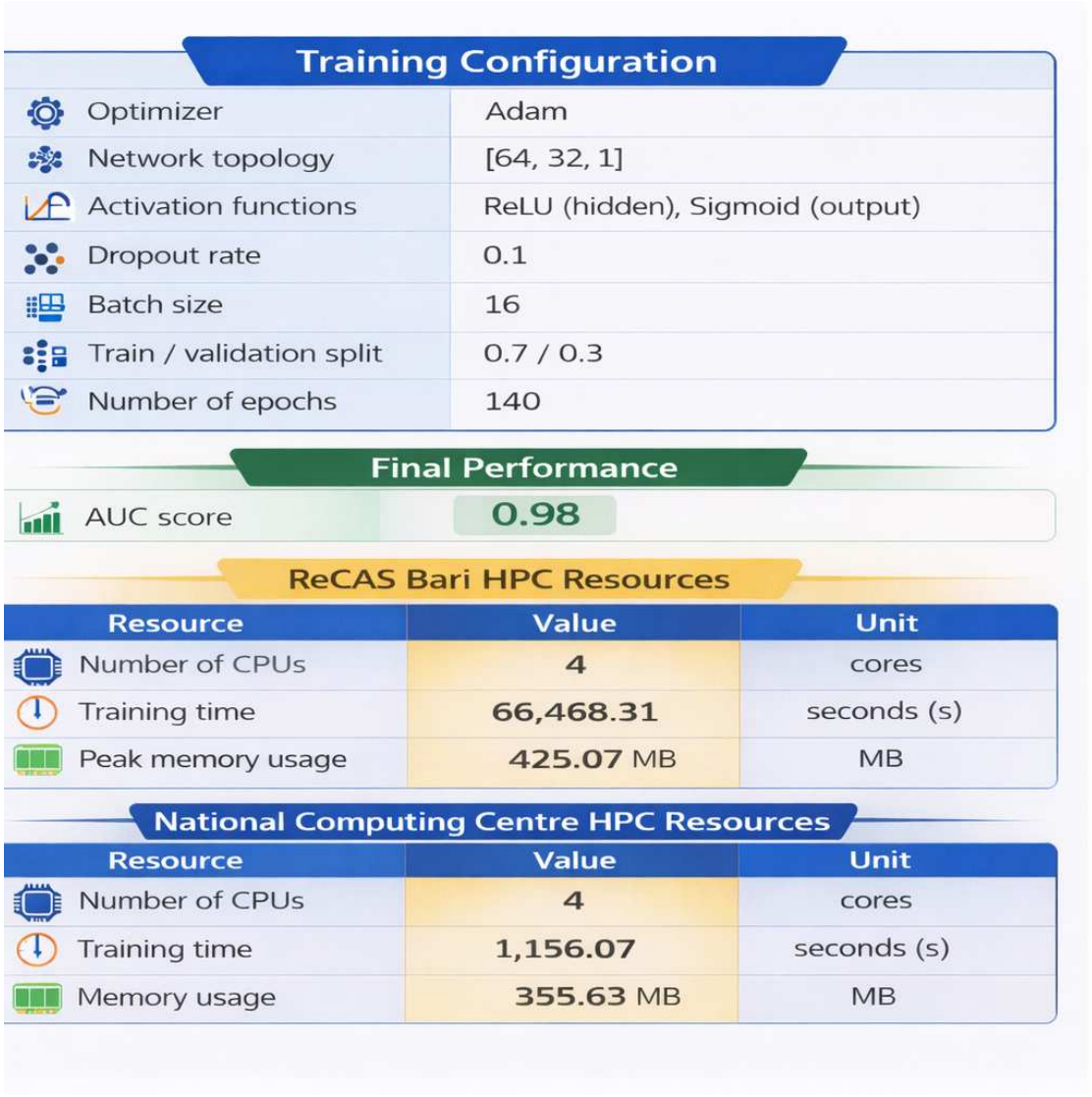


Figure 6.9: ROC curve of the selected LSTM model evaluated on muon momenta 180 GeV (AUC $\approx$ 0.98).

HPC Resources and Hyperparameters of Best LSTM Model

The table 6.3 and 6.4 provides an overview of the hyperparameters and final performance of the best LSTM model, along with detailed HPC resource usage from both local ReCaS Bari and the National Computing Centre resources. The training configuration section shows the hyperparameters of the best LSTM model includes the optimizer, neurons in different layers, activation functions, dropout rate, batch size, train/validation split, and the number of epochs. The final performance of the model is summarized by an AUC score of 0.98. Additionally, the table displays the resources used during training, including the number of CPUs, training time, and memory usage for both computing platforms.



Training Configuration	
Optimizer	Adam
Network topology	[64, 32, 1]
Activation functions	ReLU (hidden), Sigmoid (output)
Dropout rate	0.1
Batch size	16
Train / validation split	0.7 / 0.3
Number of epochs	140

Final Performance	
AUC score	0.98

ReCAS Bari HPC Resources

Resource	Value	Unit
Number of CPUs	4	cores
Training time	66,468.31	seconds (s)
Peak memory usage	425.07 MB	MB

National Computing Centre HPC Resources

Resource	Value	Unit
Number of CPUs	4	cores
Training time	1,156.07	seconds (s)
Memory usage	355.63 MB	MB

Table 6.3: The hyperparameters and HPC resource usage for the best LSTM model, trained on 180 GeV momentum, are reported in the table.

Training Configuration

Optimizer	AdamW
Network topology	[96, 64, 1]
Activation functions	ReLU (hidden), Sigmoid (output)
Dropout rate	0.0
Batch size	8
Train / validation split	0.7 / 0.3
Number of epochs	200

Final Performance

AUC score	0.9784

ReCaS Bari HPC Resources

Resource	Value	Unit
Number of CPUs	4	cores
Training time	38,729.93	seconds (s)
Peak memory usage	401.97	MB

National Computing Centre HPC Resources

Resource	Value	Unit
Number of CPUs	4	cores
Training time	13,095.21	seconds (s)
Memory usage	78.42	MB

Table 6.4: The hyperparameters and HPC resource usage for the best LSTM model, trained over the 2–20 GeV momentum range, are reported in the table.

6.4 Hyperparameter Optimization of the CNN Clusterization Model

After selecting the best LSTM peak-finding model, the next step is to estimate the **mean number of primary ionization clusters** on the behalf of total detected peaks. In this section, a **Convolutional Neural Network (CNN)** is also trained for this task. The CNN is a **regression model**, which means it does not output a class label. Instead, it estimate the mean number of primary clusters based on the detected peaks found in peak finding algorithm. This algorithm is known as clusterization which is the second important step of cluster counting tecniques.

It is important to note that the CNN is trained using the **same simulation setup**

and operating conditions that were described previously. These simulation parameters are chosen to match the real test-beam data collected in 2022(180 GeV) and 2023(2-10 GeV), so that the trained model can later be applied in a consistent way on the real test beam data.

To train many CNN configurations efficiently, I prepared an HPC workflow where many jobs are submitted at the same time as discussed in more details in the previous section. Each job corresponds to one CNN hyperparameter configuration. This approach reduces manual effort, and it also makes the full optimization process easier to repeat. In each run, I used differnt sets of hyperparameters such as optimizer, activation functions, early-stopping patience, batch size, the number of filters in convolution layers, dropout, and number of epochs etc. Table 6.5 shows the hyperparameter ranges used in the CNN grid search. During training, I also managed different local recas bari HPC resources such as CPU usage, memory request, and runtime.

Table 6.5: Hyperparameter space used for the CNN clusterization model during grid search.

6.4.1 Criteria for Selecting the Best CNN Model

Because the CNN is a regression model, the selection rule is based on an error value, not on AUC. In this work, the **Mean Squared Error (MSE)** is used as the main metric. The MSE measures how close the predicted cluster count $\hat{y}_i$ is to the true target value y_i. A smaller MSE means the model is predicting the number of clusters more accurately.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 . \tag{6.1}$$

After all CNN configurations finish running on HPC, I compute the validation MSE for each configuration and then rank all models from lowest to highest MSE. The model with the **minimum validation MSE** is selected as the best CNN model. Figure 6.10 shows the distribution of MSE values across many tested configurations, and the best model corresponds to the lowest point highlighted in the scan.

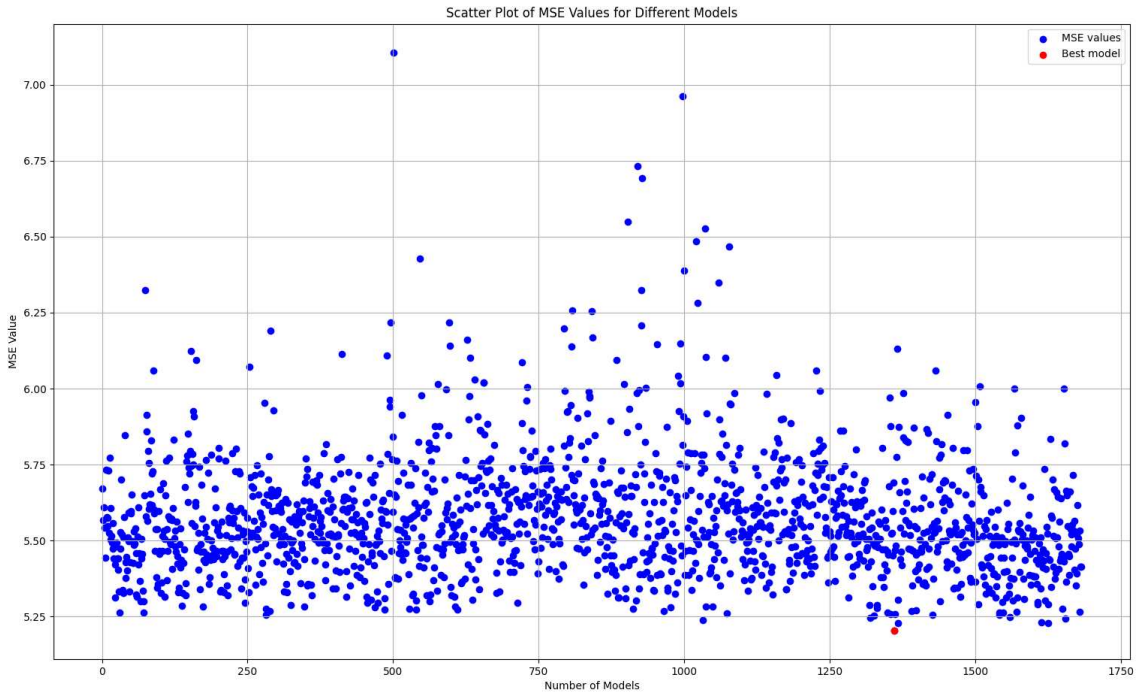


Figure 6.10: MSE values for different CNN configurations obtained during the grid search over the momentum range of 2–20 GeV. The configuration with the lowest MSE is selected as the best-performing model.

After selecting the best CNN configuration, I study its training behavior. The training and validation loss curves are shown in Figure 6.11. The loss decreases quickly at the beginning and then becomes almost stable, which indicates that the model has learned a consistent mapping from detected peaks to the cluster count.

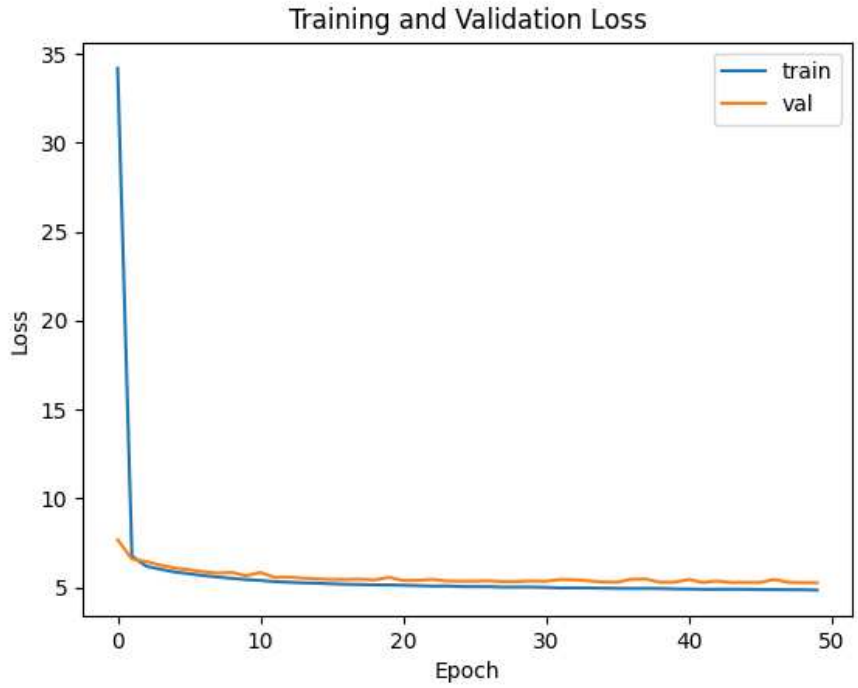


Figure 6.11: Training and validation loss curves for the best CNN regression model over the momentum range of 2–20 GeV. Both curves decrease rapidly during the initial epochs and then gradually stabilize, indicating good convergence and consistent learning performance of the model.

The final selected hyperparameters and HPC resources of the best CNN model are summarized in table 6.6. This configuration is the one that produced the lowest MSE among all tested models.

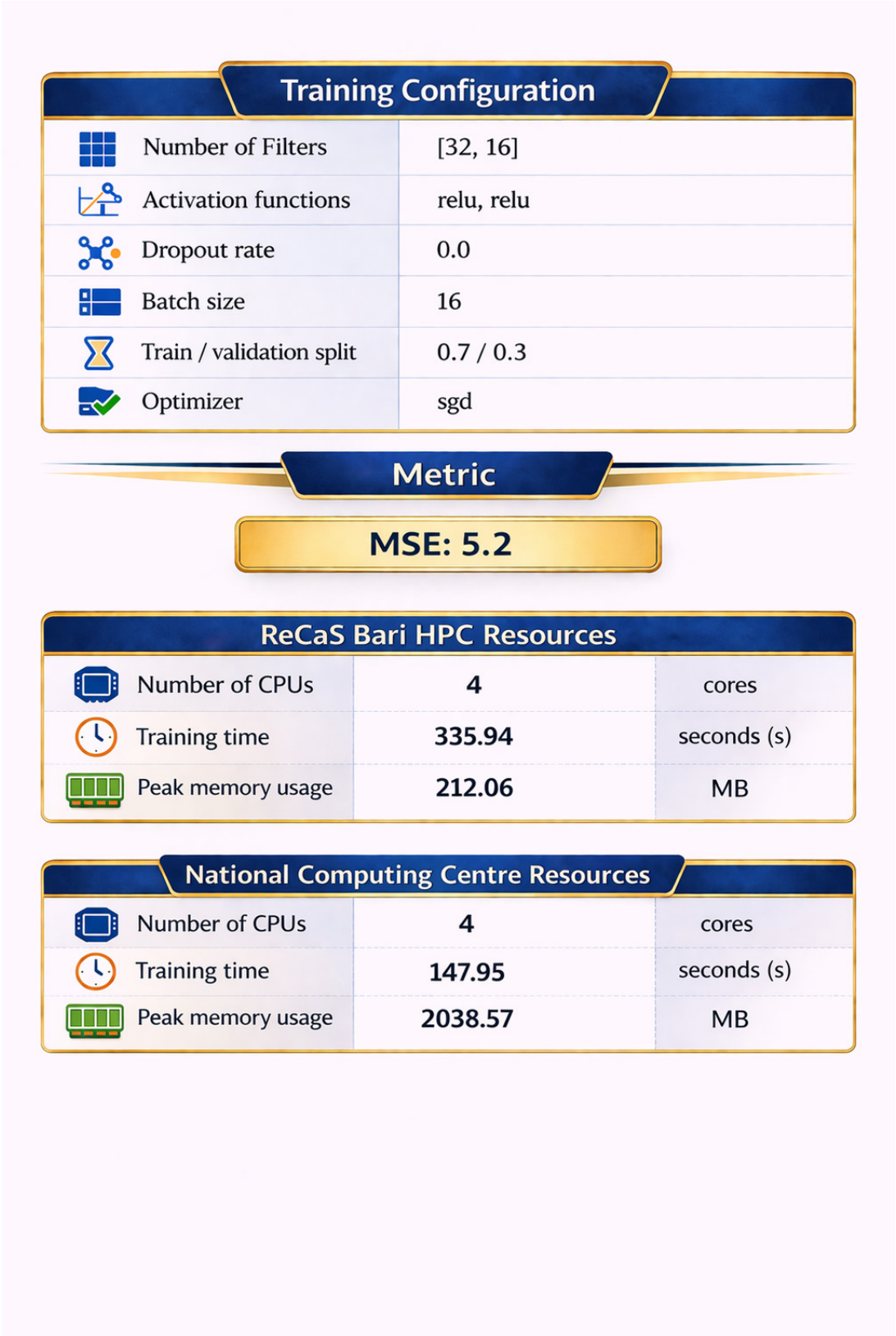


Table 6.6: The hyperparameters and HPC resource usage for the best CNN regression model, trained over the momentum range of 2–20 GeV, are reported in the table.

In addition, the figure 6.12-6.14 and table 6.7 shows the different configurations of CNN model, selection of best CNN model on the behlaf of lowest mean sqaure error value

among all configurations and loss (mse) vs epochs plot, summary of the best CNN model using HPC resources respectively for muons with momentum $p_\mu = 180$ GeV.

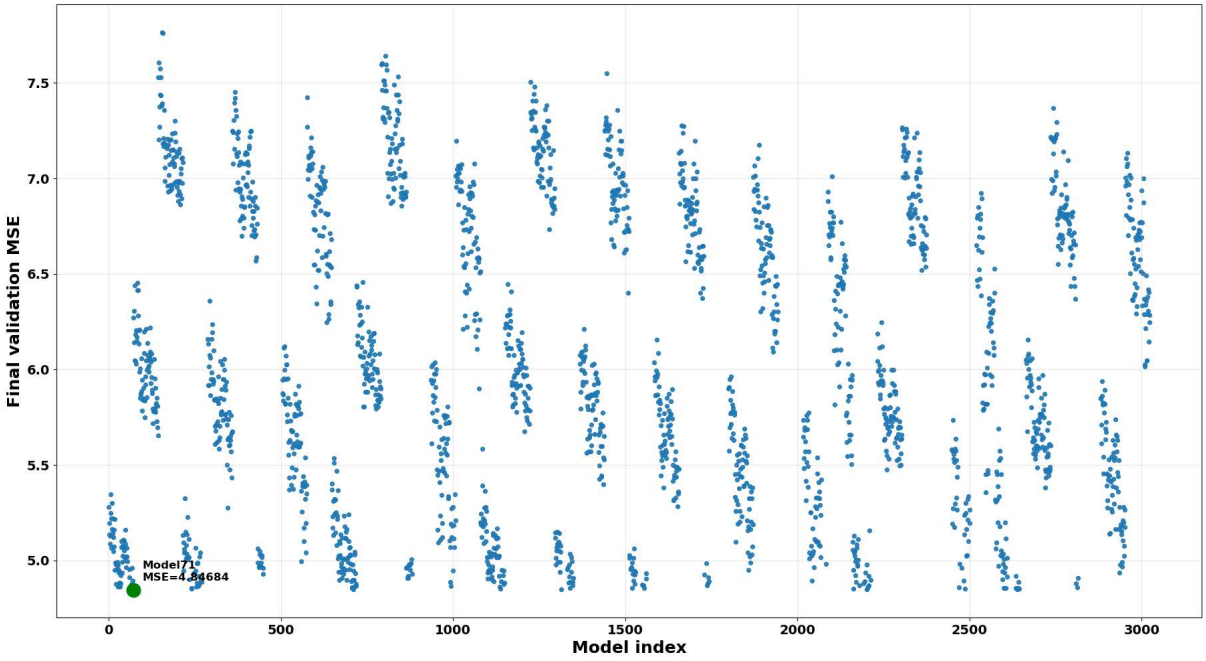


Figure 6.12: MSE values for the CNN model using muon momenta 180 GeV, for all grid-search configurations.

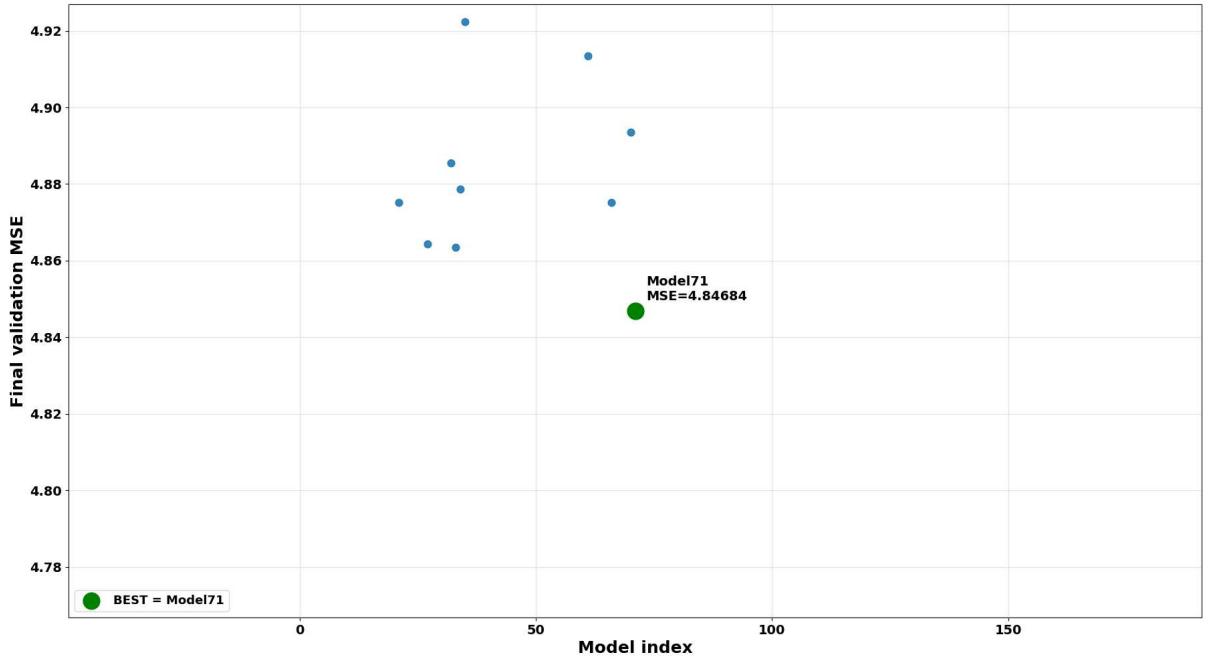


Figure 6.13: Zoomed-in scatter plot focusing on the region with the lowest mean sqaure error values for 180 GeV muon momenta

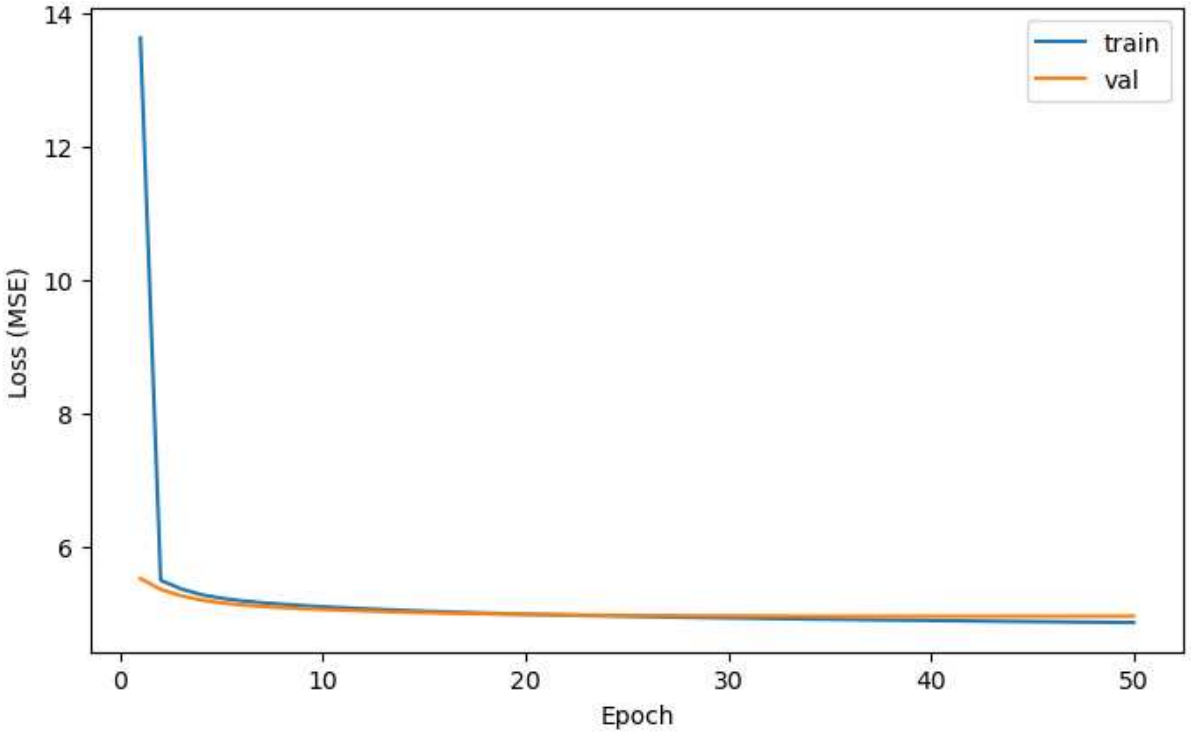


Figure 6.14: The plot shows the loss as a function of epochs for the 180 GeV muon momentum sample. The loss decreases rapidly in the initial epochs and then becomes nearly stable, indicating convergence of the training process.

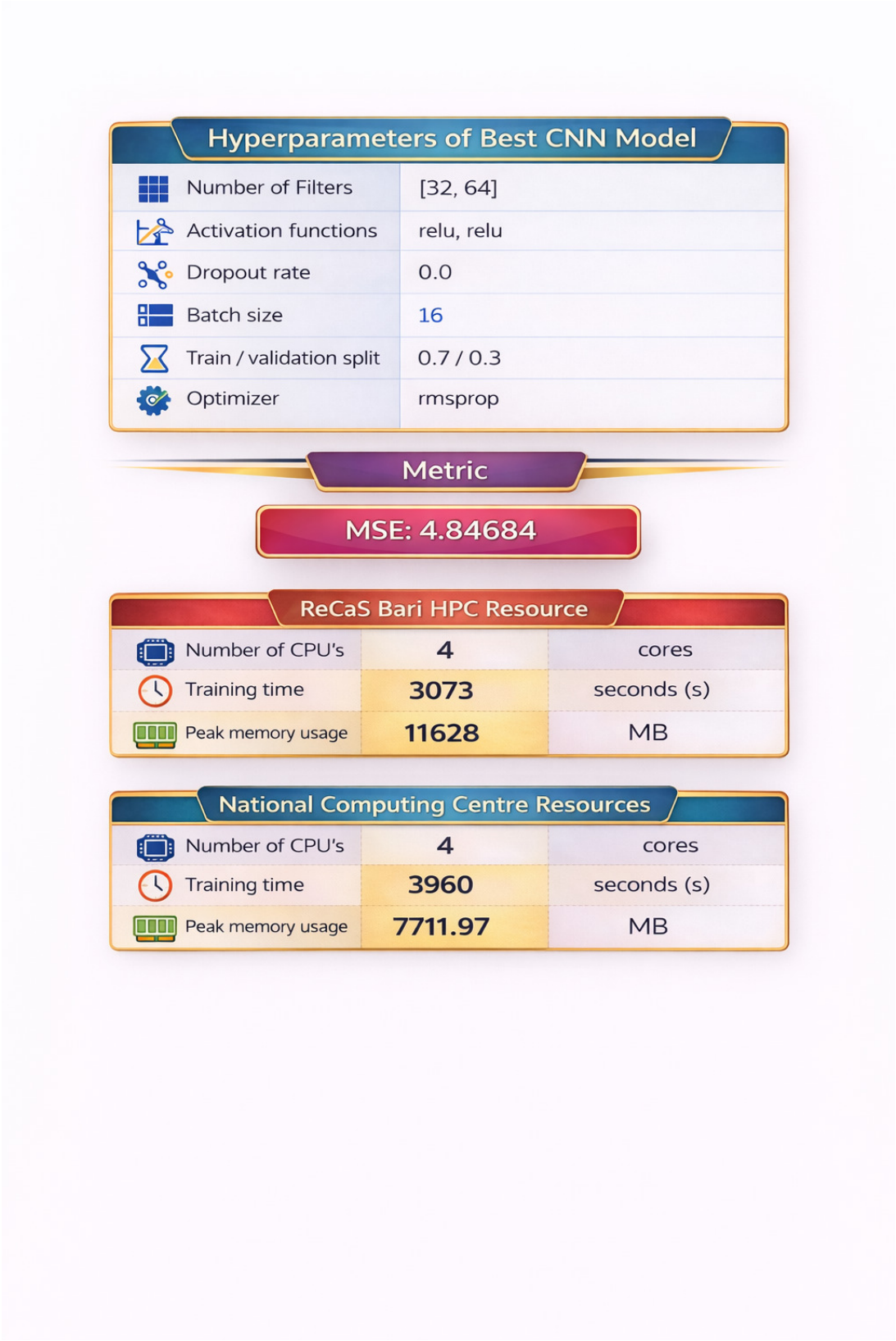


Table 6.7: Hyperparameters and HPC resource usage for the best CNN model selected based on the lowest MSE on 180 GeV momentum.

6.5 Monte Carlo Truth and Final Reconstructed Results of Neural Network Models

This section presents the final results for the mean number of primary clusters from **Monte Carlo (MC)** truth and **neural network (NN)** reconstruction across lower momenta (2, 4, 6, 8 and 10 GeV) and higher momentum(180 GeV) of muon, pion and kaon. The MC truth is generated directly with `Garfield++` and is used as the reference. We compare it with the primary-cluster distributions reconstructed by the LSTM and CNN model.

In this study, we compare **three distributions** of the number of primary clusters:

- **MC truth (`Garfield++`):** the distribution of the number of primary clusters produced by the detector simulation based on Garfield ++.
- **LSTM reconstruction:** the distribution of the number of primary clusters reconstructed by the LSTM model (classification), matched with MC truth.
- **CNN reconstruction:** the distribution of the number of primary clusters reconstructed by the CNN model (regression) using the total number of detected peaks from the peak-finding algorithm.

6.5.1 MC and Neural Network Reconstruction for 180 GeV Muons

We first present the results of MC and final reconstruction by Neural Network of **180 GeV** muon. The distributions (figure 6.15-6.17) show the mean number of primary clusters for 180 GeV muons, including the **MC truth** primary cluster, the number of primary clusters reconstructed by **CNN**, and the **LSTM matched to MC truth** using the simulation parameters shown in table 6.1.

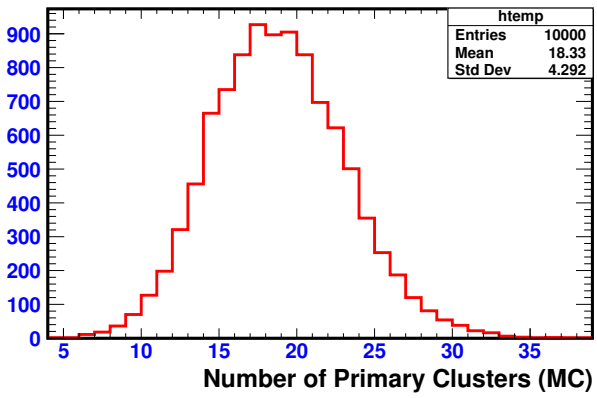


Figure 6.15: Above distrinution shows the number of Primary Clusters (MC) generated directly based on garfield++ for 180 GeV muon momentum

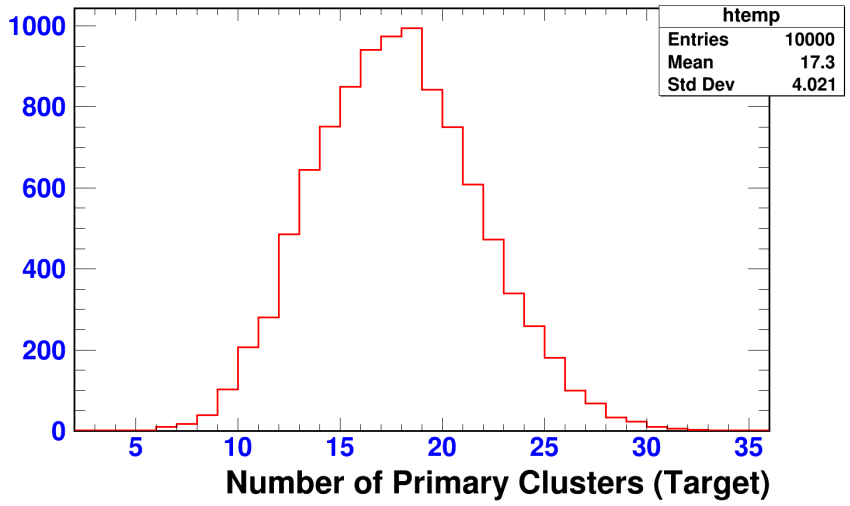


Figure 6.16: Above distributions shows the number of primary clusters reconstructed by an LSTM model matched with MC truth for 180 GeV muon momentum

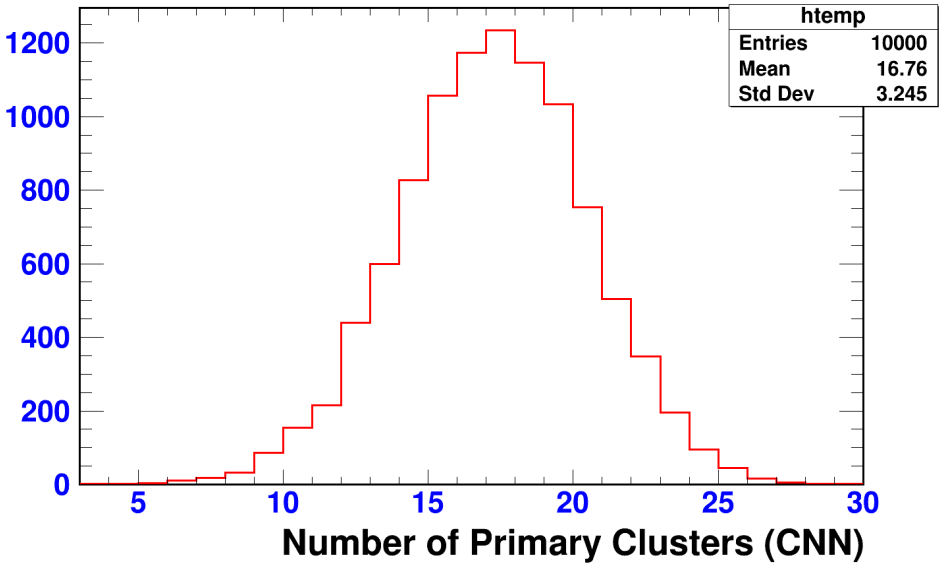


Figure 6.17: Above distribution shows the number of Primary Clusters reconstructed by CNN model for 180 GeV muon momentum

6.5.2 MC and Neural Network Reconstruction for Lower Momenta

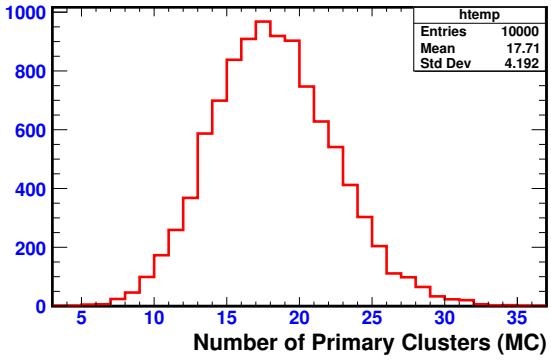
After validating the reconstruction at 180 GeV, we extend the comparison to the **lower-momentum region**. The final results for the MC truth and neural network-based reconstruction are presented for muons, pions, and kaons at **2, 4, 6, 8, and 10 GeV**. The distributions below show the mean number of primary clusters, including the **MC truth**, the mean number of primary clusters reconstructed by **CNN**, and the **LSTM matched to MC truth** using the simulation parameters shown in table 6.1.

Figures 6.18–6.22 show a comparison between the number of primary clusters from Monte Carlo (MC) simulation, the number of primary clusters reconstructed using the LSTM model matched to the MC truth, and the number of primary clusters reconstructed using the CNN regression model for muon momenta of 10, 8, 6, 4, and 2 GeV.

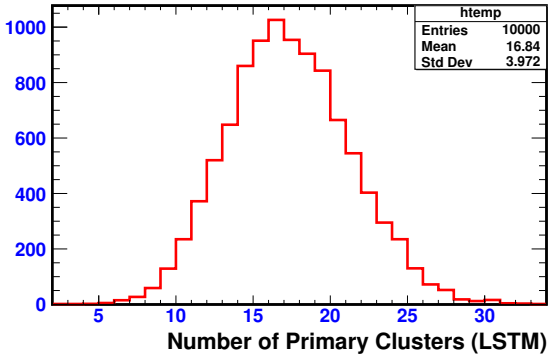
Figures 6.23–6.27 show a comparison between the number of primary clusters from Monte Carlo (MC) simulation, the number of primary clusters reconstructed using the LSTM model matched to the MC truth, and the number of primary clusters reconstructed using the CNN regression model for pion momenta of 10, 8, 6, 4, and 2 GeV.

Figures 6.28–6.32 show a comparison between the number of primary clusters from Monte Carlo (MC) simulation, the number of primary clusters reconstructed using the LSTM model matched to the MC truth, and the number of primary clusters reconstructed using the CNN regression model for kaon momenta of 10, 8, 6, 4, and 2 GeV.

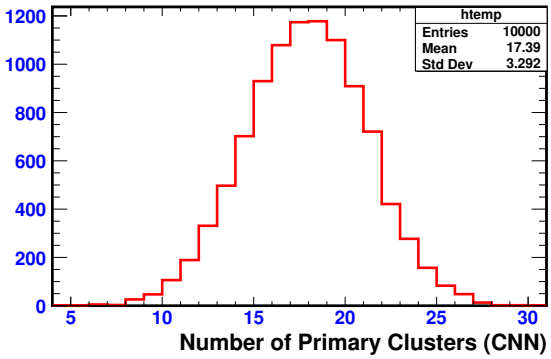
Overall the mean number of primary clusters reconstructed by best selected neural network models (LSTM and CNN) using HPC resources are close to the number of primary clusters (MC) generated based on Garfield++.



(a) Monte Carlo truth (MC)

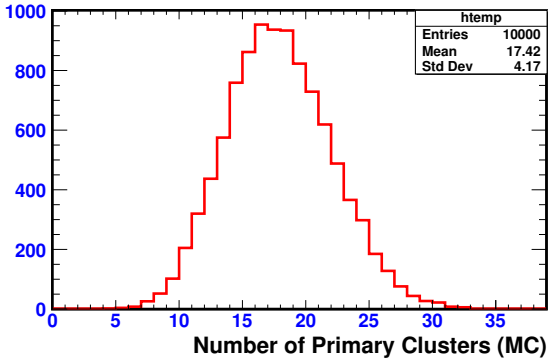


(b)Number of primary clusters reconstructed by an LSTM matched with MC

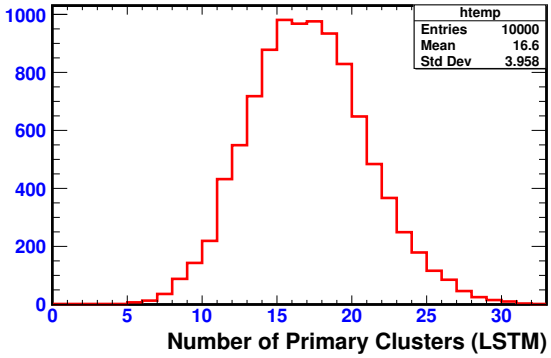


(c) Number of primary clusters reconstructed by CNN

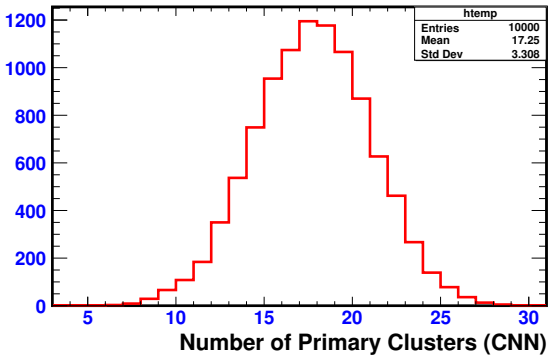
Figure 6.18: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a muon momentum of 10 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

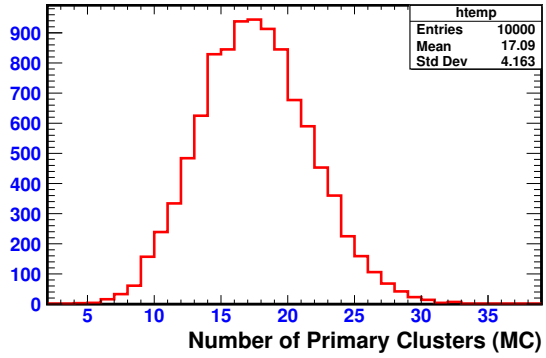


(b)Number of primary clusters reconstructed by an LSTM matched with MC

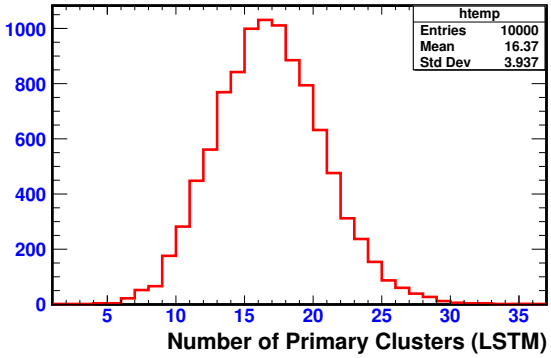


(c) Number of primary clusters reconstructed by CNN

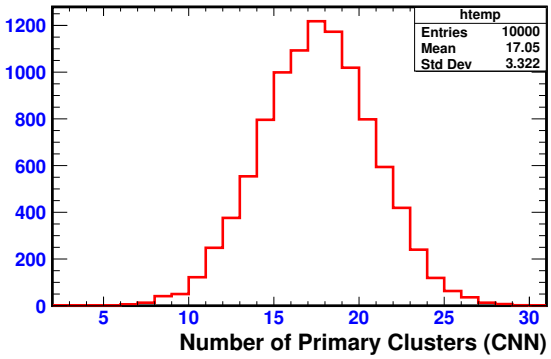
Figure 6.19: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a muon momentum of 8 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

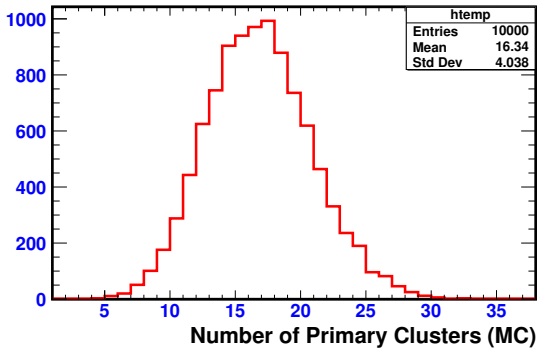


(b)Number of primary clusters reconstructed by an LSTM matched with MC

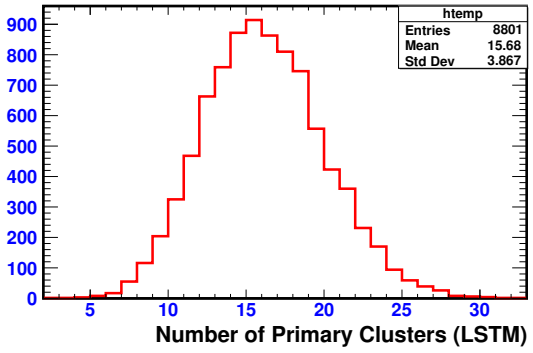


(c) Number of primary clusters reconstructed by CNN

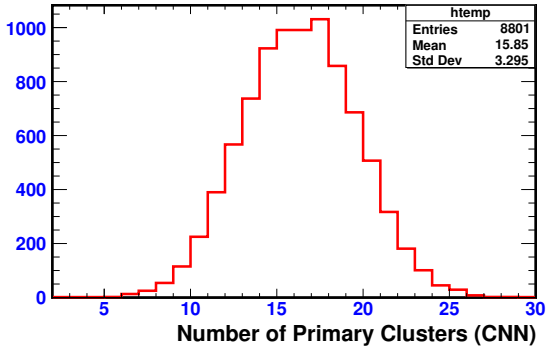
Figure 6.20: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a muon momentum of 6 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

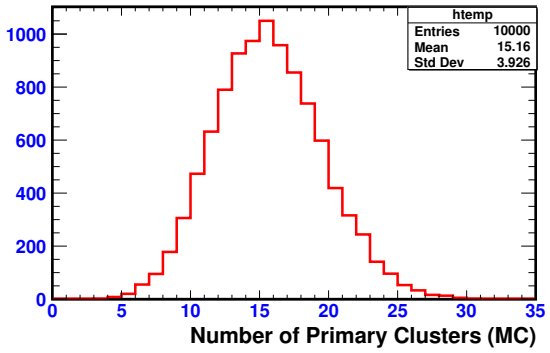


(b) LSTM reconstruction of primary clusters matched with MC

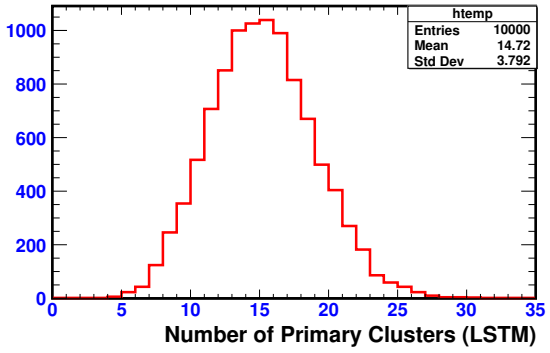


(c) CNN clusterization reconstruction

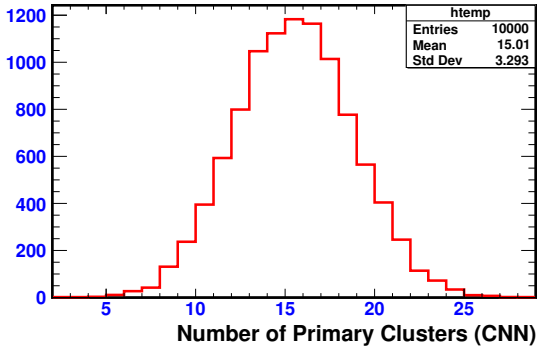
Figure 6.21: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a muon momentum of 4 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

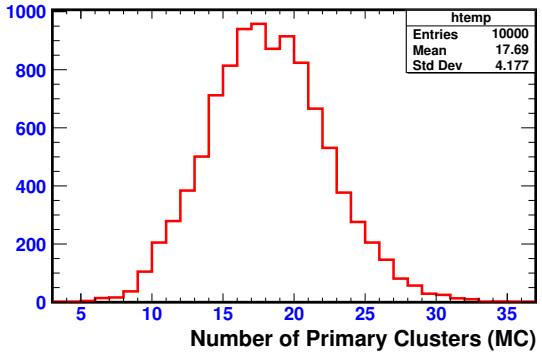


(b) LSTM reconstruction of primary clusters matched with MC

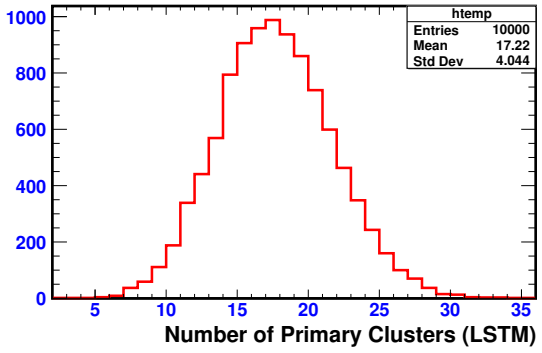


(c) CNN clusterization reconstruction

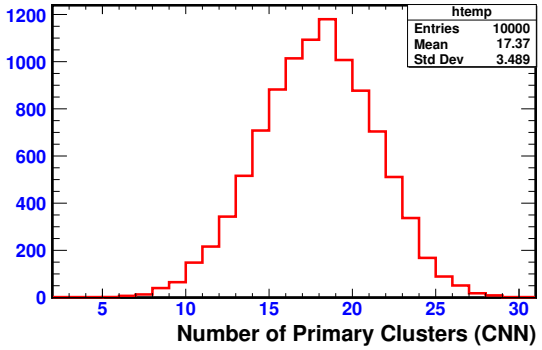
Figure 6.22: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a muon momentum of 2 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

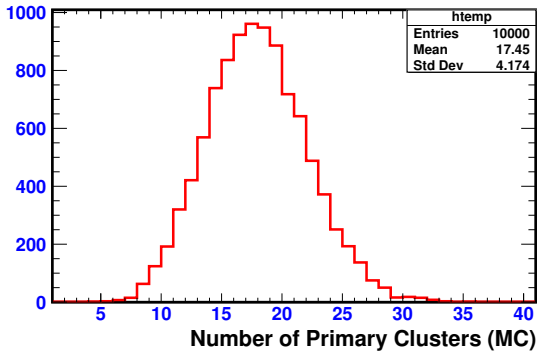


(b) LSTM reconstruction of primary clusters matched with MC

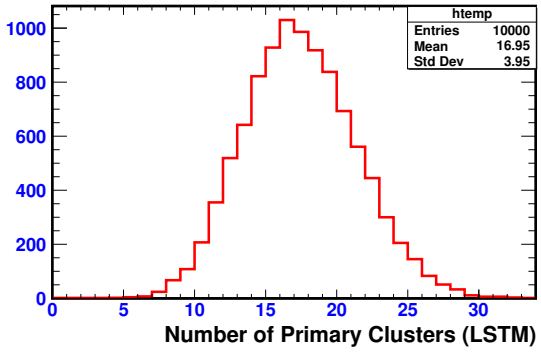


(c) CNN clusterization reconstruction

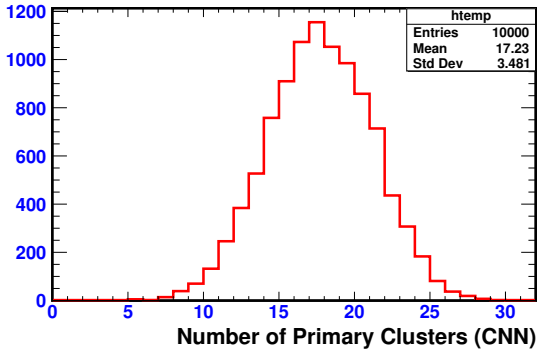
Figure 6.23: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a pion momentum of 10 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

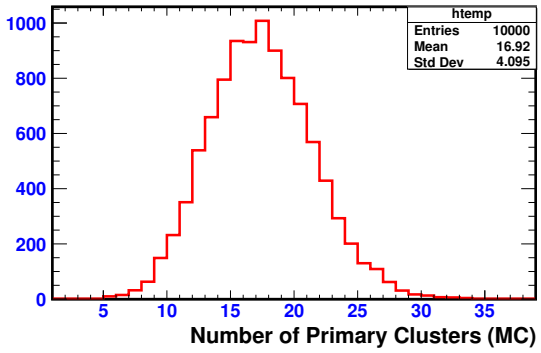


(b) LSTM reconstruction of primary clusters matched with MC

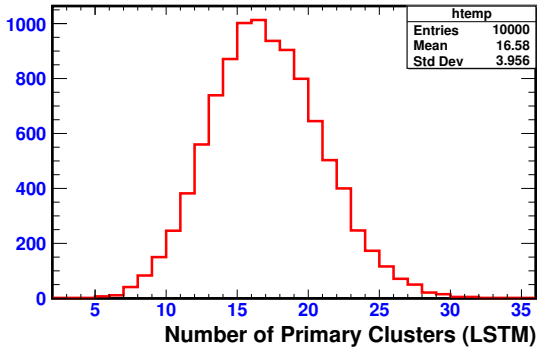


(c) CNN clusterization reconstruction

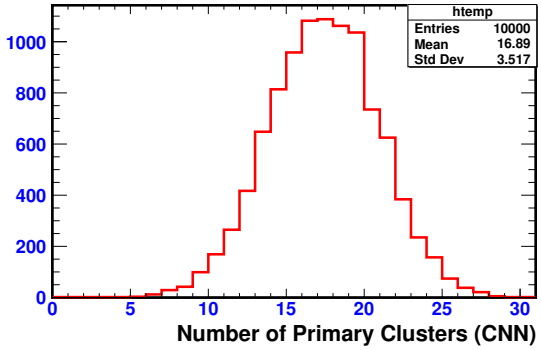
Figure 6.24: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a pion momentum of 8 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

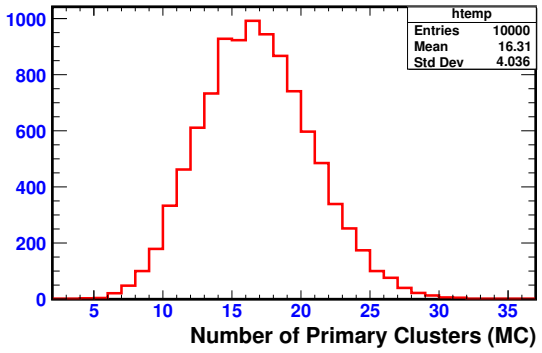


(b) LSTM reconstruction of primary clusters matched with MC

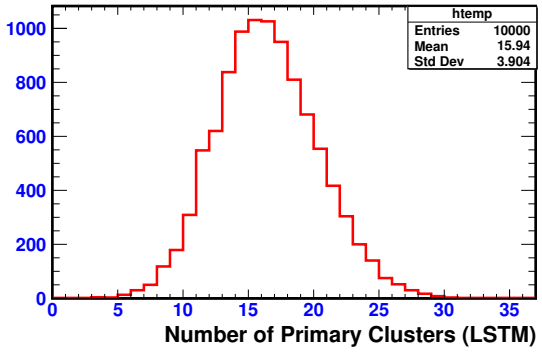


(c) CNN clusterization reconstruction

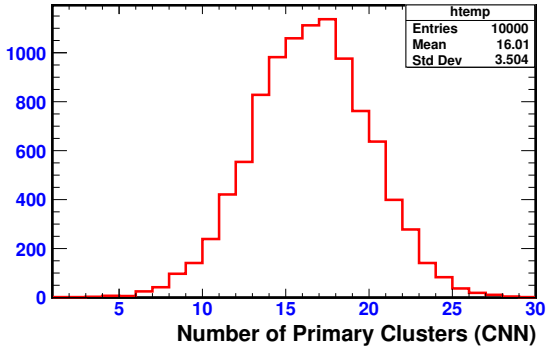
Figure 6.25: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a pion momentum of 6 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

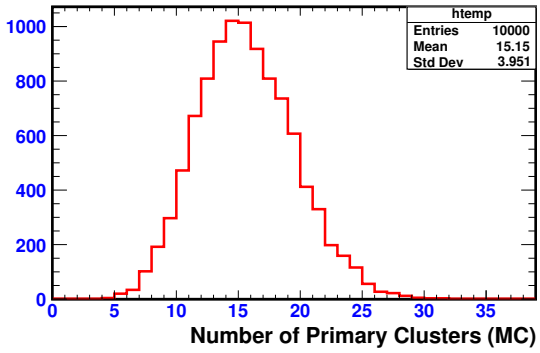


(b) LSTM reconstruction of primary clusters matched with MC

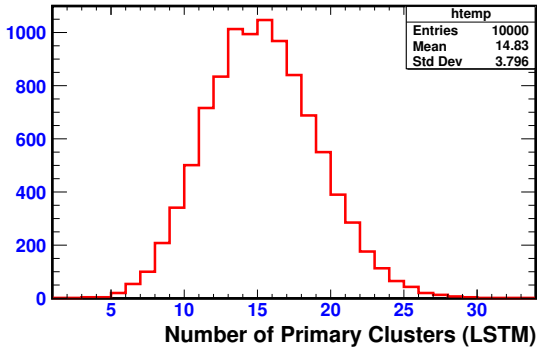


(c) CNN clusterization reconstruction

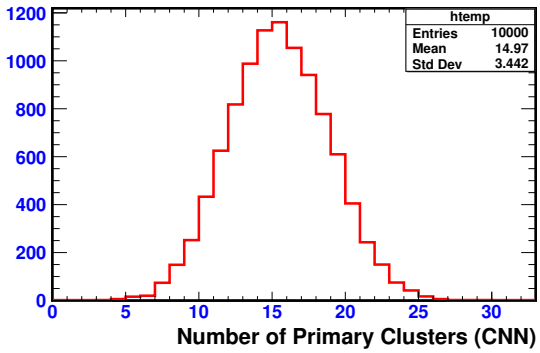
Figure 6.26: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a pion momentum of 4 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

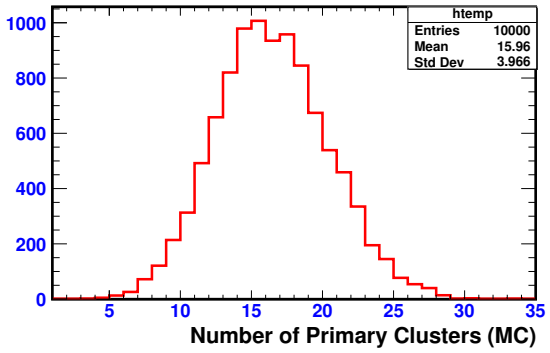


(b) LSTM reconstruction of primary clusters matched with MC

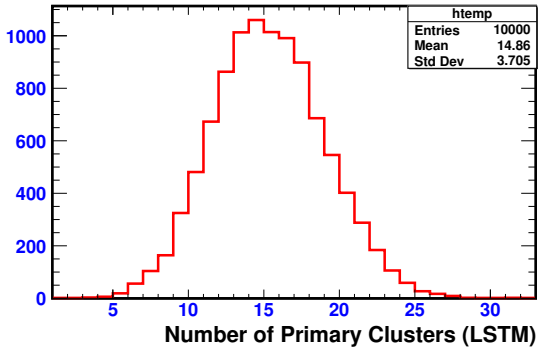


(c) CNN clusterization reconstruction

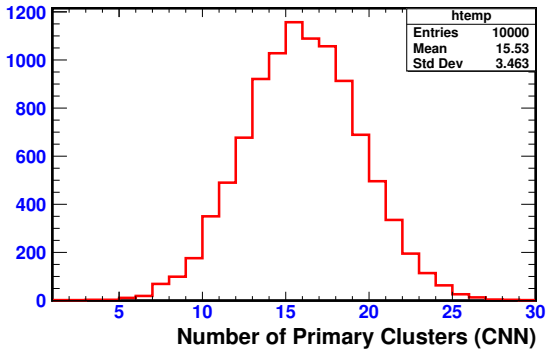
Figure 6.27: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a pion momentum of 2 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

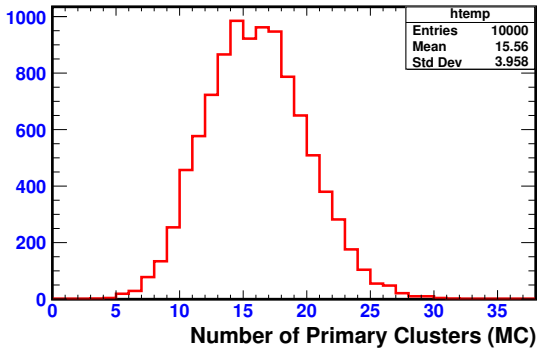


(b) LSTM reconstruction of primary clusters matched with MC

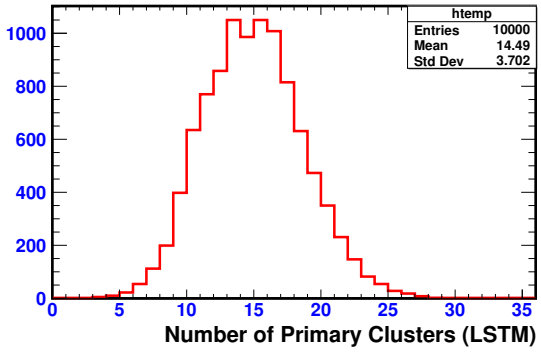


(c) CNN clusterization reconstruction

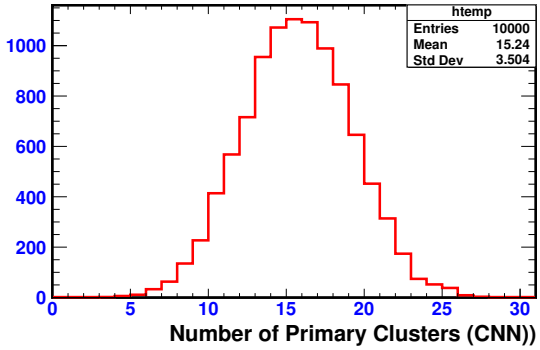
Figure 6.28: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a kaon momentum of 10 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

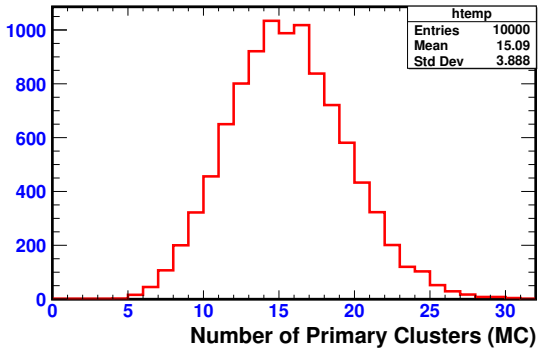


(b) LSTM reconstruction of primary clusters matched with MC

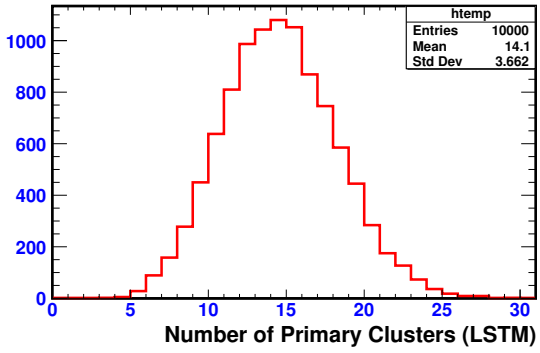


(c) CNN clusterization reconstruction

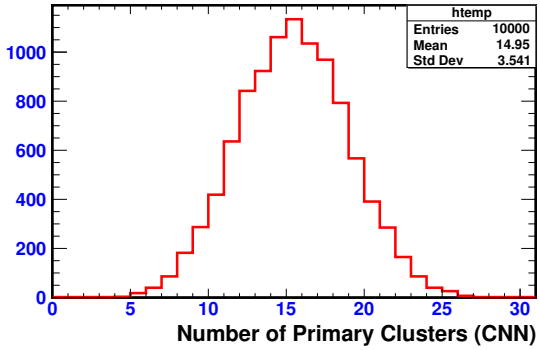
Figure 6.29: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a kaon momentum of 8 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

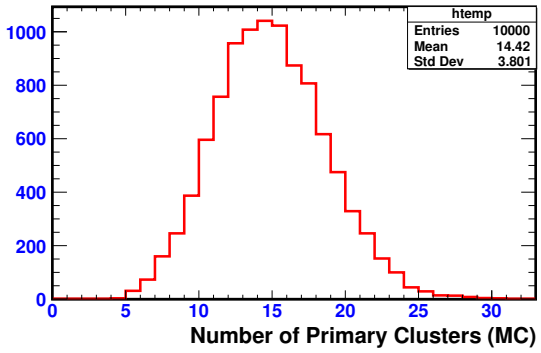


(b) LSTM reconstruction of primary clusters matched with MC

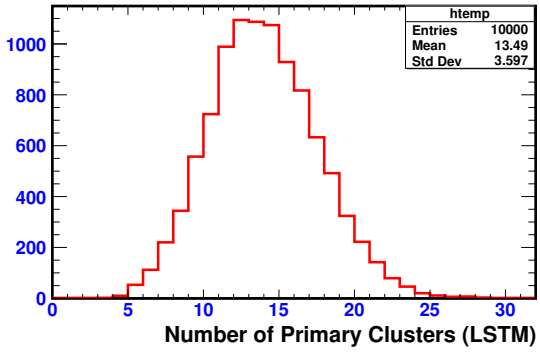


(c) CNN clusterization reconstruction

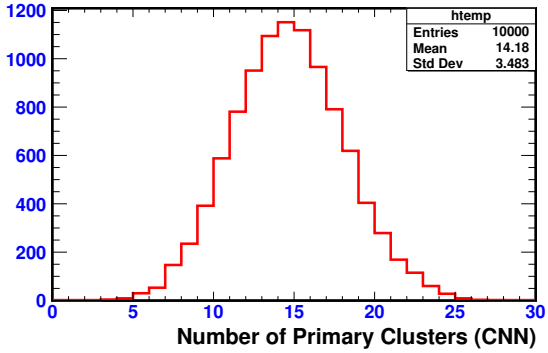
Figure 6.30: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a kaon momentum of 6 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)

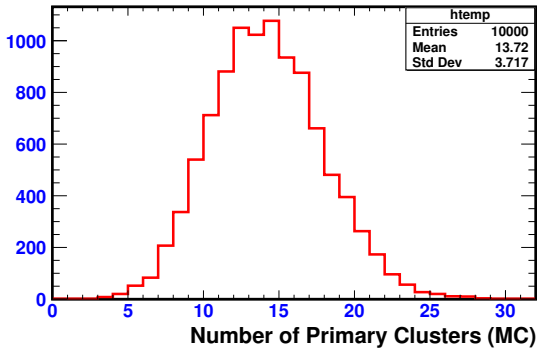


(b) LSTM reconstruction of primary clusters matched with MC

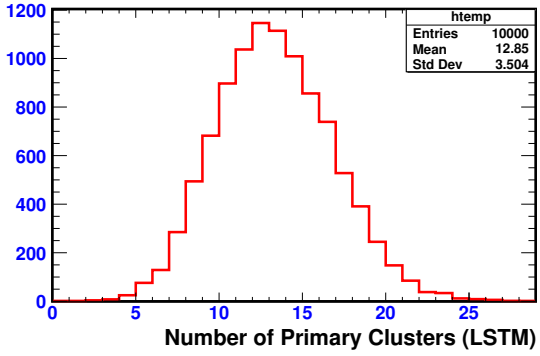


(c) CNN clusterization reconstruction

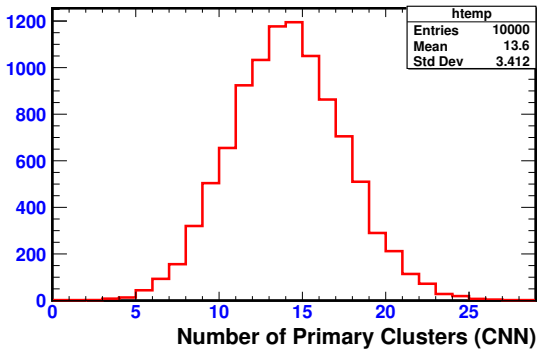
Figure 6.31: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a kaon momentum of 4 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.



(a) Monte Carlo truth (MC)



(b) LSTM reconstruction of primary clusters matched with MC



(c) CNN clusterization reconstruction

Figure 6.32: The above distributions show a comparison of the number of primary clusters from Monte Carlo simulation (MC), the number of primary clusters reconstructed using an LSTM model matched with MC truth, and the number of primary clusters reconstructed using a CNN regression model at a kaon momentum of 2 GeV. The three distributions exhibit similar Gaussian-like behavior, with mean values that are closely aligned with the MC truth.

Summary: To make the comparison easier and more quantitative, the mean number of primary clusters from MC, LSTM, and CNN for all tested muon, pion, and kaon momenta are summarized in the tables below. Tables 6.8–6.10 directly answer the main question of this section: how close the numbers of primary clusters reconstructed by the LSTM and

CNN models are to the MC truth in terms of the central value (mean) for muons, pions, and kaons in the momentum range 2–10 GeV.

Number of Primary Clusters for Kaon

Kaon momentum	MC (Mean, Std Dev)	LSTM (Mean, Std Dev)	CNN (Mean, Std Dev)
10 GeV	15.96, 3.966	14.86, 3.705	15.53, 3.463
8 GeV	15.56, 3.958	14.49, 3.702	15.24, 3.504
6 GeV	15.09, 3.888	14.10, 3.662	14.95, 3.541
4 GeV	14.42, 3.801	13.49, 3.597	14.18, 3.483
2 GeV	13.72, 3.717	12.85, 3.504	13.60, 3.412

Table 6.8: Mean number of primary clusters of Monte Carlo (MC), LSTM, and CNN for kaon momenta of 10, 8, 6, 4, and 2 GeV.

Number of Primary Clusters for Pion

Pion momentum	MC (Mean, Std Dev)	LSTM (Mean, Std Dev)	CNN (Mean, Std Dev)
10 GeV	17.69, 4.177	17.22, 4.044	17.37, 3.489
8 GeV	17.45, 4.174	16.95, 3.95	17.23, 3.481
6 GeV	16.92, 4.095	16.58, 3.956	16.89, 3.517
4 GeV	16.31, 4.038	15.94, 3.904	16.01, 3.504
2 GeV	15.15, 3.951	14.83, 3.796	14.97, 3.442

Table 6.9: Mean number of primary clusters of Monte Carlo (MC), LSTM, and CNN for pion momenta of 10, 8, 6, 4, and 2 GeV.

Number of Primary Clusters for Muon

Muon momentum	MC (Mean, Std Dev)	LSTM (Mean, Std Dev)	CNN (Mean, Std Dev)
180 GeV	18.33, 4.292	17.30, 4.021	16.76, 3.245
10 GeV	17.71, 4.192	16.84, 3.972	17.39, 3.292
8 GeV	17.42, 4.17	16.60, 3.958	17.25, 3.308
6 GeV	17.09, 4.163	16.37, 3.937	17.05, 3.322
4 GeV	16.34, 4.038	15.68, 3.867	15.85, 3.295
2 GeV	15.16, 3.926	14.72, 3.792	15.01, 3.293

Table 6.10: Mean number of primary clusters of Monte Carlo (MC), LSTM, and CNN for muon momenta of 10, 8, 6, 4, and 2 GeV.

6.6 Separation Power Comparison: Cluster Counting vs. Energy Loss

Particle identification performance is quantified using the separation power between two particle species. In this subsection, the separation study is performed for pion and kaon.

The separation power S is defined as

$$S = \frac{|\mu_A - \mu_B|}{(\sigma_A + \sigma_B)/2}, \tag{6.2}$$

where μ_A and μ_B represent the mean number of primary clusters of the PID observable for the two particle species, while σ_A and σ_B denote their corresponding standard deviation.

A larger value of S indicates stronger separation and better particle identification capability.

Two different algorithms are compared:

- Traditional energy loss measurement (dE/dx)
- Cluster counting measurement (dN/dx)

The conventional dE/dx method estimates the deposited energy along the track. However, this measurement is strongly affected by Landau fluctuations and secondary ionization effects, which degrade PID performance as shown.

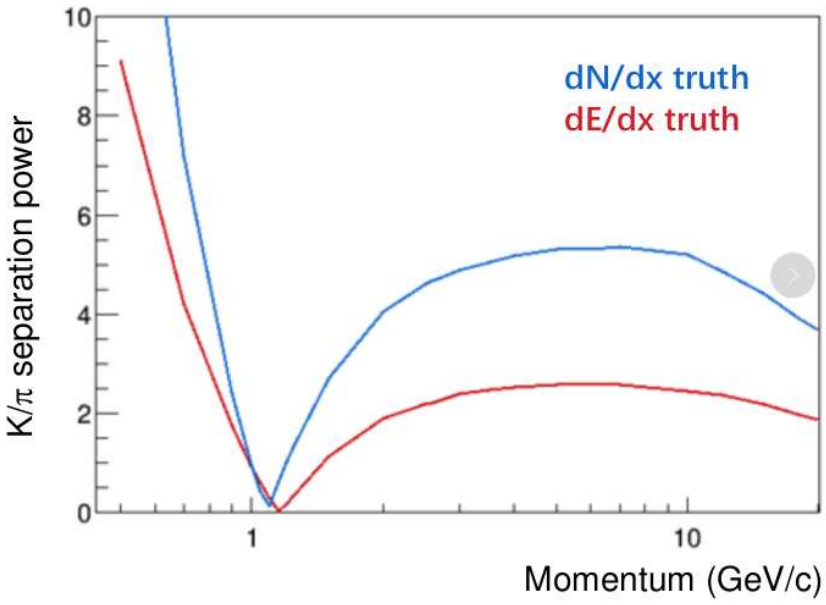


Figure 6.33: Kaon–pion separation power obtained using cluster counting (dN/dx) compared with the traditional energy-loss method (dE/dx) as a function of momentum.

In contrast, the cluster counting method measures the number of primary ionization clusters per unit length. Since primary clusters follow Poisson statistics, the fluctuation level is significantly reduced, leading to improved resolution.

Figure 6.33 represents the separation power as a function of momentum. The cluster counting algorithm consistently outperforms than the traditional energy-loss method.

In the momentum range from 2 GeV to 10 GeV, which is the focus of this study, the improvement is clearly visible. The dN/dx curve remains significantly two times higher than the dE/dx curve, demonstrating stronger pion–kaon discrimination.

This performance gain confirms that cluster counting provides a major advancement in particle identification. By reducing ionization fluctuations and focusing on primary clusters, the method enhances detector PID capability without requiring hardware modifications.

Therefore, cluster counting can be considered a breakthrough technique for next-generation drift chamber detectors.

6.6.1 Separation Power for Particle Identification (dN/dx VS CNN)

In addition to that, now the final objective of cluster counting is particle identification (PID). In this part, the separation capability is evaluated between muons and kaons by comaprison of the mean number of primary clusters and its resolution based on Monte carlo and reconstructed number of primary clusters by neural network model (CNN), denoted as (dN/dx).

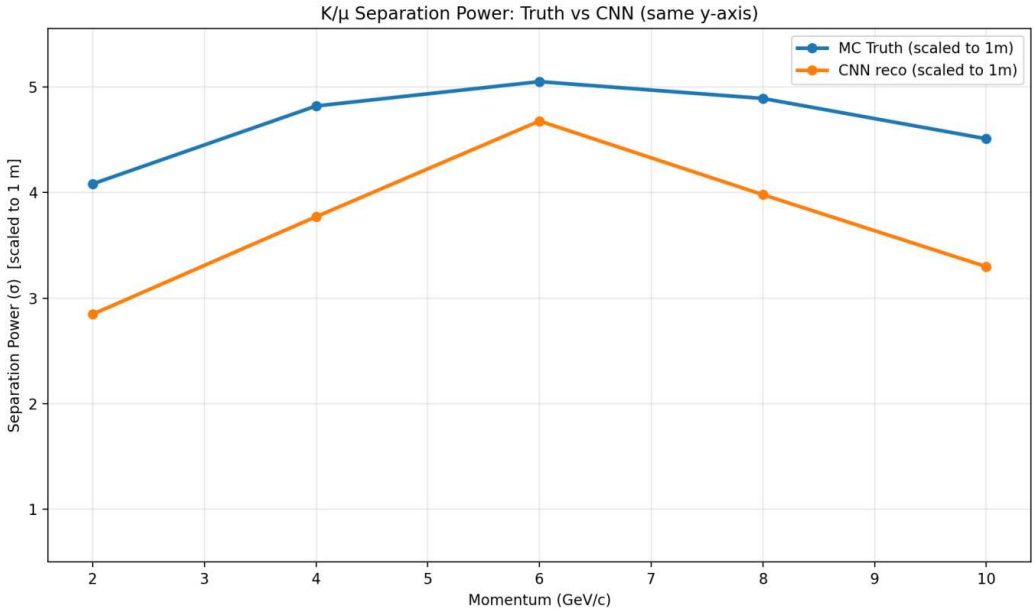


Figure 6.34: Particle identification separation power using cluster counting (dN/dx) of Monte Carlo (MC) and Neural Netwok Model (CNN) for different momenta of muon and kaon (2-10 GeV).

The separation power S is defined as

$$S = \frac{\left| \left(\frac{dN}{dx}\right)_{\mu} - \left(\frac{dN}{dx}\right)_{K} \right|}{(\sigma_{\mu} + \sigma_{K})/2}. \tag{6.3}$$

Here, $\left(\frac{dN}{dx}\right)_{\mu}$ and $\left(\frac{dN}{dx}\right)_{K}$ represent the mean number of primary clusters for muons and kaons, while σ_{μ} and σ_{K} denote their corresponding standard deviation. A larger value of S indicates better particle separation and improved PID performance.

The separation study is performed over the momentum range from 2 GeV to 10 GeV. In figure 6.34 clearly it showed that seperation power of muon and kaon for monte carlo (blue curve) and Neural network CNN model (green curve) are close and look reasonable.

In addition to that, figure 6.35 clearly it showed that seperation power of pion and kaon for monte carlo (blue curve) and Neural network CNN model (green curve) are close and look reasonable.

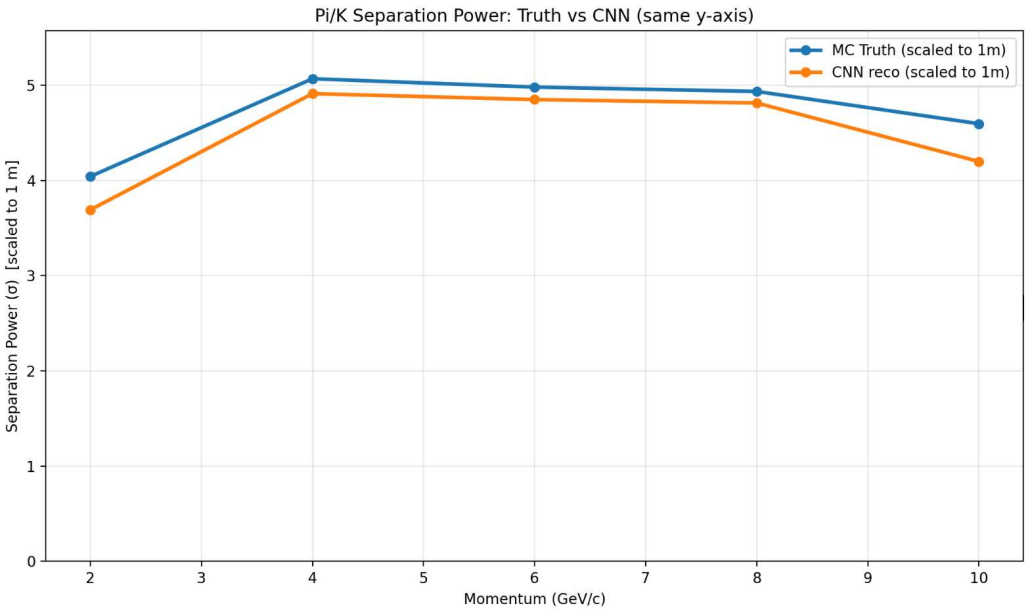


Figure 6.35: Particle identification separation power using cluster counting (dN/dx) of Monte Carlo (MC) and Neural Netwok Model (CNN) for different momenta of pion and kaon (2-10 GeV).

Chapter 7

Analysis of Real Test Beam Data

This chapter presents the analysis of real test-beam data, with the main focus on reconstructing the number of primary clusters from digitized waveforms using machine-learning-based algorithm and comparing the results with those obtained from the Running Template Algorithm (RTA). To achieve this, the simulated samples were generated using the same experimental conditions as the real test-beam data, including the cell size, muon momentum, angle between the muon track and the sense wire, and gas mixture etc. These samples were then tuned in terms of noise and maximum signal amplitude to improve the agreement between simulation and data

After the tuning procedure, the LSTM and CNN models are trained on the tuned simulated samples for peak finding and clusterization, respectively. The best LSTM peak-finding model is selected based on the highest area under the curve (AUC), while the best CNN clusterization model is chosen based on the lowest mean squared error (MSE) among all tested configurations. Finally, the selected LSTM model for peak finding and the selected CNN model for clusterization are applied to the real test-beam data to perform cluster counting. In particular, the final number of primary clusters reconstructed by the CNN is compared directly with the result obtained from the Running Template Algorithm (RTA).

7.1 Beam Test Overview

The beam test setup used in 2021, shown in Figure 7.1, was built as a complete detection and readout system. The same general setup was also used in the later beam test campaigns. Figure 7.2 (left) shows the drift tubes used in the beam tests. The sense wires of the drift tubes were connected to a high-voltage (HV) supply card at one end. At the other end, an HV decoupling termination card with an impedance of 330 Ω was used to match the characteristic impedance of the drift tubes. Even though the coaxial cables were relatively long, 40 cm in 2021 and 60 cm in 2023, no preamplifier was used.

The trigger system selected events by using a coincidence of scintillators coupled to

silicon photomultipliers (SiPMs). This allowed precise event selection. The detector was operated with a well-controlled gas mixture. The gas flow and gas mixing were regulated by mass flowmeters under normal temperature and pressure (NTP) conditions, which helped keep the detector response stable. The portable gas system used for gas-flow monitoring and gas mixing is shown in Figure 7.2 (right).

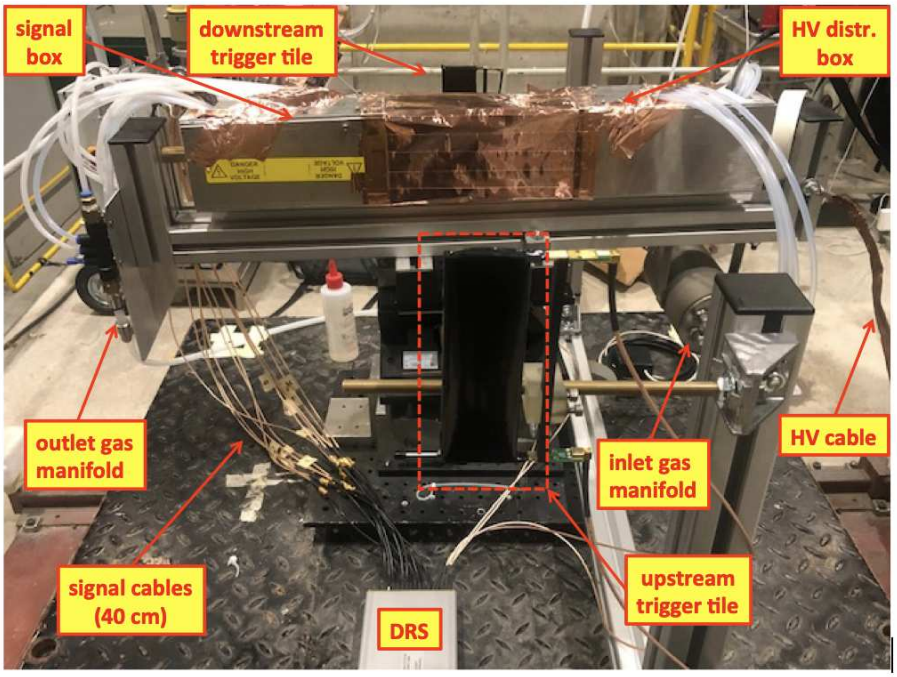


Figure 7.1: Experimental setup used for the 2021 beam test, viewed from the beam upstream side. The labeled components include the signal box, downstream trigger tile, HV distribution box, outlet gas manifold, signal cables (40 cm), DRS, inlet gas manifold, upstream trigger tile, and HV cable.

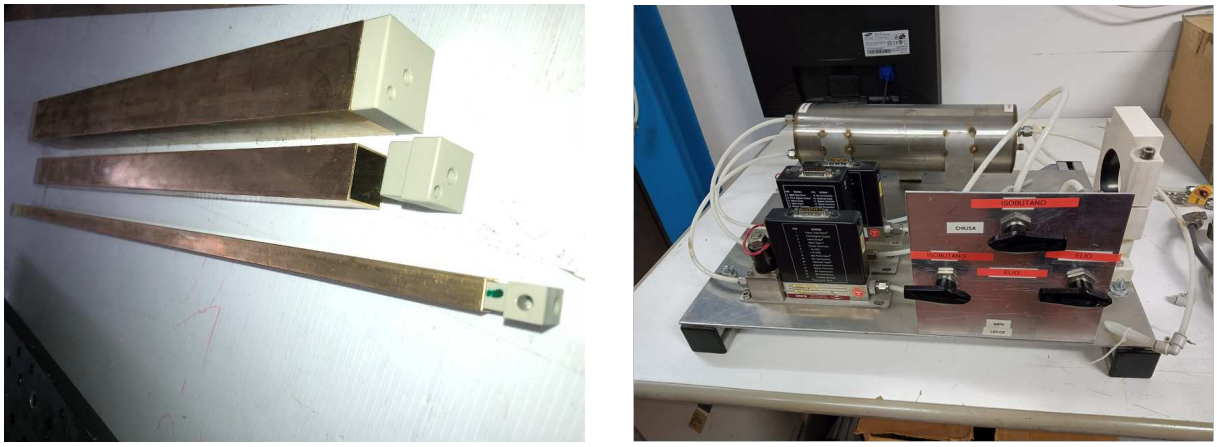


Figure 7.2: Left: set of drift tubes with different cell sizes used in the beam tests. Right: portable gas system used for gas-flow monitoring and gas mixing.

The main data acquisition unit was the Wave Dream Board (WDB)[76], shown in Figure 7.3 (left). The WDB has 16 channels and provides variable gain amplification and

programmable pole-zero cancellation. These features were available, but they were not used during the beam tests. The board contains two onboard DRS4 chips, and each chip is connected to an 8-channel ADC. This allows digitization at 80 MSPS with 12-bit resolution. In transparent mode, the DRS4 continuously samples the input signals with an analog ring buffer at up to 5 GSPS. At the same time, an FPGA applies trigger algorithms based on threshold cuts on the summed input signals. The WDB can operate autonomously, the data are read out through Gigabit Ethernet, and an ultra-low-noise bias voltage generator provides stable operation for the SiPMs.

This setup gave several important advantages. Muon beams with different $\beta\gamma$ values made it possible to study particle identification (PID) over a wide momentum range. The stacked metal drift tubes had a low material budget, which reduced multiple scattering and allowed more precise ionization measurements. External tracking systems were not needed because the drift path inside the tube was already well defined. The setup also removed the need for internal tracking, so extra steps such as t_0 calibration, alignment, and time-to-distance conversion were not required. As a result, the analysis could focus directly on counting ionization clusters in the time domain, which simplified the measurement procedure.

Figure 7.3 (right) shows an event display from the WDB, which works in a way similar to an oscilloscope. The first four channels correspond to the trigger scintillators, which are read out by SiPMs and internally processed by the WDB. The next six channels record signals from a row of 1 cm drift tubes crossed by the same muon. The control panel on the right side of the display gives access to channel settings such as gain, threshold, bias voltage, sampling rate, and trigger definition.

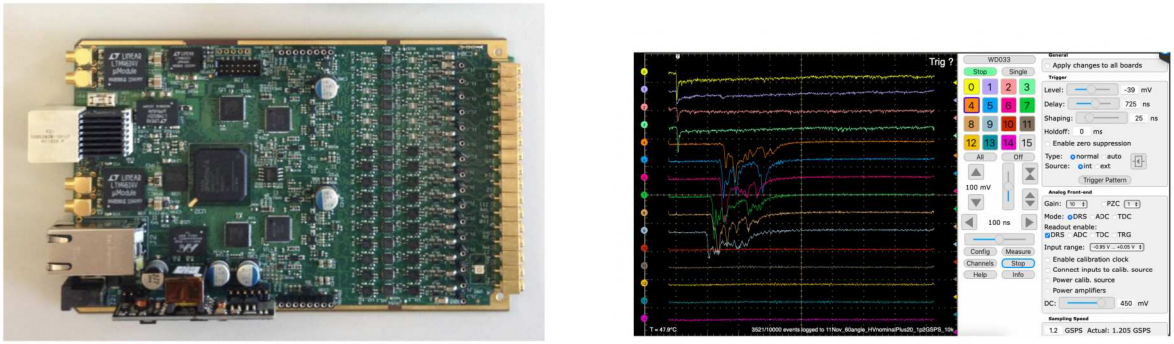


Figure 7.3: Left: Wave Dream Board (WDB). Right: event display and control panel of the WDB. The panel allows the configuration of channel settings such as gain, threshold, bias voltage, sampling rate, and trigger definition.

Besides comparing the PID performance of the traditional dE/dx method and the cluster-counting dN/dx method, the beam tests were also designed to study the basic properties of cluster counting and to improve the method. One main goal was to confirm the Poisson nature of cluster counting and to study how it depends on the $\beta\gamma$ of the

ionizing particle. Another goal was to find the most effective cluster-counting algorithms by studying several parameters, including detector geometry, gas mixtures, gas gain, electric field strength, and the performance of the front-end and digitization electronics. The study also examined possible effects that could reduce cluster-counting efficiency, such as cluster size, space-charge distribution, and electron–ion recombination.

To reach these goals, several sets of square brass drift tubes were tested, with cross sections ranging from 1.0 cm to 3.0 cm. The tubes used sense wires with diameters between 10 and 40 μm and were operated at gas gains between 1 and 5×10^5. Different helium–isobutane gas mixtures, namely 90/10, 85/15, and 80/20, were used to study the effect of gas composition on the detector response. The setup was exposed to muon beams with different momenta in three beam test campaigns carried out over the last three years.

The first beam tests were performed at CERN SPS-H8 in November 2021 and July 2022. In these tests, muon beams with momenta between 40 and 180 GeV/c were used. This momentum region corresponds to the Fermi plateau. Later beam tests were carried out at CERN PS-T10 in July 2023 and July 2024, where muon beams with momenta between 2 and 10 GeV/c were used. This region corresponds to the relativistic rise. These campaigns are summarized in Figure 7.4. They were also complemented by an earlier beam test at PSI[76]. Together, these measurements provide important experimental validation of the cluster-counting technique and help improve particle-identification methods for future applications.

Figure 7.4 shows the beam test campaigns on top of a generic Bethe–Bloch curve. The figure covers the low-momentum PSI region, the CERN PS region, and the CERN SPS-H8 region. It also marks kaon momentum ranges relevant for FCC-ee and FNAL studies. In the same figure, two example drift-tube layouts used in the measurements are shown. One layout uses a tube length of 30 cm with cell sizes of 1 cm, 2 cm, and 3 cm, helium–isobutane mixtures of 90/10 and 80/20, and wire diameters of 10, 15, 20, 25, and 40 μm. The second layout also uses a 30 cm tube length and includes helium–isobutane mixtures of 80/20, 85/15, and 90/10, with wire diameters of 15, 20, and 25 μm. The figure also shows example beam conditions, including a 165 GeV muon beam with about 1500 muons per spill, and the trigger coverage used for the 40, 80, and 180 GeV runs.

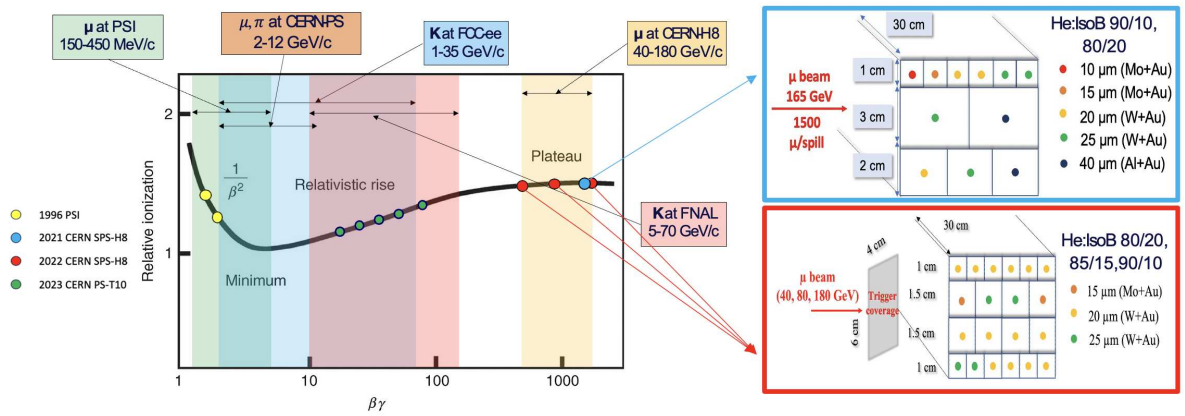


Figure 7.4: Beam test campaigns with muons at the indicated $\beta\gamma$ values superimposed on a generic Bethe–Bloch curve. The explored $\beta\gamma$ values cover the main regions relevant for particle identification studies at future lepton colliders. The figure also shows example drift-tube layouts, gas mixtures, wire diameters, and beam conditions used during the measurements.

7.2 Analysis Strategy Using ML and RTA at 180 GeV

In this work, two different algorithms are used to reconstruct the number of primary clusters from the digitized waveforms in cluster counting techniques. The first one is a machine-learning (ML) algorithm. The second one is the Running Template Algorithm (RTA). So, it is useful to introduce both algorithms before discussing their application to the 180 GeV muon beam data.

7.2.1 ML and Running Template Algorithm (RTA)

The **ML algorithm** follows a two-step strategy. In the first step, a long short-term memory (LSTM) model is used for peak finding in the digitized waveform. Its task is to detect electron peaks in the waveform and separate them from noise. This step is treated as a classification task. The detected peaks can come from both primary and secondary electrons. In the second step, a convolutional neural network (CNN) model is used for clusterization. The CNN plays a role in estimating the number of primary ionization clusters from the peaks detected in the first step. Therefore, in the ML approach, the LSTM is used as the peak-finding model for the classification task, while the CNN is used as the clusterization model for the clusterization task.. This Algorithm is explained in more detail in chapter 5.

The second reconstruction method used in this work is the **Running Template Algorithm (RTA)**. This method uses a predefined electron pulse template. The template has a rising part and a falling part, and it is obtained from experimental data. It is then digitized according to the sampling rate. The algorithm scans the waveform inside

a search window and compares the normalized template with the measured signal. A peak is identified when the agreement between the template and the waveform is larger than a predefined threshold. Once a peak is found, its contribution is subtracted from the waveform, and its position and amplitude are stored. The same procedure is then repeated over the remaining part of the waveform until no further peaks are found.

After peak finding, the RTA applies a **clusterization** step to identify the primary ionization clusters. Clusterization means grouping together all ionization electrons that come from the same ionization event. In this procedure, the peaks found in the signal waveform are associated into clusters, and each cluster is assigned a drift time and an amplitude.

The grouping follows a simple physical idea. Electron peaks that are close in time and have time differences compatible with diffusion-induced spreading are assigned to the same ionization cluster. After that, the total number of electrons inside each cluster is determined. The drift time and amplitude of the cluster are taken from the electron peak with the largest amplitude inside that group.

7.3 How to Apply NN on Data

This section focuses on the strategy to apply the neural network model on real data for the cluster counting studies. So, first, We will generate simulated samples exactly with parameters such as cell size, momentum, angle between sense wire and muon, pressure etc which are used in the real test beam data. Then we will compare both simulated samples and real data. if in case there are some discrepancies between data and simulated samples, then we need to tune the simulated samples on the behalf of noise amplitude, maximum amplitude, pulse rise time and total collected charge. Onward, we will train the NN models on the tuned simulated samples, and then we will apply the trained models on the real data for peak finding and clusterization in the digitized waveforms

The full procedure related to the application of NN models on data is summarized in the Figure 7.5

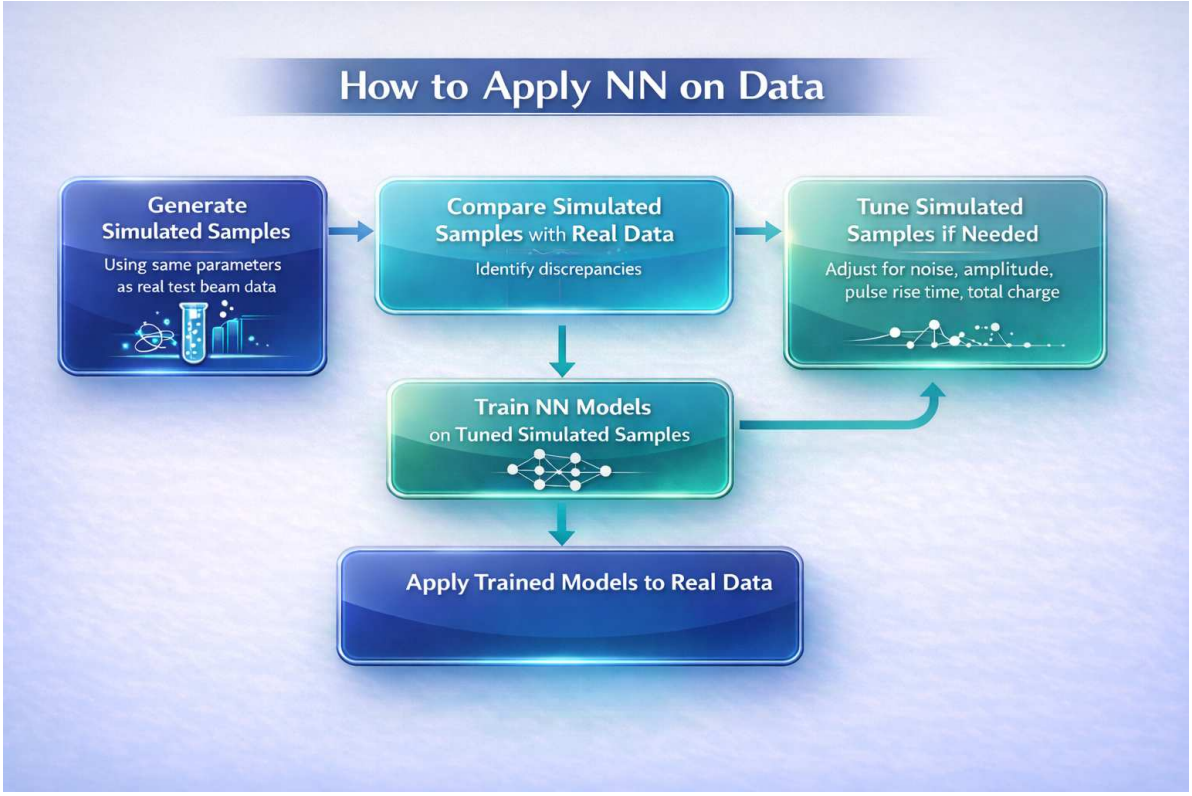


Figure 7.5: Workflow for applying neural network models to real test beam data. Real data are used as the reference. Simulated samples are generated, compared with data, and tuned it. The neural network models are then trained on tuned simulated samples and then applied to the real data .

7.3.1 Beam-Test Data Selection

Channel 5 refers to the data-acquisition readout channel associated with the 1 cm drift tube. A total of 4,832 beam-test waveforms recorded through this channel were analyzed. The detector was operated with a gas mixture of 90% helium and 10% isobutane and used a sense wire with a diameter of 20 μm. The incident muon beam had a momentum of 180 GeV and crossed the drift tube at an angle of 45° with respect to the sense wire. Following the waveform-selection procedure adopted in the beam-test study, each waveform was characterized by its maximum amplitude in the signal region, while the baseline noise was evaluated in the pre-signal region before 50 ns, where no physical pulse is expected. The noise RMS was defined from the fluctuations of the baseline around its average value. A waveform was classified as signal-like if its maximum amplitude was greater than 3 mV and its noise RMS was smaller than 1.5 mV(typically in the order of 1 mV). Waveforms that did not satisfy these conditions and did not show any visible physical peak were classified as pure-noise events.

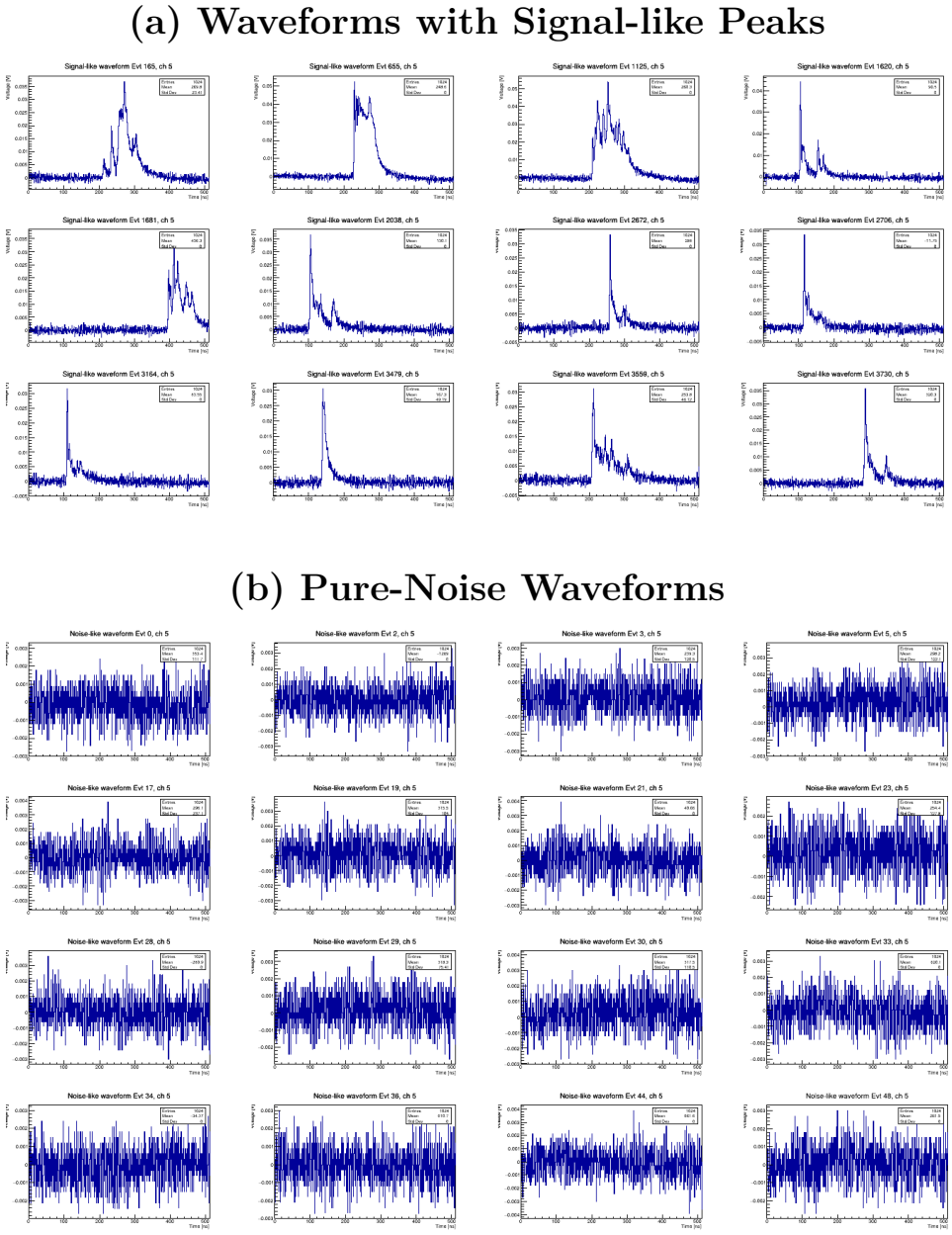


Figure 7.6: Above: waveforms with signal-like peaks. Below: pure-noise waveforms after the selection cut.

7.3.2 Noise Amplitude

The first tuning step was based on the measurement and comparison of the electronics noise in the simulation and the real data. To do this, the frequency response of the noise was extracted from pure-noise waveforms in the experimental data by applying a Fast Fourier Transform (FFT). The resulting frequency-dependent amplitudes describe how the noise power is distributed over the frequency spectrum, as shown in Figure 7.7. To generate noise in the time domain, random phases were assigned to this frequency response, and an inverse Fast Fourier Transform (IFFT) was then applied. In addition, the RMS noise was measured for the simulated waveforms and compared with the RMS noise observed in

the real test muon beam data. This comparison was used to check whether the simulated noise reproduced the rms noise seen in the experimental waveforms as shown in figure 7.8.

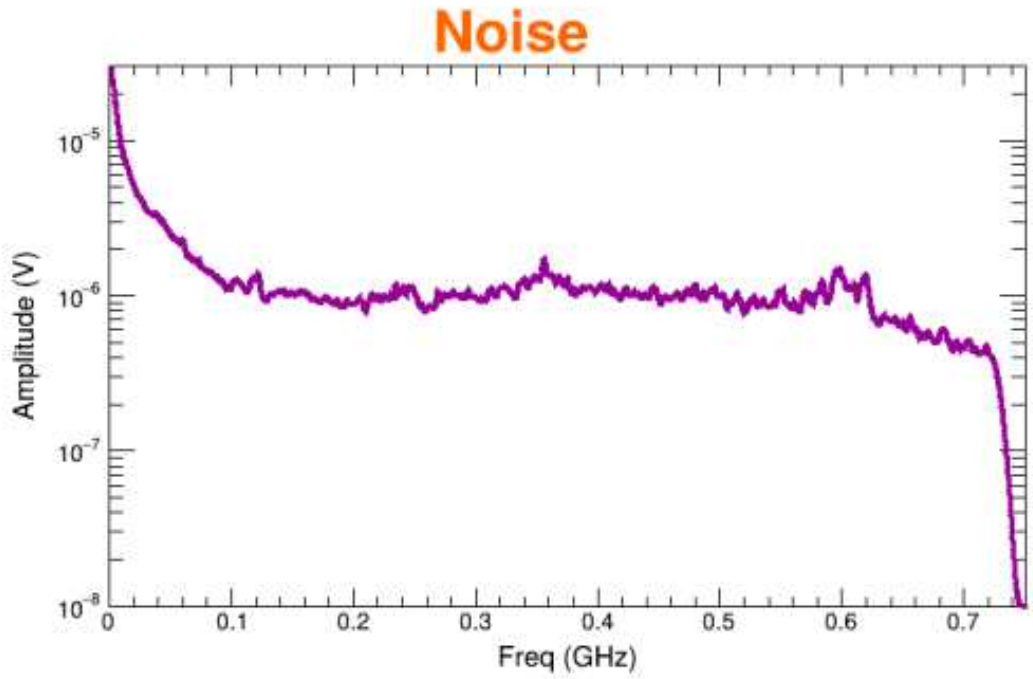


Figure 7.7: Noise amplitude is measured by using fast fourier which is distributed over the range of frequencies.

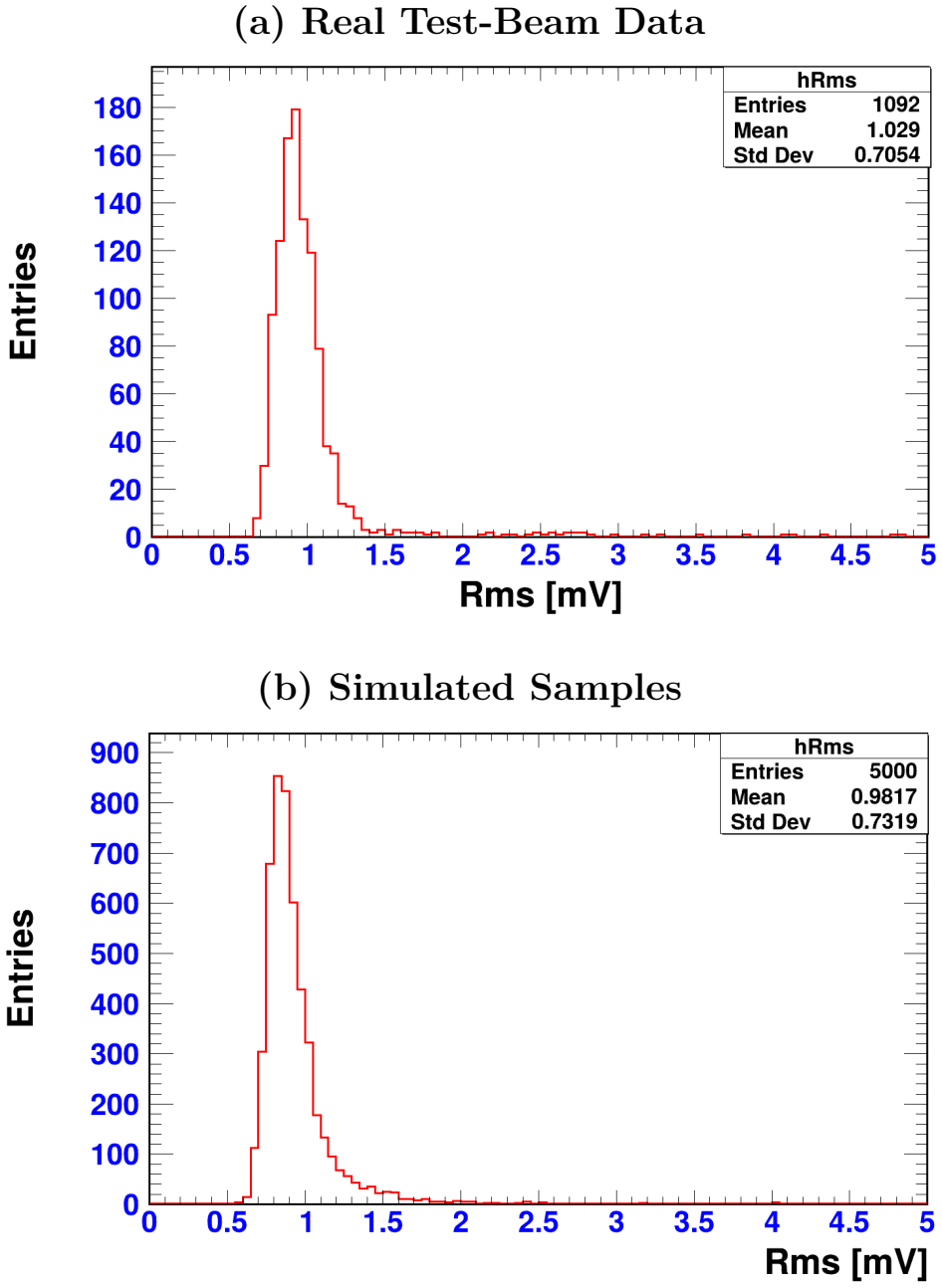


Figure 7.8: RMS noise distributions for real test-beam data and simulation. The top panel shows the RMS noise distribution for the real test-beam data, while the bottom panel shows the corresponding RMS noise distribution for the simulated samples.

As shown in Figures 7.8, the simulated samples were tuned on the basis of the RMS noise in comparison with real test beam data. In particular, the mean RMS noise is 1.029 mV in data and 0.9817 mV in simulation, while the corresponding standard deviations are 0.7054 mV and 0.7319 mV, showing that the tuned simulation is very close to the real data in terms of both mean value and width of the distribution.

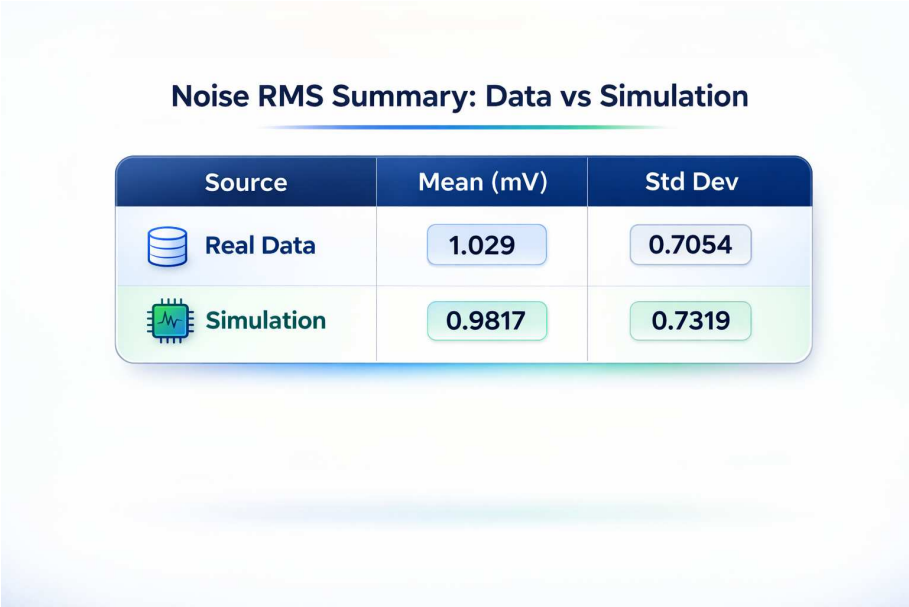


Table 7.1: Comparison of the RMS noise between real test-beam data and the tuned simulation.

7.3.3 Maximum-Amplitude

After the noise tuning, the signal amplitude of the simulated waveform was adjusted to match the real test-beam data. This was done with an iterative scaling procedure. The tuning started from an initial value of

$$\text{ScaleFactor} = 1.0, \tag{7.1}$$

which means that no scaling was applied in the first step. After generating the simulated waveforms, the mean signal amplitude of the MC (3.6502 V) was compared with the mean signal amplitude measured (0.00765813) from the 2022 test-beam data. The scale factor was then updated step by step using

$$\text{ScaleFactor}_{\text{new}} = \text{ScaleFactor}_{\text{current}} \cdot \frac{\text{mean}_{\text{data}}}{\text{mean}_{\text{MC}}}. \tag{7.2}$$

In this expression, $\text{ScaleFactor}_{\text{current}}$ is the value used in the previous tuning step, $\text{mean}_{\text{data}}$ is the mean signal amplitude measured from the real test-beam waveforms, and mean_{MC} is the mean signal amplitude obtained from the simulated waveforms.

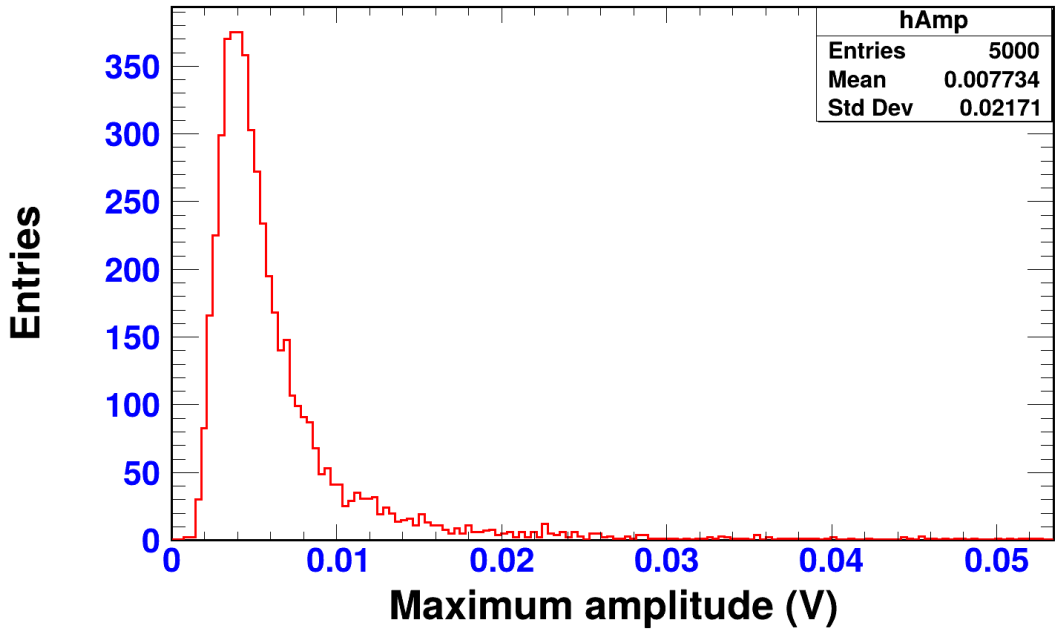
The tuning was repeated until the scaled MC mean amplitude matched the real data as closely as possible. The final scale factor was 0.002. With this value, the mean MC amplitude became 0.007734 V, which is in good agreement with the real data value of 0.007658 V. The tuning steps of the simulation on the behalf of maxima amplitude are shown in the table below.

Step	ScaleFactor	Mean (V)	Std dev (V)
Step 1	1.00000	3.6502	9.66536
Step 2	0.00210	0.00741983	0.0171354
Step 3 (final MC)	0.00200	0.007734	0.02171
Real Data	—	0.007658	0.020610

Table 7.2: Step-by-step update of the scale factor used to tune the MC signal amplitude to the 2022 test-beam data.

The distribution of the maximum amplitude of tuned monte carlo and data are showed in figure 7.9

(a) Simulated Samples



(b) Real Test-Beam Data

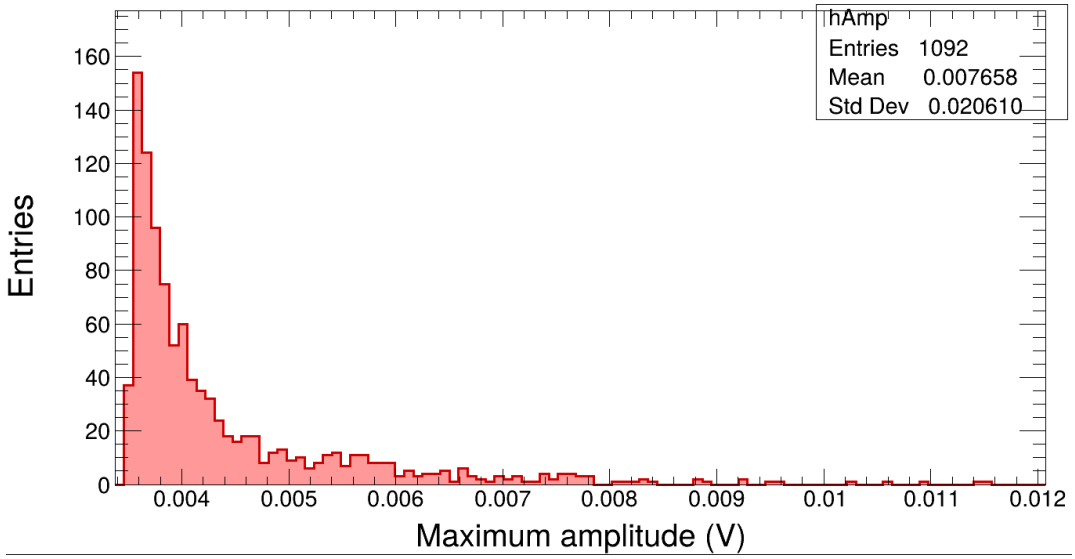


Figure 7.9: Comparison of the maximum-amplitude distributions for the simulated samples and the real test-beam data. The tuned simulation compares with data well, on the behalf of mean values and standard deviation.

7.3.4 Comparison of Tuned MC and Real Data

After tuning both the noise and the amplitude, the updated MC waveforms compare to the waveforms of the real data as shown in 7.10. This agreement is important because neural network models should be trained on simulated waveforms that are as realistic as possible before they are applied to the real beam data.

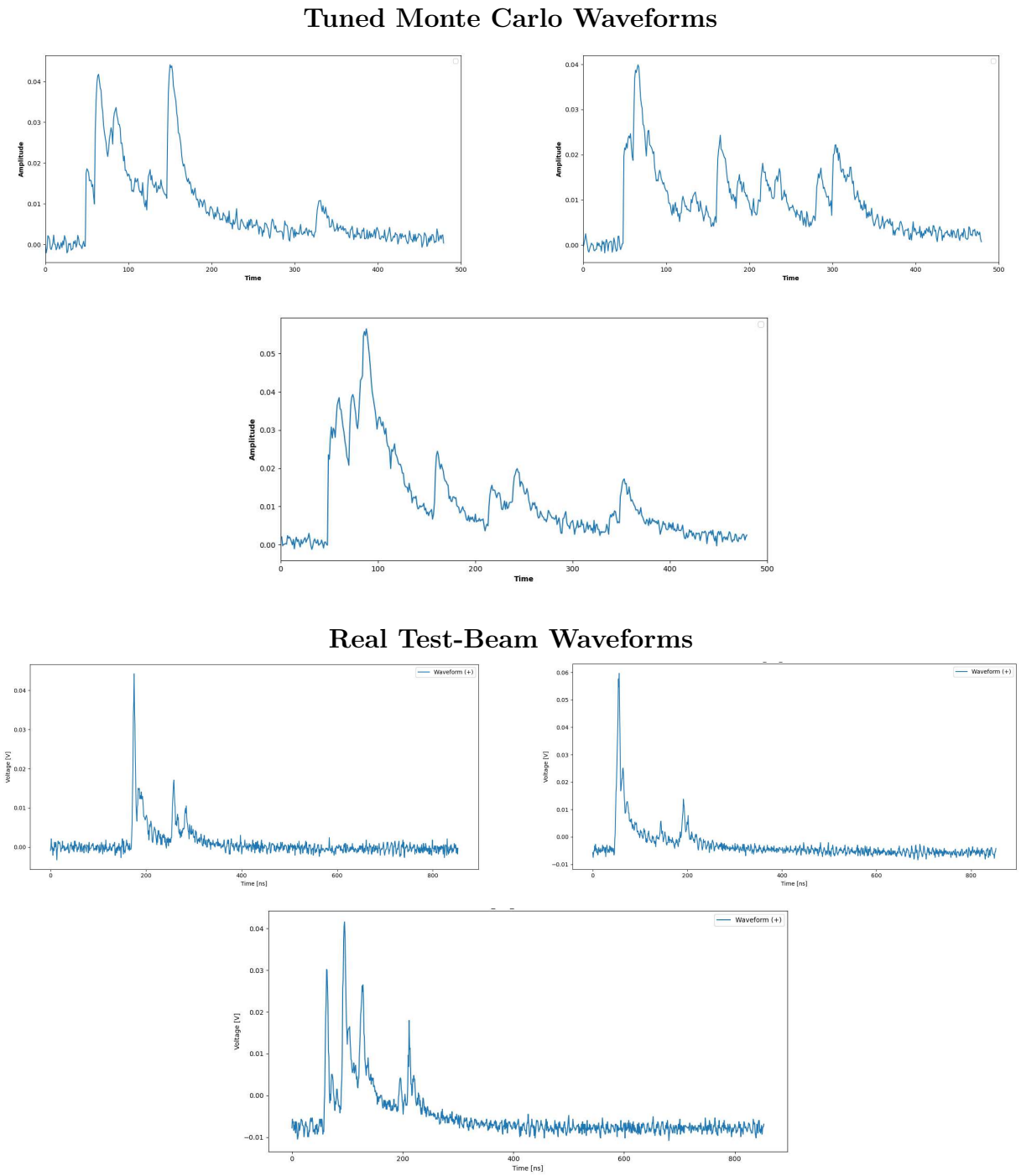


Figure 7.10: Comparison between the tuned MC waveforms and the real test-beam waveforms. The top panels show the simulated waveforms, while the bottom panels show the corresponding real-data waveforms.

7.3.5 Training of LSTM Model on Tuned Waveform

As the first task, the long short-term memory (LSTM) model should be trained on the tuned 180 GeV simulated waveforms with parameters showed in the table 6.1 for 2022 real data [13]. The purpose of this step is to find the best peak-finding model before applying it to the real beam data.

Grid-search optimization. A grid-search study was carried out to train the LSTM-based peak-finding model using many different hyperparameter configurations on the local ReCaS Bari HPC resources. The tested hyperparameters included the activation functions, optimizer type, number of epochs, batch size, early-stopping patience, and dropout rate. At the same time, practical computing quantities were also monitored, such as memory usage, training time for each configuration, and execution on 4 CPU cores.

Figure 7.11 summarizes the performance of all tested LSTM configurations. In this scatter plot, the x-axis shows the model index, where each index corresponds to one unique hyperparameter set, and the y-axis shows the corresponding AUC value obtained for that model. This figure gives an overall view of how the model performance changes across the tested configurations.

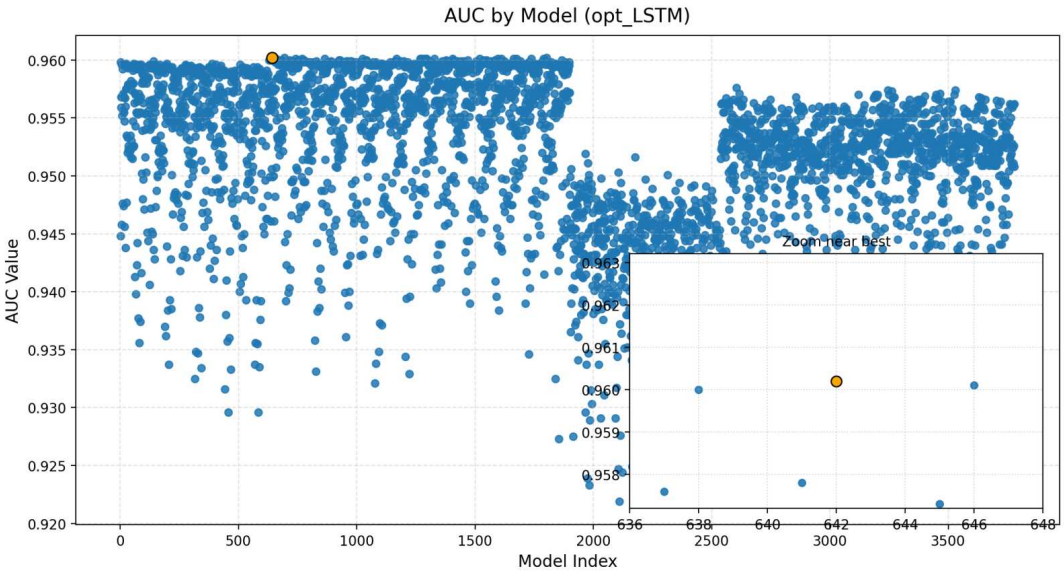


Figure 7.11: Grid-search performance of the LSTM models trained on the tuned 180 GeV simulated waveforms. Each point corresponds to one tested hyperparameter configuration. The x-axis shows the model index, and the y-axis shows the corresponding AUC value.

Selection of the best LSTM model. Since each hyperparameter configuration gives a different AUC value, the best LSTM model was selected as the one with the highest AUC among all tested models. In this study, the best model achieved an AUC of 0.9602. In the zoomed plot, this model is highlighted by the red point. The corresponding ROC curve is shown together with the zoomed AUC view in Figure 7.12. These plots confirm that

the selected model provides the best classification performance among the tested LSTM configurations.

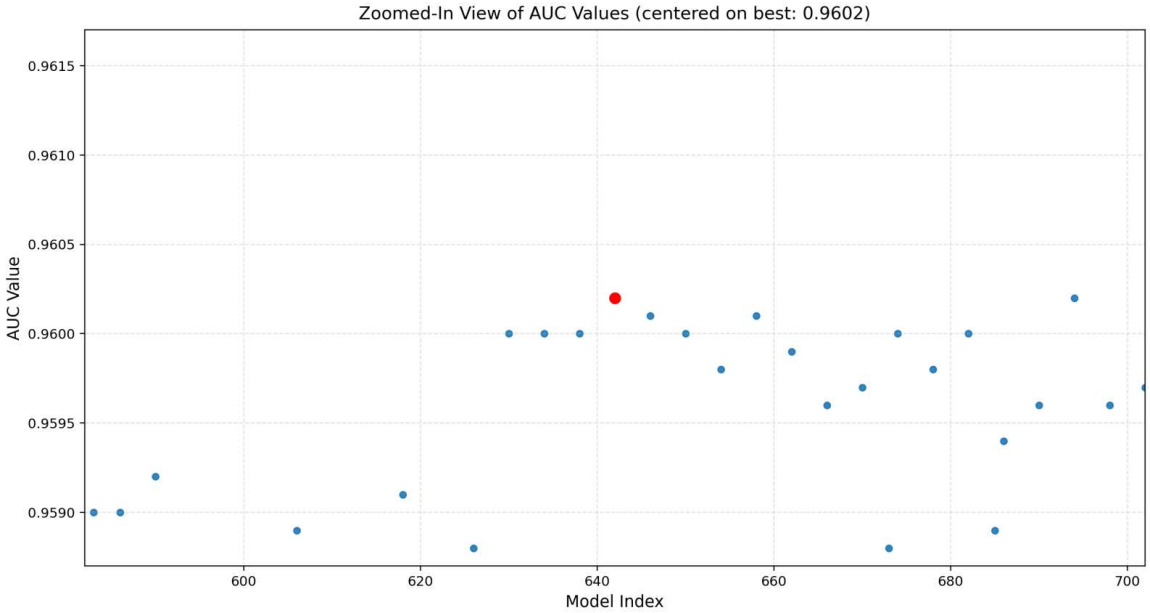


Figure 7.12: Zoomed-in view of the AUC values around the best-performing LSTM configurations. The selected model is highlighted by the red point and achieves an AUC of (0.9602).

Best hyperparameters and computing resources. The hyperparameters and HPC resource usage of the best LSTM model are summarized in table 7.3. The best LSTM model was selected based on the hyperparameter such as Adam optimizer, 100 epoch, bach size 16 etc as showed in the table below. In addition to computing resources of local bari recas resources such as training time, memory usage, and number of CPU's are showed in the table below.

Training Configuration	
Optimizer	Adam
Network topology	[64, 32, 1]
Activation functions	GELU (hidden), Sigmoid (output)
Dropout rate	0.0
Batch size	16
Train / validation split	0.7 / 0.3
Number of epochs	100
Final Performance	
AUC score	0.9602
HPC Resource Usage	
Number of CPUs	4
Training time	51 992 s
Peak memory usage	1173 MB

Table 7.3: Hyperparameters, final performance, and HPC resource usage of the best LSTM model trained on the tuned 180 GeV simulated waveforms.

Accuracy and Loss of Best LSTM model. Using the above steps, the best configuration reached an AUC of about **0.96**, which indicates strong separation of signal from background in the digitized waveform. Figure 7.13 show the training behavior of the selected model. The loss decreases with epochs and then becomes almost constant, while the accuracy increases and then stabilizes. This stable behavior is a good sign that the model has learned well.

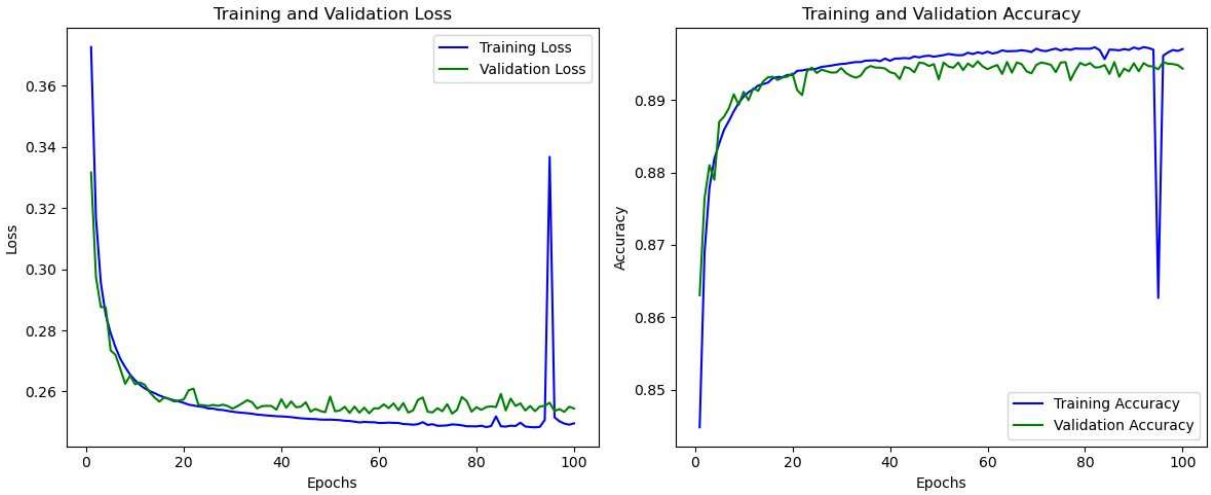


Figure 7.13: Training history of the selected LSTM peak-finding model (muon momentum: 180 GeV). The loss and accuracy decreases and increases with epochs and stabilizes after several epochs, indicating good convergence.

ROC curve. The figure 7.14 shows the Receiver Operating Characteristic (ROC) curve of the selected model. The x-axis represents the False Positive Rate (FPR), while the y-axis represents the True Positive Rate (TPR). The solid blue curve shows the performance of the classifier, and the dashed diagonal line represents the performance of a random classifier. Since the ROC curve lies close to the upper-left corner and the area under the curve is approximately AUC ≈ 0.98, the model has excellent discrimination ability between signal and background.

The quantities are defined as

$$\text{TPR} = \frac{TP}{TP + FN} \qquad \text{and} \qquad \text{FPR} = \frac{FP}{FP + TN}.$$

where TP is the number of correctly detected peaks and FP is the number of wrong detected peaks, (TP+FP) is the total number of detected peaks, and (TP+FN) is the total number of peaks in MC truth of the waveform

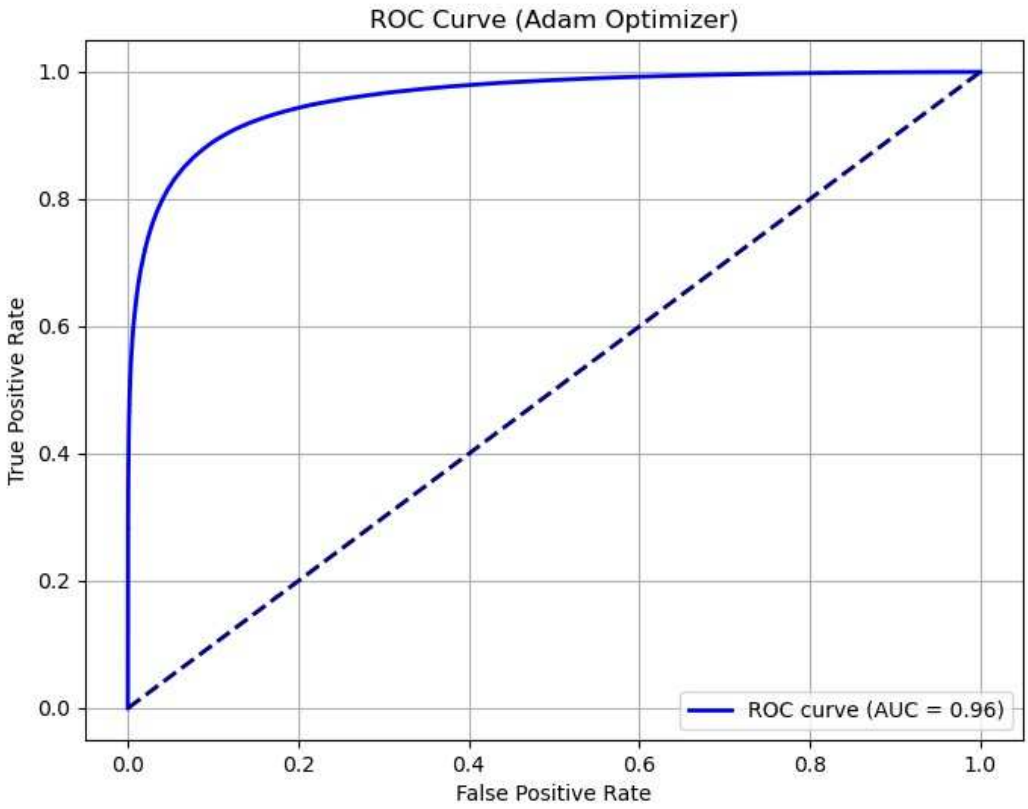


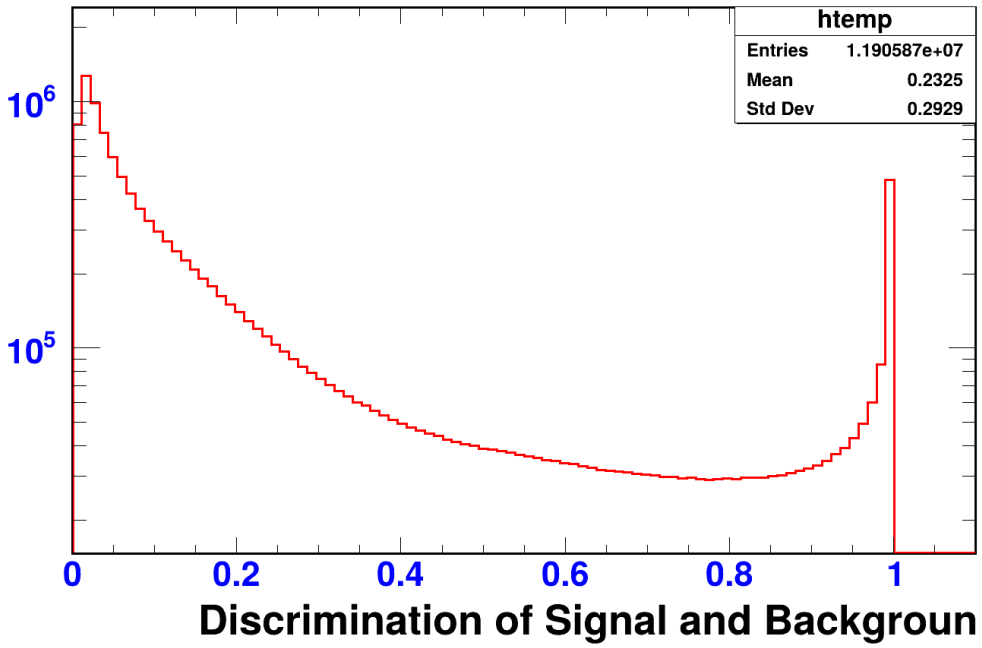
Figure 7.14: ROC curve of the selected LSTM model evaluated on muon momenta 180 GeV (AUC ≈ 0.96).

7.3.6 Peak Finding Algorithm

After selecting the best LSTM model, the next step was to apply it to both the tuned 180 GeV simulated waveforms and the real 180 GeV test-beam data on channel 5. The purpose of this step was to detect signal peaks in the digitized waveform and separate them from the noise. In this analysis, the signal peaks are associated with both primary and secondary electrons. Therefore, this step is a binary classification problem and represents the central part of the peak-finding procedure used in the cluster-counting technique.

Discrimination of signal and background. The trained LSTM model was first applied to both the tuned Monte Carlo and the real test-beam data in order to separate signal peaks from background. The output distributions are shown in Figure 7.15. In both cases, two clear regions appear: one close to 0, which mainly corresponds to background events, and one close to 1, which mainly corresponds to signal events. This behavior shows that the trained LSTM model can distinguish physical peaks from noise-like fluctuations in both the tuned simulation and the real data.

(a) Tuned Monte Carlo



(b) Real Data

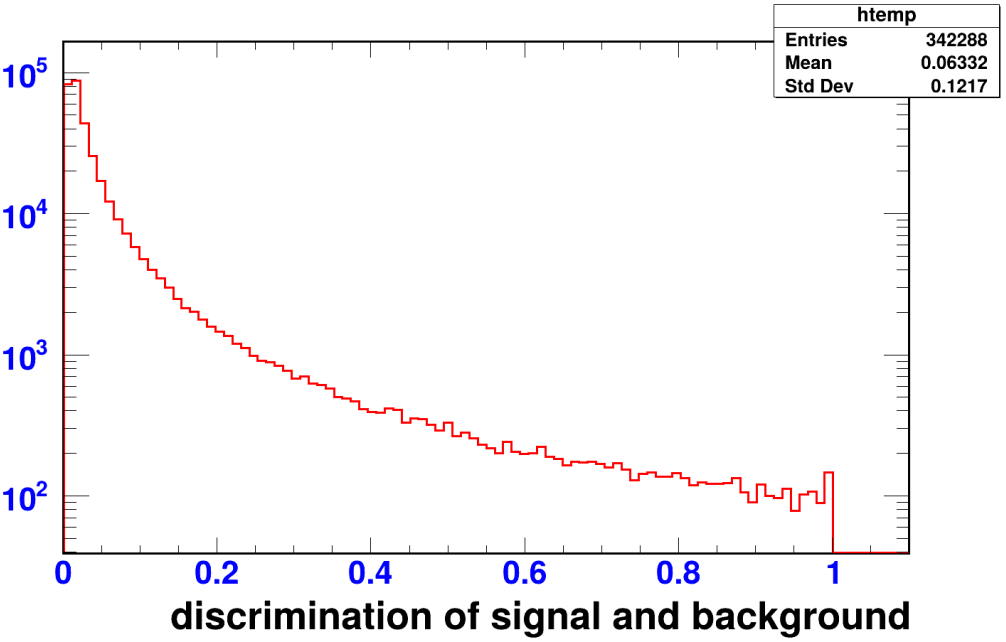


Figure 7.15: Output distributions of the trained LSTM peak-finding model for tuned Monte Carlo and real data. The top panel shows the result for the tuned simulation, while the bottom panel shows the result for the real data on the behalf of discrimination of signal and background. A value near 0 mainly corresponds to background events.

Peak-finding results for data and MC compared with RTA. After the signal–background discrimination step, a selection threshold of 0.55 was applied to the LSTM output to identify signal peaks above the noise. For the 180 GeV muon data sample, which

contains 1092 waveforms, the LSTM model detects an average of 31.48 peaks per waveform. For comparison, the Running Template Algorithm detects an average of 35 peaks per waveform. Example waveforms from tuned Monte Carlo, real data, the distribution of the total detected peaks, and the waveform reconstructed using the RTA algorithm are shown in Figure 7.16.

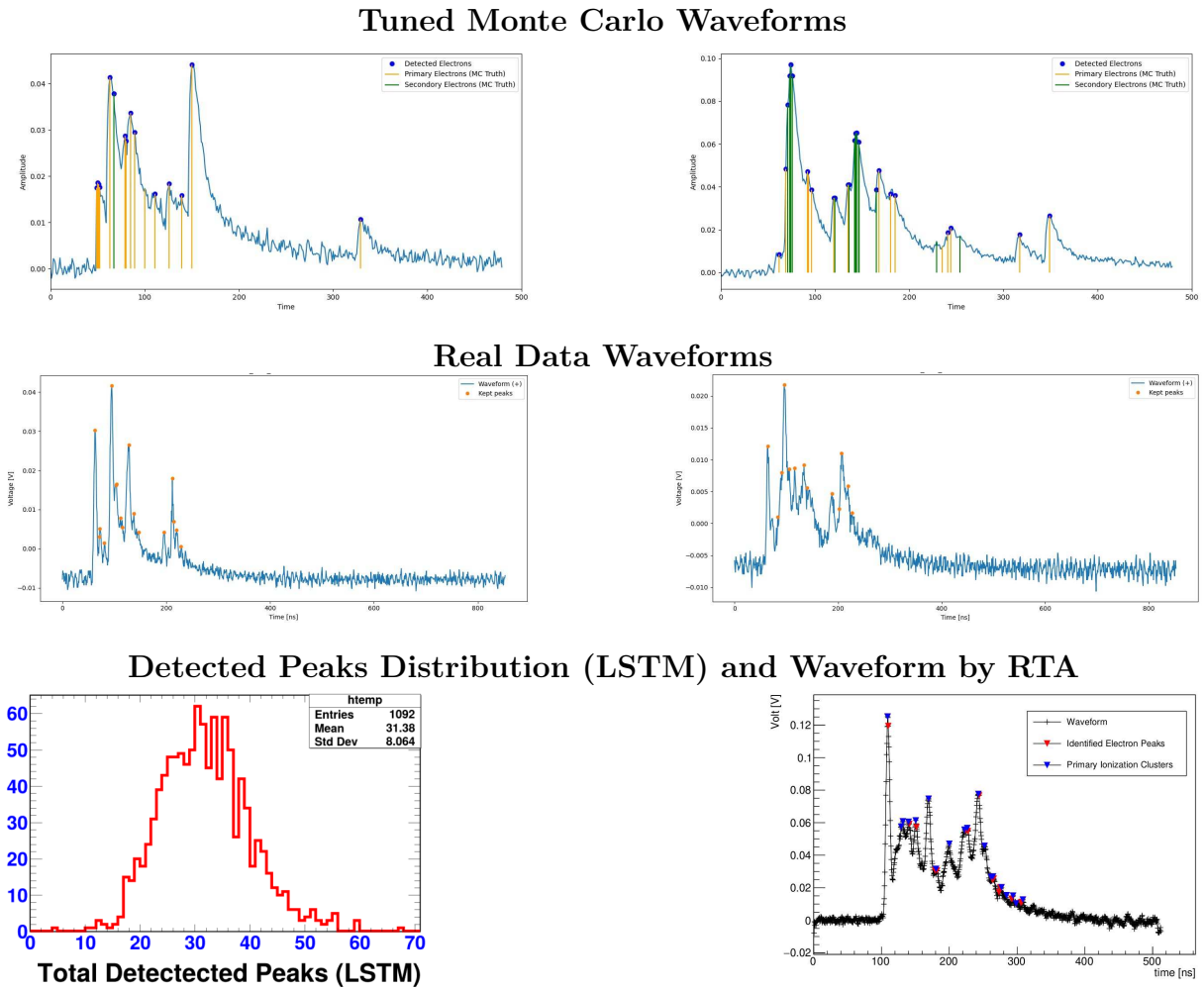


Figure 7.16: Peak-finding results for tuned Monte Carlo and real data using the trained LSTM model. The top panels show tuned Monte Carlo waveforms, the middle panels show real-data waveforms, and the bottom panels show the detected-peaks distribution from the LSTM and the waveform obtained by the Running Template Algorithm (RTA).

7.3.7 Clusterization Algorithm

In this part of the work, the main goal was to develop a machine-learning-based clusterization method for cluster-counting studies in drift-chamber waveforms. The activity was divided into two main tasks. The first task was to train a convolutional neural network (CNN) model using tuned 180 GeV simulated samples with parameters showed in table 6.1 for 2022 real data. The second task was to define a clear logic to select the best-performing CNN model among all tested configurations. This training-and-selection step is necessary

before applying the best CNN model to real test-beam data to estimate the number of primary clusters based on the detected peaks and to compare the results with the Running Template Algorithm (RTA).

Grid-search optimization. A grid-search study was carried out to train the CNN-based clusterization model using many different hyperparameter configurations on the local ReCaS Bari HPC resources. The tested hyperparameters included the activation functions, optimizer choice, number of epochs, batch size, early-stopping patience, and dropout rate. At the same time, practical computing quantities were monitored, such as memory usage, training time for each configuration, and execution on 4 CPU cores.

The overall performance of the tested CNN models is summarized in Figure 7.17. In this scatter plot, the mean square error (MSE) is shown for different hyperparameter configurations. The x-axis represents the model index $(0, 1, 2, \dots)$, where each index corresponds to one unique hyperparameter set, and the y-axis shows the corresponding MSE value obtained for that model.

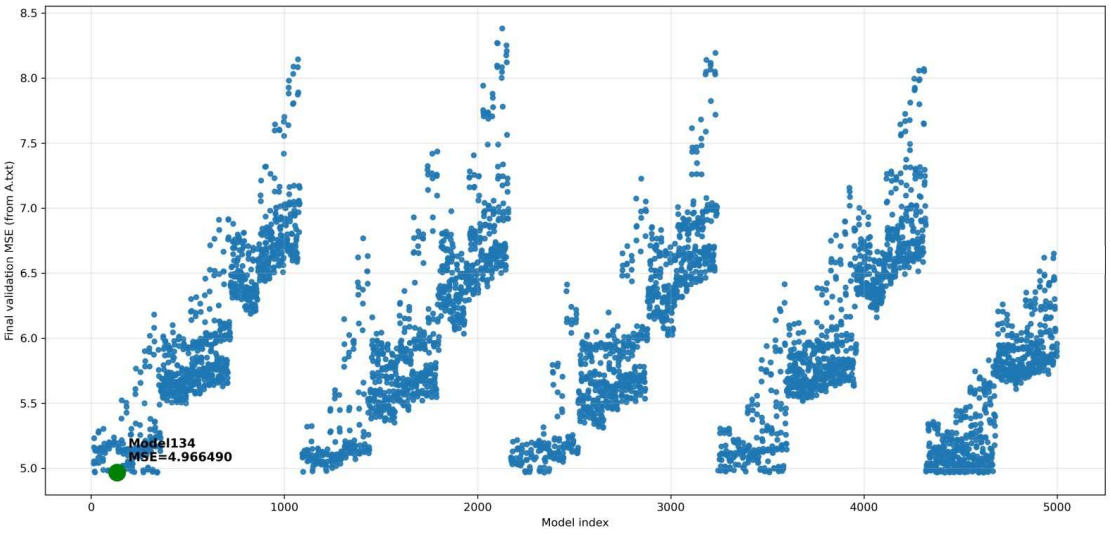
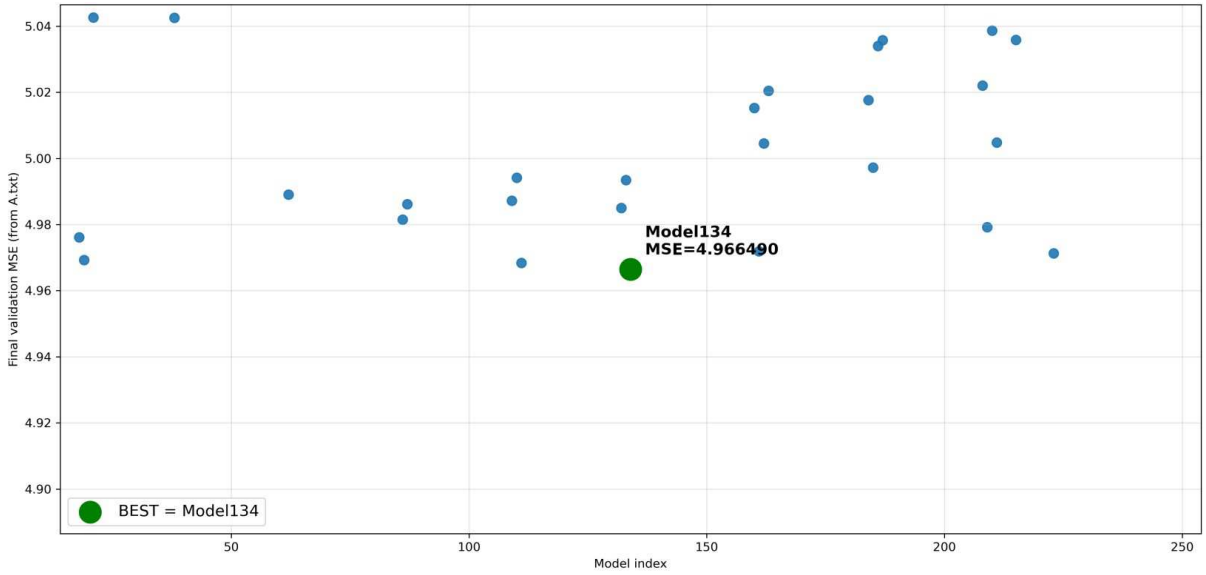


Figure 7.17: Grid-search performance of the CNN clusterization models trained on the tuned 180 GeV simulated samples. Each point corresponds to one tested hyperparameter configuration. The x-axis shows the model index, and the y-axis shows the final validation MSE. The best model is highlighted by the green point.

Logic to select the best CNN model. Since each hyperparameter configuration gives a different MSE value, the best CNN model was selected as the configuration with the lowest MSE among all tested models. In this study, the best model was identified as Model134, with a final validation MSE of 4.966490 . In the scatter plot, this model is marked by the green point. A zoomed view around the best configurations and the corresponding training and validation loss curve are shown in Figure 7.18.

The loss curve shows that the model converges quickly during training and then becomes stable. This behavior indicates that the selected CNN model provides the best

regression performance among the clusterization models tested.



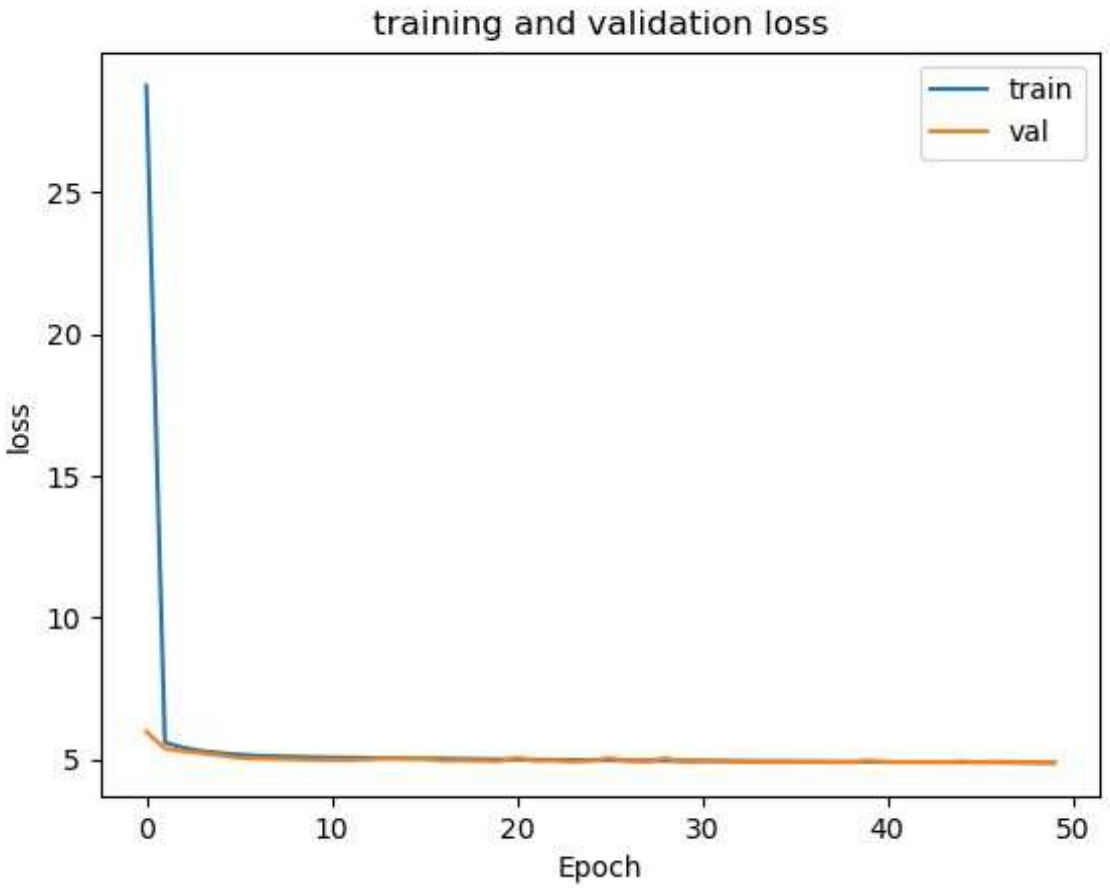


Figure 7.18: Selection of the best CNN clusterization model. Top: zoomed view of the best-performing configurations, where Model134 is highlighted by the green point. Bottom: training and validation loss curves of the selected model.

Best hyperparameters and HPC resources. The hyperparameters of the selected CNN clusterization model and the corresponding HPC resource usage are summarized in table 7.4. Its architecture uses filters `[32,16]`. The model was trained for 50 epochs with a batch size of 96. The training and validation split was 0.7/0.3, and the early-stopping patience was set to 20. The neuron activation functions were `relu, relu`, and the optimizer was `AdamW`. The final MSE of this best CNN model was 4.9664.

In terms of computing resources, the training of the best CNN model used 4 CPUs on the local ReCaS Bari HPC system. Memory usage was 11628 MB, and the total training time was 3073 s. The hyperparameters and HPC resources for the best CNN clusterization model are summarized in the table below.

Hyperparameters	
Hyperparameters	Value
Filters	[32, 16]
Epochs	50
Batch size	96
Train/Validation split	0.7 / 0.3
Patience / Early Stopping	20
Neuron activations	relu, relu
Optimizer	AdamW
Final Metrics	
Final MSE	4.9664

HPC ReCaS Bari Resources	Value
Total CPUs used	4
Peak memory usage	11628 MB
Training Time	3073 sec

Table 7.4: Hyperparameters, final performance metric, and HPC resource usage of the best CNN clusterization model trained on the tuned 180 GeV simulated samples.

Final Results of Reconstruction

After training the CNN clusterization model, the next step was to apply it to both the tuned simulated waveforms and the real 180 GeV test-beam data. The purpose of this step was to reconstruct the distribution of the number of primary ionization clusters and to compare the CNN result with the reference result obtained from the Running Template Algorithm (RTA).

Figure 7.19 and 7.20 shows three distributions. The first distribution corresponds to the number of primary clusters reconstructed by the CNN on the tuned simulation. The second distribution corresponds to the number of primary clusters reconstructed by the CNN on the real data. The third distribution shows the number of clusters reconstructed by the RTA on data.

For the tuned simulation, the CNN distribution contains 50 000 entries and has a histogram mean of 16.85 with a standard deviation of 3.871.

For the real data, the CNN distribution contains 1092 entries and has a histogram mean of 14 with a standard deviation of 4.185. For comparison, the RTA result on data contains 4894 entries and has a mean of 16.71 with a standard deviation of 4.617. The summary related to the three distribution on the behalf of cluerizations are shown in the table 7.5 .

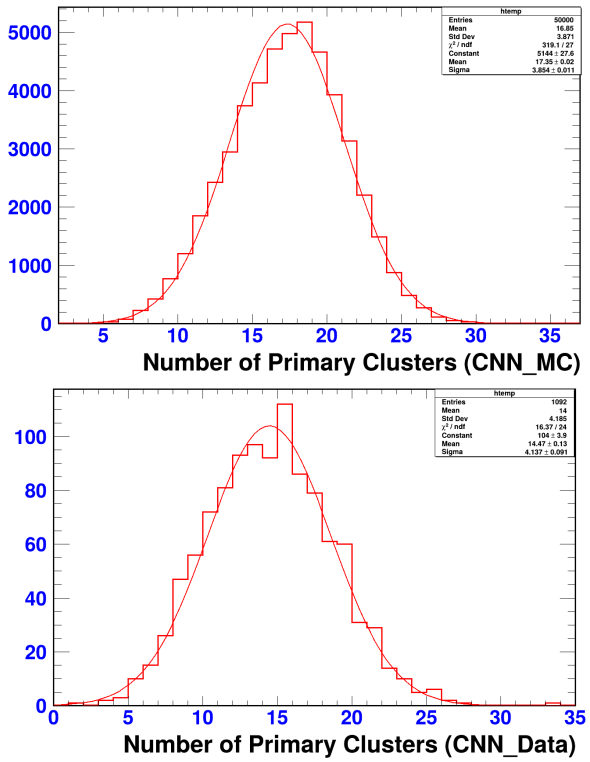


Figure 7.19: Distributions of the reconstructed number of primary clusters obtained with the CNN model. Top: CNN result for the tuned simulated sample. Bottom: CNN result for real data.

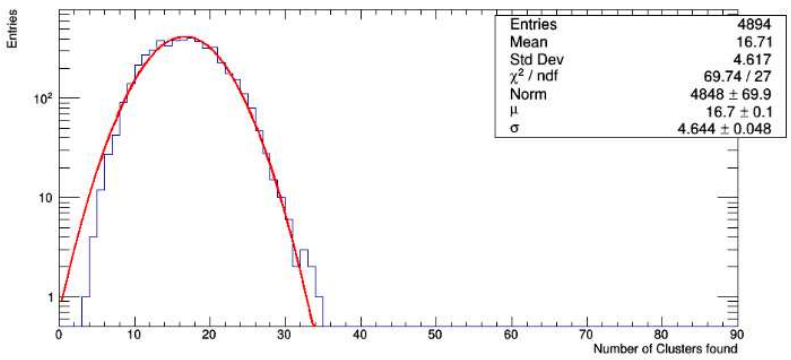


Figure 7.20: Distribution of the reconstructed number of primary clusters obtained with the Running Template Algorithm (RTA) for real data.

Conclusion related to clusterization using Machine Learning and Running Template Algorithm (RTA) are showed in the table below:

Algorithm	Mean	Standard Deviation	Total Entries (Waveforms)
CNN_Simulation	16.85	3.871	50000
CNN_Data	14	4.185	1092
RTA Algorithm	16.71	4.617	4894

Table 7.5: The table shows the mean number of primary clusters and the standard deviation for the three distributions. The second row corresponds to the number of primary clusters reconstructed by the CNN on simulation, the third row corresponds to the number of primary clusters reconstructed by the CNN on real data, and the last row corresponds to the number of primary clusters reconstructed by the Running Template Algorithm (RTA).

Conclusion

The main objective of this thesis was to develop and optimize deep-learning models for cluster counting in drift chambers using high-performance computing (HPC) resources. The study focused on estimating the number of primary ionization clusters per unit length from digitized waveforms. The proposed reconstruction chain consists of a Long Short-Term Memory (LSTM) model for peak finding, treated as a binary classification problem, followed by a Convolutional Neural Network (CNN) model for estimating the number of primary clusters, treated as a regression problem.

The training samples were generated with Garfield++ and configured to reproduce the main detector conditions of the 2022 and 2023 beam-test campaigns, including the drift-cell geometry, gas mixture, sense-wire diameter, and track angle. These simulated samples provided a controlled environment in which the reconstructed cluster distributions could be compared directly with the Monte Carlo truth.

The LSTM and CNN models were trained and optimized using the local ReCaS Bari and national HPC infrastructures. Many hyperparameter configurations were executed simultaneously as independent jobs. This parallel approach made the model-selection process faster, systematic, and reproducible, while also allowing the CPU usage, memory consumption, and training time of the different configurations to be evaluated.

For the simulated muon samples, the selected LSTM peak-finding model achieved an AUC of approximately (0.98), showing strong discrimination between ionization peaks and waveform noise. Its training required approximately (66468.31 s) and (425.07 MB) of memory on the local ReCaS Bari system. The selected CNN model achieved the lowest mean squared error, approximately (4.8), and required approximately (3073 s) and (11628 MB) of memory.

The mean numbers of primary clusters reconstructed by the optimized LSTM–CNN chain were generally close to the corresponding Monte Carlo truth values across the investigated momentum ranges and particle species. This agreement confirms that the selected models reproduce the main features of the simulated primary-cluster distributions. In addition, the particle-identification capability was evaluated using the separation power for muon–kaon and pion–kaon pairs in the momentum range (2–10 GeV/c). The separation powers calculated from the CNN-reconstructed primary clusters were close to those obtained from the Monte Carlo truth for both particle pairs.

The optimized framework was also applied to real 2022 beam-test data. Before retraining, the simulated waveforms were tuned to reproduce the measured noise and maximum-amplitude characteristics. Using HPC resources, new LSTM and CNN hyperparameter searches were performed on the tuned samples. The selected LSTM model achieved an AUC of approximately (0.96), while the selected CNN model achieved a mean squared error of approximately (4.966).

The final and important task is related to the application of "Machine Learning Algorithm on Real Test Beam Data Using HPC Resources" in order to reconstruct the number of primary clusters in digitized waveforms in comparison with Runing Template Algorithm (RTA). For this purpose, the simulated samples corresponding to the 2022 test-beam data are further tuned in terms of noise and maximum amplitude. After this tuning procedure, the LSTM and CNN models are trained on the tuned simulated samples using HPC resources. Among all tested hyperparameter configurations, the best LSTM model for peak finding is selected according to the highest area under the curve (0.96), while the best CNN model for clusterization is selected according to the lowest mean squared error (4.966). Finally, the best selected CNN model is applied to the real digitized waveforms to reconstruct the number of primary clusters (14), and its performance is compared with the Running Template Algorithm (16).

Overall, this thesis demonstrates that hyperparameter optimization using HPC resources is an effective method for developing neural-network models for drift-chamber cluster counting. HPC enabled the parallel training and comparison of many LSTM and CNN configurations and supported model selection based on both predictive performance and computational efficiency. The optimized models reconstructed primary-cluster multiplicities close to Monte Carlo truth and preserved meaningful muon–kaon and pion–kaon separation power.

Future work will extend the analysis to the 2023 and 2024 beam-test data in the (2–10 GeV) momentum range. The ML reconstruction will be compared with both the RTA and the derivative-based cluster-counting method. Further studies should focus on improved simulation tuning, systematic uncertainty evaluation, and direct validation of particle-identification performance under realistic experimental conditions.

Bibliography

[1] Eurohpc. https://eurohpc-ju.europa.eu. Accessed: 2026-03-27.

[2] Eosc. https://eosc.eu. Accessed: 2026-03-27.

[3] European processor initiative (epi). https://www.european-processor-initiative.eu/. Accessed: 2026-03-27.

[4] Quantum technologies (qt). https://qt.eu. Accessed: 2026-03-27.

[5] Icsc supercomputing. https://www.supercomputing-icsc.it/en/icsc-home/. Accessed: 2026-03-27.

[6] Terabit project. https://www.terabit-project.it/en. Accessed: 2026-03-27.

[7] Garr-t network. https://www.garr.it/it/infrastrutture/rete-nazionale/rete-garr-t. Accessed: 2026-03-27.

[8] Eurohpc joint undertaking. https://eurohpc-ju.europa.eu/index_en. Accessed: 2026-03-27.

[9] Eosc open science cloud. https://open-science-cloud.ec.europa.eu/. Accessed: 2026-03-27.

[10] Garr. https://www.garr.it/it/. Accessed: 2026-03-27.

[11] Cineca. https://www.cineca.it/en. Accessed: 2026-03-27.

[12] Infn. https://home.infn.it/en/. Accessed: 2026-03-27.

[13] Gaia-x. https://gaia-x.eu/. Accessed: 2026-03-27.

[14] Inspirehep: Literature 1635816. https://inspirehep.net/literature/1635816. Accessed: 2026-03-27.

[15] Jupyter documentation. https://docs.jupyter.org/en/latest/. Accessed: 2026-03-27.

[16] Dask documentation. https://docs.dask.org/en/stable/. Accessed: 2026-03-27.

[17] Htcondor documentation. https://htcondor.readthedocs.io/en/latest/. Accessed: 2026-03-27.

[18] Openstack documentation. https://docs.openstack.org/2023.1. Accessed: 2026-03-27.

[19] Rancher rke documentation. https://rke.docs.rancher.com. Accessed: 2026-03-27.

[20] Docker. https://www.docker.com/. Accessed: 2026-03-27.

[21] M. Benedikt et al. Future circular collider study. volume 2: The lepton collider (fcc-ee) conceptual design report. CERN accelerator reports CERN-ACC-2018-0057, CERN, 2018.

[22] B. Richter. Nucl. instr. methods. *Nucl. Instr. Methods*, 136:47, 1976.

[23] CEPC-SPPC Study Group. Cepc-sppc preliminary conceptual design report. physics and detector. Technical Report IHEP-CEPC-DR-2015-01, IHEP, 2015.

[24] T. Behnke et al. The international linear collider technical design report – volume 1: Executive summary. arXiv:1306.6327 [physics.acc-ph], 2013.

[25] M. Aicheler et al. A multi-tev linear collider based on clic technology. Technical report, CERN Yellow Reports: Monographs, 2012.

[26] CEPC Study Group. Cepc conceptual design report. Technical Report IHEP-CEPC-DR-2018-01, IHEP, 2018. arXiv:1809.00285 [physics.acc-ph].

[27] K. Fujii et al. Physics case for the 250 gev stage of the international linear collider. arXiv:1710.07621 [hep-ex], 2017.

[28] CLIC Collaboration CLICdp and M. J. Boland et al. Updated baseline for a staged compact linear collider, 2016.

[29] A. Apollonio et al. Fcc-ee operation model, availability, and performance. In *Proceedings of ICFA Advanced Beam Dynamics Workshop eeFACT2018*, Hong Kong, 2008.

[30] M. Antonello. Idea: A detector concept for future leptonic colliders. *C*, 43:27, 2020.

[31] P. Azzi and E. Perez. Exploring requirements and detector solutions for fcc-ee, 2021.

[32] M. Benedikt et al. Fcc-ee: The lepton collider: Future circular collider conceptual design report volume 2. Technical report, CERN, Geneva, 2018.

[33] G. Morello G. Bencivenni, R. D. Oliveira and M. P. Lener. The micro-resistive well detector: a compact spark-protected single amplification-stage mpgd. *Instrumentation*, 10:P02008, 2015.

[34] L. Pezzotti and A. Villa. Idea dual-readout calorimeter simulation. In *PoS EPS-HEP2021*, page 777, 2022.

[35] S. Braibant and P. Giacomelli. Muon detection at fcc-ee. *Eur. Phys. J. Plus*, 136(11):1143, 2021.

[36] F. Cuna et al. A proposal of a he based drift chamber as central tracker for the idea detector concept for a future ee collider. In *PoS EPS-HEP2021*, page 786, 2022.

[37] G. F. Tassielli. A proposal of a drift chamber for the idea experiment for a future e+ e collider. In *PoS ICHEP2020*, page 877, 2021.

[38] F. Grancagnolo M. Cascella and G. Tassielli. Cluster counting/timing techniques for drift chambers. In *Nuclear Physics B - Proceedings Supplements*, volume 248-250, pages 127–130, 2014.

[39] F. Grancagnolo et al. Drift chamber for the cmd-3 detector. *Nucl. Instrum. Meth. A*, 623:114–116, 2010.

[40] I. B. Smirnov. Modeling of ionization produced by fast charged particles in gases. *Nucl. Instr. Meth. A*, 554:474–493, 2005.

[41] Monte carlo simulation of electron drift and diffusion in counting gases under the influence of electric and magnetic fields. *Nucl. Instr. Meth. A*, 421:234–240, 1999.

[42] S. Ramo. Currents induced by electron motion. *Proceedings of the I.R.E.*, 27:584–585, 1939.

[43] O. Sahin. Excitation energy transfer model for ne-co2 and ne-n2 mixtures. *JINST*, 16:P03026, 2021.

[44] E. N. Ozmutlu R. Veenhof O. Sahin, I. Tapan. Penning transfer in argon-based gas mixtures. *JINST*, 5:P05002, 2010.

[45] W. Shockley. Currents to conductors induced by a moving point charge. *J. Appl. Phys.*, 9:635–636, 1938.

[46] Martin Erdmann, Jonas Glombitza, Gregor Kasieczka, and Uwe Klemradt. *Deep Learning for Physics Research*. World Scientific, Hackensack, New Jersey, 2021.

[47] N. Jere I. D. Mienye. Deep learning for credit card fraud detection: A review of algorithms, challenges, and solutions. *IEEE Access*, 12:96893–96910, 2024.

[48] I. H. Sarker. Ai-based modeling: Techniques, applications and research issues towards automation, intelligent and smart systems. *SN Comput. Sci.*, 3:158, 2022.

[49] E. Ileberi I. D. Mienye, Y. Sun. Artificial intelligence and sustainable development in africa: A comprehensive review. *Mach. Learn. Appl.*, 18:100591, 2024.

[50] A. Athaiya S. Sharma, S. Sharma. Activation functions in neural networks. Towards Data Sci. 6, 310, 2017.

[51] A. Gachagan S. Marshall C. Nwankpa, W. Ijomah. Activation functions: Comparison of trends in practice and research for deep learning. arXiv:1811.03354, 2018.

[52] V. Jain A. K. Dubey. Comparative study of convolution neural network's relu and leaky-relu activation functions. In *Applications of Computing, Automation and Wireless Systems in Electrical Engineering: Proceedings of MARC 2018*, pages 873–880. Springer, 2019.

[53] M. Sabuncu Z. Zhang. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Adv. Neural Inf. Process. Syst. 31*, 2018.

[54] S. Jadon A. Jadon, A. Patil. A comprehensive survey of regression-based loss functions for time series forecasting. In *International Conference on Data Management, Analytics Innovation, Pune, India*, pages 117–147. Springer, 2014.

[55] G. Chen W. Zheng. An accurate gru-based power time-series prediction approach with selective state updating and stochastic optimization. *IEEE Trans. Cybern.*, 52:13902–13914, 2021.

[56] M. Stern N. Srebro B. Recht A. C. Wilson, R. Roelofs. The marginal value of adaptive gradient methods in machine learning. In *Adv. Neural Inf. Process. Syst. 30*, 2017.

[57] W. P. Lee Y. L. Peng. Practical guidelines for resolving the loss divergence caused by the root-mean-squared propagation optimizer. *Appl. Soft Comput.*, 153:111335, 2024.

[58] D. P. Kingma. Adam: A method for stochastic optimization. arXiv:1412.6980, 2014.

[59] J. Schmidhuber S. Hochreiter. Long short-term memory. *Neural Computation*, 9:1735–1785, 1997.

[60] A. Zisserman K. Simonyan. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556, 2014.

[61] H. Schopper C. W. Fabjan, editor. *Particle Physics Reference Library Detectors for Particles and Radiation*. Springer Nature, 2020.

[62] G. F. Knoll. *Radiation detection and measurement*. Wiley, 4th edition, 2010.

[63] R. L. Workman et al. Review of particle physics. *PTEP*, 2022:083C01, 2022.

[64] H. H. Cui et al. Y. F. Zhu, S. Z. Chen. Requirement analysis for de/dx measurement and pid performance at the cepc baseline detector. *Nucl. Instrum. Meth. A*, 1047:167835, 2023.

[65] S. S. AbdusSalam et al. A. Abada, M. Abbrescia. Fcc-ee: The lepton collider. *Eur. Phys. J. Spec. Top.*, 228:261–623, 2019.

[66] T. Bressani et al. G. Charpak, R. Bouclier. The use of multiwire proportional counters to select and localize charged particles. *Nucl. Instr. Meth.*, 62:262–268, 1968.

[67] L. Rolandi W. Blum, W. Riegler. *Particle Detection with Drift Chambers*. 2008.

[68] A. H. Walenta. The time expansion chamber and single ionization cluster measurement. *IEEE Trans. Nucl. Sci.*, 26:73–80, 1979.

[69] J. Schmidhuber S. Hochreiter. Long short-term memory. *Neural Computation*, 9:1735–1785, 1997.

[70] P. Frasconi Y. Bengio, P. Simard. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.*, 5:157–166, 1994.

[71] C. H. Hu et al. Y. Yu, X. S. Si. A review of recurrent neural networks: Lstm cells and network architectures. *Neural Comput.*, 31:1235–1270, 2019.

[72] C. Moraga J. Han. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *From Natural to Artificial Neural Computation*, 1995.

[73] C. Soares R. Vilalta P. Brazdil, C. G. Carrier. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[74] D. Montgomery. *Design and analysis of experiments*. John Wiley & Sons, Inc., 8th edition, 2013.

[75] Y. Bengio J. Bergstra. Random search for hyper-parameter optimization. *J. Machine Learning Research*, 13:281–305, 2012.

[76] A. Baldini et al. The ultra-light drift chamber of the meg2 experiment. *Nucl. Instrum. Meth. A*, 958:162152, 2020.