Graph Neural Network-based Track Finding Algorithm for the CBM Experiment

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachberich Informatik und Mathematik
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

von
Oddharak Tyagi
aus India

Frankfurt am Main 2025
(D 30)

Vom Fachbereich Informatik und Mathematik
der Johann Wolfgang Goethe-Universität
als Dissertation angenommen

Dekan:	Prof. Dr. Bastian von Harrach-Sammet
Gutachter:	Prof. Dr. Ivan Kisel
	Prof. Dr. Volker Lindenstruth
Datum der Disputation:	19.03.2026

Abstract

The Compressed Baryonic Matter (CBM) experiment at the future FAIR facility will study strongly interacting matter at high net-baryon densities. Achieving its physics program requires operating at interaction rates of up to 10 MHz, which, in combination with the high charged particle multiplicities of heavy-ion collisions, results in raw data rates of the order of 1 TB/s. To comply with storage constraints, this data stream must be reduced in real time by more than two orders of magnitude. Because the observables of interest exhibit complex topologies and cannot be selected by simple low-level hardware triggers, CBM employs a free-streaming readout architecture with online event reconstruction and selection.

This thesis presents a novel Graph Neural Network (GNN)-based track-finding algorithm for the main tracking system of the CBM experiment. Unlike other recent deep-learning approaches to tracking, this algorithm explicitly integrates Kalman Filter (KF)-based filtering and traditional selection cuts into the reconstruction pipeline. This reduces reliance on neural networks in areas where geometric or physics-motivated criteria are sufficient. Consequently, the method combines the representational power of data-driven deep learning models with the physical robustness and interpretability of KF-based parameter estimation. Additionally, a new cooperative selection scheme is introduced as an alternative to the widely used Cellular Automaton (CA), maximizing the number of reconstructed tracks.

The algorithm has been integrated into the CBMRoot software framework and evaluated at realistic experimental conditions. The reconstruction efficiency improves relative to the CA track finder for all track classes, while maintaining comparable clone and ghost rates and high reconstruction quality. The most significant gains are obtained for low-momentum secondary tracks, whose reconstruction efficiency increases by 5.7% and 7.1% for central and minimum-bias Au+Au collisions at 10 AGeV, respectively. The yield of indirectly reconstructed

short-lived particles K^0_S and Λ particles increases by 3.8% and 3.4% for central Au+Au collisions at 10 AGeV. Across all tested beam energies and collision systems, the GNN-based track finder outperforms the CA-based track finder. Finally, an initial GPU implementation is presented, demonstrating the algorithm's potential for online processing.

Overall, this thesis establishes a novel and effective track-finding algorithm for the CBM experiment. It demonstrates that combining deep learning with physically motivated reconstruction techniques is a viable strategy for track finding in high-rate, high-multiplicity experiments.

Kurzfassung

Das Compressed Baryonic Matter (CBM) Experiment an der zukünftigen FAIR-Anlage wird stark wechselwirkende Materie bei hohen Netto-Baryonendichten untersuchen. Die Umsetzung des Physikprogramms erfordert den Betrieb bei Wechselwirkungsraten von bis zu 10 MHz, was in Kombination mit den hohen Multiplizitäten geladener Teilchen aus Schwerionenkollisionen zu Rohdatenraten in der Größenordnung von 1 TB/s führt. Um die Speicherbeschränkungen einzuhalten, muss dieser Datenstrom in Echtzeit um mehr als zwei Größenordnungen reduziert werden. Da die interessierenden Observablen komplexe Topologien aufweisen und nicht durch einfache Low-Level-Hardware-Trigger selektiert werden können, verwendet CBM eine Free-Streaming-Auslesearchitektur mit Online-Ereignisrekonstruktion und -selektion.

Diese Arbeit stellt einen neuartigen, auf Graph Neural Networks (GNN) basierenden Spurfindungsalgorithmus für das Haupt-Trackingsystem des CBM-Experiments vor. Im Gegensatz zu anderen aktuellen Deep-Learning-Ansätzen für das Tracking integriert dieser Algorithmus explizit Kalman-Filter (KF)-basierte Filterung und traditionelle Selektionsschnitte in die Rekonstruktionspipeline. Dies reduziert die Abhängigkeit von neuronalen Netzen in Bereichen, in denen geometrische oder physikalisch motivierte Kriterien ausreichen. Folglich kombiniert die Methode die Leistungsfähigkeit datengetriebener Deep-Learning-Modelle mit der physikalischen Robustheit und Interpretierbarkeit der KF-basierten Parameterschätzung. Zusätzlich wird ein neues kooperatives Selektionsschema als Alternative zum weit verbreiteten Zellularautomaten (CA) eingeführt, welches die Anzahl der rekonstruierten Spuren maximiert.

Der Algorithmus wurde in das CBMRoot-Software-Framework integriert und unter realistischen experimentellen Bedingungen evaluiert. Die Rekonstruktionseffizienz verbessert sich relativ zum CA-Spurfinder für alle Spurklassen, während vergleichbare Clone- und Ghost-Raten sowie eine hohe Rekonstruktionsqualität

beibehalten werden. Die signifikantesten Gewinne werden für Sekundärspuren mit niedrigem Impuls erzielt, deren Rekonstruktionseffizienz für zentrale bzw. Minimum-Bias-Au+Au-Kollisionen bei 10 AGeV um 5,7 % bzw. 7,1 % steigt. Die Ausbeute an indirekt rekonstruierten kurzlebigen K^0_S- und Λ-Teilchen steigt für zentrale Au+Au-Kollisionen bei 10 AGeV um 3,8 % bzw. 3,4 %. Über alle getesteten Strahlenergien und Kollisionssysteme hinweg übertrifft der GNN-basierte Spurfinder den CA-basierten Spurfinder. Schließlich wird eine erste GPU-Implementierung vorgestellt, die das Potenzial des Algorithmus für die Online-Verarbeitung demonstriert.

Insgesamt etabliert diese Arbeit einen neuartigen und effektiven Spurfindungsalgorithmus für das CBM-Experiment. Sie zeigt, dass die Kombination von Deep Learning mit physikalisch motivierten Rekonstruktionstechniken eine tragfähige Strategie für die Spurfindung in Experimenten mit hohen Raten und hohen Multiplizitäten ist.

Contents

Chapter 1

Introduction

The large variety of natural phenomena observed in the universe arise from a small set of elementary particles interacting through four fundamental forces: gravitation, electromagnetism, and the weak and strong interactions. While gravitation is negligible at typical particle physics scales, the remaining three forces are unified within the Standard Model (SM) of particle physics. The SM provides a framework for elementary particles which has been experimentally validated to exceptional precision (Fig. 1.1). Despite its immense success, the SM remains incomplete. For instance, the observation of neutrino oscillations [1] implies non-zero neutrino masses, necessitating physics beyond the minimal SM. Furthermore, within the SM, complex emergent phenomena arising from the strong interaction, such as the binding of quarks into hadrons and the dynamical generation of the vast majority of visible mass, are not yet fully understood.

The strong interaction, described by Quantum Chromodynamics (QCD), exhibits two defining features: confinement and asymptotic freedom. These properties lead to a rich phase structure of strongly interacting matter, which is mapped as a function of temperature T and baryon chemical potential μ_B (Fig. 1.2). First-principles calculations using Lattice QCD predicts a smooth crossover from hadronic matter to a deconfined Quark-Gluon Plasma (QGP) at small μ_B [3], with a transition temperature $T_c \approx 155\,\mathrm{MeV}$ [4, 5]. However, at finite μ_B, direct numerical techniques cannot be applied [6]. In the high-μ_B regime, effective models suggest that the transition becomes first-order. If the phase diagram features a smooth crossover at low μ_B and a first-order transition at high μ_B, there must exist a critical endpoint where the first-order phase boundary terminates. The search for this critical point, and the mapping of the phase diagram in the

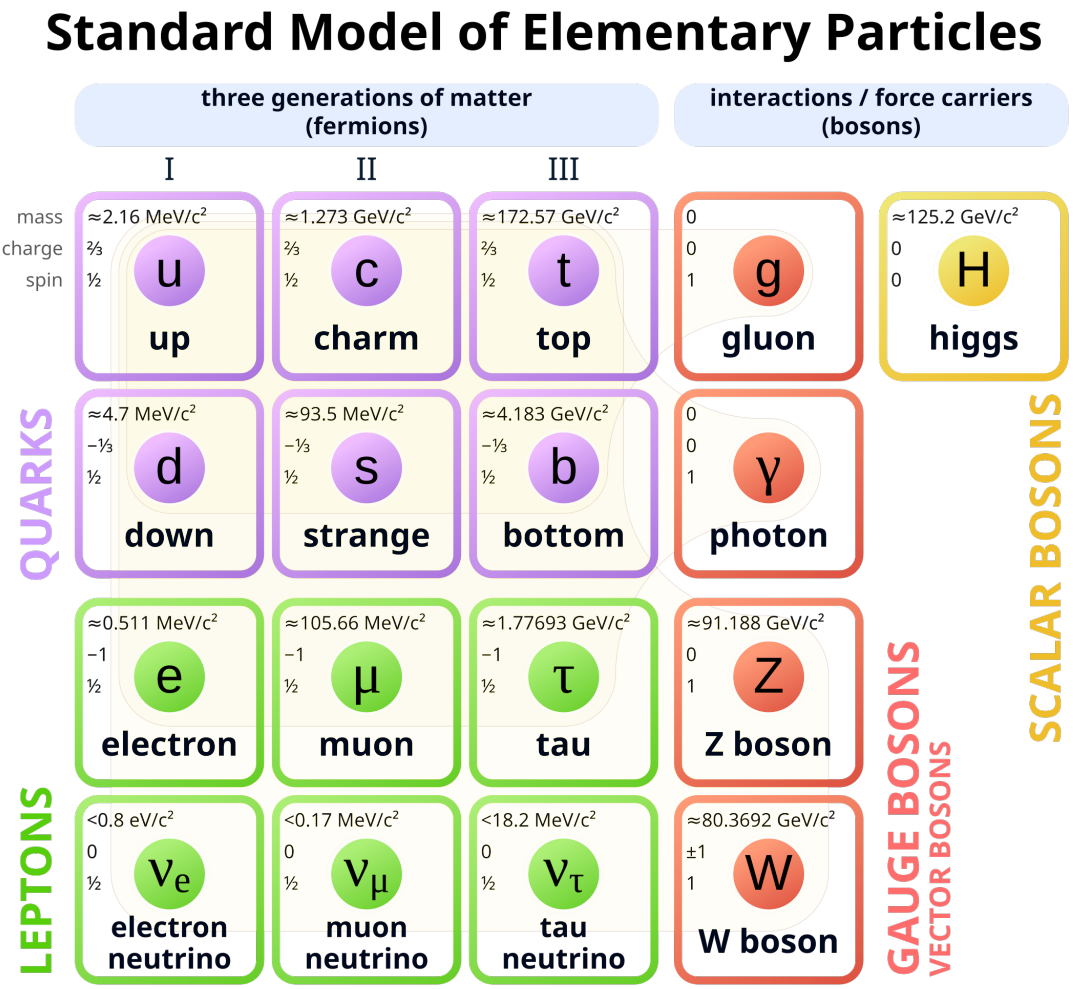


Figure 1.1: Standard model of elementary particles: the 12 fundamental fermions, arranged in three generations, and 5 fundamental bosons. Brown loops indicate which bosons (red) couple to which fermions (purple and green). Figure from [2].

high-density region, stands as a central challenge in modern nuclear physics.

The primary experimental tool for exploring this phase diagram is relativistic heavy-ion collisions. When heavy nuclei, such as gold or lead, collide at relativistic velocities, they create an interaction region where the energy density exceeds that of ordinary nuclear matter by several orders of magnitude. Under these extreme conditions, the formation of QGP is expected. Cosmological models predict that the universe was filled with this QGP state microseconds after

the Big Bang. Consequently, high-energy nuclear collisions provide the unique opportunity to recreate and study the primordial matter that filled the early universe in a laboratory setting.

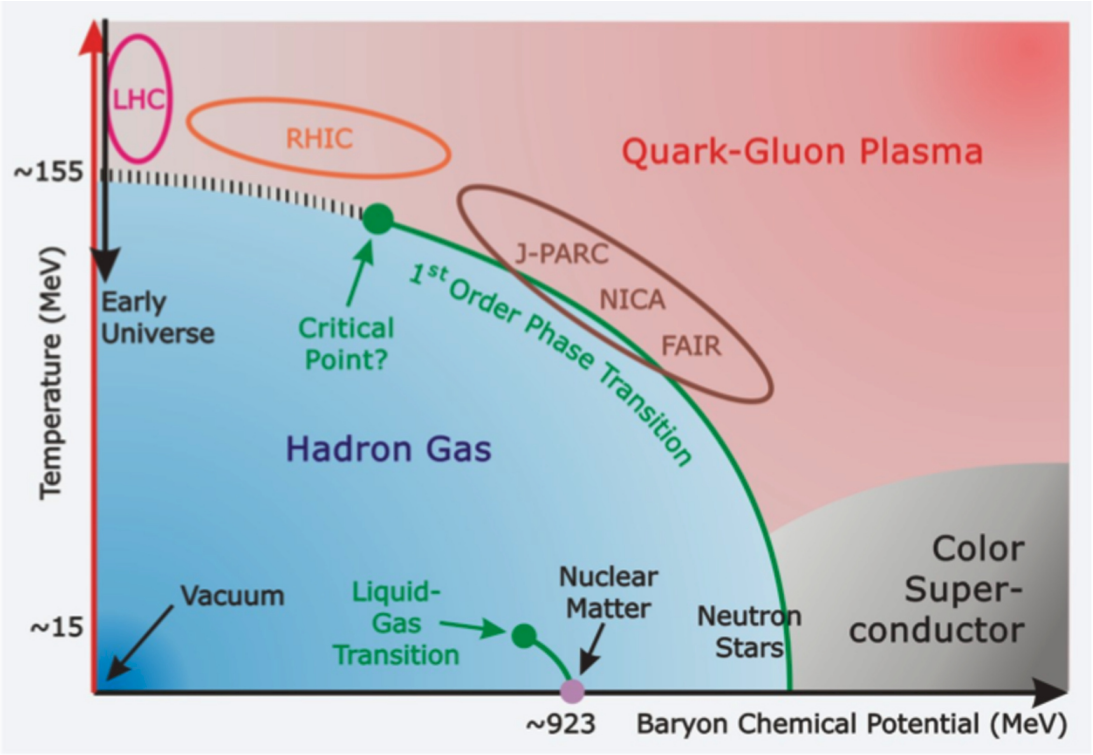


Figure 1.2: Current theoretical predictions about the QGP phase diagram in the temperature (T) and baryon chemical potential (μ_B) plane. The baryon chemical potential is a measure of the difference between number of baryons and anti-baryons. The regions accessible to different experiments are marked. Figure from [7].

The Compressed Baryonic Matter (CBM) experiment at the future Facility for Antiproton and Ion Research (FAIR) in Darmstadt, Germany, is designed to address these open questions. While collider experiments at RHIC (STAR [8], PHENIX [9]) and the LHC (ALICE [10]) have successfully characterized the QGP at low μ_B, CBM will explore the high-μ_B, moderate temperature region of the QCD phase diagram (Fig. 1.2). Using fixed-target heavy-ion collisions at beam energies of 2–11 AGeV, the experiment aims to create matter with baryon densities several times that of normal nuclear matter. This regime offers access to rare diagnostic probes, including multi-strange hyperons, charmed hadrons,

dileptons, and collective flow. These signals are highly sensitive to the equation of state of dense nuclear matter and in-medium hadronic modifications, making them crucial for identifying phase transitions and the critical endpoint [11].

To collect sufficient statistics of these rare probes, the CBM experiment must operate at extremely high interaction rates of up to 10^7 collisions/s (10 MHz). This rate generates raw data volumes on the order of terabytes per second which makes archiving all raw event data infeasible. Crucially, the complex signatures of the most interesting physics channels, such as multi-strange hyperons and charm hadrons, preclude the use of simple, low-level hardware triggers [12]. Therefore, CBM employs a continuous, free-streaming readout architecture combined with real-time event reconstruction and selection to reduce the data volume before permanent storage. To achieve the required throughput, event reconstruction must be highly efficient, parallel, deployable across heterogeneous architectures (CPUs, GPUs, and FPGAs), and sufficiently robust against high track multiplicities and detector backgrounds.

A central component of this online processing chain is track reconstruction: the identification and parameter estimation of charged-particle trajectories. This process is divided into two sub-tasks: *track finding* (associating detector hits to a single particle) and *track fitting* (estimating the momentum and trajectory parameters). In the high track multiplicity environment of CBM, track finding is particularly challenging. Algorithms must rapidly resolve massive combinatorial complexity while accounting for material effects and tolerating detector noise. Furthermore, as track reconstruction serves as a high-level software trigger for event selection, it must maintain high efficiency across all event topologies. Any inefficiency correlated with a particular track type could introduce a selection bias against some signatures, potentially compromising the core physics analysis. Consequently, the quality and throughput of the tracking algorithms directly determine the experiment's sensitivity to rare physics signals.

This thesis introduces a new Graph Neural Network (GNN)-based track finding algorithm for the main tracking system of the CBM experiment. Unlike other recent approaches which apply deep learning to the track finding problem, the algorithm presented in this thesis is distinguished by incorporating KF-based filtering and traditional selection cuts into the pipeline, reducing the reliance on neural networks where geometric heuristics suffice. Thus the algorithm optimally combines the representational power of data-driven deep learning models with the physical robustness of KF-based parameter estimation.

The remainder of this thesis is organised as follows. Chapter 2 describes the CBM detector and sub-systems. Chapter 3 summarises relevant high-performance computing concepts and introduces the portable C++ library `xpu` used in this work. Chapter 4 introduces the required deep learning background: loss function, optimisation, generalisation, neural network architectures (MLP, GNN) and the ANN4FLES package. Chapter 5 explains the GNN-based track finding algorithm in detail. Chapter 6 presents the performance of the GNN-based algorithm for 10 A GeV Au+Au collisions and compares with the Cellular Automaton baseline. The impact on short-lived particle yields, runtime performance and results from a GPU implementation are also discussed. Results at other energies and collision type (p+Au) are included in Appendix A.

Chapter 2

The Compressed Baryonic Matter (CBM) Experiment

The Compressed Baryonic Matter (CBM) experiment will be located at the future Facility for Antiproton and Ion Research (FAIR) in Darmstadt, Germany. It is aimed at investigating the properties of strongly interacting matter at extreme conditions. This chapter gives a brief overview of the main detectors which comprise the CBM experiment. There is a more detailed discussion on the detectors responsible for track finding and the computational infrastructure for even reconstruction.

2.1 The Facility for Antiproton and Ion Research (FAIR)

The Facility for Antiproton and Ion Research (FAIR) is an accelerator complex currently under construction at the GSI Helmholtz Centre for Heavy Ion Research in Darmstadt, Germany. The FAIR research program is divided into four experimental pillars:

- Atomic, Plasma Physics and Applications (APPA)
  APPA encompasses an array of experiments to understand how matter, from an individual atom to plasma and complex matter such as biological tissue, interacts with radiation. High intensity heavy-ion and plasma beams will be used for these investigations. Applications of this research include cancer therapy with radiation, finding materials best suited for use as

probes in space and testing Quantum Electrodynamics (QED) in extremely strong electromagnetic fields [13].

- Compressed Baryonic Matter (CBM)
  CBM is focused on the exploration of the QCD phase diagram and nuclear matter equation of state in the region of high baryon densities. Measurements of the rare probes needed for these studies will be collected using nucleus-nucleus collisions at interaction rates of up to 10 MHz [14].

- Nuclear Structure, Astrophysics and Reactions (NUSTAR)
  NUSTAR is dedicated to exploring the structure and interactions of nuclei, especially those far from stability. Using radioactive ion beams delivered through the Super-FRS separator [15], NUSTAR will enable precision studies relevant to nuclear astrophysics (e.g., nucleosynthesis), nuclear structure near drip lines, and fundamental reactions in exotic systems [16].

- Antiproton Annihilation at Darmstadt (PANDA)
  PANDA will use proton–antiproton annihilation to study the physics of strong interactions at medium energies. This will allow many studies such as hadron spectroscopy, search for exotic hadrons (multiquarks and gluonic hadrons), understanding origin of hadron masses, nucleon structure and hypernuclei [17].

This thesis is focused on the CBM experiment which will be described in more detail in the rest of this chapter.

2.2 The Compressed Baryonic Matter (CBM) Experiment

The primary scientific motivation for CBM lies in its access to the high baryon-density region of the QCD phase diagram (Fig. 2.2), where theoretical predictions suggest the existence of a first-order phase transition and a critical endpoint in the transition from confined hadronic matter to deconfined quark-gluonic matter. To address these questions, CBM targets key observables such as dileptons production, charm production, multi-strange, especially hyperon, particle production, and global observables like yield and collective flow of hadrons, and event-by-event fluctuations [11].

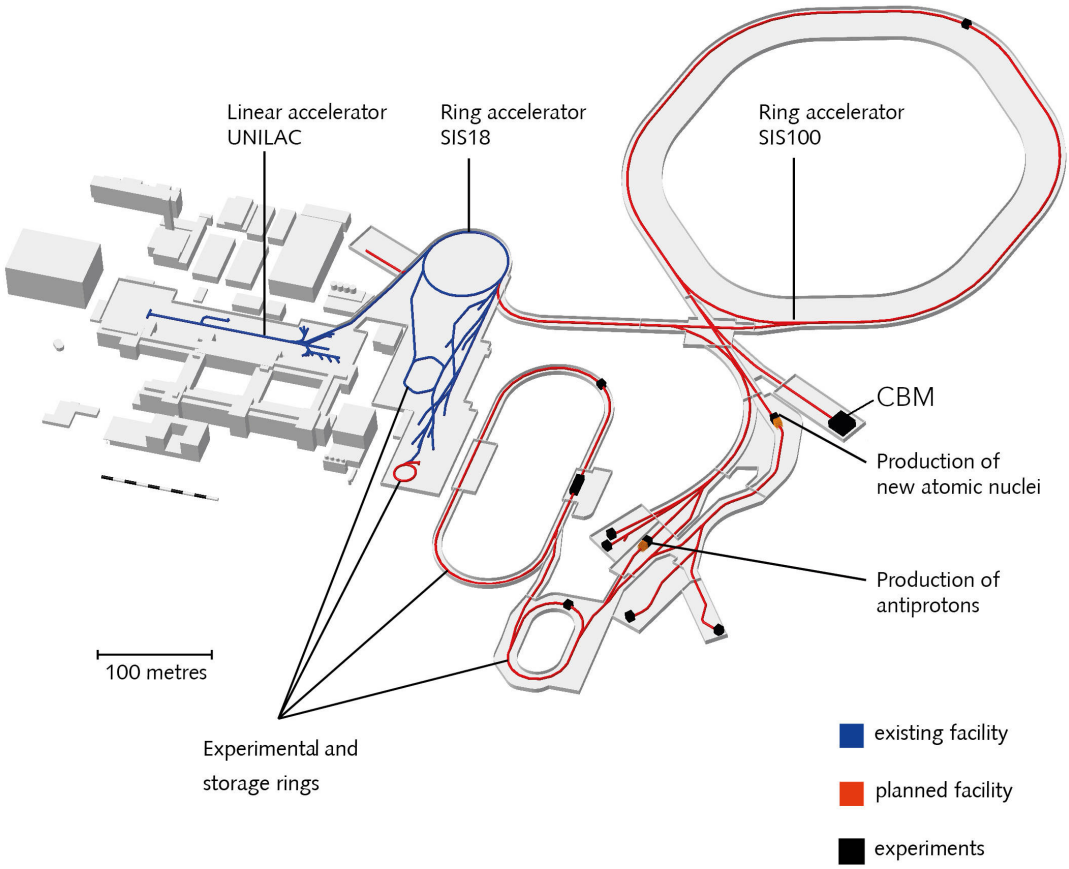


Figure 2.1: Layout of the FAIR accelerator complex and its experimental halls [18].

CBM will operate in fixed-target mode with heavy-ion and proton beams delivered by the SIS100 synchrotron (Fig. 2.1). Protons will be accelerated up to 29 GeV, gold (Au) up to 11 AGeV (GeV per nucleon) and other nuclei with $Z/A = 0.5$ up to 14 AGeV. The results of transport-model calculations for central Au+Au collisions show that densities exceeding five times the nuclear saturation density (0.17 fm^{-3}) can be achieved under SIS100 beam conditions [20]. By systematically varying the beam energy and colliding ions, CBM will explore a new region of the QCD phase diagram and enable tests of theoretical predictions regarding deconfinement and chiral symmetry restoration.

The observables of interest to CBM occur rarely, thus, to collect statistically significant results, CBM must operate at extremely high interaction rates. The

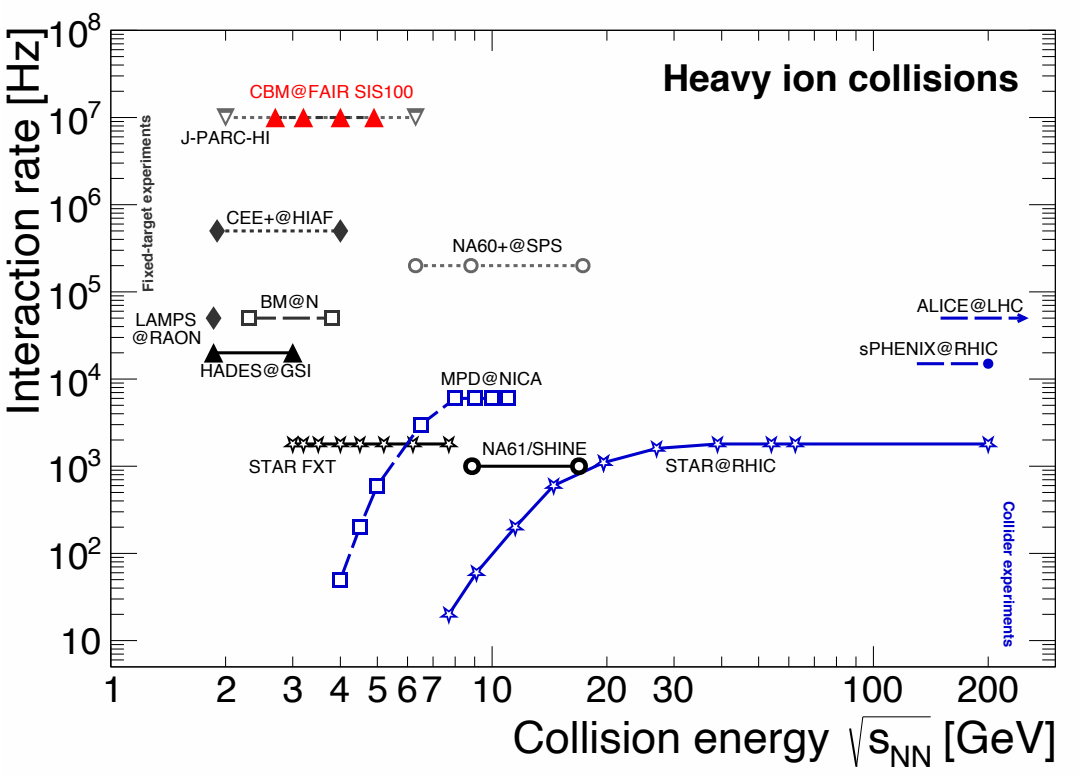


Figure 2.2: The interaction rates and collision energies of existing and upcoming heavy ion experiments. The CBM (marked in red) is at the top left. Figure from [19].

raw data produced is of the order of terabytes per second which is infeasible to archive completely. Furthermore, the signatures of the most interesting physics channels are too complex for the use of low-level hardware triggers and require full event reconstruction. This demanding environment requires CBM to develop fast, radiation-hard detector systems with a free-streaming, self-triggering readout architecture and computing hardware and software capable of real-time event reconstruction and selection [21]. In this context, algorithms using machine learning, and especially deep learning, are being explored for online classification and selection of physics events [22].

2.3 CBM Experimental Setup

The CBM experimental strategy is designed to perform both integral and differential measurements of nearly all particles produced in nuclear collisions. The

measured quantities include yields, phase-space distributions, correlations, and fluctuations. The experimental setup is an array of sub-detectors aligned along the beam axis, each specialized for different tasks. Figure 2.3 shows the detector arrangement in the setup for electron measurements.

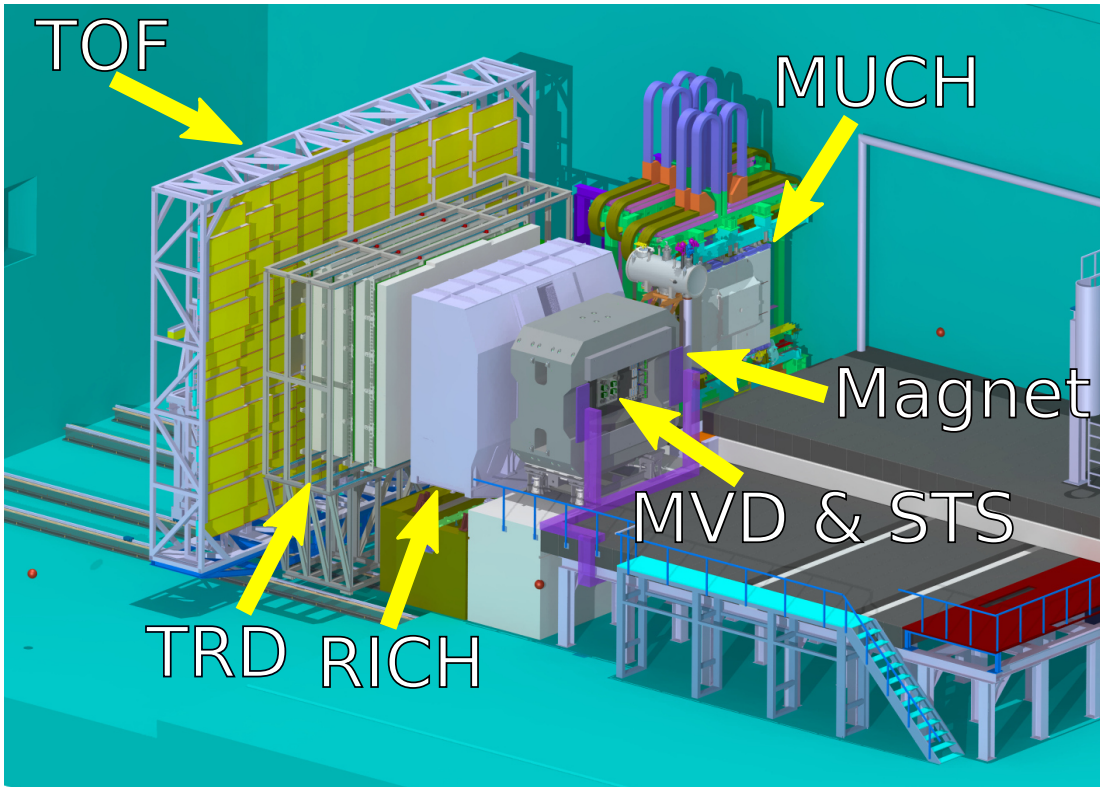


Figure 2.3: A Computer Aided Design (CAD) rendering of the CBM detector in the electron configuration. The unused Muon Chamber (MUCH) is shown in the parking position. Beam enters from the right. Figure from [23].

The rest of this section provides a brief overview of these sub-detectors.

2.3.1 Dipole Magnet

The main tracking system of CBM is placed inside a superconducting dipole magnet for precise momentum reconstruction. A magnetic field integral of 1 Tm enables momentum resolution of $\Delta p/p \lesssim 1\%$ in track reconstruction. It has a large aperture, vertical and horizontal acceptance of $\pm 25°$ and $\pm 30°$ respectively, so that the target and all the tracking stations can be placed in the magnet gap. The dipole magnet produces an inhomogeneous magnetic field which must be

accurately mapped for track parameter reconstruction [24].

Simulations show fringe magnetic fields strong enough to disturb the electron trajectories and the quality of Cherenkov rings in the downstream RICH detector necessitating the use of a solid iron shielding box.

2.3.2 Micro Vertex Detector (MVD)

The Micro Vertex Detector (MVD) provides precise hit measurements in the highest track density region close to the target. It is located immediately downstream of the target and inside the dipole magnet. The main utility of the MVD is in enabling the reconstruction of particles with momenta down to $\sim 300\,\mathrm{MeV}/c$. MVD also allows for secondary vertex reconstruction with precision better than 100 μm along the beam axis for daughter particles with momenta greater than $\sim 1\,\mathrm{GeV}/c$ [25]. The high spatial precision of MVD ($\sim 5\,\mu m$) makes it especially useful to background rejection in dielectron measurements through identification of $e^+\,e^-$ pairs from photon conversion and π^0 Dalitz decay.

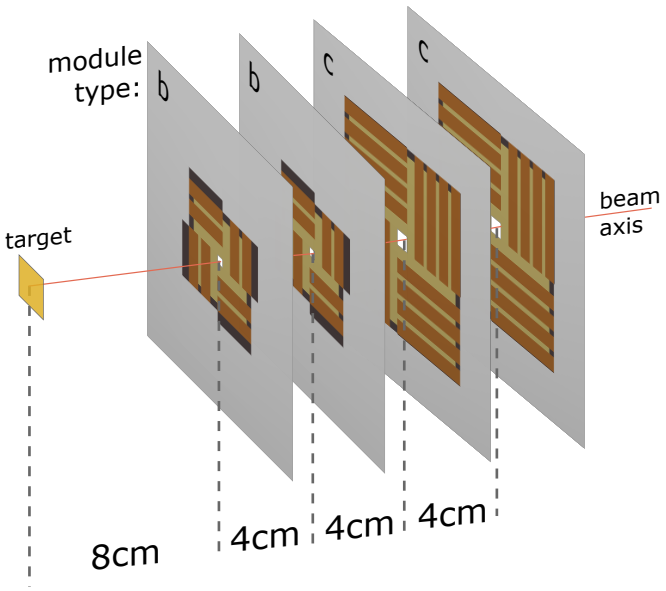


Figure 2.4: Layout of MVD stations in the track reconstruction setup. The MVD is placed closest to the target and inside the dipole magnet. Figure from [25].

The MVD consists of four ultra-thin detection planes mounted inside the vacuum chamber, placed equally apart at distances between 5 or 8 and 20 cm down-

stream of the target. Two setups are provided, *TR* for track reconstruction and *VX* for vertex finding. In the TR setup, the four detection planes are placed equidistantly between 8 and 20 cm (Fig. 2.4), while in the VX setup, the planes are placed closer to the target from 5 to 20 cm. The focus of this thesis is on track reconstruction and thus only the TR setup is used.

Each plane is subdivided into four quadrant modules arranged around the beam axis, covering fully the azimuthal direction and the polar range $2.5° \leq \theta \leq 25°$. The modules are instrumented with 50 μm thick, radiation-hard CMOS Monolithic Active Pixel Sensors (MAPS) mounted in a back-to-back arrangement on thin support sheets made of material which efficiently dissipates heat. The material budget of the stations is extremely light. It is of the order of 0.5% x/X_0 in the TR configuration, and the first station in the VX geometry is even lighter at ∼0.3% x/X_0.

2.3.3 Silicon Tracking System (STS)

The Silicon Tracking System (STS) is the core detector of the CBM experiment. Collisions in CBM are expected to produce up to 1000 charged particles in the acceptance region $2.5° \leq \theta \leq 25°$. To meet the physics goals, the tracking system must obtain track reconstruction efficiency of at least 95% along with momentum resolution $\Delta p/p \lesssim 1\%$. These requirements impose constraints on the sensor technology, material budget, read-out electronics and also on the reconstruction algorithm which is the topic of this thesis.

The STS is situated inside the magnet and downstream of the MVD. It is composed of eight planar tracking stations placed at distances between 30 cm and 100 cm downstream of the target. Each STS station is made of carbon fiber support ladders carrying 320 μm thick double-sided silicon micro-strip sensors. In total, there are 876 sensors each with 2048 strips [27]. The silicon detector is a reverse-biased diode with the depleted zone acting analogous to an ionization chamber (Fig. 2.6). Charged particles passing through the detector create electron-hole pairs which drift under the applied electric field to the electrodes and produce a signal that is registered by the electronics. A group of neighbouring strips, in addition to the front and back, in a detector are commonly triggered by a single particle. A procedure called hit reconstruction is responsible for converting clusters to hits that are used by the track reconstruction algorithm.

The front and back sides of the strips are tilted by a relative angle of 7.5°.

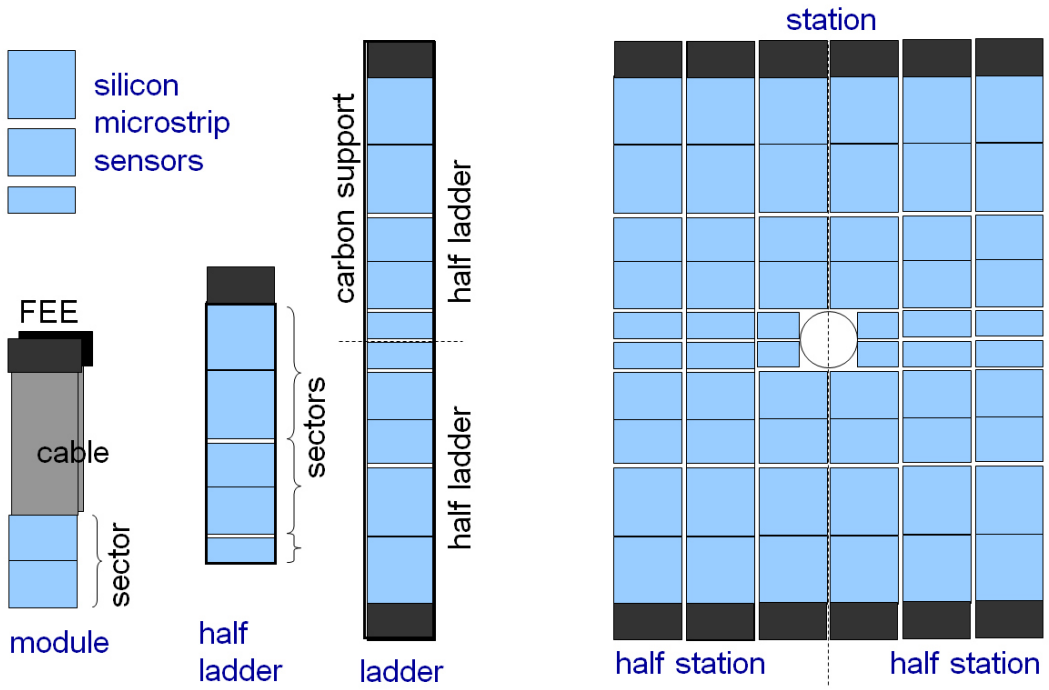


Figure 2.5: Layout of a STS station and the intermediate building blocks. The three sizes of the sensors can be seen in the top left. Note how the read-out electronics are all at the periphery. Figure from [26].

This allows reconstructing of multiple hits on the same sensor at the expense of a poorer vertical direction resolution and minimization of noise hits (Fig. 5.8 and discussion in Sec. 5.7). Three different dimensions of microstrips are used with the smaller strips used in the innermost regions with the highest hit density (Fig. 2.5). The sensors are read out via ultra-thin multi-line micro-cables which connect to self-triggering front-end electronics placed at the periphery of the station. Micro-strip sensors allow the placement of read-out outside the acceptance. This reduces the material budget and thus multiple scattering significantly. Since the momentum resolution depends primarily on multiple scattering, the use of micro-strip sensors is key for achieving the desired resolution.

2.3.4 Ring Imaging Cherenkov Detector (RICH)

Efficient identification of electrons and positrons over a wide momentum range up to 10 GeV/c is essential for accessing rare probes such as vector-meson decays

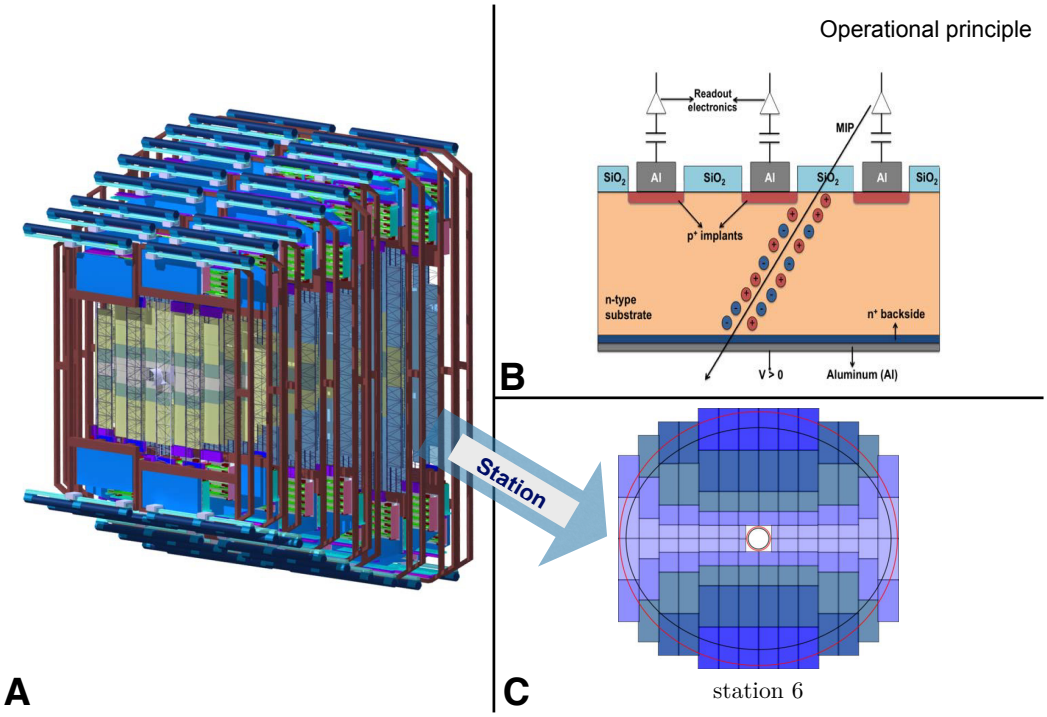


Figure 2.6: STS Detector (A) Layout of the stations. (B) Operational principle of the silicon strip detectors. (C) Layout of the STS station 6. The cutout for the beam pipe can be seen in the centre. Figure from [26, 28, 29].

($\rho, \omega, \phi \to e^+e^-$) and charm-pair decays. These electromagnetic channels provide clean probes of the dense baryonic matter created in nucleus–nucleus collisions. The RICH detector is designed to deliver the required electron identification capability, achieving a pion-suppression factor of order 10^2.

The operating principle is based on the Cherenkov effect [30]. A charged particle traversing a medium with speed v exceeding the local phase velocity of light, c/n, emits a cone of Cherenkov photons. The half-angle of the cone, θ_C, satisfies

$$\cos\theta_C = \frac{1}{\beta n},$$

where $\beta = v/c$ and n is the refractive index of the radiator. Measuring θ_C provides the particle velocity, which combined with the momentum from tracking, allows inference of particle mass. The emitted photons, mostly in the visible and UV range, are reflected by a focusing mirror system onto a pixelated photon-detector plane, allowing the reconstruction of characteristic Cherenkov rings.

RICH is located downstream of the dipole magnet and the silicon tracking system, approximately 1.6 m from the target. The radiator consists of a 1.7 m long CO_2 gas volume operated at 2 mbar overpressure. Charged pions begin to emit Cherenkov light in CO_2 only above a threshold momentum of about 4.6 GeV/c. Below this momentum, pions are fully suppressed; above threshold, further pion rejection is achieved because θ_C remains particle-dependent via the Lorentz factor.

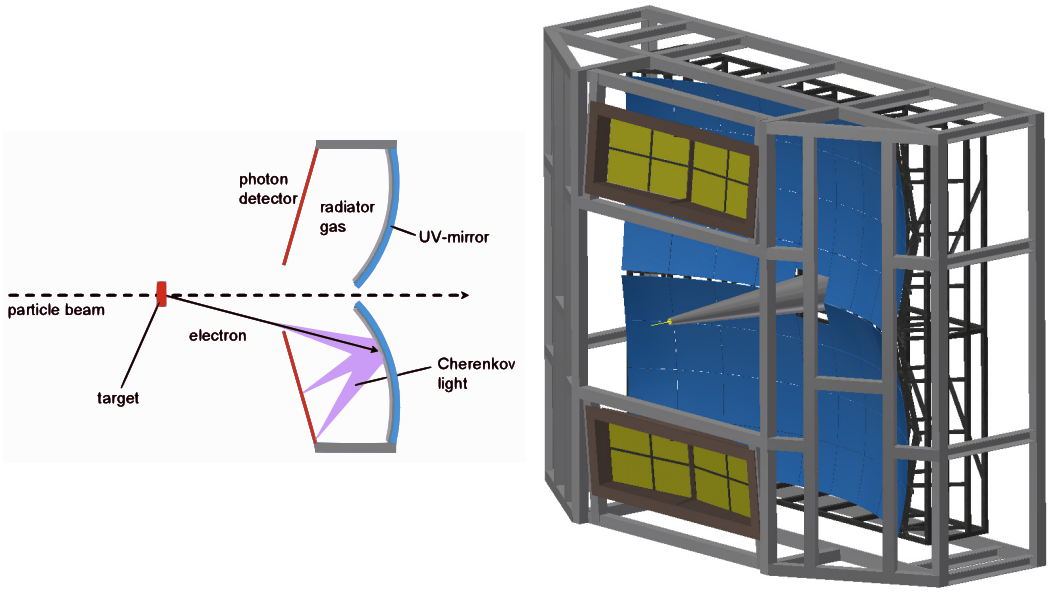


Figure 2.7: Schematic drawing of the RICH detector. Cherenkov radiation, produced by electrons, is reflected towards photon detectors (green) using mirrors (blue) [31].

The Cherenkov photons are collected by two arrays of segmented spherical mirrors coated with Al+MgF_2 for enhanced UV sensitivity. They are focused onto two photon-camera planes equipped with approximately 55 000 channels based on Multi-anode PhotoMultiplier Tubes (MaPMTs). These provide high granularity, excellent photon-detection efficiency, and stable operation at the high interaction rates expected in CBM. A magnetic shielding box around the cameras (not shown) suppresses residual fringe fields from the dipole magnet. The entire RICH detector is mounted on rails and can be retracted from the beam line to permit installation of the muon-detection system (MuCh) when operating in the muon configuration [31].

Beam-test measurements of a full-length prototype indicate that central Au+Au collisions at 25 A GeV produce $\mathcal{O}(100)$ Cherenkov rings per event, each containing on average around 22 detected photons.

2.3.5 Muon Chamber System (MuCh)

In the muon configuration of CBM, the Muon Chamber System (MuCh) replaces the RICH detector in order to enable identification of muon pairs. Measurements of both electron and muon pairs provide complementary information on dilepton production, allowing a more complete reconstruction of vector-meson and charmonium decays in dense matter. The MuCh is designed to operate in the high particle-density environment expected at beam energies from 4 to 40 A GeV.

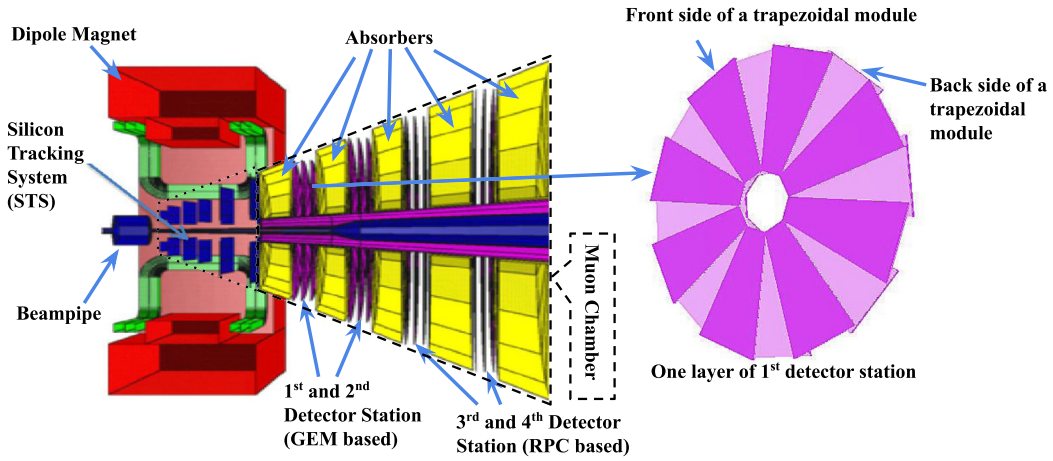


Figure 2.8: Left: The CBM experiment in the muon configuration with the MuCh detector installed in place of the RICH. Right: Schematic representation of the detector modules within a MuCh station [32].

MuCH uses the technique of hadron-absorber filtering combined with tracking chambers. Dense absorber layers stop or strongly scatter hadrons and low-energy particles, while muons penetrate the absorber system with only moderate energy loss. Tracking stations placed between absorber layers record the residual track segments, which can then be matched to STS tracks to identify muon candidates and suppress hadronic background. The alternating absorber-tracker geometry (Fig. 2.8) provides momentum-dependent muon identification under high multiplicity conditions.

2.3.6 Transition Radiation Detector (TRD)

The Transition Radiation Detector (TRD) complements the RICH in electron identification by providing electron-pion separation at momenta above approximately 1.5 GeV/c. In addition, the TRD contributes to intermediate tracking and the identification of nuclear fragments in the forward region.

Transition radiation (TR) is emitted when a charged particle crosses the interface between two media with different dielectric constants [33, 34]. The sudden change in the electromagnetic field induces the emission of soft X-ray photons, with intensities scaling with the Lorentz factor γ and so only highly relativistic particles ($\gamma \gtrsim 1000$) emit TR efficiently. At CBM energies, this means that electrons, having much higher γ values than hadrons of the same momentum, can be distinguished based on their additional TR photon yield. By detecting both the ionization energy loss (dE/dx) and the transition-radiation photons, the TRD enables a statistical separation between electrons and pions.

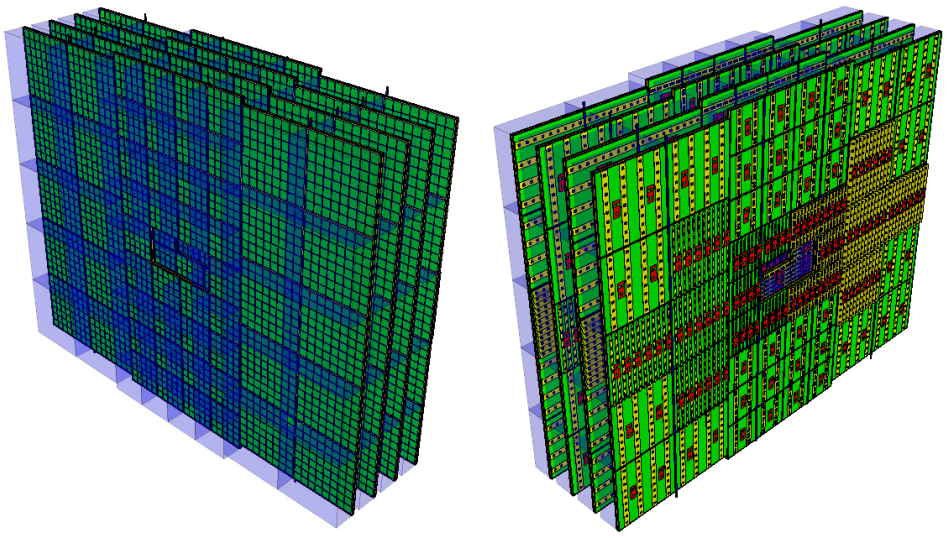


Figure 2.9: CBM-TRD geometry for SIS100, consisting of one station with four detector layers. The front view (left) shows the radiator boxes, while the rear view (right) displays the backplanes and front-end electronics [35].

The CBM TRD for SIS100 consists of a single station composed of four identical layers of detector modules (Fig. 2.9). Each layer includes a radiator made of polyethylene-foam foils to generate TR photons, followed by a gaseous drift chamber for detection. The drift chamber comprises a 5 mm drift region and a

7 mm amplification region based on multiwire proportional chamber (MWPC) technology. Charged particles ionize the detector gas, while TR photons produce additional secondary electrons.

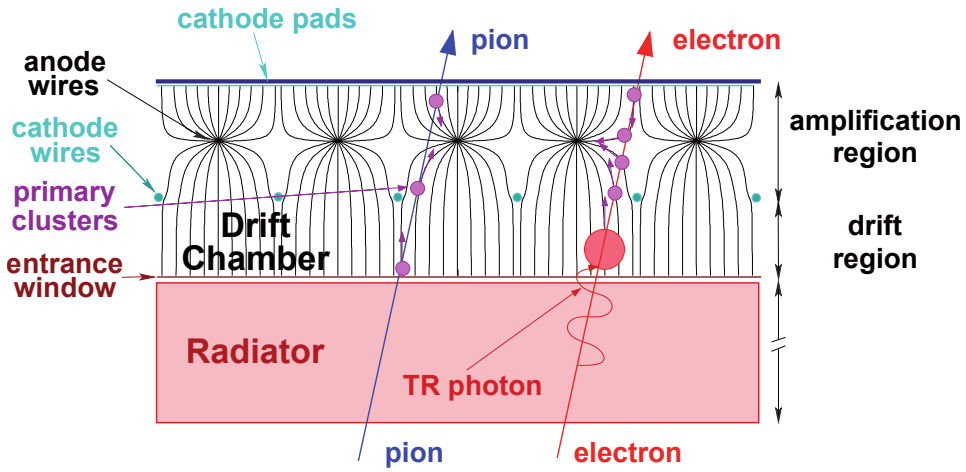


Figure 2.10: Illustration of the operating principle of a TRD module in CBM. While all charged particles deposit energy in the readout chamber, electrons also produce TR traversing the radiator. Electrons can thus be identified by the total energy deposited [35].

Electron–pion separation is achieved by analysing the energy-deposition distributions over the four layers; electrons produce on average larger signals due to the absorbed TR photons (Fig. 2.10). In addition to particle identification, the TRD provides space points for tracking between the STS and the TOF, improving pattern recognition and momentum reconstruction in the forward region [35].

2.3.7 Time Of Flight detector (TOF)

The Time-of-Flight (TOF) detector provides charged-hadron identification over a wide momentum range. Particle identification is performed by combining the momentum measurement from the upstream tracking detectors with an independent velocity measurement from the TOF system. Using the reconstructed track length L and measured flight time t, the velocity is obtained as $\beta = L/(ct)$. Together with the momentum p estimated from the curvature in the STS, the particle mass is determined using,

$$m^2 = p^2 \left(\frac{1}{\beta^2} - 1 \right). \tag{2.1}$$

At CBM, the dominant contribution to the mass uncertainty arises from the TOF time resolution σ_t, while uncertainties in the momentum and path length are subleading. Thus, the error in m^2, given by,

$$\sigma_{m^2} = \frac{2p^2}{\beta^2}\frac{\sigma_t}{t}. \tag{2.2}$$

is effectively independent of m and is quadratic in p. Consequently, the particle identification capability of TOF decreases with momentum.

Simulations and prototype tests demonstrate that the TOF system provides the required hadron-identification capability, enabling, for example, pion–kaon separation up to $\sim$3.5 GeV/c and proton–kaon separation up to $\sim$6 GeV/c at high rate.

2.4 Data acquisition system (DAQ) and online event processing

Many of the key observables in CBM's physics programme rely on rare processes that occur at rates even smaller than once per one million collisions (Fig. 2.11). To collect statistically significant samples of such rare processes, CBM must operate at interaction rates of up to 10 MHz in Au+Au collisions. This extremely high interaction rate sets strong constraints on the readout electronics and the data acquisition system. The average raw event size of a minimum bias Au+Au collision is approximately 50 kB. At the peak 10 MHz collision rate, the raw data production rate will be around 500 GB/s. The storage of this large raw data is constrained by not only the storage bandwidth but also, and more importantly, the cost of storage media.

Practical considerations require that the raw data rate must be reduced by two orders of magnitude before writing to permanent storage. To achieve this data reduction, raw data from collisions must be analyzed for characteristics specific to the observables of interest. Only events possessing interesting signatures *trigger* permanent storage. For the CBM experiment, a trigger signature could be the off-target decay of hyperons. Detecting this signature requires event reconstruction at least up to the track level, which is computationally nontrivial. Most high energy physics experiments rely on front-end electronics that buffer data for short periods (typically a few microseconds) until a trigger decision is issued. Data from

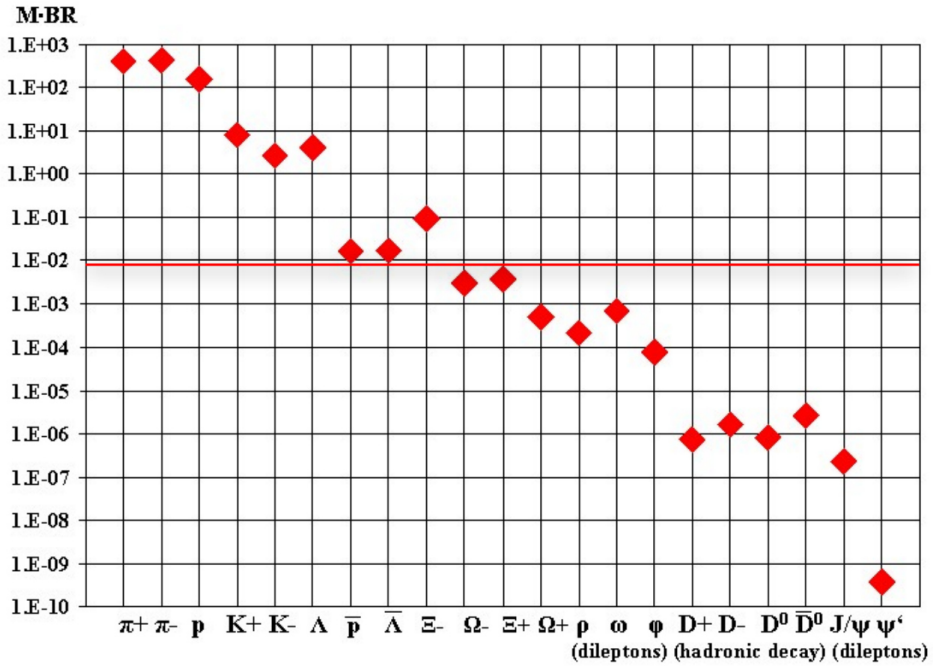


Figure 2.11: The predicted yields (particle Multiplicity (M) × Branching Ratio (BR)) of some observables to be measured in the CBM experiment in 25 A GeV central Au+Au collisions. For the vector mesons (ρ, ω, ϕ, J/ψ, ψ'), the decay into lepton pairs was assumed, while for the D mesons the hadronic decay into kaons and pions. The particles below the red line have not yet been accurately measured in heavy ion experiments in the FAIR energy range. Figure from [11].

these accepted events is then passed to either higher-level triggers or storage [36]. However, triggers in the CBM experiment are too complex to be implemented in conventional trigger logic, they must be evaluated in software on CPUs or GPUs. The magnitude of data produced and the absence of a low-level hardware trigger make conventional DAQ architectures unsuitable for the CBM experiment.

CBM adopts a *triggerless, time-stamped streaming* architecture. All readout electronics are self-triggered, i.e. they send time marked hit data on activation of analog channels above pre-defined thresholds. All data from a collision recorded across all the detectors, and the many readout electronics, must be combined together using the time stamps. The Front-End Electronics (FEE) must be synchronized to sub-nanosecond precision to a central timing system. The FEE are connected to the entry cluster of the First Level Event Selector (FLES) through

optical links that handle control signals, clock information and data transfer as shown in Fig. 2.12.

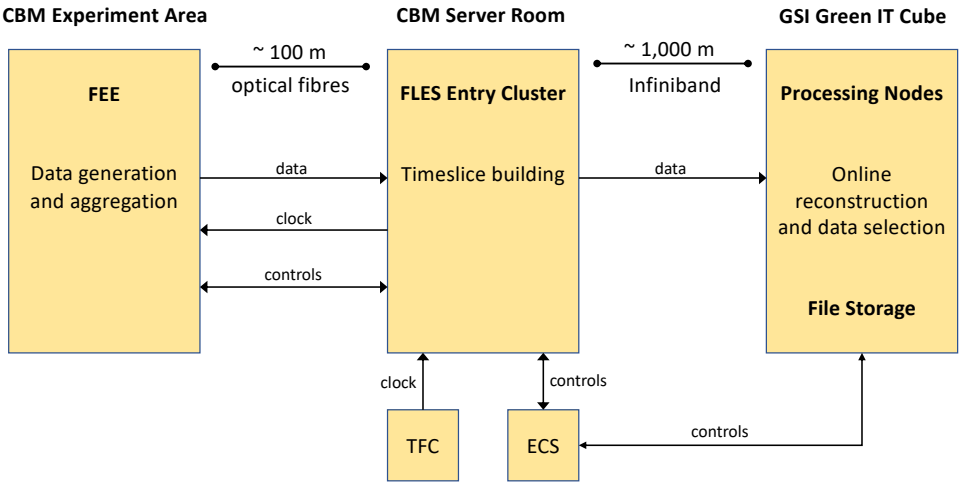


Figure 2.12: A high level schematic of the CBM DAQ structure, illustrating data flow from the front-end electronics (FEE) to the FLES Entry Cluster and finally the Green IT Cube [21].

2.4.1 First Level Event Selector (FLES)

The First Level Event Selector (FLES) is the core data handling, processing and event selection system of the CBM experiment. The FLES is responsible for the interface with the detector readouts, combining and organizing data into appropriate containers and performing the full online reconstruction and analysis for event selection. While the basic design is similar to other high-level software triggers used in modern high-energy physics experiments, the software-only event selection makes it uniquely challenging.

The FLES Entry Cluster is composed of Common Readout Interface (CRI) units which are custom PCIe cards placed on commercial computers. The CRI forwards the clock information from the Timing and Fast Control (TFC) system to the FEE. It also processes and packages the data transmitted from the FEE into a data container suitable for processing at the FLES compute farm (Fig. 2.13). The high-level trigger decision, event building and writing to storage are carried out at the compute farm. The entry cluster of FLES will be located at the CBM server room while the compute farm is located at the Green IT

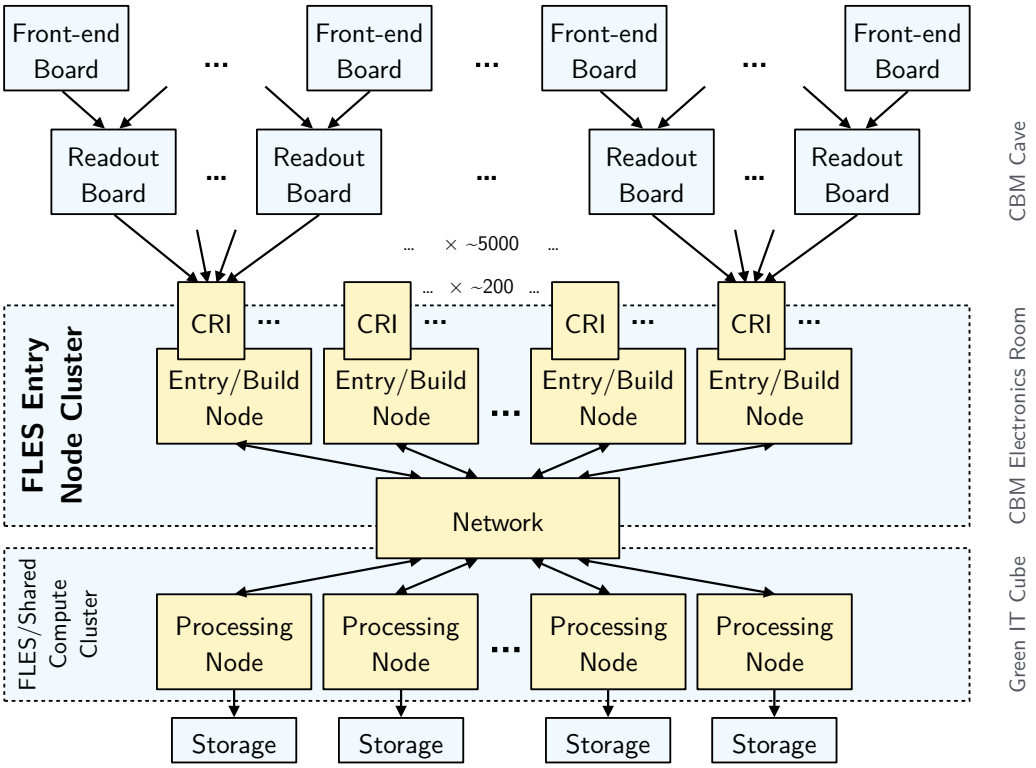


Figure 2.13: Schematic of the First-level Event Selector (FLES) architecture, showing the entry node cluster to be situated in the CBM electronics room and the processing cluster located in the Green IT Cube. Figure from [21].

Cube (see Sec. 2.4.2), approximately 1 km away, and connected by a InfiniBand connection.

The data from a collision event is transferred into FLES distributed across the many entry nodes. For efficient event selection on the compute cluster, it is crucial that all the data from an event, collected across all FEE, is packaged together into a container so that the analysis can be performed on a single node without extensive inter-node communication. Input data streams, which contain time marked hit messages, are divided into small, global time intervals. All the data from a stream which lies in an interval is called a *timeslice component* (TSC). The TSCs from all input data streams of a given period are aggregated into a container called a *timeslice*. A timeslice contains data from all the CBM detectors, for a given time period. The process of building a timeslice is performed in the FLES entry cluster (Fig. 2.14). Timeslices are then transferred to and distributed

across the many processing nodes of the FLES compute cluster for analysis and potential storage.

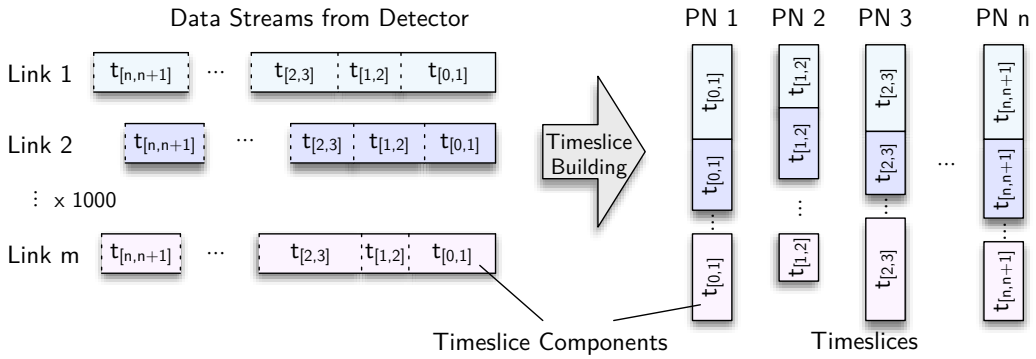


Figure 2.14: Concept of timeslice building in the FLES system. Data streams from different detectors are combined, using time markers, into timeslices covering the same time period and sent to one of the many processing nodes (PN) of the compute cluster for reconstruction, analysis and potential storage. Figure from [21].

2.4.2 The Green IT Cube

To meet the extraordinary data volume and processing demands of the CBM experiment, a dedicated high-performance computing facility, the *Green IT Cube*, was constructed [37]. This facility fulfills the real-time event selection, filtering and storage required by CBM's free-streaming data-acquisition chain, while also addressing energy-efficiency and scalability from the outset.

CBM's online workflow demands the continuous ingestion of interaction rates of up to 10 MHz, immediate reconstruction and selection of physics events, and sustained data transfer to long-term storage. The Green IT Cube provides the compute nodes, storage systems and network infrastructure necessary for these operations. Tightly integrated with the CBM DAQ system, the facility enables communication between front-end electronics, data concentrators and the reconstruction farm, without latency bottlenecks.

The Green IT Cube is designed to deliver high computing performance with minimal overhead in cooling and power consumption. Instead of conventional air-cooling, water cooling is used directly in the server cabinets. It achieves

a power-usage-effectiveness (PUE) of less than 1.07, meaning that the cooling overhead remains under 7% of the total IT power. The facility has space for up to 768 server racks which are interconnected by high-speed InfiniBand links.

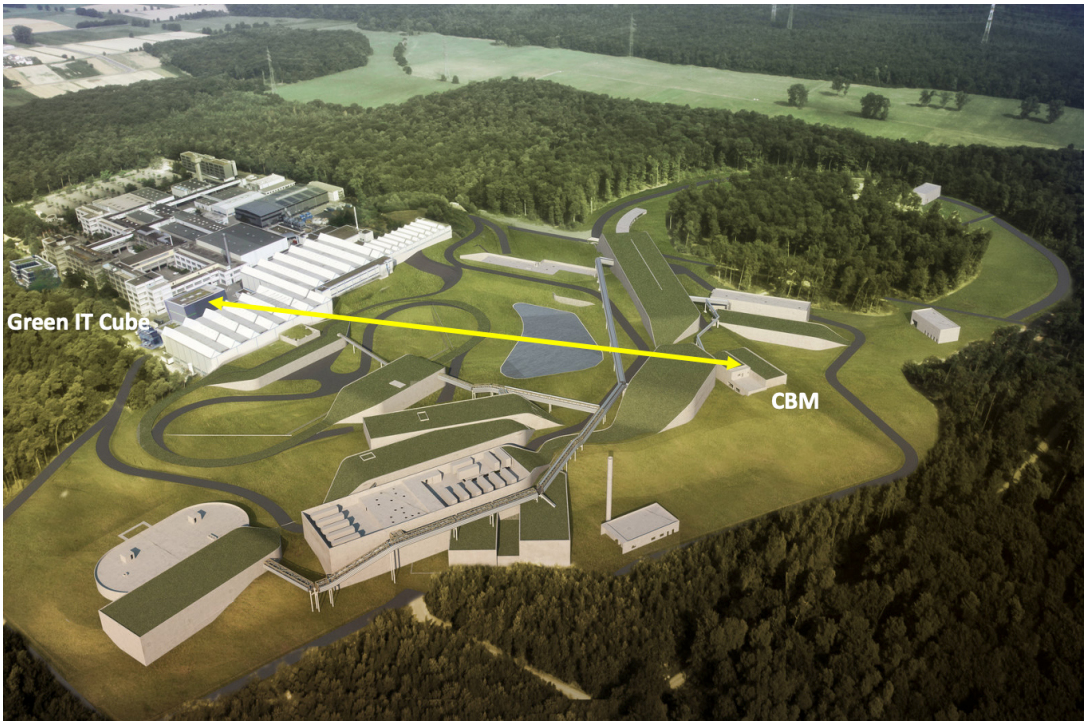


Figure 2.15: A drawing of the planned FAIR campus marking the location of the CBM experiment, where data will be produced, and the Green IT Cube where the data will be processed. Figure from [21].

Raw data from the FLES entry cluster, located on-site, are transferred through fibre optic links to the Green IT Cube. The facility's scalable architecture allows the addition of further compute nodes or GPU clusters. Its physical separation from the experimental hall ensures that computing resources are insulated from beam-induced noise or radiation backgrounds.

Summary

The Compressed Baryonic Matter (CBM) experiment will use fixed-target heavy-ion collisions to investigate the QCD phase diagram at high net-baryon densities. This chapter provides an overview of the major detector subsystems of which the

dipole magnet, Micro Vertex Detector (MVD) and Silicon Tracking System (STS) constitute the core tracking system. Charged particles traversing these detectors leave discrete signals (“hits”), which are subsequently processed by track-finding and track-fitting algorithms to reconstruct the collision geometry and particle trajectories. Because the key physics observables targeted by CBM require real-time event selection, the experiment relies on online reconstruction performed by the First-Level Event Selector (FLES). The FLES architecture consists of an entry cluster, which ingests data from all front-end electronics (FEE), assembles the raw data into time slices, and forwards them to a compute cluster responsible for online processing and data storage. Achieving the necessary throughput and latency constraints demands techniques from high-performance computing, which are discussed in the following chapter.

Chapter 3

High Performance Computing

High Performance Computing (HPC) refers to the aggregation of computing resources to deliver performance far exceeding that of a typical desktop workstation. HPC is ubiquitous in academic research and industry, facilitating complex simulations in science, engineering, and business.

Until the early 2000s, performance gains were primarily achieved through vertical scaling: improvements in semiconductor manufacturing allowed for steady increases in processor frequency, thereby increasing the number of instructions processed per second. However, around 2004, frequency scaling stalled due to the breakdown of Dennard Scaling; power density constraints created a *power wall* around 4 GHz, as seen in Fig. 3.1. Concurrently, the widening disparity between processor speed and memory latency became a critical bottleneck. These physical limitations necessitated a paradigm shift toward horizontal scaling and parallel computing using multi-core processors [38]. Today, HPC relies almost exclusively on parallel computing techniques to maximize throughput.

3.1 Parallel Computing

Traditional software is generally serial: a sequence of instructions executed one at a time on a single processing unit. However, many computational problems can be decomposed into independent sub-tasks suitable for simultaneous execution. Parallel computing exploits this data or task independence to reduce execution time by distributing the workload across multiple processing units. These units range from shared-memory multi-core CPUs to distributed memory clusters and specialized accelerators, such as Graphics Processing Units (GPUs).

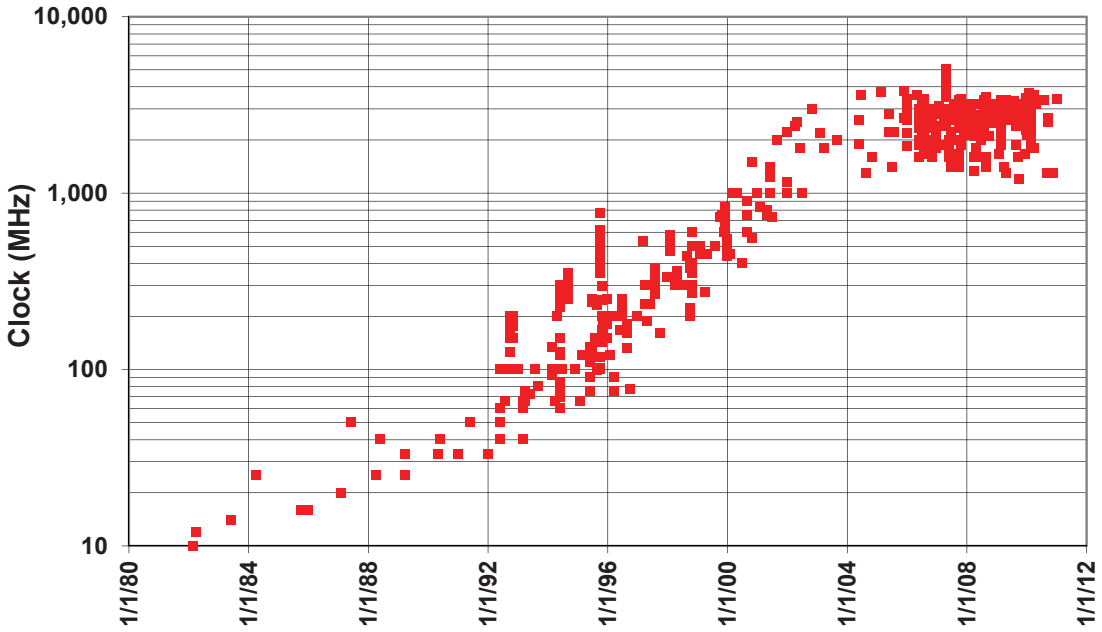


Figure 3.1: Processor frequency (clock rate) trends from 1980 to 2012, illustrating the "power wall" plateau reached around 2004. Figure from [39].

3.1.1 Flynn's Taxonomy

In 1966, Michael J. Flynn introduced a taxonomy classifying computer architectures based on the number of concurrent instruction and data streams available [40]. The four classes are:

1. Single Instruction Single Data (**SISD**): A sequential architecture that exploits no parallelism in instruction or data streams. A single Control Unit (CU) fetches an Instruction Stream (IS) to direct a Processing Unit (PU) to operate on a single Data Stream (DS). This represents the classic von Neumann architecture found in early computers.

2. Single Instruction Multiple Data (**SIMD**): A single instruction is applied simultaneously to multiple data elements. This architecture is ideal for problems with high regularity, such as image processing or matrix operations. Modern CPUs employ SIMD via vector registers (e.g., AVX, NEON). While manual implementation is possible, compiler auto-vectorization is a standard method for utilizing SIMD lanes [45]. An illustration is provided in Fig. 3.3.

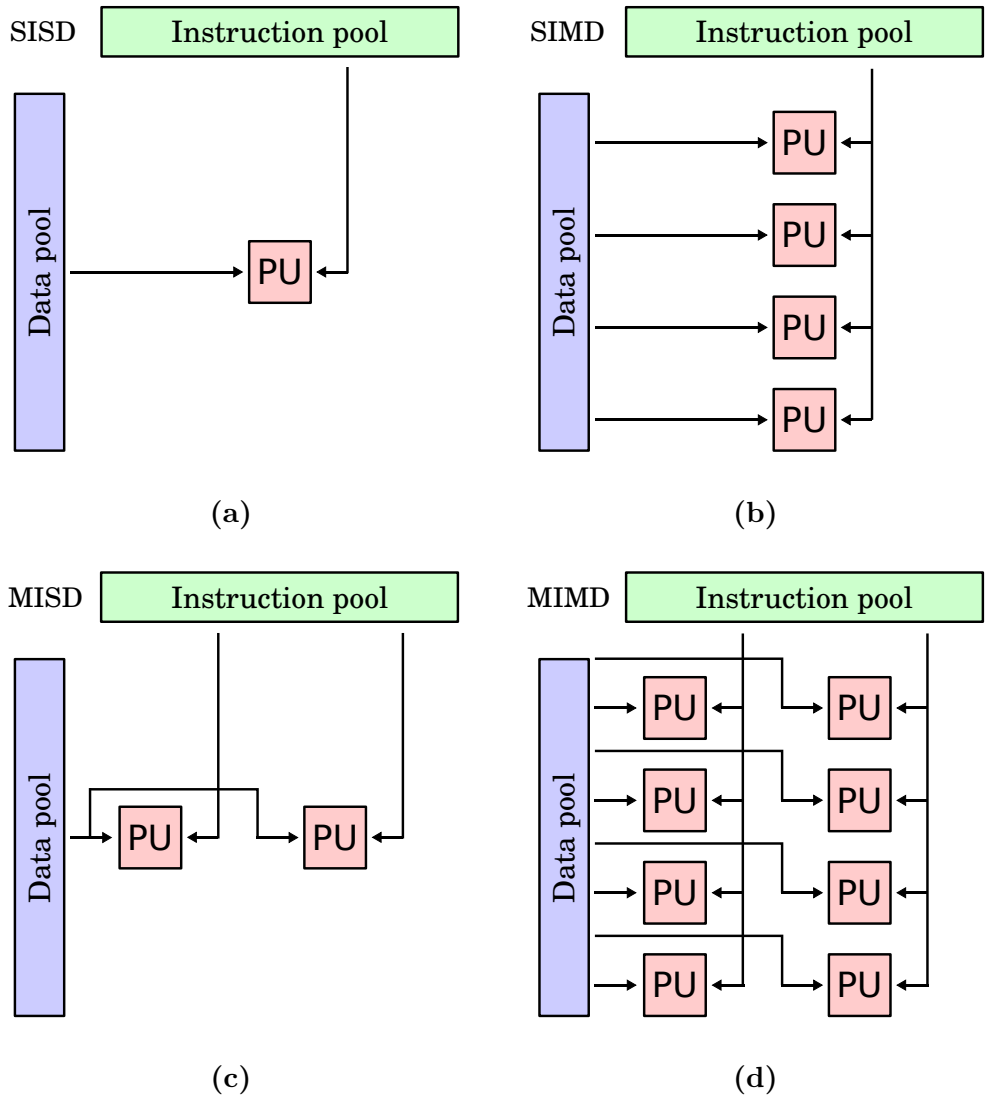


Figure 3.2: Flynn's taxonomy with architectures arranged in a table. The vertical axis is divided into two parts based on single or multiple data streams. The horizontal axis is divided similarly but for the instruction stream. The Processing Units (PUs), instruction and data pool are shown. (a) Single Instruction Stream, Single Data Stream (SISD), (b) Single Instruction Stream, Multiple Data Stream (SIMD), (c) Multiple Instruction Stream, Single Data Stream (MISD), (d) Multiple Instruction Stream, Multiple Data Stream (MIMD). Figures from [41–44].

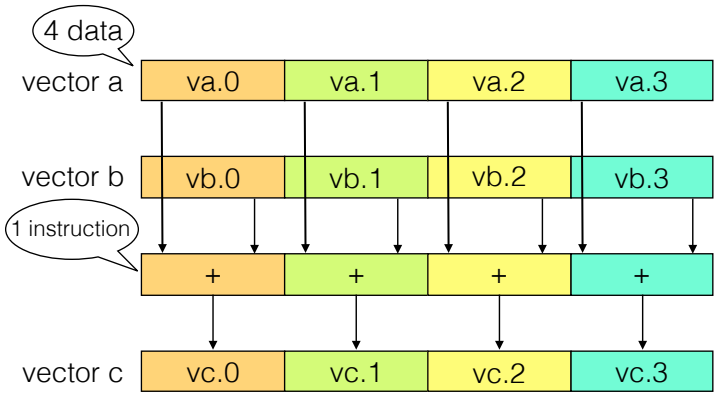


Figure 3.3: Schematic of SIMD calculations. A single operation, addition, iis applied to four data elements in a single clock cycle. Figure from [46]

3. Multiple Instruction Single Data (**MISD**): Multiple instructions operate on a single data stream. This is a rare architecture generally reserved for fault tolerance (e.g., the Space Shuttle flight control system [47]) where heterogeneous systems process the same data to verify results. Modern implementations include systolic arrays (often used in AI accelerators), where data flows through a pipeline of processing units [48].

4. Multiple Instruction Multiple Data (**MIMD**): Multiple processors execute different instructions on different data simultaneously. This is the most common parallel architecture, encompassing multi-core CPUs and computing clusters. A crucial sub-category (or programming model) within MIMD is Single Program Multiple Data (**SPMD**). In SPMD, multiple autonomous processors execute the same program at independent points. While GPUs are physically **SIMT** (Single Instruction, Multiple Threads) architectures, a hardware variant of SIMD, they are typically programmed using the SPMD model.

3.1.2 Task-level parallelism

Task-level parallelism focuses on distributing distinct computational tasks across different processors for concurrent execution. Unlike data parallelism, where the same operation is performed on different data, task parallelism involves executing different instructions on different data streams, necessitating a MIMD

architecture. This paradigm can be implemented at both the process level (*multiprocessing*) and the thread level (*multithreading*).

Achieving effective task-level parallelism requires a careful balance of task granularity and scheduling. An excessively fine-grained division introduces significant scheduling overheads, whereas a coarse division may result in load imbalance and suboptimal resource utilization. Dynamic scheduling is often handled by the operating system or runtime environment to ensure tasks are distributed evenly.

Processes and Threads

Multiprocessing utilizes independent processes with separate address spaces. Communication occurs explicitly, commonly through Message Passing Interface (MPI) or Inter-Process Communication (IPC) mechanisms. This model is advantageous for distributed systems or clusters such as the Green IT Cube (Sec. 2.4.2) where memory isolation provides fault tolerance, albeit at the cost of higher communication latency.

Multithreading operates within shared-memory architectures. Multiple threads exist within a single process, sharing the same address space while maintaining individual instruction pointers and stacks. This facilitates low-latency data exchange but introduces hazards related to shared state. To maximize performance, *thread affinity* (or pinning) policies are often employed to bind threads to specific cores, thereby reducing context switching and cache invalidation.

Hazards and Synchronization

Parallel execution introduces non-deterministic hazards, most notably *race conditions*. These occur when multiple threads access shared data simultaneously without synchronization, leading to undefined behaviour that is difficult to reproduce and debug. Synchronization primitives such as locks, mutexes (mutual exclusions), and atomic operations are required to prevent these hazards. However, these constructs introduce performance penalties: coarse-grained locking effectively serializes execution, while fine-grained locking increases the complexity and the risk of deadlocks.

Application in CBM

The CBM experiment leverages these concepts extensively. The First-Level Event Selector (FLES, see Sec. 2.4.1) organizes raw detector data into timeslices, allowing independent collisions to be reconstructed simultaneously on the distinct compute nodes of the Green IT Cube. Furthermore, fine-grained parallelism is utilized by offloading specific reconstruction tasks, such as tracking, to GPUs.

3.1.3 OpenMP

OpenMP (Open Multi-Processing) [49, 50] is a high-level Application Programming Interface (API) for shared-memory parallel programming in C, C++, and Fortran. Rather than explicitly managing threads and synchronization primitives, programmers express parallelism declaratively using compiler directives (pragmas) and runtime library routines. This model simplifies the development of multithreaded applications while maintaining close-to-hardware performance.

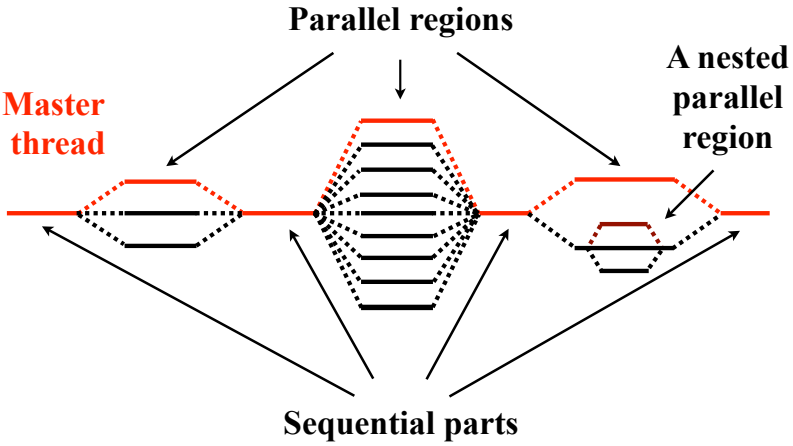


Figure 3.4: Illustration of the fork-join model of OpenMP. Multiple child threads (black) can be spawned from the master thread (red) in parallel regions of the code. Figure from [29]

The core construct is the `parallel` directive, which forks a team of threads to execute a region of code (Fig. 3.4). Work-sharing constructs, such as `for`, `sections`, and `task`, distribute iterations or logic blocks among threads. Synchronization is managed via `critical`, `atomic`, and `barrier` directives.

3.1.4 Performance Limits and Scaling Laws

Even with an optimal implementation, parallel performance is bounded by the serial portion of the code and communication overheads. Two primary laws describe these theoretical limits.

Amdahl's Law

Amdahl's Law addresses the theoretical speedup of a task with a *fixed problem size* as the resource count increases. It states that the performance improvement is limited by the sequential fraction of the code [51]. The speedup S is defined as:

$$S_{\text{Amdahl}} = \frac{1}{(1-p) + \frac{p}{N}}, \tag{3.1}$$

where N is the number of processors and p is the proportion of execution time that can be parallelized. As $N \to \infty$, the speedup converges to $1/(1-p)$, implying that even a small sequential fraction significantly inhibits performance (Fig. 3.5).

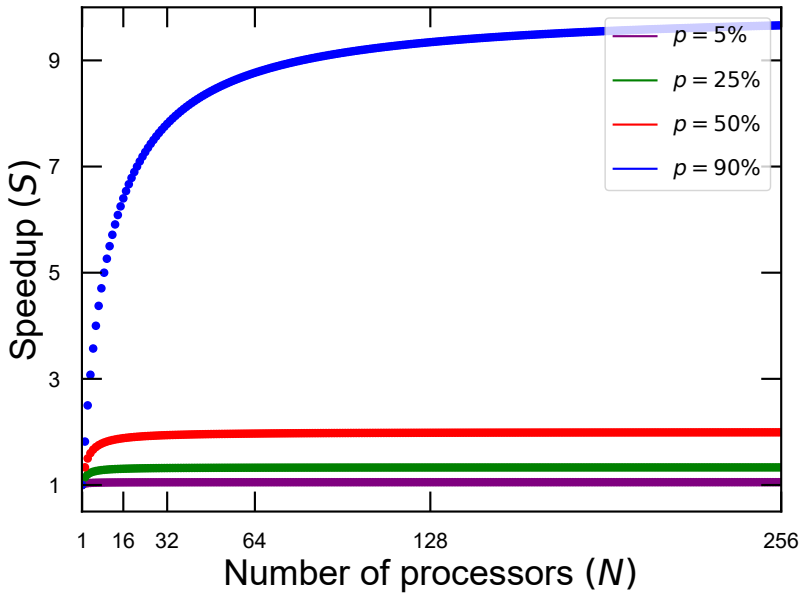


Figure 3.5: Theoretical speedup (S) according to Amdahl's Law as a function of processor count (N) for different parallel portions (p).

Gustafson's Law

In 1988, John L. Gustafson argued that in practice, problem sizes are rarely fixed; as computing power increases, researchers tend to solve larger problems in the same amount of time [52]. Gustafson's Law proposes that the parallel part of the workload scales with the number of processors, while the serial part remains constant. The scaled speedup is calculated as:

$$S_{\text{Gustafson}} = 1 + (N - 1) \cdot p, \tag{3.2}$$

where N is the number of processors and p is the parallelizable fraction of the workload. This perspective suggests that limitations imposed by sequential code can be effectively countered by increasing the magnitude of the computation as shown in Fig. 3.6.

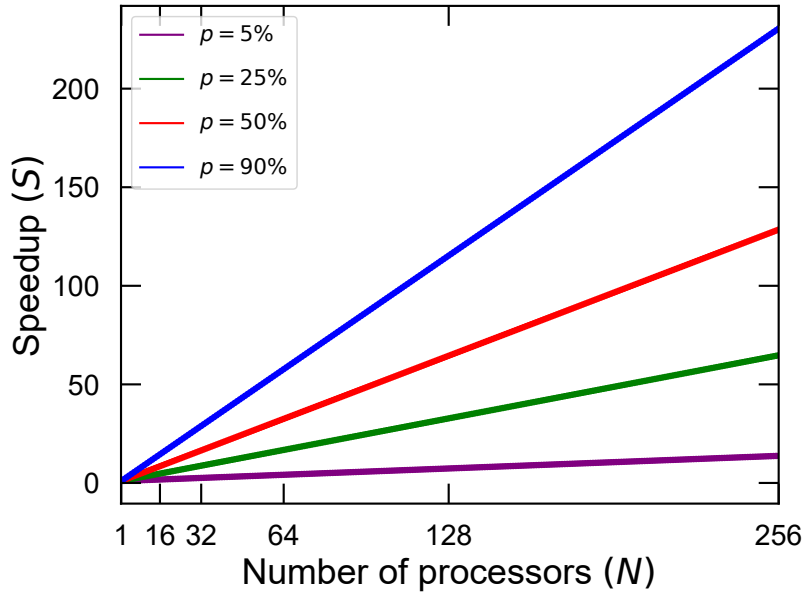


Figure 3.6: The speedup (S) as a function of number of processors (N) according to Gustafson increases linearly instead of the saturating.

3.2 Introduction to GPUs

Graphics Processing Units (GPUs) were originally developed in the 1970s and 80s as specialized hardware for handling graphics in arcade games. The transition

to programmable hardware in the early 2000s gave rise to *GPGPU* (General-Purpose computing on Graphics Processing Units). Researchers recognized that many scientific and engineering workloads shared characteristics with graphics rendering: specifically, the need to perform identical operations on large sets of independent data. The release of NVIDIA's CUDA (Compute Unified Device Architecture) [53] in 2006 provided the first unified programming model to exploit this hardware directly. Today, GPUs are ubiquitous in scientific computing, data analysis, and deep learning.

In the field of high-energy physics, GPUs are increasingly adopted for next-generation experiments. At CERN, the ALICE experiment [10] utilizes GPUs in the trigger system for its Time Projection Chamber [54]. Similarly, the CMS collaboration is implementing track reconstruction algorithms on GPUs [55]. The CBM experiment [56] is also actively integrating GPUs for event reconstruction and selection tasks.

3.2.1 GPU Architecture

Modern GPUs are highly parallel, many-core architectures designed to maximize computational throughput. Unlike Central Processing Units (CPUs), which employ large caches and complex branch prediction to minimize the *latency* of a single thread, GPUs are optimized for the execution of thousands of lightweight threads to maximize *throughput*. This architecture is ideal for workloads that can be expressed as a collection of independent, data-parallel operations.

Execution Model

The computational resources of a GPU are partitioned into *Streaming Multiprocessors* (SMs). Each SM is capable of supporting hundreds of active threads. Logically, programmers organize threads into *blocks*, which are further grouped into a *grid*. This grid defines the execution domain of a *kernel*: a function executed in parallel across the device.

At the hardware level, threads within a block are grouped into bundles called *warps* (sets of 32 threads in NVIDIA architectures) or *wavefronts* (64 threads in AMD architectures). Warps are the fundamental unit of execution in the Single Instruction, Multiple Thread (**SIMT**) model. In SIMT, all threads in a warp execute the same instruction simultaneously on different data elements.

Each block has access to a block-specific shared memory region, which is physically located on the SM that executes it. Threads within a block can communicate and synchronize through this shared memory, while threads in different blocks cannot directly share data. The mapping of blocks to SMs is handled by the GPU scheduler, which dynamically distributes workloads across available hardware resources. In this hierarchy, warps are hardware execution units, while blocks and grids are programmer-defined logical constructs that determine how computation is decomposed and executed on the GPU.

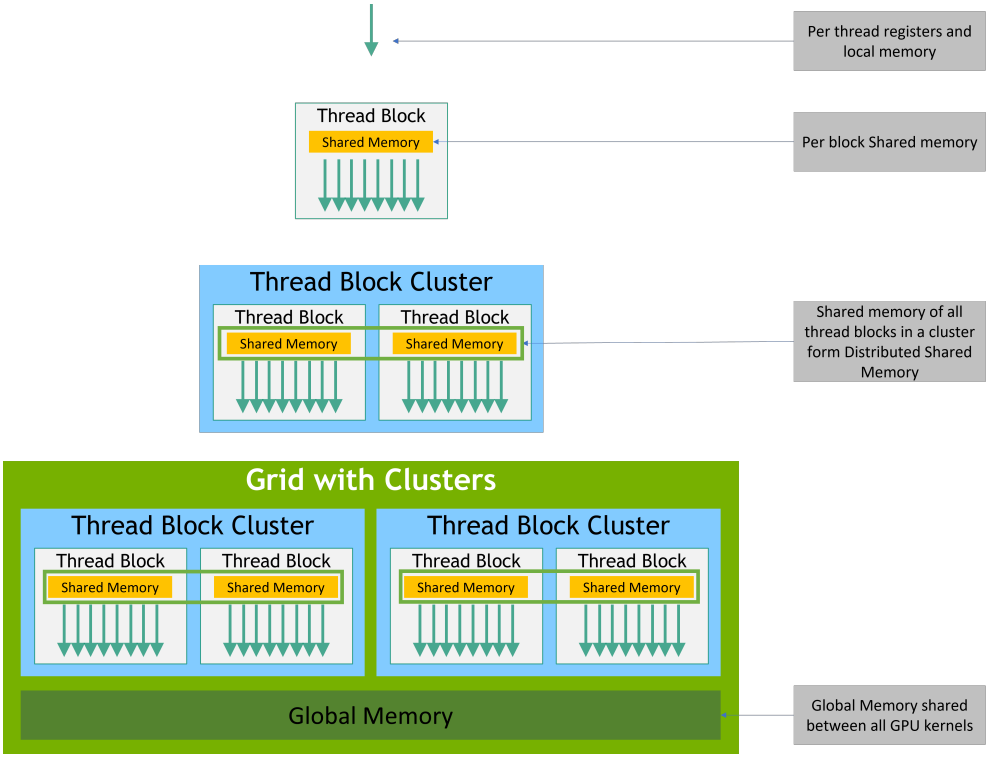


Figure 3.7: The CUDA hierarchy of threads, blocks, and grids. Figure adapted from [57].

To achieve peak performance, algorithms must minimize *thread divergence*. Divergence occurs when threads within a single warp follow different execution paths due to conditional statements (e.g., `if-else`). Because a warp must execute instructions in lockstep, divergent paths are serialized: the warp executes the ‘if‘ path while disabling threads that took the ‘else‘ path, and vice versa. This serialization significantly reduces effective parallelism.

3.2.2 Memory Hierarchy

- **Registers:** The fastest on-chip memory, private to each thread. Registers store local variables and operands. While the total register file per multiprocessor is large, the number of registers available per thread is limited and can constrain occupancy.
- **Shared Memory:** A low-latency, programmable on-chip memory shared by all threads in a block. It is extensively used for inter-thread communication and data reuse to reduce accesses to global memory.
- **L1/L2 Caches:** Modern GPUs include hardware-managed caches. The L1 cache is typically unified with shared memory, while the L2 cache services memory requests before accessing the global memory.
- **Constant Memory:** A specialized, read-only cache optimized for broadcasting a single value to all threads in a warp resulting in latency comparable to register access. It is ideal for storing parameters or lookup tables that remain fixed during kernel execution.
- **Global Memory:** The primary off-chip DRAM (e.g., GDDR6 or HBM). It is the largest memory space, accessible by all threads, but incurs high latency.

While GPUs offer very high arithmetic throughput their memory capacity is limited to around tens of gigabytes compared to hundreds of gigabytes available on CPUs. Moreover, GPU global memory is allocated and managed by the host before kernel launch and is accessible to all threads across the grid. Although threads can read and write global memory during execution, dynamic allocation or resizing within a kernel is generally not supported. Therefore, efficient memory management and minimizing data transfers between the host and device are essential considerations in GPU programming.

The efficiency of global memory access is a major determinant of GPU performance. GPUs read and write global memory in chunks (typically 32, 64 or 128 bytes). When threads in a warp access contiguous memory locations, these requests can be *coalesced* into a single transaction, maximizing bandwidth utilization. However, if threads access scattered or misaligned memory addresses, multiple memory accesses must be issued, resulting in performance degradation.

To ensure coalescing, data structures are often reorganized from an *Array-of-Structures (AoS)* layout (e.g., an array of 'Particle' objects) to a *Structure-of-Arrays (SoA)* layout (e.g., separate arrays for 'x', 'y', 'z'). The SoA layout ensures that neighbouring threads access neighbouring memory addresses, aligning perfectly with the SIMT execution model.

The hierarchical thread and memory structure of GPUs introduces distinct challenges in algorithm design. Achieving optimal performance requires a careful balance between computational workload, memory usage, and control flow regularity. In summary, efficient GPU algorithms must:

1. Maximize *arithmetic intensity* (the ratio of computation to memory access),
2. Exploit shared memory and registers to minimize global memory traffic,
3. Maintain regular control flow to avoid warp divergence, and
4. Utilize SoA data layouts to ensure coalesced memory access.

3.3 XPU: A Portable GPU Programming Library

The computational demands of online event reconstruction in CBM require the use of hardware accelerators, in particular GPUs. Developing GPU-accelerated software for a long-term experiment such as CBM poses several challenges. First, the experiment will begin data taking in 2028, and the specific accelerator hardware available at that time cannot be predetermined. The software stack must therefore remain portable across different GPU vendors and programming models to avoid architectural lock-in. Second, the reconstruction framework consists of a large, modern C++ codebase, which necessitates seamless integration of accelerator kernels with existing data structures, build systems, and development workflows. Finally, for testing, debugging, and continuous integration, the same code must remain executable on CPUs without modification.

To address these requirements, the `xpu` library was developed as a lightweight and portable C++ abstraction layer for heterogeneous computing [58, 59]. The library provides a unified interface for HIP, CUDA, and SYCL backends, while also supporting a pure CPU execution mode. This is achieved through a low-overhead intermediate abstraction layer that maps a common set of programming primitives onto the native backend APIs. Device code is compiled separately for

each backend, which allows developers to write hardware-agnostic kernels while retaining the performance benefits of vendor-specific compilers. The design of `xpu` emphasizes minimal intrusion into user code and modern C++ compatibility, enabling the integration of GPU kernels into the existing reconstruction framework with minimal modifications.

The ability to target multiple GPU architectures while preserving a single code path is essential for ensuring long-term maintainability and reproducibility of CBM reconstruction software. The `xpu` library therefore plays a central role in enabling portable high-performance computations for the experiment.

Using xpu requires the following steps:

1. Initialize runtime:

```
// default settings
xpu::initialize();
// for custom settings
xpu::settings settings;
settings.device = "cuda0";  // choose device
xpu::initalize(settings);
```

2. Create command queue

```
xpu::queue queue;  // use default device
xpu::queue queue(xpu::device{"cuda2"}); // specific device
```

3. Allocate memory in `xpu::buffer` objects

```
// Allocate memory for transfer between host and device
xpu::buffer<float> data(1024, xpu::buf_io);
// Allocate memory on host that is accessible from device
xpu::buffer<float> data_pinned(1024, xpu::buf_pinned);
// Allocate memory only on device
xpu::buffer<float> data_device(1024, xpu::buf_device);
```

4. Data transfer

```
// transfer data from host to device
queue.copy(data, xpu::h2d);
/*... run kernels ...*/
// transfer data from device back to host
queue.copy(data, xpu::d2h);
```

5. Kernel launch
   Kernels inherit from the `xpu::kernel` class and must overload the function call `()` operator with the `XPU_D` macro to mark it for device compilation. The block size, shared memory layout, constant memory access and the context are specified through the parent kernel class. The context is responsible for storing the thread position in grid, pointer to shared memory and access constant memory. The first argument must be a context object. An example of a kernel declaration and definition is shown.

```
#include <xpu/device.h> // Contains device side API

struct DeviceLib {}; // Dummy type for device library

struct VectorAdd : xpu::kernel<DeviceLib> {
   // set block size to 64 threads
   using block_size = xpu::block_size<64>;
   using context = xpu::kernel_context<>;

   XPU_D void operator()(
      context& ctx,
      xpu::buffer<const float> a,
      xpu::buffer<const float> b,
      xpu::buffer<float> c,
      size_t N);
};
```

Listing 3.1: Kernel declaration in vector_add.h

```
#include "vector_add.h"

XPU_IMAGE(DeviceLib);
XPU_EXPORT(VectorAdd);

XPU_D void VectorAdd::operator()(
   context& ctx,
   xpu::buffer<const float> a,
   xpu::buffer<const float> b,
```

```
10    xpu::buffer<float> c,
11    size_t N)
12 {
13    // Calculate global thread index
14    int idx = ctx.block_idx_x() * ctx.block_dim_x() + ctx.
      thread_idx_x();
15    if (idx >= N) return;
16
17    c[idx] = a[idx] + b[idx];
18 }
```

Listing 3.2: Kernel definition in vector_add.cpp

Finally the kernel can be launched with,

```
1 queue.launch<VectorAdd>(xpu::nblocks(1024), a, b);
```

The kernel definition must follow after call to `XPU_EXPORT` so that the kernel is registered with the runtime. `xpu` can then dynamically load the appropriate compiled device code for the backend in use. For CPU, the kernel is compiled with OpenMP as a regular C++ function.

A GPU-based reconstruction chain for the Silicon Tracking System (STS) was developed using `xpu` and integrated into the free-streaming data framework of the CBM experiment [60]. By employing parallel algorithms for hit reconstruction and cluster finding, this implementation achieved a speedup of approximately 122× compared to the previous CPU-based version. The GPU chain was successfully deployed during the May 2024 mCBM run, demonstrating both performance and operational robustness [61]. Following this validation, `xpu` is now being used to implement the Graph Neural Network (GNN)-based Track Finder on GPUs (see Chapter 6). The unified, C++-like abstraction significantly reduces the complexity of porting existing algorithms to heterogeneous architectures. In addition, the CA Track Finder is currently being adapted for GPU execution using `xpu` [62].

Summary

This chapter introduces the essential concepts of high performance computing. It begins with Flynn's taxonomy as a framework for understanding modern parallel

architectures, followed by a brief discussion of multiprocessing and multithreading, and the use of OpenMP for shared-memory parallelism and race condition control. The limits of parallel performance are outlined through Amdahl's and Gustafson's laws. The chapter then provides a short introduction to general-purpose GPU programming, describing the execution model, memory hierarchy, and key optimisation strategies such as reducing warp divergence and exploiting shared memory. Finally, the `xpu` library is presented as a backend-independent interface enabling portable GPU development, with short examples illustrating its practical use. The next chapter introduces the deep learning methods underpinning the algorithm developed later in the thesis.

Chapter 4

Deep Learning

Many real-world problems are too complex or ill-defined to be solved through explicit, rule-based programming. Such tasks often require the ability to learn abstract, high-level representations from data. Deep learning, a subfield of machine learning, addresses this challenge by learning hierarchical representations through multi-layer neural networks. These models can automatically discover intricate patterns in raw data without manual feature engineering. Over the past decade, deep learning has transformed numerous scientific and technological domains, including computer vision [63], natural language processing [64], and computational biology [65]. In this chapter, a brief introduction to deep learning is provided, with an emphasis on Multi-Layer Perceptrons (MLPs) and Graph Neural Networks (GNNs), which are core components of the track finding algorithm developed in this thesis.

4.1 Introduction

Artificial Neural Networks (ANNs) are computational systems inspired by the network of neurons found in biological brains. The concept originates from simplified mathematical models of biological neurons proposed by McCulloch and Pitts [66]. While the term "neural network" retains this biological metaphor, modern deep learning research is concerned with their mathematical and computational properties over biological connection.

At their core, ANNs are systems that learn to perform tasks by optimizing their parameters based on examples, rather than by following explicit, handcrafted rules. For instance, an ANN can learn to recognize images containing

cars by training on labelled datasets, a task that is impractical with rule-based code, given the variability in shapes, colours, perspectives and viewing conditions. ANNs therefore excel in domains where it is easier to provide examples of correct behaviour than to define the behaviour explicitly.

An ANN consists of interconnected nodes, or neurons, each of which holds a real-valued state. Neurons are connected by weights, which quantify the strength and direction of connection between them. During training, these weights are iteratively adjusted so that they collectively encode all the knowledge the model acquires. Each neuron applies an activation function to a weighted sum of its inputs, producing an output that becomes the input to subsequent layers. Neurons are typically organized into a series of layers with the number of neurons in a layer called the width, and the number of layers called the depth of the network. The specific configuration of these layers and their connectivity constitutes the network's architecture. A deep neural network (DNN) is an ANN with many hidden layers separating the input and output layers.

The expressive power of neural networks is formalized by a family of universal approximation theorems, which establish that neural networks can approximate any continuous function to arbitrary precision, given sufficient capacity. In 1989, Hornik et al. [67] demonstrated that multi-layer perceptrons (MLPs) with a single hidden layer of arbitrary width are universal approximators. More recent work has extended these results to MLPs with bounded width but arbitrary depth [68, 69], and even to networks with both bounded width and depth under specific conditions [70, 71]. However, these theorems are purely existential: they guarantee that suitable weights exist, but offer no method to find them.

The first implementation of an ANN, the *perceptron*, is credited to Frank Rosenblatt in 1958 [72]. The arrangement of these perceptrons into layers is called the Multi-Layer Perceptron (MLP). Stochastic gradient descent [73] as a training method was applied to these architectures as early as 1967 [74]. The development and application of backpropagation [75, 76] to MLPs provided an efficient means of computing gradients for these networks, enabling practical large-scale learning. In the 1990s, LeCun and colleagues extended this framework to Convolutional Neural Networks (CNNs), applying them successfully to handwritten digit recognition [77].

Despite promising early developments, the field faced periods of stagnation, often referred to as AI winters, primarily due to limited computational power and training data. The resurgence of deep learning in the mid-2000s was driven by

several converging factors: exponential growth in computational resources with the introduction of NVIDIA's CUDA [53] framework enabling general-purpose GPU programming, and the availability of high-quality, large-scale datasets. The watershed moment came in 2012, when a CNN-based model achieved a dramatic breakthrough in image classification on the ImageNet benchmark [63], marking the beginning of the modern deep learning revolution.

Today, DNNs are employed for a variety of tasks such as language translation [78], speech recognition [79], text [80] and image generation [81]. The training procedure of DNNs follows this general pipeline: define a parameterized model and a task-specific loss, use batches of data from training set and calculate loss, minimize loss by backpropagating gradients and updating parameters using an optimizer. The performance is evaluated on held-out data. Training proceeds over multiple passes, called *epochs*, through the dataset and is controlled by hyper-parameters such as learning rate, batch size, and regularization strength.

Depending on the availability of labelled data, machine learning methods are typically categorized as supervised, unsupervised or semi-supervised. In supervised learning, each training example is associated with a known ground-truth label or target value. The model learns to map input to output by minimizing a loss function that quantifies the discrepancy between the model's predictions and true labels. The performance of supervised models is typically evaluated using task-specific metrics such as accuracy, precision, recall, the area under the ROC curve (AUC), or the mean absolute error. For example, the track candidate classifier and the metric learning tasks described in Chapter 5 are both supervised learning problems.

In contrast, unsupervised learning methods operate on unlabelled data, seeking to uncover latent patterns or structure inherent in the data itself. Such methods are particularly valuable when obtaining labelled examples is costly or impractical. Some use cases are in clustering and dimensionality reduction. Semi-supervised approaches bridge the gap between these two extremes by combining a small amount of labelled data with a larger pool of unlabelled samples. Another interesting technique is reinforcement learning which involves learning to make sequential decisions through interaction with an environment and feedback in the form of rewards or penalties [82].

Some characteristics common to all ANNs will be discussed in the next sections.

4.2 Loss Function

The loss function is a fundamental component of any deep learning algorithm. It provides a quantitative measure of how well a model's predictions align with the ground-truth data. The loss function quantifies the difference between model output and ground truth for a given input into a scalar value called the *loss*. A model that produces accurate predictions will have a low loss value, while poor predictions result in a higher loss.

During training, the parameters of the model are optimized to minimize the loss function. Thus, the loss serves not only as an evaluation criterion but also as a feedback signal for optimization algorithms (see Section 4.3), guiding parameter updates through gradient-based methods. For most optimization schemes, it is advantageous for the loss function to be differentiable with respect to the model parameters, ensuring that gradients can be computed efficiently.

The loss function should be chosen appropriately for the task at hand since it has a significant influence on the stability of training and performance. Different losses encode different assumptions about the distribution of errors and emphasize different aspects of prediction quality. For instance, some losses penalize large deviations more strongly, making them sensitive to outliers, whereas others are designed to be more robust to noise. Some commonly used loss functions are discussed in the following.

4.2.1 Regression Loss Functions

Regression problems involve predicting a continuous output variable $y \in \mathbb{R}$ from one or more input features $\mathbf{x}$. The objective of a regression loss function is to quantify how far a predicted value $\hat{y}$ deviates from its true value y.

Mean Squared Error (MSE) The *Mean Squared Error* (MSE), also known as the *L2 loss*, is the most widely used objective for regression tasks. It is defined as the average of the squared differences between predicted and true values:

$$\mathcal{L}_{\text{MSE}}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2.$$

By squaring the residuals, MSE places a disproportionately high penalty on large errors, making it highly sensitive to outliers. While this can improve convergence

in cases where large deviations are rare, it may lead to instability when the data contains many extreme values. MSE considers only the magnitude of the error and not its direction.

Mean Absolute Error (MAE) The *Mean Absolute Error* (MAE), or *L1 loss*, measures the average absolute magnitude of the errors:

$$\mathcal{L}_{\text{MAE}}(y, \hat{y}) \;=\; \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|.$$

MAE is more robust to outliers than MSE because all deviations contribute linearly to the total loss. However, it penalizes small errors more heavily relative to MSE and is not differentiable at zero (i.e., when $y_i = \hat{y}_i$). Although this non-differentiability can be handled in practice, it can make optimization trickier.

Huber Loss The *Huber loss* [83] provides a balance between MSE and MAE by combining their advantages. It introduces a threshold parameter $\delta > 0$ that controls the transition between quadratic and linear behaviour:

$$\mathcal{L}_{\text{Huber}}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & \text{if } |y - \hat{y}| \leq \delta, \\ \delta\big(|y - \hat{y}| - \frac{1}{2}\delta\big), & \text{otherwise.} \end{cases}$$

For small residuals (within δ), the loss behaves like MSE, ensuring smooth differentiability and efficient gradient-based optimization. For larger errors, it transitions to an MAE-like regime, reducing the influence of outliers. The Huber loss is thus particularly effective in applications where robustness is desired without sacrificing differentiability.

4.2.2 Classification Loss Functions

Classification tasks involve predicting the class $y^\star \in \mathcal{Y} = \{1, \ldots, C\}$ to which an observation x belongs. The model outputs a probability distribution $q_\theta(c \mid x)$ over all possible classes, which is compared against the true distribution $p(c \mid x)$. Typically, $p(c \mid x)$ is represented as a *one-hot* vector, assigning unit probability to the true class $y^\star$ and zero to all others.

Cross-Entropy Loss *Cross-entropy* is the standard loss function for classification tasks. It measures the dissimilarity between the true and predicted distributions

$$H(p, q) \;=\; -\sum_{c=1}^{C} p(c) \log q(c).$$

When the ground-truth distribution p is one-hot, the expression simplifies to the *negative log-likelihood* of the true class:

$$H(p, q) \;=\; -\log q(y^\star).$$

Minimizing cross-entropy encourages the model to assign higher probability to the correct class while reducing the probabilities assigned to incorrect ones.

A neural network for multi-class classification typically outputs *logits* $z_c(x) \in \mathbb{R}$, which are converted to probabilities through the *softmax* function

$$q_\theta(c \mid x) \;=\; \frac{\exp(z_c(x))}{\sum_{k=1}^{C} \exp(z_k(x))}.$$

The *categorical cross-entropy* loss for a single example $(x, y^\star)$ is therefore

$$\mathcal{L}_{\text{cat}}(x, y^\star; \theta) \;=\; -\log q_\theta(y^\star \mid x).$$

Binary Cross-Entropy (BCE) The binary classification case ($C = 2$) is a special instance of the categorical cross-entropy loss. In this case, the model produces a single logit $z(x)$, which is transformed into a probability using the *sigmoid* function

$$\sigma(z) \;=\; \frac{1}{1 + \exp(-z)}.$$

The *binary cross-entropy* loss is then defined as,

$$\mathcal{L}_{\text{BCE}}(x, y; \theta) \;=\; -\big[\, y \log \sigma(z(x)) + (1 - y) \log(1 - \sigma(z(x))) \,\big],$$

where $y \in \{0, 1\}$ denotes the true label.

This formulation is algebraically equivalent to categorical cross-entropy with two classes but is computationally more efficient. For numerical stability, modern machine learning frameworks implement this as `BCEWithLogitsLoss`, which combines the sigmoid activation and the logarithmic operations into a single, stable computation.

Custom loss functions can be used for specific problems. In practice, a number of loss functions can be used for a particular task and there are no general rules to determine which function would be optimal.

4.3 Optimization via Gradient Descent

Optimization refers to the process of adjusting a model's parameters to minimize (or, in some cases, maximize) a given objective function. In the context of deep learning, this objective is to minimize the loss function $\mathcal{L}$, which quantifies the discrepancy between the model's predictions and the true target values. The parameters of the neural network, denoted collectively by $\boldsymbol{\theta}$, are iteratively updated to minimize $\mathcal{L}(\boldsymbol{\theta})$.

The gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ indicates the direction of the steepest ascent of the loss function. Consequently, to minimize $\mathcal{L}$, the parameters are updated in the opposite direction of the gradient according to

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}), \tag{4.1}$$

where α is the *learning rate*, a scalar hyperparameter controlling the magnitude of each update step. The algorithm proceeds iteratively, computing gradients and applying updates until convergence or until a predefined number of training iterations (epochs) is reached. Each epoch corresponds to a complete pass through the training dataset.

This basic approach, known as *gradient descent*, forms the foundation of nearly all optimization algorithms used in deep learning. However, computing the full gradient over large datasets can be computationally expensive, motivating the use of stochastic variants that provide more efficient, scalable alternatives.

4.3.1 Stochastic Gradient Descent

The *Stochastic Gradient Descent* (SGD) algorithm approximates the true gradient, computed over the entire dataset, using a randomly selected subset (or *mini-batch*) of training examples. Let

$$\mathcal{B} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m')}, y^{(m')})\}$$

denote a mini-batch containing m' samples drawn uniformly at random from the training set. The stochastic estimate of the gradient is then given by

$$\mathbf{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} \mathcal{L}(x^{(i)}, y^{(i)}, \boldsymbol{\theta}), \tag{4.2}$$

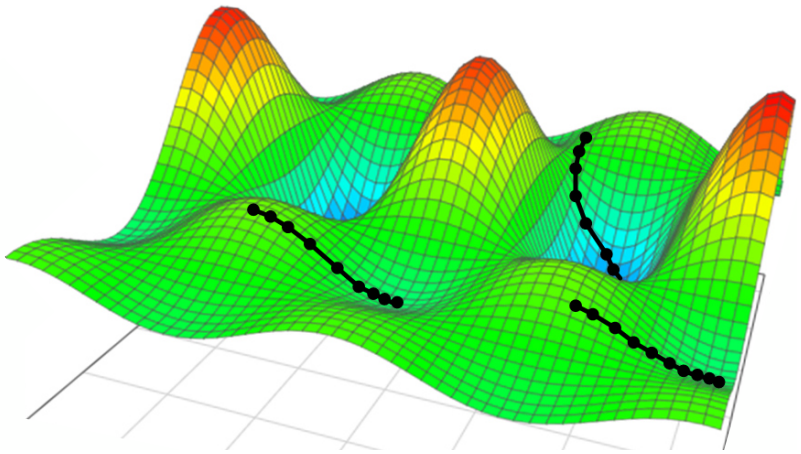


Figure 4.1: Illustration of gradient descent in a two-dimensional parameter space. Different trajectories can lead to different local minima giving qualitatively different results. Figure from [84].

where $\mathcal{L}(x^{(i)}, y^{(i)}, \boldsymbol{\theta})$ denotes the loss for a single training example. The parameters are subsequently updated according to

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \mathbf{g}. \tag{4.3}$$

The batch size m' is typically a small fixed number (e.g., 32, 64, or 128) that remains constant even as the total dataset size increases, ensuring that computational cost per iteration remains manageable. This stochastic approximation introduces noise into the gradient estimates, which can help escape shallow local minima and improve generalization. In practice, SGD tends to converge faster than full-batch gradient descent for large-scale learning problems.

A schematic illustration of gradient descent trajectories in parameter space is shown in Fig. 4.1. Different optimization paths can lead to distinct local minima, highlighting the importance of the optimizer's configuration and its hyperparameters in determining the final performance of the trained model.

4.3.2 Adaptive Optimization Methods: Adam

While SGD provides the conceptual foundation for most optimization algorithms in deep learning, numerous extensions have been proposed to improve convergence speed, stability, and robustness to hyperparameter choices. Among these, the *Adaptive Moment Estimation* (Adam) optimizer [85] is one of the most widely used methods due to its strong empirical performance and ease of use.

Adam combines two important ideas: *momentum* and *adaptive learning rates*. Momentum accumulates information from past gradients, enabling more consistent progress along directions of persistent descent while damping oscillations along high curvature directions. Adaptive learning rates adjust the effective step size for each parameter individually, based on the historical magnitude of its gradients.

At each iteration t, Adam maintains exponentially decaying moving averages of both the gradients and their squared values:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t, \tag{4.4}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2, \tag{4.5}$$

where $\mathbf{g}_t$ is the stochastic gradient at time step t, and $\beta_1, \beta_2 \in [0, 1)$ are hyperparameters controlling the decay rates (typically $\beta_1 = 0.9$, $\beta_2 = 0.999$). To correct for initialization bias, bias-corrected estimates are computed as:

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}, \tag{4.6}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}. \tag{4.7}$$

The parameter update rule is then given by

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon}, \tag{4.8}$$

where ε is a small constant (e.g., 10^{-8}) added to avoid division by zero.

Adam typically converges faster than vanilla SGD and requires less manual tuning of the learning rate. However, it has been observed that Adam can sometimes yield slightly poorer generalization performance compared to well-tuned SGD. To address this, variants such as *AdamW* [86] introduce explicit weight decay regularization.

4.3.3 Backpropagation

All optimization algorithms rely on the computation of gradients with respect to all model parameters. In deep neural networks, the loss function depends on the parameters through a nested composition of linear transformations and nonlinear activation functions across multiple layers. The efficient computation of all partial derivatives is achieved through the *backpropagation* algorithm.

Backpropagation is an application of the chain rule of calculus to computational graphs. During the forward pass, the input $\mathbf{x}$ propagates through the network, producing intermediate activations and ultimately the output $\mathbf{y}$ and a computational graph is built. The loss function $\mathcal{L}(\boldsymbol{\theta})$ is then computed from this output and the ground truth. Next, in the backward pass, the derivative of the loss with respect to each parameter is computed by traversing the computational graph in reverse order, from the output layer back to the input, accumulating gradients along the way.

Formally, consider a neural network composed of layers indexed by l, where the output of layer l is given by

$$\mathbf{h}^{(l)} = f^{(l)}\left(\mathbf{h}^{(l-1)}; \boldsymbol{\theta}^{(l)}\right), \quad \text{with } \mathbf{h}^{(0)} = \mathbf{x}.$$

The gradient of the loss with respect to the parameters of layer l is obtained via the chain rule as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(l)}} \frac{\partial \mathbf{h}^{(l)}}{\partial \boldsymbol{\theta}^{(l)}}. \tag{4.9}$$

The crucial recursive relationship that enables efficient computation is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(l)}} = \left(\frac{\partial \mathbf{h}^{(l+1)}}{\partial \mathbf{h}^{(l)}}\right)^{\top} \frac{\partial \mathcal{L}}{\partial \mathbf{h}^{(l+1)}}. \tag{4.10}$$

By reusing intermediate derivatives computed during this backward traversal, backpropagation computes all gradients in time proportional to the number of model parameters, rather than exponentially with the number of layers. This efficiency made the training of deep networks practically feasible and remains fundamental to modern deep learning.

Deep learning frameworks such as `PyTorch` [87] and `TensorFlow` [88] implement backpropagation automatically through a mechanism known as *automatic differentiation*. These systems dynamically construct computational graphs during the forward pass and apply the backpropagation algorithm in the backward pass to compute exact gradients.

4.3.4 Under- and Over- fitting

Minimizing the loss on the training set constitutes the core optimization problem in deep learning. However, achieving low training loss alone is insufficient, the true objective is to develop a model that performs well on previously unseen data.

This ability to transfer learned patterns from the training data to new inputs is known as *generalization*.

To assess generalization, model performance is evaluated on a *test set*, which is comprised of data not used for training. The training and test sets are assumed to be drawn independently and identically distributed (i.i.d) from the same underlying data-generating process. Under this assumption, a well-generalized model should achieve comparable loss values on both sets.

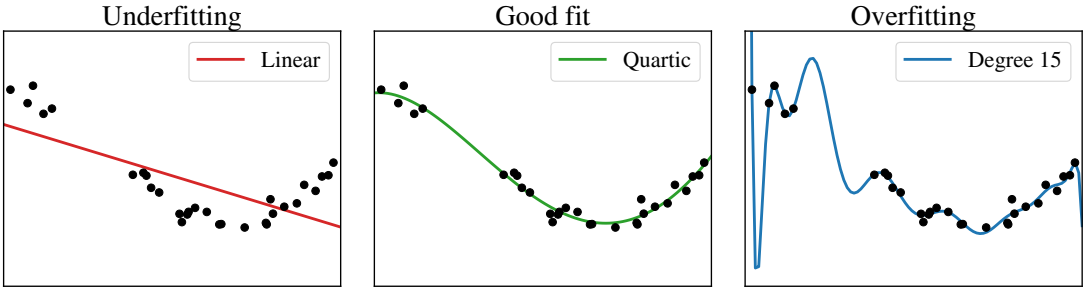


Figure 4.2: Illustration of underfitting and overfitting on a synthetic dataset with a cosine structure. Left: A linear model underfits, failing to capture curvature. Center: A quartic model captures the underlying pattern and generalizes well. Right: High-degree polynomial model has tried to fit the fluctuations, resulting in poor performance on unseen data.

Two common problems arise in this context: *underfitting* and *overfitting* (Fig. 4.2). Underfitting occurs when a model fails to capture the underlying structure of the data, resulting in high error on both training and test sets. This typically indicates that the model is too simple, either due to insufficient capacity, inadequate features, or insufficient training.

In contrast, overfitting arises when a model fits the training data too closely, including its noise and spurious fluctuations. While such a model may achieve very low training loss, its performance on unseen data is worse because it has different noise and fluctuations. Overfitting is often associated with excessive model capacity (too many parameters) relative to the true complexity of the task.

Choosing an appropriate model complexity is therefore essential to achieving good generalization. This trade-off between model bias and variance, illustrated in Fig. 4.3, lies at the heart of model selection and regularization strategies in machine learning.

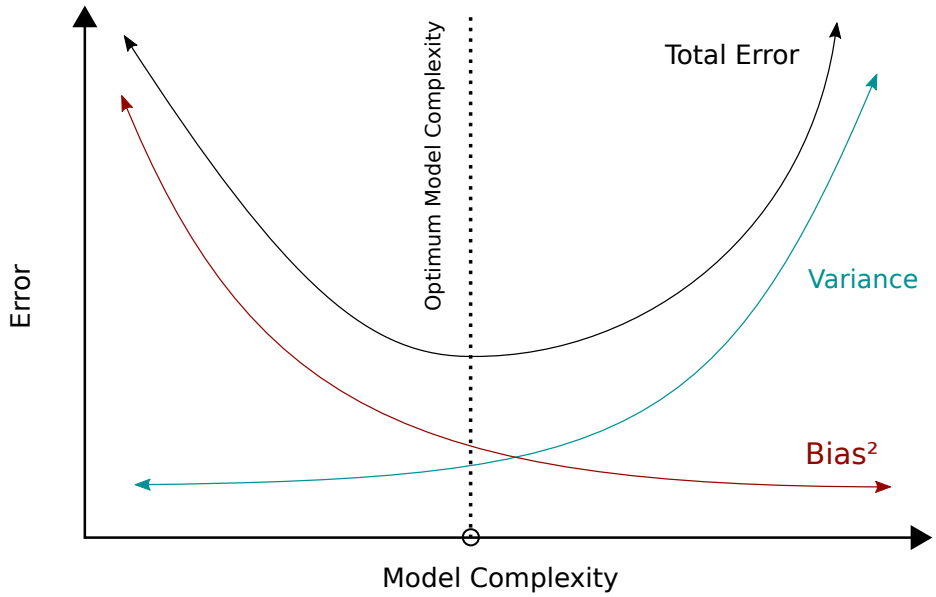


Figure 4.3: Relationship between bias, variance, and model complexity. Increasing complexity typically reduces bias but increases variance, illustrating the bias–variance trade-off. Figure from [89].

4.4 Multi-Layer Perceptrons (MLPs)

A *Multi-Layer Perceptron* (MLP) is the canonical form of a deep feedforward neural network and serves as the foundation for most modern neural network architectures. It consists of an input layer, one or more hidden layers, and an output layer. Each layer performs a set of affine transformations followed by nonlinear activations. A schematic representation of a feedforward network with three hidden layers and fully connected (dense) connections is shown in Fig. 4.4. These networks are also called *fully connected*, because every neuron in a layer is connected to every neuron in the adjacent layers.

4.4.1 Layer Operations

Each layer in an MLP performs two fundamental operations (Fig. 4.5). First, the input vector $\mathbf{x}$ undergoes a linear transformation defined by a weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}. \tag{4.11}$$

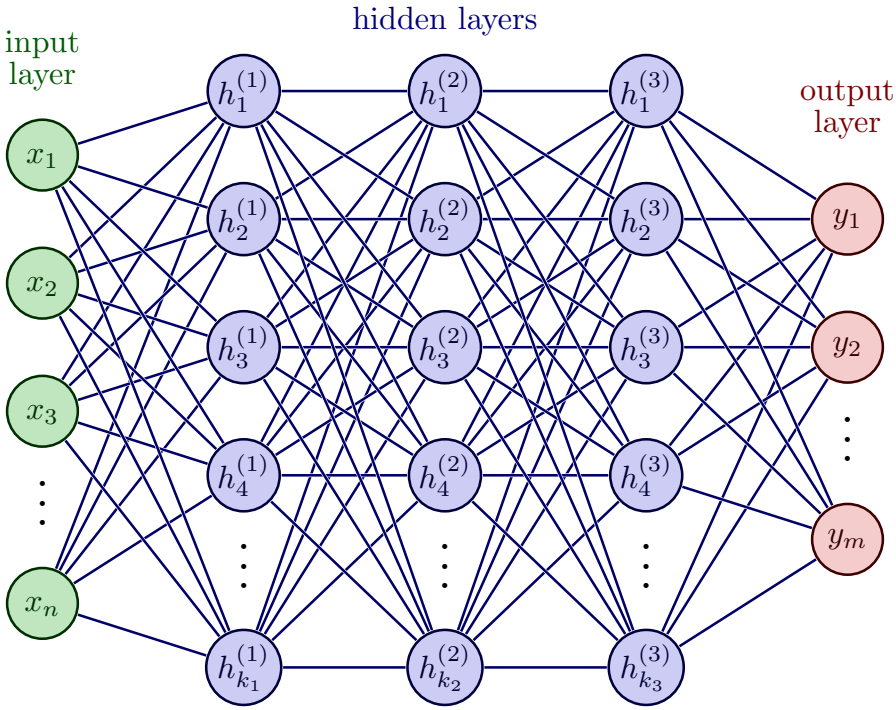


Figure 4.4: Illustration of a deep feedforward neural network, highlighting its input, hidden and output layers. Adapted from [90].

Second, a nonlinear activation function $\sigma(\cdot)$ is applied element-wise to produce the layer output:

$$\mathbf{h} = \sigma(\mathbf{z}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}). \tag{4.12}$$

Without the nonlinear activation, stacking multiple layers is equivalent to a single linear transformation, fundamentally limiting the expressive power of the network. Nonlinearity enables MLPs to approximate highly complex input-output mappings, including those that are not linearly separable.

4.4.2 Activation Functions

The choice of activation function has a substantial impact on both the representational capacity and the trainability of deep networks. In particular, activation functions mitigate issues such as vanishing gradients, enable sparse representations, or introduce smooth nonlinearities that improve optimization dynamics. Commonly used activation functions include:

- **Rectified Linear Unit (ReLU):** $\mathrm{ReLU}(x) = \max(0, x)$. ReLU is compu-

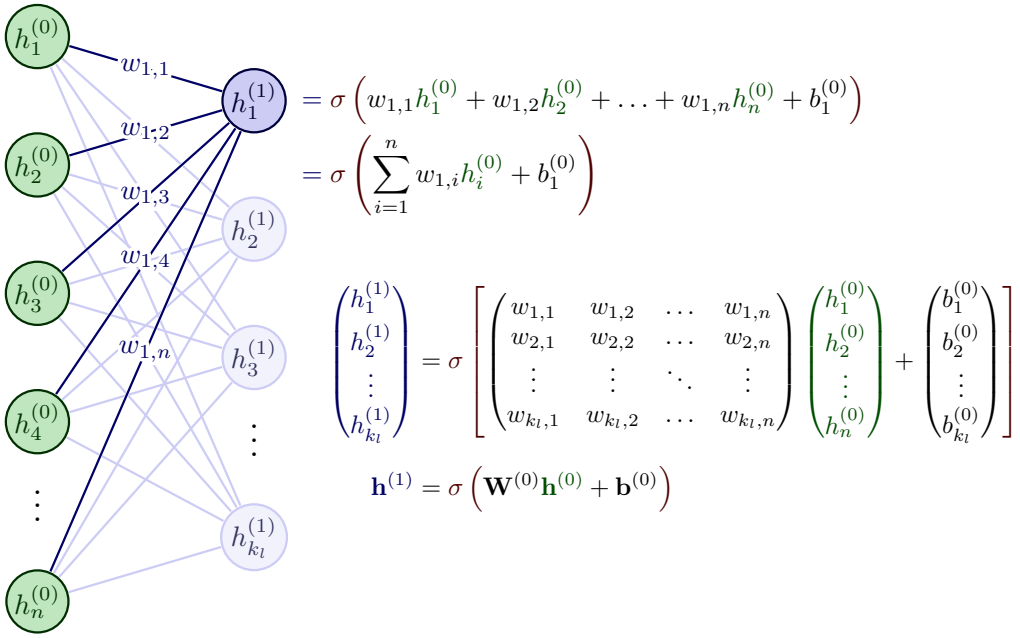


Figure 4.5: The affine and nonlinear activation function operations between a layer are shown. Weights are denoted as w, biases as b, and the activation function as σ. Adapted from [90].

tationally efficient and alleviates the vanishing gradient problem, making it the default choice in many deep architectures.

- **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$. Historically important, the sigmoid maps inputs to $[0, 1]$ but suffers from saturation for large $|x|$, which severely attenuates gradients.
- **Hyperbolic Tangent (tanh):** $\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$. The tanh function outputs values in $[-1, 1]$, centering activations and often improving convergence, though it still exhibits gradient saturation.
- **Swish:** $\text{swish}(x) = x\,\sigma(x)$. Swish provides smooth nonlinearity and retains negative inputs with small magnitude, leading to improved performance in very deep models at the cost of slightly higher computational overhead.

4.4.3 Layer Composition

A MLP can be viewed as a composition of multiple layer transformations. For a network with k layers producing a vector output, the overall mapping can be

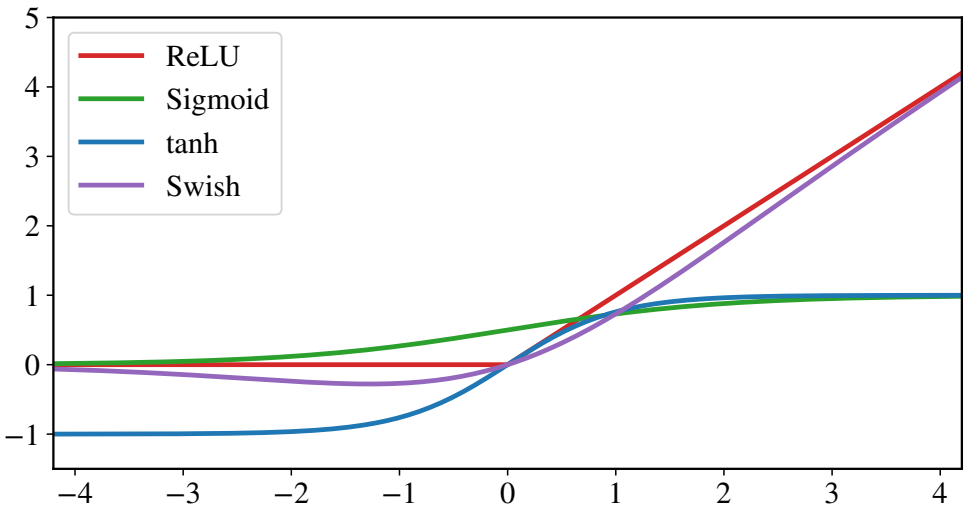


Figure 4.6: Some commonly used activation functions.

expressed as:

$$\mathbf{y} = f_{\mathrm{NN}}(\mathbf{x}) = f_k\big(f_{k-1}(\cdots f_2(f_1(\mathbf{x}))\big), \tag{4.13}$$

where each intermediate layer is defined as

$$f_l(\mathbf{z}) = \sigma_l(\mathbf{W}_l\mathbf{z} + \mathbf{b}_l), \tag{4.14}$$

with learnable weights $\mathbf{W}_l$, biases $\mathbf{b}_l$, and activation function σ_l. The final layer f_k typically uses a linear activation for regression tasks or a softmax activation for multi-class classification.

This hierarchical composition of affine and nonlinear transformations allows MLPs to approximate arbitrary continuous functions on compact domains, a result formalized by the universal approximation theorem. As such, MLPs serve as a flexible and general-purpose architecture, even though modern specialized architectures (e.g., CNNs, RNNs, and Transformers) are often better suited to particular tasks.

4.5 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are a class of models specifically designed to process data exhibiting a grid-like topology, most notably images. When inputs such as images are treated as a long, unstructured vector (as in a MLP), important spatial information encoded in the relative positions of pixels are lost.

CNNs preserve and exploit this local structure by operating on the image in its native two-dimensional form using local, shared filters.

Consider a single-channel image of spatial resolution $H \times W$. Flattening this array yields HW input units. A fully connected layer mapping this vector to M outputs would require $M \cdot HW$ learnable weights, which becomes quickly impractical as image size grows. For instance, a high-definition (HD) frame of size 1280×720 already contains 921,600 pixels. Convolutional layers address this by parameterising local interactions using $k_h \times k_w$ kernels (e.g., 3×3). These kernels are applied at every spatial location using the same set of parameters, giving rise to two key properties:

- **Locality:** Each output depends only on a small neighbourhood of the input. The extent of local information incorporated is controlled by the kernel dimensions and, indirectly, by network depth.
- **Parameter sharing:** The same kernel weights are reused across all spatial positions. This dramatically reduces the number of learnable parameters, mitigating risk of overfitting, and enabling construction of deeper networks. It also induces the learning of (approximately) shift-equivariant features.

4.5.1 The Discrete Convolution Operation

The continuous convolution of functions f and g is defined as

$$(f * g)(t) \coloneqq \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau.$$

Note that this operation is commutative. In the discrete, two-dimensional case relevant for image processing, the convolution of an input image I with a kernel K can be written as

$$S(i,j) \;=\; (K * I)(i,j) \;=\; \sum_{m}\sum_{n} I(i+m,\, j+n)\, K(m,n), \tag{4.15}$$

where the sums range over the dimensions of the kernel.

For a multi-channel input with C_{in} channels (e.g., $C_{\text{in}} = 3$ for RGB images), each filter is a tensor of shape $k_h \times k_w \times C_{\text{in}}$. The output of the c-th filter is the sum of channelwise convolutions:

$$S_c(i,j) = \sum_{p=1}^{C_{\text{in}}} (K_{c,p} * I_p)(i,j) + b_c, \tag{4.16}$$

where $K_{c,p}$ denotes the spatial kernel applied to input channel p, I_p is the p-th input channel, and b_c is a bias term. With C_{out} such filters, the layer produces an output tensor of size $H' \times W' \times C_{\text{out}}$. The spatial dimensions H', W' depend on the convolution *stride* and *padding*. A stride greater than one subsamples the feature map, while padding controls boundary behaviour and enables preservation of spatial resolution.

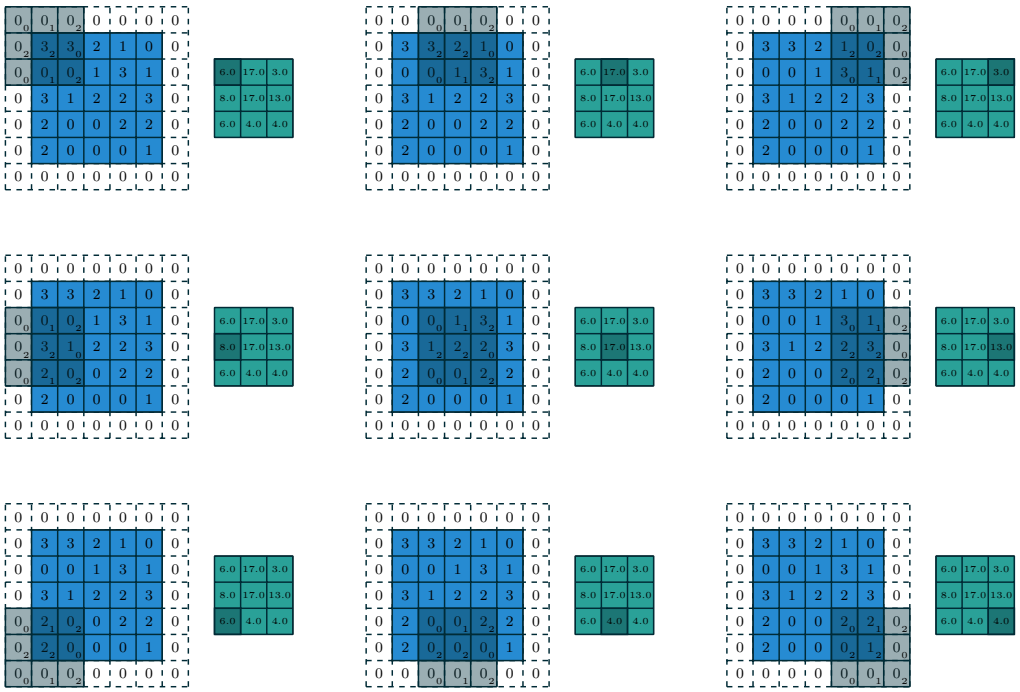


Figure 4.7: The result of applying a 3 ×3 kernel on a 5×5 image with padding of size 1 and a stride of size 2 in both directions. Figure from [91].

4.5.2 Translation Equivariance and Invariance

A central property of convolution is *translation equivariance*: translating the input results in an equivalently translated output (up to boundary effects). If T_Δ denotes translation by an offset Δ, then for a convolutional operator $\mathcal{K}$,

$$\mathcal{K}[T_\Delta I] = T_\Delta\big[\mathcal{K}[I]\big].$$

This property reflects the intuition that local patterns such as edges or corners carry semantic meaning independent of their absolute position in the image.

Many architectures combine convolutional layers with pooling or global aggregation to obtain a form of *translation invariance*, where the final prediction (e.g., object class) does not change under small image translations. For example, local max-pooling reduces sensitivity to small translations by selecting the local maximum, while a global average pooling collapses spatial dimensions to produce a translation-invariant descriptor for classification.

4.5.3 Hierarchical Feature Learning and Receptive Fields

By stacking convolutional layers, CNNs learn hierarchical feature representations. Early layers typically detect simple, low-level structures (edges, corners, colour contrasts), whereas deeper layers compose these primitives into progressively more abstract and semantically meaningful concepts (object parts, textures, full objects). The *receptive field*, the region of the input that influences a given activation, grows with network depth (Fig. 4.8), enabling deeper layers to integrate global image context.

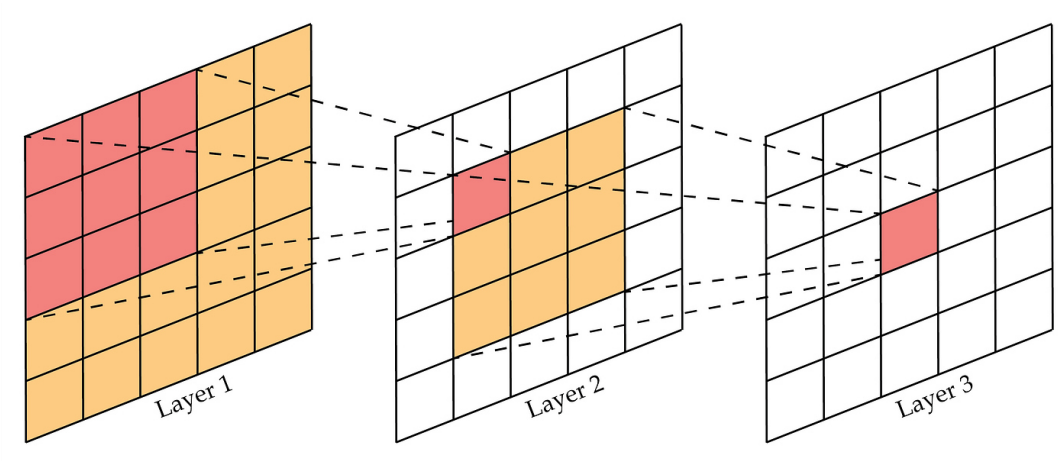


Figure 4.8: Illustration of how the receptive field expands in a CNN when a 3×3 kernel is used. Figure from [92]

4.6 Graph Neural Networks (GNNs)

Similarly to how CNNs exploit the underlying grid structure of images, many real-world tasks involve data that are more naturally represented as graphs. Examples include social networks, where individuals are linked through social interactions,

protein–protein interaction networks in molecular biology, and web graphs in which pages are connected by hyperlinks. In such settings, the connectivity structure carries crucial information about the relationships, dependencies, and dynamics of the underlying system. Neural architectures must therefore be able to reason not only over node attributes, but also over the relational structure encoded by edges [93, 94].

A graph is formally defined as a pair $G = (V, E)$, where V is a finite set of vertices (or nodes), and E is the set of edges connecting pair of nodes. A convenient mathematical representation of a graph is the *adjacency matrix* $\mathbf{A}$, defined as:

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if there is a link from node } i \text{ to node } j, \\ 0, & \text{otherwise.} \end{cases} \tag{4.17}$$

For undirected graphs, $\mathbf{A}$ is symmetric. When edges carry weights, the graph is said to be *weighted*. Furthermore, each node $i \in V$ may also be associated with a feature vector $\mathbf{x}_i \in \mathbb{R}^d$, while each edge (i, j) can carry its own feature vector $\mathbf{e}_{ij}$.

Unlike images, where each pixel has a fixed number of neighbours arranged on a regular grid, graphs allow more general relationships. Nodes can have different degrees, their connections need not follow a regular pattern, and there is no inherent ordering of nodes. Applying standard MLPs or CNNs directly to graphs is therefore inappropriate, since any permutation of node indices should leave the graph's meaning, and hence the model's output, unchanged. This requirement is known as permutation invariance, and it motivates the development of neural architectures tailored specifically to graph-structured data.

Graph Neural Networks (GNNs) are architectures specialized to deal with graphical data [95–98]. They retain the core properties that make CNNs effective: locality, weight sharing, and hierarchical feature extraction. GNNs can be thought of as a generalization of CNNs with the convolution operation replaced by neighbourhood aggregation, whereby each node updates its representation using information from its immediate neighbours rather than spatially adjacent pixels [99, 100]. Stacking GNN layers increases the receptive field, enabling the network to capture increasingly global structure.

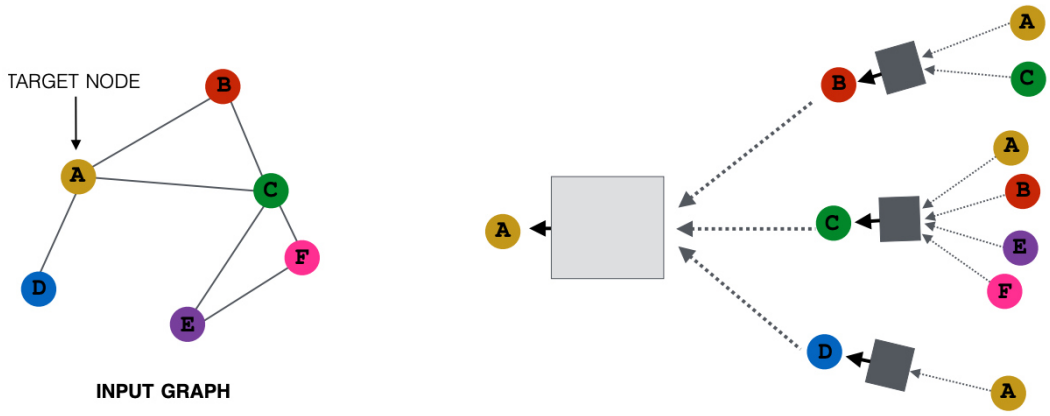


Figure 4.9: Illustration of the process of message passing. Every node defines its own computation graph based on its neighborhood. Left: The input graph and the target node based on which the series of computations is defined. Right: The message passing steps for two hops away from the target node. Gray rectangles represent neural networks. Figure from [101].

4.6.1 Message Passing

The most general framework for defining GNNs is the Message Passing Neural Network (MPNN) [102]. In this framework, the graph processing is divided into distinct phases occurring over K layers (or iterations). For a given layer k, the update of a node u involves three distinct steps:

1. **Message Calculation:** A message $\mathbf{m}_{uv}^{(k)}$ is computed for every edge (u, v) connecting node u to its neighbour v. This function often depends on the hidden states of both nodes and the edge features:

$$\mathbf{m}_{uv}^{(k)} = \psi^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}, \mathbf{e}_{uv}^{(k-1)}\right). \tag{4.18}$$

2. **Aggregation:** The messages from all neighbors $\mathcal{N}(u)$ are aggregated into a single context vector $\mathbf{m}_u^{(k)}$ using a permutation-invariant operator $\bigoplus$ (such as sum, mean, or max):

$$\mathbf{m}_u^{(k)} = \bigoplus_{v \in \mathcal{N}(u)} \mathbf{m}_{uv}^{(k)}. \tag{4.19}$$

3. **Update:** The node's current hidden state is updated using the aggregated message and its previous state:

$$\mathbf{h}_u^{(k)} = \phi^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathbf{m}_u^{(k)}\right). \tag{4.20}$$

Here, $\mathbf{h}_u^{(0)}$ is initialized as the input features $\mathbf{x}_u$. The functions ψ and ϕ are typically learned differentiable functions, such as Multi-Layer Perceptrons (MLPs). This modular framework allows for significant flexibility; by choosing different functions for the message and aggregation steps, one can recover various popular GNN architectures.

4.6.2 Permutation Symmetry

A fundamental requirement for learning on graphs is handling the lack of a node ordering. In a standard image, swapping pixel $(0, 0)$ with $(0, 1)$ changes the image content. In a graph, however, swapping the indices of two nodes does not change the underlying topology of the graph. Neural networks applied to graphs must satisfy one of two properties depending on the task:

- **Permutation Invariance:** The output of the model remains unchanged regardless of the node ordering: $f(\mathbf{PAP}^\top, \mathbf{PX}) = f(\mathbf{A}, \mathbf{X})$, where $\mathbf{P}$ is some permutation matrix. This is essential for graph-level classification tasks.
- **Permutation Equivariance:** The output of the model permutes in the same way as the input: $f(\mathbf{PAP}^\top, \mathbf{PX}) = \mathbf{P}f(\mathbf{A}, \mathbf{X})$.

For track finding equivariance must be satisfied. It ensures that the model's prediction for a specific detector hit depends only on its geometric relationships and features, not on the arbitrary order in which the hit is provided.

4.6.3 Types of Prediction

Depending on the information required, GNNs can be trained for tasks at three different levels:

1. **Graph-level:** Predicting a property for the whole graph
2. **Node-level:** Predicting the class or value of individual nodes
3. **Edge-level:** Predicting the existence or label of an edge.

In the context of particle tracking, the problem is most naturally formulated as an edge-level classification task. The GNN infers the probability that an edge connecting two detector hits represents a true segment of a particle trajectory (a "true" link) versus a spurious connection (a "false" link).

4.7 ANN4FLES: Neural Networks for First Level Event Selection

Artificial Neural Networks for First Level Event Selection (`ANN4FLES`) [103] is a lightweight `C++` package that simplifies the use of neural network-based triggers and algorithms in the CBM software environment. It was originally developed for the detection of Quark Gluon Plasma (QGP) signatures in heavy-ion collisions in the CBM experiment [104]. It's design philosophy prioritizes computational efficiency, ease of integration, and maintainability within the existing CBM software stack. Unlike general-purpose deep learning frameworks such as TensorFlow or PyTorch, `ANN4FLES` is not intended to act as a general-purpose deep learning library. Instead, its goal is to offer a minimal-overhead solution tailored to the specific requirements of high-performance event reconstruction and selection in the CBM context. To this end, the framework uses `OpenMP` (see Sec. 3.1.3) for multithreading and SIMD vectorization to achieve high throughput on modern CPU architectures while avoiding the overhead associated with larger frameworks.

The correctness and performance of `ANN4FLES` have been validated through comparative benchmarks with PyTorch on standard reference problems such as handwritten digit classification on the MNIST dataset [105]. Using identical network architectures and hyperparameters, both implementations achieved nearly identical classification performance, confirming the algorithmic consistency of `ANN4FLES` with established deep learning tools. Moreover, for small- and medium-size neural networks, the regime relevant for real-time processing, `ANN4FLES` demonstrated inference times comparable to those of optimized PyTorch models, underscoring its suitability for the CBM online processing environment.

`ANN4FLES` provides the underlying infrastructure for all neural network components developed in this thesis, including the MLP-based metric learning and track-candidate classification models, as well as the GNN-based edge classification algorithm. The framework is fully integrated into the CBMRoot codebase, allowing users to access its functionality through standard header inclusion. This

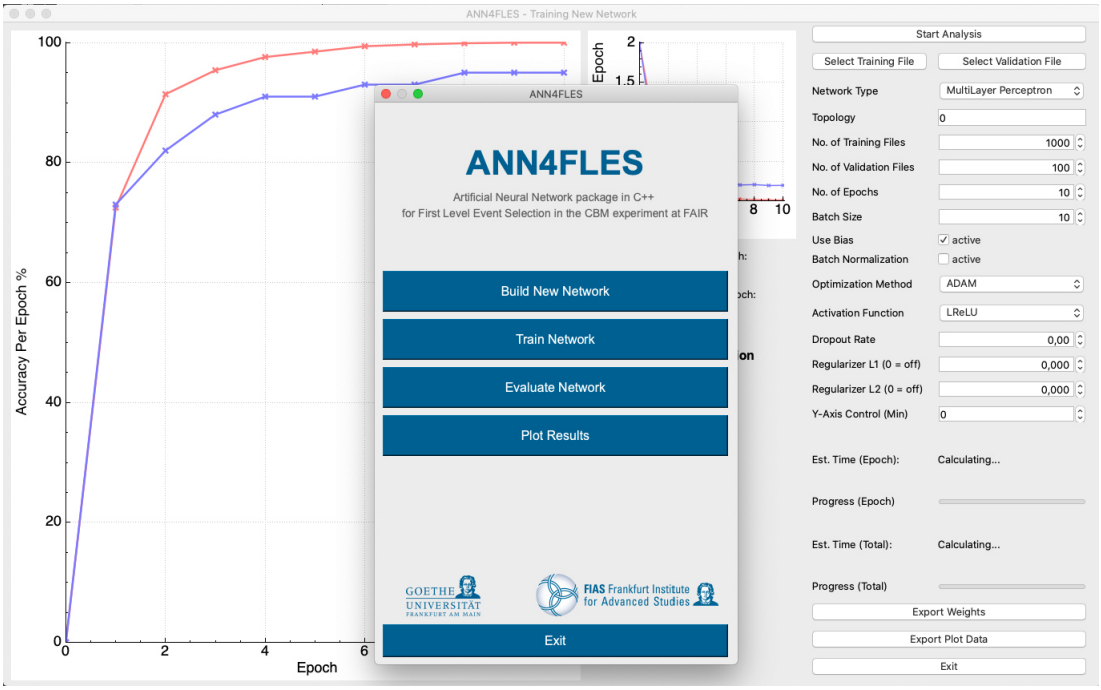


Figure 4.10: Graphical interface of the `ANN4FLES` package.

tight integration ensures seamless deployment of neural models within existing CBM reconstruction and triggering workflows, without requiring additional runtime dependencies or external libraries.

GPU acceleration has also been explored within this framework. The metric learning network has been successfully ported to GPU architectures using the `xpu` library [58], and ongoing work aims to extend GPU support to the remaining components of the tracking pipeline. This effort is expected to yield further performance improvements for the full reconstruction chain.

A comprehensive technical description of the `ANN4FLES` framework can be found in [22].

Summary

This chapter introduces the deep learning concepts required for the development of the GNN-based track finder. It begins with a brief overview of the field, from its historical roots to its wide deployment in modern pattern-recognition tasks. The chapter then explains how neural networks are trained: model parameters are optimized by minimizing a loss function that quantifies the discrepancy be-

tween predicted and target values. This optimization is performed using stochastic gradient descent, where gradients of the loss with respect to all parameters are computed efficiently using backpropagation. The role of batches, learning rates, and iterative updates is highlighted to clarify how networks progressively improve their predictive accuracy. The chapter next outlines the architectures most relevant to this thesis: Multi-Layer Perceptrons (MLP) and Graph Neural Networks (GNN), with an emphasis on message-passing in GNNs. Finally, the ANN4FLES package is introduced as the package used within CBMRoot to deploy neural networks in the GNN-based track finding algorithm.

This chapter marks the end of the introductory part of this thesis where all relevant background for understanding the main topic of this thesis was discussed. The next chapter will describe in detail the GNN-based algorithm that has been developed for track finding in the CBM experiment.

Chapter 5

GNN-based Track Finding Algorithm

The multitude of charged particles produced in high-energy collisions collectively carry information which can be used to understand the fundamental laws governing particle interactions. This information is encoded in the position and momentum of the produced particles. To reconstruct these kinematics, an array of detectors, called tracking stations, are set up to observe the particle's trajectories. These tracking stations record discrete position measurements, called *hits*, along the trajectory of the particle. Track reconstruction is the process of recovering particle kinematics from these hits. The output of track reconstruction feeds directly into all physics analyses. Therefore, developing efficient and accurate track reconstruction algorithms is a central task in every collider experiment.

5.1 Track Finding

Track reconstruction is typically divided into two subtasks, track finding and track fitting. Track finding is a classification problem in which hits originating from the same particle must be grouped into a track candidate. Track fitting is the problem of estimating the parameters describing the state of a particle using its hits. These estimated, also called *fitted*, parameters are used in physics analyses and thus must be determined with high precision and accuracy. Achieving this requires accounting for detector inefficiencies, energy losses, inhomogeneities in the magnetic field, multiple scattering in detector material and measurement precision.

Modern collider experiments operate at increasingly high energies and interaction rates. Higher energies generate denser hit environments, which increases

the combinatorial complexity of track finding. Also, as interaction rates rise, stricter time performance requirements are placed on track finding algorithms since they are the most computationally demanding part of real-time event reconstruction. The recent successes of deep learning across many domains has motivated investigations of its application to track finding. In this chapter, a Graph Neural Network (GNN)-based track finding algorithm developed for the CBM experiment is presented.

Although collider experiments differ significantly in their physics goals and detector designs, many tracking concepts are shared. Most contemporary tracking algorithms are variants of the Combinatorial Kalman Filter (CKF) [106]. The Kalman Filter (KF) [107] provides an optimal iterative estimation framework for linear dynamical systems by incorporating measurement uncertainties and system evolution. CKF algorithms begin with a track seed, a single hit or a small group of hits, and an initial estimate of the track parameters, obtained either from the seed itself or from external constraints such as the primary vertex in fixed-target experiments. The seed is extrapolated to the next tracking station, where a region of interest is defined to search for compatible hits. If multiple hits fall within this region, several viable track candidates are spawned. These candidates are propagated through subsequent detector stations, adding hits iteratively. After all seeds are processed, a large set of track candidates remains, often with shared hits. Since a hit can be assigned only to one track, a track selection procedure is required to determine the optimal set of tracks.

In the CBM experiment, a Cellular Automaton (CA)-based track finding algorithm has been developed. A CKF is used to find triplets of hits which are used to construct track candidates. Track selection is performed using a CA-inspired method. CA has been applied to many high energy physics experiments with great success [108–111]. The use of the KF naturally incorporate multiple scattering, magnetic-field inhomogeneities, and energy losses in detector material, but they also introduce significant computational overhead. Furthermore their parallelization can present problems with numerical stability [112–114].

The use of artificial neural networks for track finding dates back to early work by Denby [115]. It was followed by elastic neural networks which were developed to address efficiency loss at high hit densities [116]. Despite these efforts, CKF-based methods have remained dominant due to their robustness and performance. However, the combination of increasing hit densities in modern detectors, advances in GPU hardware, and the remarkable progress of deep neural

networks has sparked interest in deep learning-based approaches to track finding.

5.1.1 Related Work

This section provides a brief overview of recent deep learning-based approaches to track finding. Early investigations explored Convolutional Neural Networks (Sec. 4.5) and Recurrent Neural Networks, but Graph Neural Networks (Sec. 4.6) ultimately emerged as the most promising class of models and were investigated in detail by the Exa.TrkX project.

CNNs

Convolutional Neural Networks (CNNs) can operate directly on the raw data provided by pixel detectors. In Ref. [117], a CNN was trained to generate track seeds without relying on traditional cluster-splitting algorithms. Individual tracking layers were treated analogously to colour channels (RGB) in an image, enabling the network to process the full set of pixel hits in a single forward pass. Developed for the dense central region of the CMS tracker [118], the method demonstrated substantial improvements over standard seeding techniques, achieving a 60% reduction in the fake-seed rate and an 85% reduction in computation time. Although this work demonstrated CNNs as viable tools for localized pattern recognition in tracking detectors, it needs to investigated how they deal with non-pixel detectors and general detector geometries.

RNNs

Track finding can also be framed as a hit position prediction problem in which the model extrapolates the next hit along a particle's trajectory. This formulation naturally lends itself to the application of Recurrent Neural Networks (RNNs). RNNs can model the evolution of track states and predict hit positions in a manner loosely analogous to the recursive update steps of a Kalman Filter [119]. In Ref. [120], Long Short-Term Memory (LSTM) networks [121] and variants thereof were applied to toy datasets containing straight-line tracks. Pixel arrays were processed by an LSTM module, followed by a MLP that produced the predicted next hit-pixel. These models struggled at high hit multiplicities and suffer from limited scalability due to the recurrent structure.

Transformers

Transformers have also recently been explored for global, event-level track reconstruction. Ref. [122] introduced *EncCla*, a transformer-encoder architecture that assigns each hit to a class corresponding to a bin in discretized track parameter space. The model takes as input the sequence of all hit coordinates in a single event and generates class labels for each hit. The class labels map to a specific bin in the discretized track parameter space. The original hit coordinates are embedded into a high-dimensional space on which multiple encoder blocks are applied. A final classification layer yields a probability distribution over track-parameter bins, from which the most likely class is selected. This model was tested on subsets of the TrackML challenge dataset [123]. Since the model requires binning of the track parameter space it might be challenging for it to scale to high track density environments. It obtained promising results for inference time but more work is needed in pre- and post-processing to fully compare results with KF and GNN approaches.

5.1.2 Graph Neural Networks

Graph Neural Networks (GNNs) currently represent the most promising deep learning architecture for track finding. A graph is a set of nodes and a set of pairwise connections between these nodes called edges. If detector hits are considered as graph nodes and hit to hit connections on neighbouring detector stations as edges, a graph is obtained. In this graph formulation, track finding becomes the problem of reducing the set of edges down to those that connect hits from the same particle. In the language of GNNs, this is called edge classification. Ideally, the surviving edges form disjoint chains representing individual tracks which can be found with a simple connected components algorithm. However if overlapping tracks remain, an additional ranking scheme must be developed for choosing the best branch.

A lot of effort in recent years has focused on the use of GNNs in track finding with the Exa.TrkX project [124] playing a major part.

The Exa.TrkX project

The Exa.TrkX project applied geometric learning techniques [125], particularly metric learning and GNN-based edge classification to particle tracking. The

Exa.TrkX pipeline consists of

1. Hit feature embedding into high dimensional latent space,
2. Graph construction using fixed-radius or nearest-neighbour algorithms in latent space,
3. Lightweight edge filtering with MLPs to prune easily rejected edges,
4. Edge filtering using GNN,
5. Connected components on remaining edges to assemble edges into tracks.

The Exa.TrkX pipeline has shown competitive tracking efficiency and purity on the TrackML [126] dataset. Importantly, it has demonstrated near-linear inference scaling with number of hits in an event. The Exa.TrkX provided the starting point for the GNN-based algorithm developed in this thesis.

ETX4VELO

The ETX4VELO project adapted the Exa.TrkX pipeline for track reconstruction in the Vertex Locator (VELO) sub-detector of the LHCb experiment [127]. While ETX4VELO achieved marginally higher reconstruction efficiency than the traditional Kalman Filter (KF)-based track finder, it faced significant computational challenges. When implemented within the LHCb GPU pipeline, the GNN-based approach demonstrated a throughput approximately 1000 times lower than the traditional algorithm.

The algorithm developed in this work addresses these efficiency/throughput trade-offs. It differentiates itself from ETX4VELO by incorporating KF-based filtering and traditional selection cuts into the pipeline, reducing the reliance on neural networks where geometric heuristics suffice.

5.2 Overview of algorithm

This work builds upon the Exa.TrkX pipeline, which demonstrated that GNN-based methods can efficiently identify track segments in high-density environments. However, a direct application of the Exa.TrkX architecture to the CBM experiment revealed specific limitations. Notably, training a single, monolithic GNN to suppress spurious edges sufficiently for connected components clustering

proved infeasible. The presence of multiple overlapping track candidates, particularly in the low-momentum regime, necessitates additional candidate filtering and rigorous track selection.

To address this, the existing CBM Cellular Automaton (CA) track finder [128, 129] was analyzed, revealing three critical components:

1. **Iterative track finding:** Four successive iterations target tracks with distinct trajectory characteristics. Iteration-specific selection criteria are essential to achieve high purity and efficiency.

2. **Triplet fitting with KF:** Three spatial measurements are sufficient to uniquely constrain the track parameters in the CBM tracking setup. This allows the KF to generate accurate fits for hit triplets. The resulting parameter estimates and covariance matrices are vital for constructing a high-purity set of track candidates.

3. **Track selection (Competition):** When multiple track candidates share hits on the same STS strip, a heuristic based on track length and χ^2 fit quality is used to select the optimal track. This selection step is crucial in the final iteration to resolve ambiguities.

These insights guided the development of the hybrid algorithm presented in this thesis. The proposed pipeline combines the representational power of data-driven deep learning models with the physical robustness of KF-based parameter estimation.

The following sections describe each stage of the GNN-based track finding algorithm in detail.

5.3 Graph Construction

The first step is to construct a graph with the hits recorded by the tracking system. Monte Carlo simulations predict that a central Au+Au collision at 10 AGeV in CBM will produce approximately 7000 hits across the 12 layers of the MVD and STS detectors. Naively connecting all pairs of hits on adjacent detector layers would will yield a graph with $\mathcal{O}(10^6)$ edges. A graph of this size exceeds memory and runtime requirements. Thus, an effective graph construction scheme is essential which, ideally, retains all true edges while discarding as many false edges as possible.

A straightforward approach is to apply geometric cuts: thresholds on physically motivated quantities such as the spatial separation between hits or differences in their angles relative to the primary vertex. Only hit pairs satisfying these criteria are retained. For example, in CBM the difference in hit angles in the YZ plane is particularly effective for suppressing false edges from primary tracks. More elaborate schemes can involve tuning cuts separately for each detector layer, or even for different regions within a layer, to account for variations in geometry or occupancy.

However, developing geometric cuts that simultaneously accommodate the full diversity of tracks (primary and secondary, low and high momentum) is labor-intensive and highly detector-specific. Moreover, overly restrictive cuts may bias the reconstructed track sample. To avoid this, a data-driven approach based on metric learning is adopted.

5.3.1 Metric Learning

A neural network is trained to map the measured hit coordinates into a high-dimensional embedding space in which hits from the same particle lie close together, while hits from different particles lie farther apart [130]. Distances in this space are given by the Euclidean metric. Once embedded, each hit is connected to a fixed number of its nearest neighbours on adjacent tracking stations. The resulting hit pairs are referred to as *doublets*.

The embedding is learned using a hinge loss:

$$\mathcal{L} = \frac{1}{n_{true}} \sum_{true} d^2_{true} + \frac{1}{n_{false}} \sum_{false} \max(\lambda - d^2_{false}, 0) \tag{5.1}$$

where the true (false) summation is over pairs of hits on adjacent stations from the same (different) particle, d^2 is the square of the Euclidean distance in the embedding space and λ is a hyperparameter called margin. During training, loss is minimized using gradient descent. The first term, $\mathcal{L}_{true}$, 'pulls' hits belonging to the same track closer while the second term, $\mathcal{L}_{fake}$, 'pushes' hits from different tracks away. The hits are pushed away until the distance between them equals the margin. Beyond this point, $\mathcal{L}_{fake} = 0$ and the second term does not contribute to the loss.

It is important to note that metric learning faces fundamental limitations in how tightly it can cluster hits from the same track. Consider three primary

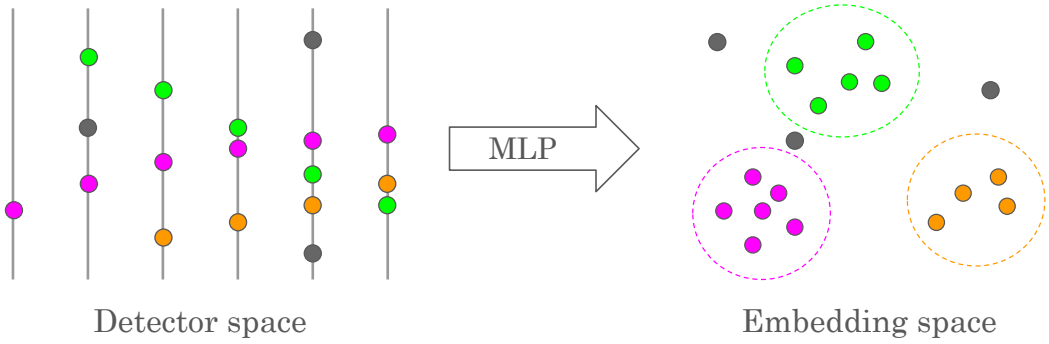


Figure 5.1: Illustration of the metric learning mapping of hits from detector coordinates to an embedding space where hits from the same track are clustered together. Grey hits represent hits from un-reconstructable tracks and ghost hits.

tracks, in three different collisions, that leave hits at the exact same positions on a given detector layer: two low-momentum tracks with opposite charges and one high-momentum track of arbitrary charge. Their subsequent hits will lie in widely separated regions with one deflecting to the left, one to the right, and one continuing nearly straight. No single embedding can map these three hits to a representation from which a unique next hit position is inferable, only a region of likely continuation can be learned. The situation is even more challenging for secondary tracks, where the absence of a known primary vertex to provide an initial bearing introduces additional ambiguity.

For these reasons, separate metric-learning models for each iteration of the tracking algorithm are trained. Each model specializes to the characteristic geometries and scattering profiles of the tracks targeted in that iteration, enabling more precise graph construction while avoiding over-constraining assumptions.

Embedding Model

The embedding model is implemented as a Multi-Layer Perceptron (MLP) comprising two hidden layers of width 16 and an output embedding dimension of six. All layers employ the hyperbolic tangent activation function (Sec. 4.4.2), constraining the embedded coordinates to the range $[-1, 1]$. The margin parameter in the hinge loss (Eq. 5.1) is set to $\lambda = 0.001$. A schematic of the network architecture is shown in Fig. 5.2.

The training data is derived from simulated Au+Au central collisions at 10 AGeV, generated using the UrQMD event generator (see Sec. 6.2 for details of the simulation setup). A training set comprising of 1000 events is used for training, with each event containing all detector hits, including noise, produced in a single collision. To promote numerical stability and accelerate convergence, input coordinates are normalized. The x and y positions are scaled by the transverse dimensions of the largest detector layer, and the z coordinate is shifted so that the primary vertex lies at the origin, reflecting the fixed-target geometry of the CBM setup.

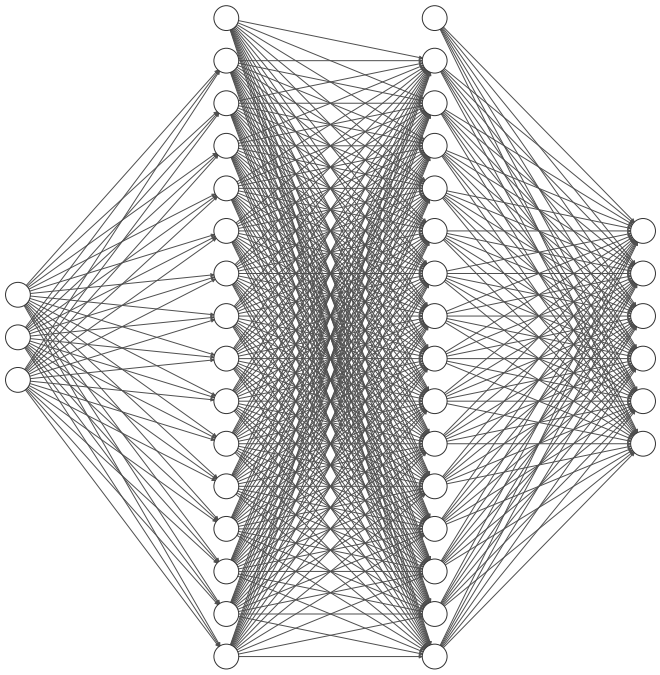


Figure 5.2: Structure of the MLP used for embedding. Figure generated using [131]

For each event, all reconstructible tracks, defined as those with at least four hits and momentum $p > 1\,\mathrm{GeV}/c$ (see Sec. 6.1), are iterated over to form the set of hit pairs on adjacent stations belonging to the same particle. These form the set of true doublets. To form fake doublets, random pairs of hits on adjacent stations that originate from different tracks are used. Because the number of possible fake combinations vastly exceeds the number of true ones, the resulting

dataset is highly imbalanced.

Class-weighted loss functions were explored to counteract this imbalance but were found to lead to unstable optimization and poorer convergence. Instead, fake doublets are randomly sampled to match the number of true doublets in each epoch, yielding a balanced training set that improves robustness and mitigates overfitting. The sampling is refreshed every epoch to promote generalization. The metric-learning loss of eq. 5.1 is then computed over the balanced sets of true and fake doublets.

The strong, inhomogeneous magnetic field within which the tracking detectors are placed, introduces substantial charge- and momentum-dependent curvature. Consequently, the spatial correlation between hits on adjacent layers differs substantially across track categories. Hits with similar positions on one layer may exhibit widely divergent positions on the next, creating an intrinsic ambiguity that cannot be adequately resolved by a single embedding model.

To address this, two specialized embedding networks are trained:

- A high-momentum primary-track model ($p > 1$ GeV/c), used exclusively in the first iteration of track finding.
- A low-momentum model, trained on both low-momentum primary tracks and secondary tracks, used in subsequent iterations.

For training the embedding model used in the first iteration, only doublets from high-momentum primary tracks are considered as true doublets. Similarly, only doublets from low-momentum tracks, both primary and secondary, are true doublets when training the second embedding model. Separating low-momentum primary and secondary tracks into distinct training sets did not yield additional performance improvements. Training the two broad categories separately, however, led to a significant increase in overall efficiency.

Training is performed using stochastic gradient descent (SGD) with a learning rate of 0.1 for 100 epochs. Gradients are accumulated across all hits in an event before each parameter update. An average training event contains roughly 400 reconstructible tracks with about 8 hits per track, which translates to $\mathcal{O}(10^3)$ true doublets per event. Over the full 1000-event dataset, the total number of true doublets reaches $\mathcal{O}(10^6)$, providing a sufficiently large and diverse sample for stable training.

Unlike classification tasks, the embedding network does not possess a natural

accuracy-based metric for performance evaluation. While the training loss provides a useful diagnostic, it is often insufficient, and even occasionally misleading, for selecting the optimal model. To ensure a robust assessment, every trained embedding is evaluated by running the full tracking chain in *doublets-as-tracks* mode. In this configuration, track finding halts at the doublet stage, and every doublet, including those sharing hits, is treated as an independent track and forwarded to the track-matching stage for performance analysis.

In this mode, track reconstruction efficiency of 100% is expected albeit with a high clone rate (see Sec. 6.1 for definition). Comparing the efficiency, clone rate and ghost rate allows reliable determine of the best embedding model. This strategy of pausing the tracking algorithm at intermediate reconstruction stages and analyzing the resulting objects proved essential throughout development. The embedding models were trained using the `ANN4FLES` package and integrated into the CBMRoot software framework which greatly simplified this debugging and testing workflow.

Visualization

Doublet construction with metric-learning provides a data driven alternative to the analytical extrapolation used in the Cellular Automaton (CA) Track Finder, where the Kalman Filter (KF) predicts the next hit position by assuming that the track originates from the primary vertex. While this assumption is well suited for primary tracks, it leads to reduced reconstruction efficiency for secondary tracks. In contrast, metric learning learns correlations between positions on consecutive stations directly from data, without imposing constraints on particle origin or momentum.

To better understand the behaviour of the embedding model, the learned distance metric is visualized. For a chosen hit on station k, its distance, in the embedding space, to every point on station $(k + 1)$ is computed and shown as a heatmap where colour encodes distance. Figure 5.3 shows four such visualizations, one in each quadrant, for the embedding model trained on low-momentum tracks. Each panel depicts a hit on station 3 (marked by a star) and the corresponding learned distance field on station 4.

In the heatmaps, dark blue regions represent embedding-space locations closest to the originating hit and are therefore prioritized during doublet construction while yellow regions are the farthest and considered last. The top-k neighbours

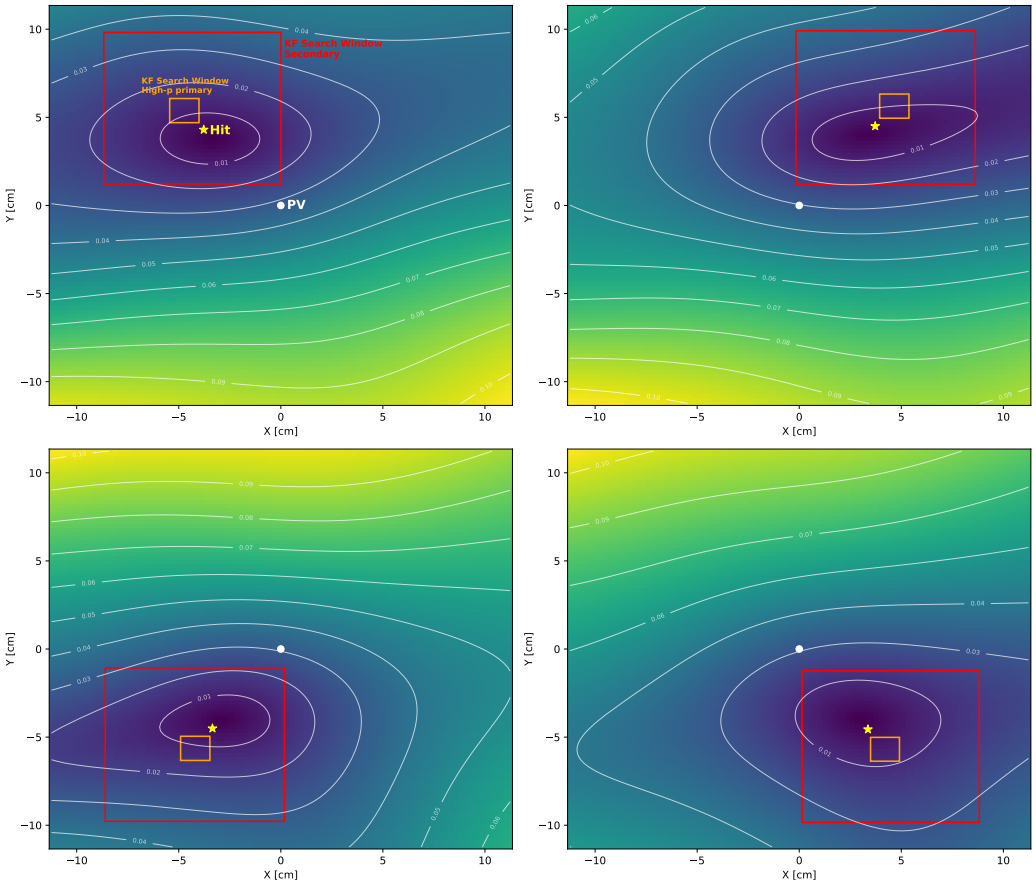


Figure 5.3: Visualization of the learned embedding distance used for finding low momentum tracks through a front-on view of the tracking system. The yellow star represents a hit on station 3 and the white circle marks the Primary Vertex (PV). Both of these points are projected onto the plane of station 4 whose surface, covered by a heatmap, is shown in the figure. Blue (yellow) colours represent areas closest (farthest) to the hit with the contours showing areas at equal distance. The contours are elongated along the x-axis because a magnetic field, directed along the y-axis, is present at the tracking stations. The red (orange) rectangle shows the search window created by extrapolation with the Kalman Filter in the iterations focused on finding secondary (high-momentum primary) tracks. The four figures show how the embedding distance varies in the four quadrants.

used to construct doublets are selected based on this distance distribution. Overlaid rectangles correspond to the KF search windows used in the CA Track Finder: the smaller orange window targets high-momentum primaries, while the larger red window targets secondary tracks. The learned search regions differ significantly from the KF predictions. Importantly, the contour elongation along the x-axis is consistent with the bending induced by the magnetic field, indicating that the network has learned physically meaningful correlations.

By replacing KF extrapolation-based search windows with a learned distance function, the metric-learning approach offers a flexible, data-driven mechanism for doublet construction. It can be trained on specific track categories and adapts naturally to changes in detector geometry or other experimental conditions, provided updated simulations are available. The embedding network itself is small, containing only 438 trainable parameters, and requires only 1000 simulated Au+Au central collisions at 10 AGeV for training. It trains in under an hour. All training and data-generation procedures are fully integrated into CBMRoot.

In Appendix A, the generalization, to different beam energies and collision systems, of the complete GNN-based tracking algorithm is shown. This implicitly confirms that the metric-learning component also generalizes well.

5.3.2 k-Nearest Neighbour Search

The metric learning loss encourages hits belonging to the same particle trajectory to cluster in the embedding space, while hits from unrelated tracks are pushed apart. Although this mapping is not perfect, many hits do not have their true successor as the first nearest neighbour, it reliably ensures that the correct successor lies among the closest neighbours. This represents a substantial improvement over naïvely considering all hits on the next station as equally likely candidates. A k-Nearest Neighbour (kNN) search algorithm, by distance in the embedding space, is used to identify candidate hits for doublet formation.

For the first iteration, optimized for high-momentum primary tracks, $k = 20$ yields the best balance of reconstruction efficiency and clone rate when evaluated in the doublets-as-tracks mode. The second and third iterations use $k = 25$ for adjacent-station doublets and $k = 10$ for next-to-adjacent (*jump*) doublets. Since a single, shared embedding model is used across all stations, hits from the same trajectory even on non-adjacent stations are clustered due to transitivity.

The identification of jump doublets does not require any additional models or training.

5.4 Doublet Filtering

The first two iterations of the tracking algorithm are designed to reconstruct primary tracks, which by definition originate from the primary vertex (PV). Given that the magnetic field in the detector is oriented along the y-axis, an additional geometric constraint can be exploited to further reduce the number of candidate doublets. Specifically, for each doublet, the intercept of the straight line connecting the two hits of the doublet in the y–z plane with the z-axis at the PV is calculated. For genuine primary tracks, this intercept is expected to be close to zero. Deviations arise primarily from measurement uncertainties and multiple scattering within the detector material.

All doublets whose extrapolated y–z intercept exceeds a set threshold are discarded. It was found that a threshold of 2 cm provides an excellent balance between retaining true doublets and suppressing spurious ones. This simple geometric filter reduces the combinatorial complexity enough to remove the need for a computationally expensive graph-based filtering in the first two iterations. In any case, constructing a full graph at this stage would be computationally prohibitive: an event contains on average $\sim$ 5000 hits which would lead to more than 150,000 candidate edges.

The third iteration targets the reconstruction of secondary tracks. By this stage, hits associated with already reconstructed primary tracks are masked, leaving on average approximately 1700 active hits per event. The resulting graph is small enough to for graph-based processing.

Graph Neural Network

A Graph Neural Network (GNN) is used for doublet filtering. The GNN updates the representation of each edge by aggregating information from its neighbouring edges, in both up- and down- stream directions. The initial edge feature vector for edge from node i $\rightarrow$ j is,

$$\mathbf{e}_{ij} = [\mathbf{r_i}, \mathbf{r_j}] \,, \tag{5.2}$$

where $[\cdot, \cdot]$ represents concatenation and $\mathbf{r_k} = (x_k, y_k, z_k)$ are the normalized coordinates of some hit k. The nodes are ordered by increasing station number in

the edge features. The edge representation is iteratively updated over a number of message passing layers (see Sec. 4.6.1). The updated edge features after a message passing layer is,

$$\mathbf{e}'_{ij} = h_\Theta \left(\mathbf{e}_{ij} + \frac{1}{\hat{d}_{\mathbf{e}_{ij}}} \sum_{\mathbf{e}_{kl} \in \mathcal{N}(\mathbf{e}_{ij})} \mathbf{e}_{kl} \right), \tag{5.3}$$

where h_Θ denotes a MLP and $\hat{d}_{\mathbf{e}_{ij}}$ is the number of neighbouring edges. The neighbours of an edge i $\rightarrow$ j consists all the edges k $\rightarrow$ i and j $\rightarrow$ l where k and l are neighbours of nodes i and j respectively. A distinct MLP (h_Θ) is used in every layer for greater expressivity.

The GNN operates in a semi-supervised setting. All edges are labeled as either true, connecting hits from the same Monte Carlo track, or fake. To mitigate class imbalance related problems, randomly selected fake edges, equal in number to true edges, are used for gradient calculation. All true edges are used in back-propagation. There are two message passing layers, each of width 16. This depth allows sufficient interaction between spatially adjacent edges without diluting information relevant to short tracks. Increasing the number of message passing layers was found to degrade performance, as edges belonging to short (four-hit) tracks were often filtered out. Since the goal of this iteration is to identify track triplets, rather than complete tracks, a shallow network depth is both sufficient and computationally efficient. The subsequent triplet fitting step naturally constrains the combinatorial explosion in the later track-building phase. A tanh activation is applied. A MLP with one hidden layer of width 16 performs the final classification. The MLP takes as input the edge representation after two layers of message passing. Finally, a sigmoid activation produces output scores in the range $[0, 1]$. A higher score signifies higher likelihood of the edge being true.

A binary cross entropy loss is used as the loss function. The sigmoid focal loss [132] was also experimented with but worse results were obtained.

For training, a dataset of 10,000 sparse graphs is prepared using the trained metric learning model for the third iteration. Each graph contains, on average, about 1700 nodes (hits) and $\sim$25,000 edges (doublets), with each node connected up to a maximum of 25 edges on the next station. Among these, the number of true edges is $\mathcal{O}(100)$. The network is trained to discriminate between true and fake edges, aiming to retain as many true connections as possible while eliminating combinatorial noise. Jump edges, connections to next-to-next stations, are

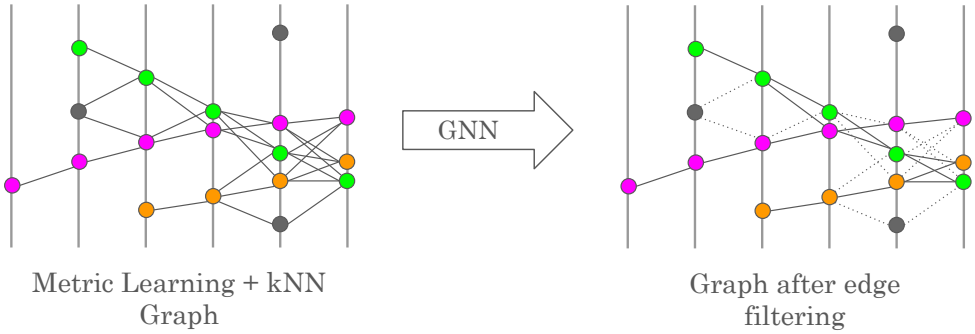


Figure 5.4: An illustration of edge filtering. The sparse graph created with metric learning and k-Nearest Neighbour (kNN) in the embedding space (left) is filtered with a Graph Neural Network (GNN) edge classification network to produce a new set of edges (right). Hits filled with the same colour belong to the same track. Filtered out edges are shown by dotted lines.

excluded from the graph since a method for handling them satisfactorily was not found. This should be investigated further in future work. For now, all jump edges are considered true and passed to the triplet construction stage.

During development, each iteration is optimized independently. The GNN training is performed after the first two iterations (primary track finding) have been optimized. This ensures that the sparse graphs used for training reflect realistic conditions encountered during inference. This stage-wise training ensures that the GNN learns to operate with the hits remaining in the third iteration.

During inference, the trained GNN processes the sparse graph generated in the third iteration and outputs a probability score for each edge. Edges with scores below a certain threshold are discarded. A threshold of 0.5 is used in this work. The edges with scores higher than the threshold are passed to the triplet-construction stage. Adjusting the threshold provides a mechanism to trade off reconstruction efficiency against computational performance. A lower threshold increases recall at the expense of additional false edges, but also increases runtime. In practice, the trained GNN achieves a reduction in the number of edges by a factor of approximately 2.5, leading to a $\sim$80% reduction in the number of triplets constructed.

The network is intentionally conservative: it prioritizes recall (retaining true edges) over precision. False edges can be filtered out in subsequent stages of the

pipeline, but true edges, once removed, cannot be recovered. This conservative filtering strategy maximizes the eventual reconstruction efficiency for secondary tracks while maintaining manageable computational complexity.

5.5 Triplet Construction and Filtering

The next stage of the algorithm involves triplet construction and filtering. A *triplet* is defined as a set of three hits located on distinct detector stations. It represents the minimal track stub required to estimate curvature, and consequently momentum, without relying on external vertex constraints.

Triplets are constructed by linking pairs of doublets that share a common central hit. During this process, a single doublet may contribute to multiple triplet candidates. While the baseline algorithm connects doublets from adjacent stations, the pipeline also supports *jump* triplets, which are formed by combining a standard doublet with a jump doublet (a doublet that skips a station). A maximum of one jump doublet is permitted per triplet. It was found that allowing more jumps increased the ghost rate and reduced hit efficiency markedly. No jump triplets are constructed in the first iteration because no strip has been assigned to a track yet.

Consequently, the maximum station index difference within a triplet is three. This yields three valid topological configurations, described by the station index k of the first hit:

- No jump: $[k, k{+}1, k{+}2]$,
- Jump on first doublet: $[k, k{+}2, k{+}3]$,
- Jump on second doublet: $[k, k{+}1, k{+}3]$.

In the first iteration, which focuses on identifying primary, high-momentum tracks, strict physics-based cuts are applied. The primary-vertex constraint is enforced during doublet formation (constraining the YZ intercept), and a momentum cut is applied during triplet formation by discarding candidates with a fitted momentum below $0.5\,\mathrm{GeV}/c$. These constraints reduce the candidate pool sufficiently to bypass computationally intensive GNN-based filtering. This approach leverages established physical knowledge, like particle motion in a magnetic field, energy losses and multiple scattering when passing through detector

material, through the KF and reserves the neural networks for more complex later iterations.

5.5.1 Kalman Filter Fitting

Each triplet provides six independent spatial measurements, corresponding to the (x, y) coordinates of the three hits. This is sufficient for a KF fit to yield a reliable initial estimate of the track parameters (position, slope and momentum). In addition to the fitted state vector, the KF returns a full covariance matrix describing the parameter uncertainties and correlations. A detailed description of the KF used in CBM is provided in [114, 133].

To assess the quality of a triplet candidate, a χ^2 goodness-of-fit value is computed:

$$\chi^2 = (m - Hx)^T V^{-1} (m - Hx), \tag{5.4}$$

where m denotes the measured hit positions, x the fitted state vector, H the projection matrix from state space to measurement space, and V the measurement covariance matrix. This χ^2 value acts as a rigorous discriminator; triplets exceeding a predefined threshold are rejected as inconsistent with a physical track hypothesis. This filtering step drastically reduces the combinatorial background propagated to the track construction stage. Additionally, the fitted parameters are stored to assess the geometric and kinematic compatibility of overlapping triplets.

5.5.2 Validation of Triplet Fitting

To verify the correctness of the triplet-fitting procedure, the distribution of the fitted χ^2 values is checked. For correctly estimated uncertainties, the calculated χ^2 values should follow the true χ^2 distribution with one degree of freedom. A common diagnostic tool is the *prob function* [134]. It is defined as the cumulative distribution function (CDF) of the χ^2 distribution evaluated at each measured χ^2. For correct fits, the resulting distribution of prob values should be approximately uniform over the interval $[0, 1]$.

The prob distributions for true and ghost triplets created in the third iteration (secondary tracks) is shown in Fig. 5.5. The flat distribution for true triplets confirms the validity of the KF fitting procedure, while deviations for ghost triplets indicate their non-physical nature and support their rejection through χ^2-based

filtering. Additionally, the peak in the prob distributions near zero can be used to set a threshold on χ^2 beyond which triplets are rejected.

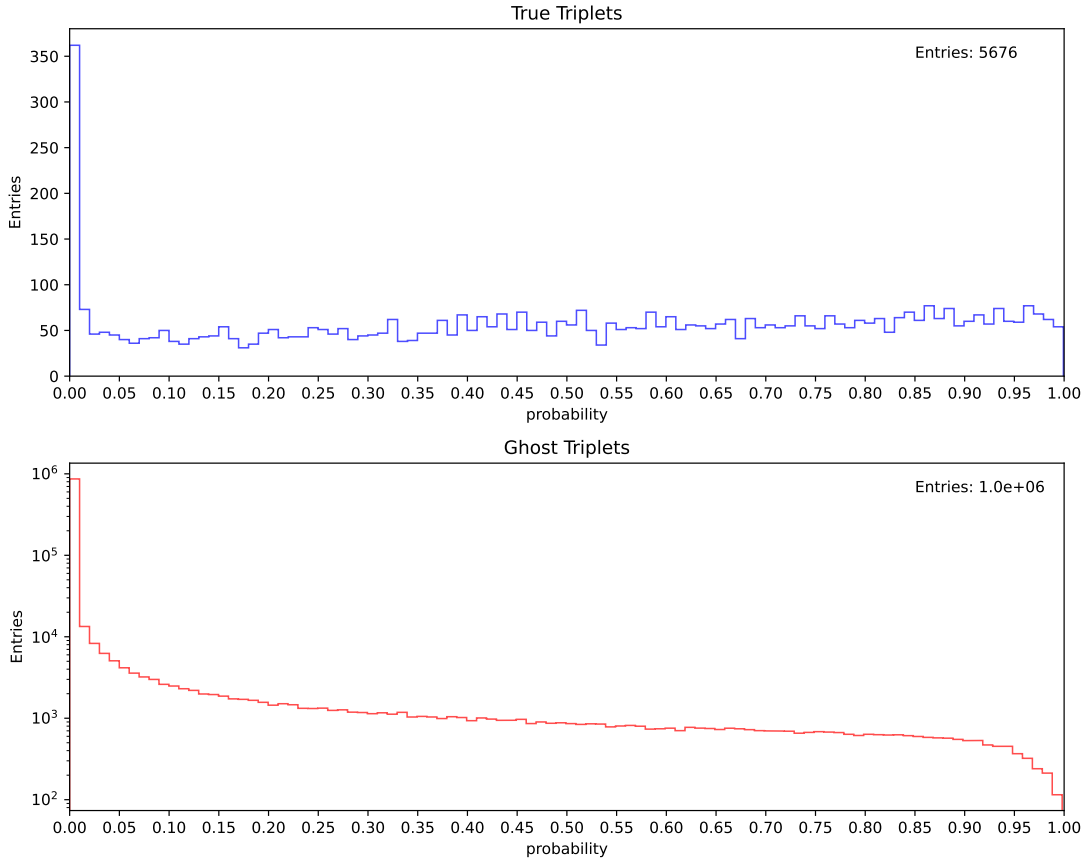


Figure 5.5: Validation of triplet fitting through the CDF of the χ^2 distribution (*prob function*) for true and ghost triplets. A flat distribution for true triplets validates correct fitting by the Kalman Filter.

Triplet construction and fitting was tested by running the algorithm in *triplets-as-tracks* mode. In this mode, triplets are considered as tracks and passed forward to the analysis modules where, similar to complete tracking, the 70% matching hits rule is used for classifying a track as reconstructed (see discussion in Sec. 6.1). For triplets this implies that all its hits must belong to the same MC track. A high efficiency and clone rate, since hit sharing is permitted, at this stage is necessary for the full algorithm to have good performance. The algorithm is optimized iteration by iteration. When checking the second iteration in triplets-as-tracks mode, the first iteration is run in full tracks mode. Furthermore, the efficiency focused on by that iteration, for instance, high momentum primary in the first

iteration, should be maximized and not the overall efficiency. The following stages operate with triplets, and doublets can be discarded.

5.6 Track Candidate Construction and Filtering

The next step in the algorithm is to construct *track candidates* from the filtered triplets. In this stage, compatible triplets are combined into longer sequences. The track candidates have to be filtered to suppress ghost tracks and improve track selection.

5.6.1 Track Candidate Construction

Track candidates are constructed by iteratively combining overlapping triplets. A pair of triplets are defined as overlapping if the last two hits of the first triplet are identical to the first two hits of the second. This overlapping condition applies a strong geometric constraint but the momenta of the two triplets can still be different which is not possible if they belong to the same track. Therefore, the momenta, provided by the KF, of the two overlapping triplets must not differ by more than five times the square root of the corresponding covariance matrix element for the momentum parameter. This ensures that only kinematically compatible triplets are combined.

All kinematically compatible combinations of overlapping triplets are enumerated and stored. Intermediate-length candidates are retained even if they can be grown into longer ones. These shorter candidates act as a fallback during track selection. Triplets that fail to merge with any compatible neighbour are discarded.

5.6.2 Track Candidate Filtering

The constructed track candidates must be filtered to suppress ghosts for effective track selection. The filtering procedure differs for the first two iterations (focused on primary tracks) and the third iteration (focused on secondary tracks).

First and Second Iterations: For the first two iterations, a χ^2-like score is used to quantify track candidate quality. The score for a given track candidate is the sum of the χ^2 value of its seed triplet (or score of the current track segment)

and a compatibility penalty term for each added triplet:

$$S = \chi^2_{\text{seed}} + \sum_i \frac{(\Delta q/p_i)^2}{C_{q/p,i}}, \tag{5.5}$$

where $\Delta q/p_i$ is the difference in the q/p parameter between the new and last triplet of the track segment, and $C_{q/p,i}$ is the corresponding covariance matrix element.

The number of degrees of freedom for a candidate is defined as $n_{\text{dof}} = 2 \times N_{\text{hits}} - 5$, and a candidate is accepted if

$$S < 5 \times n_{\text{dof}}. \tag{5.6}$$

This simple scheme is sufficient for the first two iterations. The targeted track class efficiency and ghost rate after track selection is good enough that there is no need for a more complicated, and computationally intensive, filtering process.

Third Iteration: The third iteration is more challenging because the absence of a primary-vertex constraint substantially increases the ghost rate. Each candidate is therefore re-fitted with the KF to obtain a better estimate of the track parameters and their covariance. These fitted quantities serve as inputs to a classifier trained to distinguish between true and ghost candidates.

The classifier is a MLP trained to output a continuous score between 0 and 1, with lower scores indicating a higher likelihood of being a true track. This inverted convention, assigning the label 1 to true tracks and 0 to ghosts, was chosen to maintain an analogy with the interpretation of a χ^2 statistic.

Track Candidate Classifier

The MLP is trained on a dataset of one million track candidates, evenly split between true and ghost examples. Training data is generated by running the full first and second iterations of the algorithm, followed by the third iteration up to the track candidate construction. Each candidate is matched to Monte Carlo (MC) truth using a strict condition: all hits of the candidate must correspond to a single MC track (100% hit purity). This is more stringent than the 70% criterion used for track finding performance evaluations, but was found to provide cleaner training labels and improved classifier performance. To produce a sufficiently large sample of perfectly matched candidates, thresholds in doublet

creation, triplet filtering, and candidate construction were loosened to maximize reconstruction efficiency.

The classifier takes as input a 13-dimensional feature vector:

$$[\chi^2,\, t_x,\, t_y,\, q/p,\, C_{xx},\, C_{yy},\, C_{t_x},\, C_{t_y},\, C_{q/p},\, n_{\text{dof}},\, x_{\text{start}},\, y_{\text{start}},\, z_{\text{start}}]. \tag{5.7}$$

Each feature is max-scaled to the interval $[-1, 1]$ using statistics from the training dataset.

The network comprises of three hidden layers of width 32 with tanh activation. It is trained for 200 epochs using stochastic gradient descent with a learning rate of 10^{-3}. During inference, candidates with classifier scores below 0.5 are forwarded to track selection.

To study the separability between true and ghost track candidates, the feature space using t-Distributed Stochastic Neighbour Embedding (t-SNE) [135, 136] is visualized. t-SNE is a nonlinear dimensionality reduction technique that maps high-dimensional data into a low-dimensional space while attempting to preserve the local structure of the data, i.e., points that are close in the original feature space remain close in the low-dimensional projection. It achieves this by converting pairwise distances between points into conditional probabilities and minimizing the Kullback–Leibler divergence between the two distributions. The result is an intuitive visualization of how well the two classes are separable in feature space.

Figure 5.6 shows the t-SNE projection of the training sample, with colour indicating the absolute momentum |p|. While two broad regions associated with true and ghost candidates are visible, a significant overlap remains. The overlapping region contains predominantly low-momentum true tracks, as well as ghost candidates that exhibit geometric or kinematic features similar to genuine tracks. This illustrates that perfect classification is not possible. Factors like detector resolution and inefficiency, multiple scattering and energy loss will inevitably cause some true tracks to appear similar to ghosts, and some ghost tracks, by pure combinatorial chance, will appear like true tracks.

This filtering step significantly reduces the number of track candidates while preserving the majority of true tracks. This improves the efficacy of track selection.

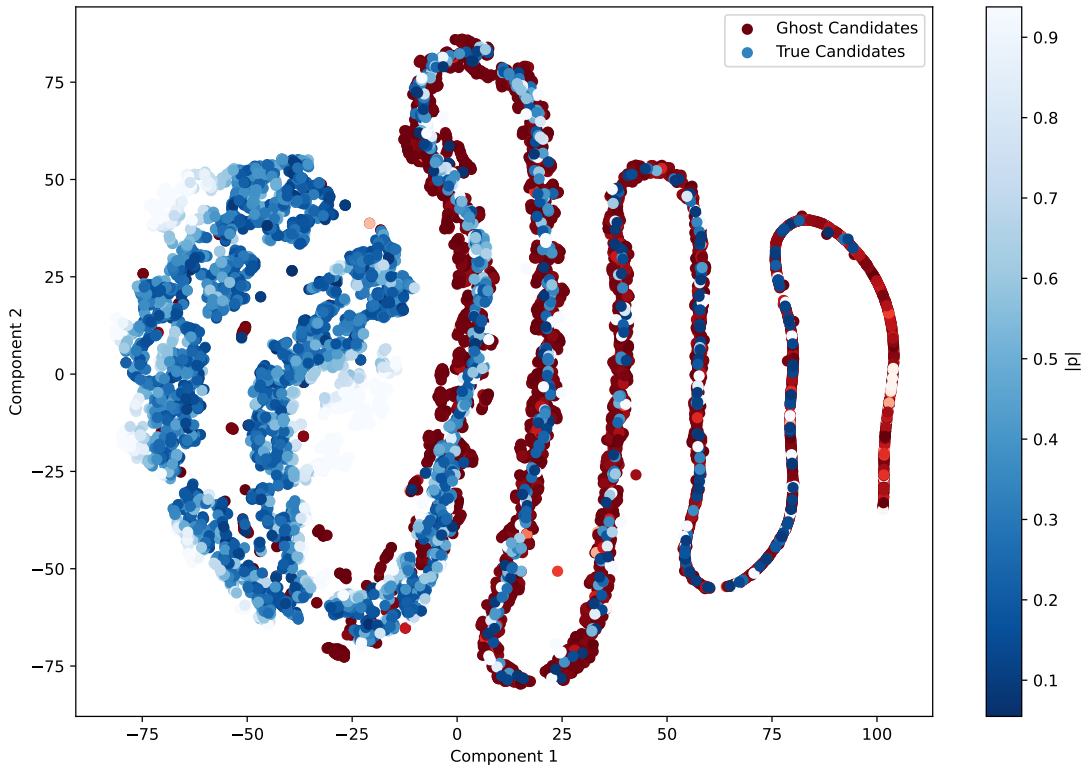


Figure 5.6: t-SNE visualization of true and ghost track candidates in the classifier feature space. The colour represents the absolute value of momentum ($|p|$) of track candidates. The mostly low-momentum tail of true candidates overlaps with the ghost tracks, indicating a limit on separability of the two classes. The split head in the true tracks arises because particles have two charges.

5.7 Track Selection

Track selection is the final stage of the tracking algorithm. It is responsible for resolving shared detector hits among the set of track candidates. Not only must every hit be assigned to be a single track but also the strip. The first four tracking stations are part of the MVD detector and use silicon pixels for detection and do not have the complication of strips. The other 8 stations are part of the STS detector which is made of front and back silicon strips. The strips are oriented at a relative angle of 7.5° to minimize *ghost hits* created by intersection of activated strips and not real particles directly (Fig. 5.8).

During earlier stages of the algorithm, track candidates of all intermediate lengths are generated and fitted using the Kalman Filter (KF). Because even

short track segments can yield good KF fits, many clones (multiple candidates corresponding to the same true track, see Sec. 6.1) survive the earlier stages. The goal of track selection is to identify and retain the best representation of each true track while suppressing clones and ghosts.

Track selection begins by establishing an ordering among track candidates so that conflicts can be resolved consistently. Two quantities are used for ranking: the number of hits in the track, called track length, and the χ^2 value from the KF fit. The χ^2 value is the deviation between the track defined by the KF fit parameters and the measured hits. Track length is the most powerful discriminator between true and ghost tracks. The probability that a random combination of hits accidentally mimics a real trajectory decreases rapidly with the number of hits. This is confirmed by the observation that the number of ghost tracks decreases with track length. There are very few ghost tracks with more than 5 hits. Furthermore, since the KF is an iterative fitter, the track parameter estimate is better for longer tracks. Classification of longer tracks is therefore easier.

Tracks are first sorted in descending order of length. Among tracks with equal length, those with smaller χ^2 values, indicating better agreement between the fitted model and observed measurements, are ranked higher. The list of track candidates is therefore ordered by (i) longer length and (ii) lower χ^2 when lengths are equal.

The ordered list is then iterated through. If all hits of a track candidate are still available (i.e., neither its front nor back strip have been claimed by a higher-ranked track), the track is accepted and all its hits are marked as used. Lower-ranked tracks that attempt to reuse these hits or strips are rejected. This procedure removes the majority of clone tracks and significantly reduces the ghost rate.

5.7.1 Cooperative Track Selection

In some cases, two real tracks may produce hits on the same strip. A simple ownership scheme, where the longer and better fitting track claims the hit, would allow only one of these tracks to be reconstructed. The other track would be lost, particularly if the two segments without the contested hit have length less than four. To deal with such a scenario, a novel *cooperative* selection scheme is used.

When a track candidate is rejected because only one of its hits conflicts with an already accepted longer track, the algorithm checks whether this conflict can

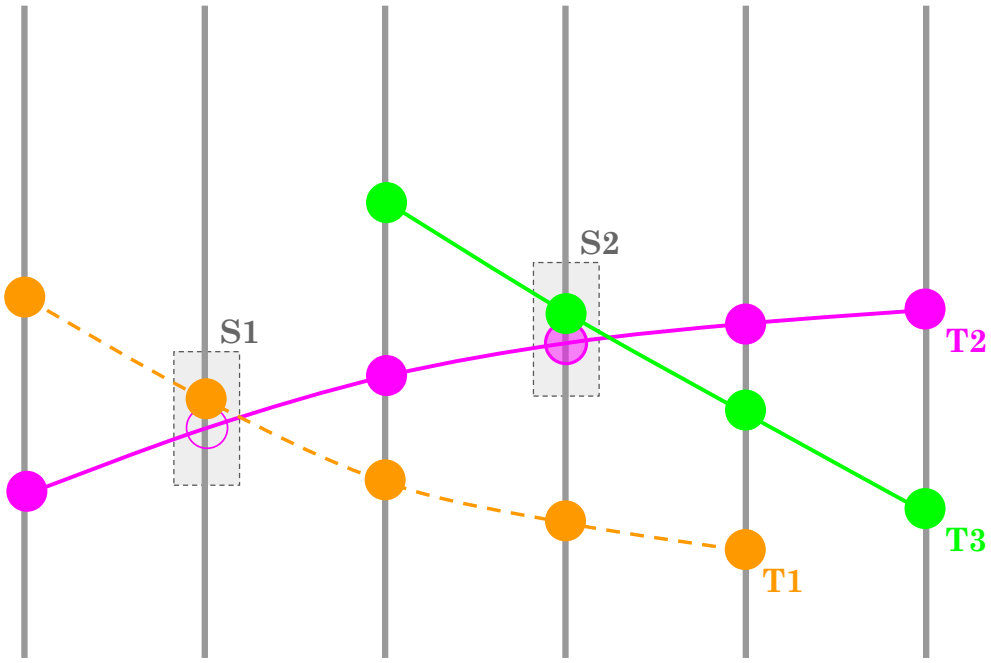


Figure 5.7: An illustration showing cooperative track selection and jump triplets. T1 is a track reconstructed in the first iteration which masks out strip S1 for subsequent iterations. T2 and T3 are tracks reconstructed in the second iteration. T2 is constructed with the help of a jump triplet over the unavailable hit in S1. Strip S2 is shared between hits of T2 and T3. With cooperative competition, T3 claims strip S2 because the longer, and typically more preferred, track T2 still has enough hits to be reconstructed without the hit on strip S2.

be resolved amicably. Specifically, it scans the list of previously accepted, longer tracks to identify which track holds the shared strip. If such a track is found and its χ^2 value is higher (i.e., it is less well fitted) than the shorter track, the strip ownership is swapped. The contested hit is reassigned to the shorter track and removed from the longer track. The shorter track is then accepted and its hits are marked as used. This cooperative adjustment ensures that both real tracks can coexist when they share a single strip, thereby improving overall tracking efficiency (Fig. 5.7).

Algorithm 1: Track Selection

Input: List of candidate tracks with scores: `trackAndScores`

Output: Cleaned list of tracks with unique hit assignments

```
1  Sort trackAndScores by track length(longer first) and then score(lower
     first);
2  foreach track T in trackAndScores do
3      NUsedHits ← 0;
4      foreach hit h in T do
5          if h.FrontStrip or h.BackStrip already used then
6              NUsedHits ← NUsedHits + 1;
7      if NUsedHits = 0 then
8          Mark all hits in T as used;
9          continue (next track);
10     else
11         if len(T) − NUsedHits ≥ 4 then
12             Remove used hits from T and mark remaining as used;
13             continue (next track);
14     remove ← true;
15     if (len(T) − NUsedHits) == 3 then
16         foreach longer track B (before T) do
17             if len(B) ≥ 5 and score(B) ≥ score(T) then
18                 if some hit b in B shares strip with a hit in T then
19                     Transfer hit from B to T;
20                     remove ← false; break;
21     if remove then
22         Remove T from trackAndScores;
23     else
24         Mark all hits in T as used;
```

5.7.2 Handling Shared Strips

Despite the cooperative competition scheme, certain true tracks may still remain unreconstructed when two tracks share a strip but one of them was reconstructed in an earlier iteration (Fig. 5.7). At the end of each iteration, all hits associated with used strips are masked, making them unusable in later iterations. While disabling masking could, in principle, recover these lost tracks, it would increase the ghost rate significantly besides increasing the combinatorial load. For central Au+Au collisions at 10 A GeV, at the sensors closest to the beam pipe on station 1 of the STS, the probability to get more than one particle hits is 3.6% [26].

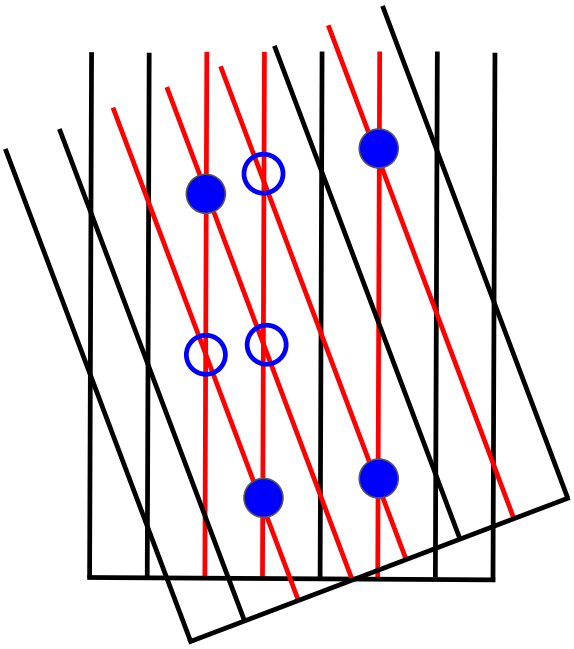


Figure 5.8: An illustration of the front and back strips in a sensor of the STS detector. The filled red circles represent hits created by real particles. The empty circles represent ghost hits that get created by the intersection of activated strips (coloured red). Two real hits, towards the right, lie on the same strip (vertical line) and cannot both be assigned to reconstructed tracks.

To address this limitation, my algorithm creates *jump tracks* after the first iteration. Jump tracks have a missing hit on one intermediate tracking station. They are formed by constructing doublets between hits on next-to-adjacent stations. The metric learning method for constructing doublets supports finding

these jump doublets quite naturally. Using jump doublets, triplets and track candidates which skip a hit on an intermediate station can be created. Jump tracks allow finding tracks that would otherwise be lost due to shared strips and detector inefficiencies. Jump tracks are treated exactly as all others in track selection.

Limitation

The fact that some tracks naturally produce hits on the same strip imposes a upper bound on the number of simultaneously reconstructable tracks. To estimate this limit, all reconstructible Monte Carlo (MC) tracks in an event were considered to calculated how many can be reconstructed under the constraint that no two tracks share a strip. When two tracks attempt to use the same strip, it is assigned to the shorter one (or randomly if both have equal length). This maximizes the number of reconstructible tracks since at least four hits are required for reconstruction. This analysis yielded a maximum achievable efficiency of 98.3%. This is an intrinsic limit of the tracking set up rather than an algorithmic deficiency.

A potential solution to overcome this limitation is a track extender method, used after the final iteration, which attempts to elongate promising triplets and track candidates with hits only on strips used by a reconstructed track.

5.7.3 Clone Suppression and Track Extension

Finally, to further reduce the number of clones, a clone-suppression step merges duplicate tracks. All pairs of tracks are extrapolated up- and down- stream, using the KF, to check if both are actually fragments of the same track. A track can be split and reconstructed again due to detector inefficiencies, strip sharing or limitations of the track finding algorithm. If two tracks are compatible (low combined χ^2), they are merged. This extension step further reduces the clone rate.

Summary

This chapter describes the Graph Neural Network (GNN)-based track finding algorithm designed for the CBM experiment developed in this thesis. The problem

of track finding and recent efforts towards applying deep learning methods to the problem are briefly reviewed. The algorithm works in three iterations, each of which is focused on reconstructing tracks with distinct trajectories. Each iteration starts with constructing pairs of hits called doublets using a MLP-based metric learning model and k-Nearest Neighbours search in the learned latent space. The first two iterations, targeted at finding primary tracks, use the primary vertex constraint to filter doublets. The third iteration, focused on finding secondary tracks, uses a more sophisticated GNN-based edge classification for doublet filtering. Doublets between hits on next to adjacent stations are created after the first iteration to allow finding tracks with missing or shared-strip hits.

Next, groups of three hits, called triplets, are constructed by combining a pair of doublets that share a middle hit. The Kalman Filter (KF) is used to fit and filter poorly fitting triplets. Track candidates are constructed by combining kinematically compatible overlapping triplets. The track candidates are filtered via a heuristic in the first two iterations and by refitting and passing track parameter estimates to a MLP-based classifier in the third iteration. Finally, a novel cooperative track selection procedure outputs a list of found tracks.

The next chapter presents the performance of the GNN-based track finding algorithm and compares it with the Cellular Automaton Track Finder.

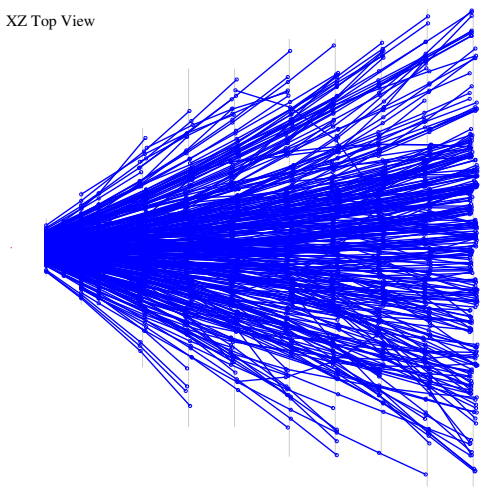
Chapter 6

Track Finding Performance

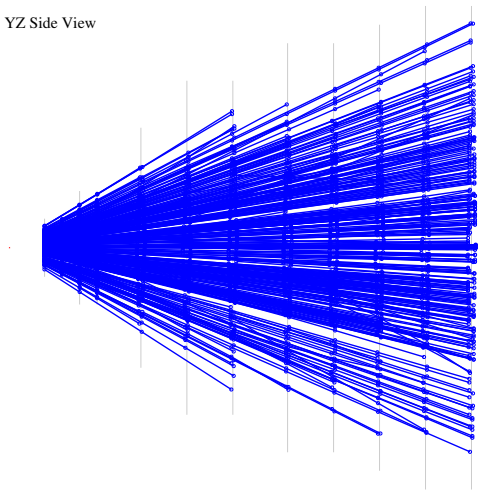
In this chapter, the performance of the Graph Neural Network (GNN)-based track finding algorithm is presented. First, the relevant evaluation metrics are defined, specifically adapted for the CBM experiment. These metrics provide a fair basis for comparison and enable the quantitative assessment of algorithm performance. Since experimental data lacks ground truth information, Monte Carlo (MC) simulations are essential for validation. The reconstruction efficiency of the GNN approach is benchmarked against the widely adopted Cellular Automaton (CA) Track Finder [128], which has been developed within the CBM collaboration for over two decades.

6.1 Track Finding Evaluation Metrics

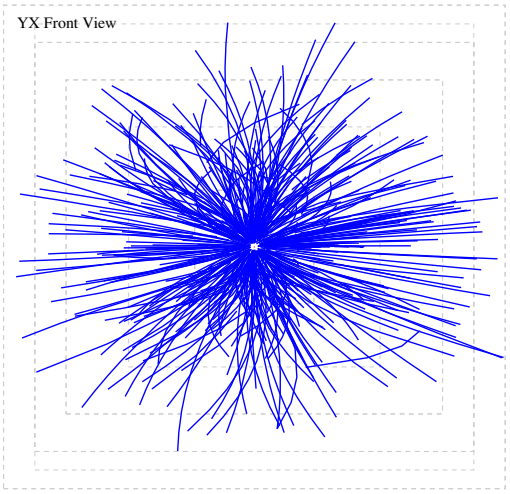
The following quantities are used to measure the performance of track finding algorithms in CBM. Similar metrics are standard in high-energy physics experiments involving track reconstruction. A central Au+Au collision at 10 AGeV produces thousands of particles on average. However, many trajectories lie partially or fully outside the detector acceptance. Tracks are classified as *unreconstructable* if the detector records insufficient information to accurately determine track parameters. Unreconstructable tracks are excluded from performance calculations. In CBM, a track is considered *reconstructable* only if it leaves at least four hits in the tracking detectors. Henceforth, references to MC tracks imply the subset of MC tracks that are reconstructable.



(a) XZ View



(b) YZ View



(c) YX View

Figure 6.1: Event display showing 395 tracks reconstructed with the GNN-based track finder algorithm from a central Au+Au collision at 10 AGeV.

The **tracking efficiency** ϵ is defined as,

$$\epsilon = \frac{N_{\text{matched}}}{N_{\text{reconstructable}}}, \tag{6.1}$$

where $N_{\text{reconstructable}}$ is the number of MC tracks with atleast four hits and N_{matched} is the number of MC tracks *matched* with a reconstructed track. A reconstructed track is considered matched if it shares more than 70% of its hits with a single MC track. Reconstructed tracks failing this criterion are classified as *ghost tracks*. For instance, a reconstructed track with four hits must have at least three hits belonging to the same MC track to be considered matched.

The **ghost rate** is defined as,

$$\text{Ghost Rate} = \frac{N_{\text{ghost}}}{N_{\text{reconstructed}}}, \tag{6.2}$$

where N_{ghost} is the number of ghost tracks in an event and $N_{\text{reconstructed}}$ is the number of tracks found by the track finding algorithm. Low ghost rates are important because in the real data there is no way to determine which of the reconstructed tracks correspond to real particles and which are just *ghosts*.

It is possible for two reconstructed tracks to match the same MC track. These duplicates are termed *clones*. The **clone rate** is defined as:

$$\text{Clone Rate} = \frac{N_{\text{clone}}}{N_{\text{reconstructed}}}, \tag{6.3}$$

where N_{clone} is the number of duplicate tracks within the reconstructed set.

While the 70% hit-matching threshold defines reconstruction, algorithms yielding tracks with higher fractions of correctly associated hits provide more precise reconstructed parameters. This is quantified by the **hit purity**:

$$\text{Hit Purity} = \frac{N_{\text{true track hits}}}{N_{\text{hits in track}}}, \tag{6.4}$$

averaged over all matched tracks in an event. A complementary measure, **hit efficiency**, quantifies the fraction of true MC hits recovered in the reconstructed tracks:

$$\text{Hit Efficiency} = \frac{N_{\text{matched hits in reco tracks}}}{N_{\text{hits in MC track}}}. \tag{6.5}$$

To account for event-by-event fluctuations, these metrics should be averaged over many events. In this thesis, all statistics are averaged over 1000 events unless otherwise stated.

It is often useful to categorize tracks by origin, *primary* (originating from the primary vertex) or *secondary* (from decays), and momentum. A threshold of $p = 1\,\mathrm{GeV}/c$ separates *high-* and *low-momentum* classes. Generally, high-p tracks follow straighter trajectories, while low-p tracks curve significantly in the magnetic field, making reconstruction more challenging. Primary tracks are generally easier to reconstruct due to the additional constraint of the primary vertex. The relative difficulty of track reconstruction increases in the following order:

1. High-p primary
2. Low-p primary
3. High-p secondary
4. Low-p secondary

This hierarchy is reflected in the reconstruction efficiencies presented in subsequent sections. The GNN algorithm employs three iterations, each targeting a specific track class, allowing for targeted optimization based on these categories.

6.2 Simulation Framework

Reliable validation of any new tracking algorithm requires a high-fidelity simulation environment that accurately reproduces the experimental conditions, including detector geometry, material budget, and physical interactions. To ensure an unbiased benchmark against the standard CA track finder, the GNN-based track finder was developed and evaluated entirely within the official CBM software framework, CBMRoot [137].

6.2.1 The CBMRoot Environment

CBMRoot is built upon the FairRoot [138, 139] architecture and the ROOT [140] ecosystem. It serves as the backbone for all simulation, reconstruction, and analysis tasks within the CBM collaboration. This integration ensures that the GNN-based track finder encounters the exact same realistic challenges as the CA track finder, specifically:

- **Detailed Geometry and Material Budget:** The simulation utilizes the precise TGeo-based geometry of the STS and MVD detectors, including passive materials such as the beam pipe. This allows for an accurate estimation of multiple Coulomb scattering and energy loss, which are critical factors in low-momentum tracking.

- **Magnetic Field Map:** A realistic map of the superconducting dipole magnet is used, ensuring that trajectory curvature and non-linear propagation effects are correctly modelled.

- **Virtual Monte Carlo (VMC):** The framework abstracts the transport engine, allowing the use of validated transport codes (e.g., GEANT4) without altering the user code.

6.2.2 Simulation Pipeline

The simulation chain proceeds in three distinct stages, transforming physics events into the detector hits used as input for the tracking neural networks:

Event Generation

Primary collision events are generated using the Ultra-Relativistic Quantum Molecular Dynamics (UrQMD) model [141, 142]. UrQMD is a microscopic transport model that describes relativistic hadron-hadron and heavy-ion collisions. It accurately predicts the high multiplicity of particles and the abundance of resonance decays typical of Au+Au collisions at CBM energies, providing a realistic occupancy background for tracking.

Particle Transport

The propagation of particles through the detector setup is handled by the GEANT4 toolkit [143, 144]. GEANT4 simulates the interaction of particles with the active and passive detector materials. It generates Monte Carlo Points (MC Points), which record the precise position, time, and energy loss wherever a particle traverses a sensitive detector volume.

Digitization and Hit Reconstruction

To mimic the hardware response, the MC Points undergo a digitization process. This step introduces realistic detector effects, including:

1. **Channel Discretization:** Continuous energy deposition is mapped onto discrete strips or pixels.

2. **Noise and Efficiency:** Electronic noise is added, and detection efficiency is applied to simulate dead channels or inefficiencies.

3. **Clustering:** Adjacent firing strips are grouped into clusters to form *hits*.

The resulting hits possess spatial uncertainties corresponding to the detector resolution. These hits serve as the sole input for the track finding algorithms, ensuring the GNN operates on data extremely similar to what will be observed in the real experiment.

6.2.3 Integration of Deep Learning

A key technical achievement of this work is the integration of neural network inference into the C++ based `CBMRoot` environment. This was facilitated by the `ANN4FLES` package (see Sec. 4.7). This direct integration allows the GNN-based tracker to be executed as a standard module in the reconstruction chain, enabling direct, event-by-event comparisons of memory usage, execution time, and physics performance against the CA track finder.

6.3 GNN-based Track Finder Performance

The performance of the GNN-based algorithm is compared against the CA Track Finder for Au+Au collisions at 10 AGeV. Both central (impact parameter $b = 0$ fm) and minimum-bias collisions are analyzed. While central collisions represent the most challenging track finding environment, minimum-bias events reflect realistic experimental conditions.

The algorithm was further tested at three more collision systems: Au+Au at 4 and 20 AGeV, and p+Au at 28.3 AGeV. The results for these systems are presented in Appendix A.

6.3.1 10 *A*GeV Au+Au

Tables 6.1a and 6.1b summarize the reconstruction efficiencies for central and minimum-bias Au+Au collisions at 10 AGeV. The collision centrality significantly impacts detector occupancy: central collisions produce an average of 5100 hits across the 12 tracking stations, whereas minimum-bias events average 1200 hits. Furthermore, the environment in central collisions is more cluttered, with fake hits constituting nearly 50% of the data compared to 30% in minimum-bias events (Fig. 5.8). The number of fake hits scales super-linearly with the total hit multiplicity.

The GNN-based track finder achieves consistently higher efficiencies than the CA-based algorithm across all categories. The improvement is most pronounced for low-momentum secondaries, which represent the most challenging track class. In central collisions, the efficiency for this group increases by approximately 6% (from 78.6% to 84.3%), while in minimum-bias collisions, the gain exceeds 7% (from 80.2% to 87.3%). Primary low-p tracks also show improvements of nearly 1% in both scenarios. High-p tracks, where efficiencies are already near saturation, exhibit modest but consistent gains. Figures 6.4a–6.5b illustrate the dependence of reconstruction efficiency on particle momentum. The GNN outperforms the CA in the low-momentum ($p < 1\,\mathrm{GeV/c}$) region and has comparable performance at high momenta.

Crucially, these efficiency gains do not compromise reconstruction purity. In central collisions, both algorithms yield a clone rate of 5.1%. In minimum-bias collisions, the GNN slightly reduces the clone rate (4.7% vs. 5.2% for the CA). The ghost rate remains stable, staying at or below 3%. The hit efficiency for central (minimum-bias) collisions is 86.7% (87.1%) for the GNN-based approach, compared to 83.8% (87.9%) for the CA track finder. Similarly, the hit purity for central (minimum-bias) collisions is 97.9% (99.1%) for the GNN, compared to 98.1% (99.2%) for the CA.

The polar and azimuthal angle dependence of the reconstruction efficiency for central collisions is shown in Fig. 6.2b. For both algorithms, the efficiency drops below 5° due to the beam pipe, which extends to 2.5° (see Sec. 2.3.3). The reconstruction efficiency as a function of track length is presented in Fig. 6.3 for all tracks and secondary tracks. The GNN-based track finder is significantly more effective at reconstructing short tracks (4 and 5 hits), demonstrating a nearly 10% improvement in efficiency. Notably, the GNN-based algorithm excels

in reconstructing geometrically challenging tracks, including shorter trajectories, those passing close to the beam pipe, and those near the acceptance boundaries of the detector.

Track quality is further evaluated through residual and pull distributions of the track parameters at the first (Fig. 6.6) and last hit (Fig. 6.7) positions of each reconstructed track. The track parameters are: track positions x and y, track slopes t_x and t_y in the XZ and YZ planes respectively, and charge over momentum q/p.

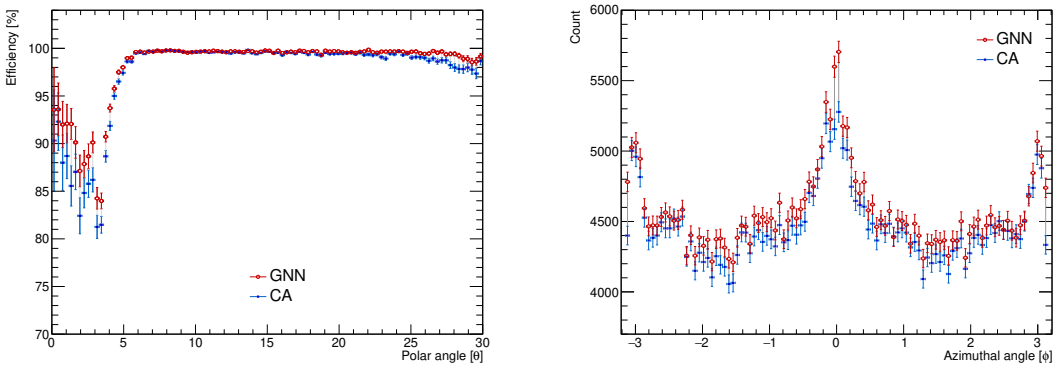
The distribution means for all parameters are very close to zero, indicating unbiased reconstruction. The residual distributions of the track positions x and y should reflect the intrinsic detector spatial resolution. The first hits of most tracks lie on a station of the MVD detector which has a spatial resolution of $5\,\mu m$ in both directions. The residual fit widths for x and y positions of the tracks are $4.86\,\mu m$ and $5.46\,\mu m$ respectively, which matches expectation. The last hits of most tracks lie in the STS detector which consists of tilted strips giving a much better resolution along the x-direction. This is reflected in residual widths which are $7.31\,\mu m$ in the x- and $65.17\,\mu m$ in the y-direction. The peak in the y-distribution of the last hit is due to tracks whose last hit is in the MVD detector. The width of the momentum residuals is 1.5% which is close to the requirement of CBM.

All the pull widths are close to unity, confirming correctness of the fitting procedure. The t_x and t_y distributions indicate accurate modelling of multiple scattering. The q/p pull distribution is the widest because momentum is sensitive to the approximations of the material used in multiple scattering, energy loss and the polynomial fits of the inhomogeneous magnetic field. Similar results were obtained for minimum bias collisions and for the configurations presented in Appendix A indicating robustness of the fitting procedure and quality of the tracks reconstructed.

The computational performance was benchmarked on a single core of an Intel(R) Xeon(R) Gold 6130 CPU with 2.1 GHz clock speed. The GNN-based track finder is approximately 10 times slower than the CA track finder. Three factors contribute to this latency: (i) neural network inference is suboptimal on single-core CPUs, whereas modern deep learning applications typically leverage GPUs; (ii) the CA track finder has benefited from two decades of optimization specific to the CBM experiment, whereas the GNN approach has been developed over three years, leaving significant room for technical optimization; and (iii) the

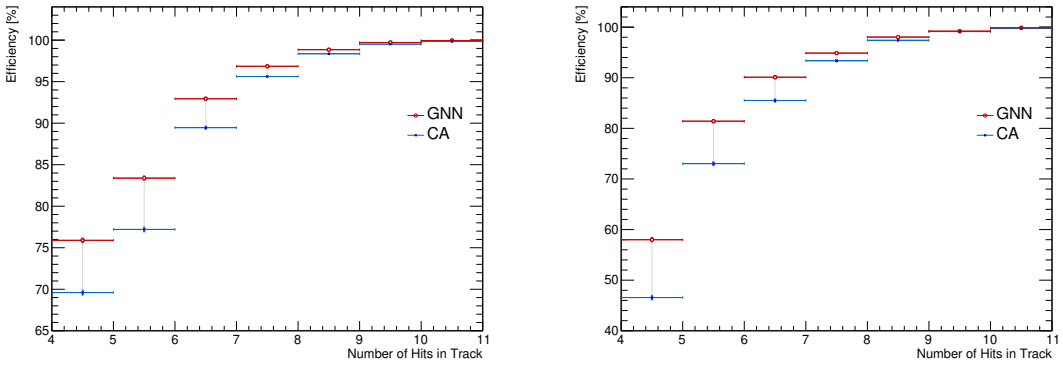
GNN-based finder considers a larger number of candidate triplets to maximize the reconstruction efficiency but this carries a higher computational cost. This tradeoff can be optimized based on practical considerations.

Optimizing the algorithm's implementation is expected to narrow the performance gap on CPUs, and a GPU implementation may potentially bring the GNN-based algorithm on par with the CA. The following section discusses a first GPU implementation of the GNN-based algorithm.



(a) Primary tracks **(b)** All tracks

Figure 6.2: Reconstruction efficiency comparison by (a) polar angle (θ) and (b) azimuthal angle (ϕ) for central Au+Au collision at 10 AGeV.



(a) All tracks **(b)** Secondary tracks

Figure 6.3: Reconstruction efficiency comparison for different track lengths for central Au+Au collision at 10 AGeV.

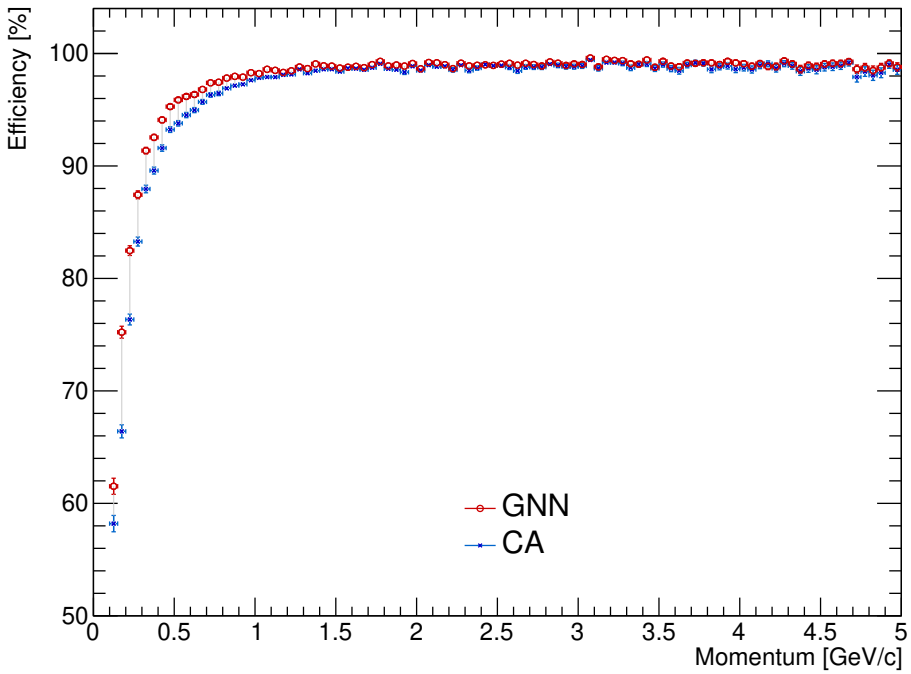
Track Category	CA (%)	GNN (%)	Change (%)	No. of MC Tracks
All tracks	95.7	**96.9**	+1.2	412
Primary high-p	99.1	**99.2**	+0.1	232
Primary low-p	97.3	**98.2**	+0.9	97
Secondary high-p	95.5	**96.7**	+1.2	29
Secondary low-p	78.6	**84.3**	+5.7	53
Clone Rate	**5.1**	**5.1**	0.0	–
Ghost Rate	**2.2**	2.9	-0.7	–
MC tracks found	394	**399**	+5(+1.2%)	412
Time, s/ev	**0.5**	4.8	–	–

(a) Central collisions

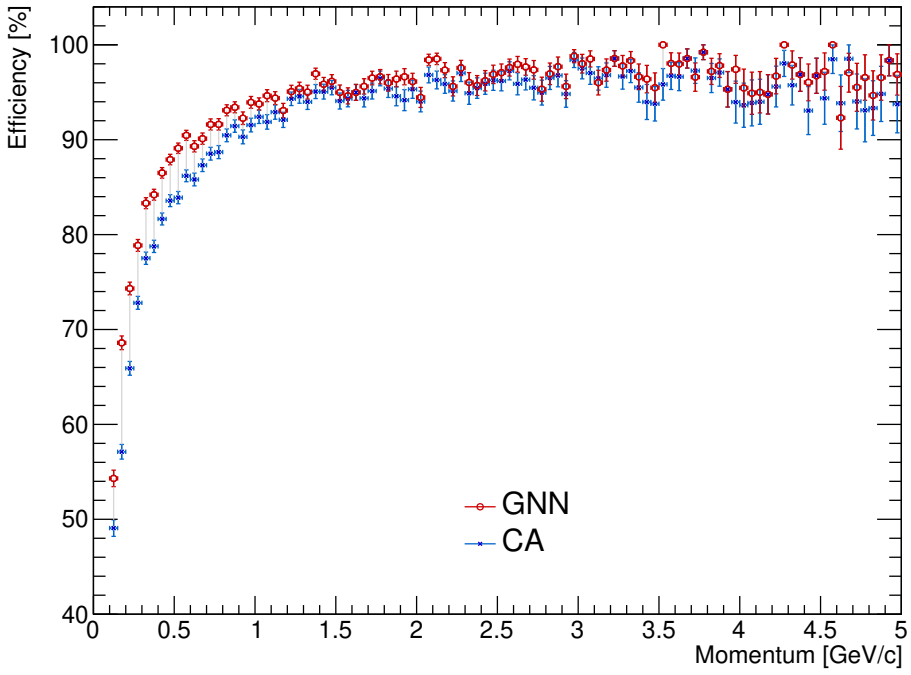
Track Category	CA (%)	GNN (%)	Change (%)	No. of MC Tracks
All tracks	95.6	**97.0**	+1.4	93
Primary high-p	99.1	**99.2**	+0.1	52
Primary low-p	95.9	**97.2**	+1.3	23
Secondary high-p	96.0	**97.5**	+1.5	6
Secondary low-p	80.2	**87.3**	+7.1	12
Clone Rate	5.2	**4.7**	+0.5	–
Ghost Rate	**0.9**	1.1	-0.2	–
MC tracks found	89	**90**	+1(+1.4%)	93
Time, s/ev	**0.06**	0.5	–	–

(b) Minimum-bias collisions

Table 6.1: Comparison of CA and GNN efficiencies across track categories for 10 AGeV Au+Au collisions. The last column shows the average maximum number of reconstructable tracks in an event. Results averaged over 1000 events.

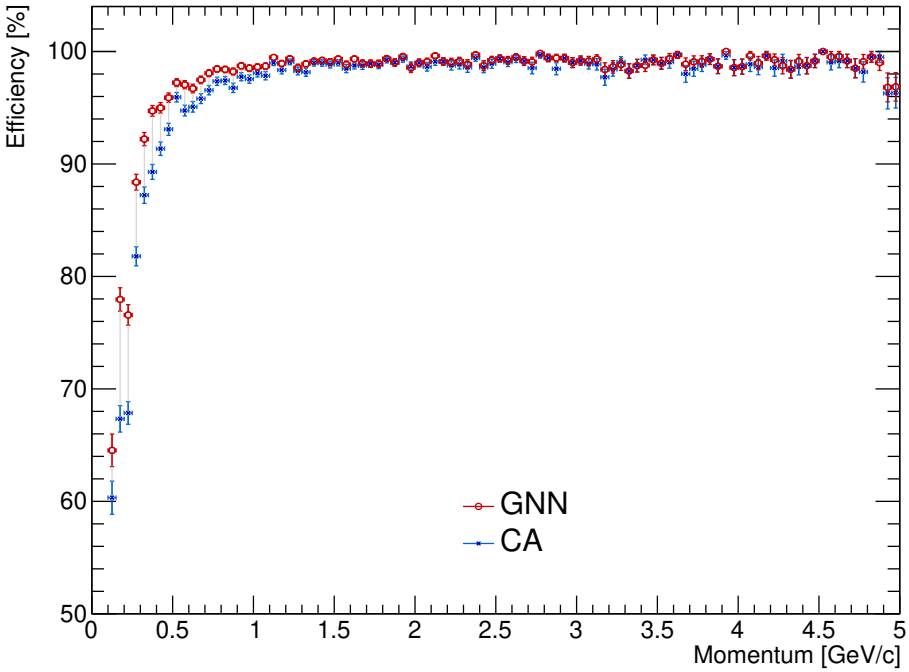


(a) All tracks

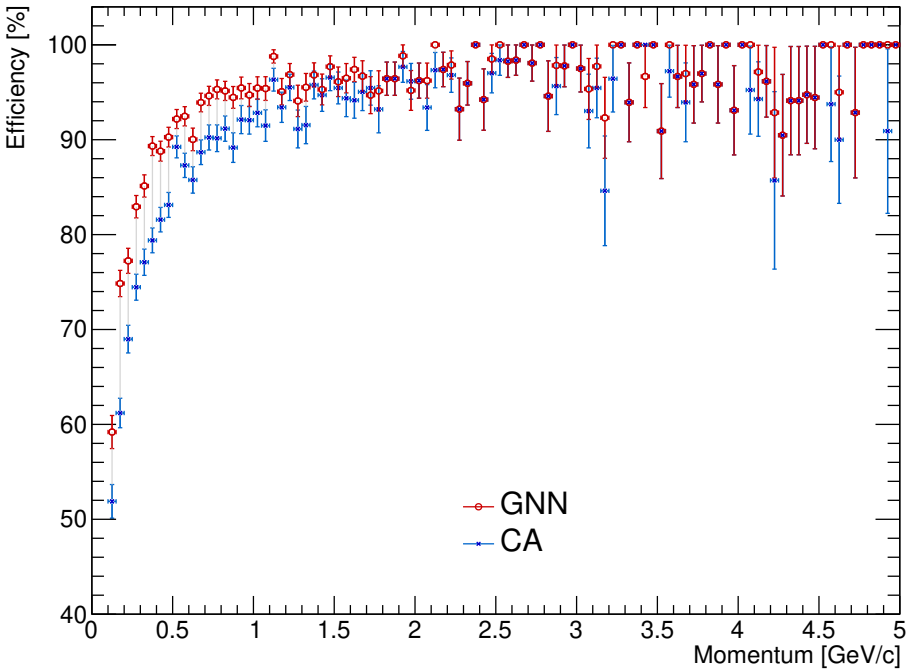


(b) Secondary tracks

Figure 6.4: Central GNN vs CA reconstruction efficiency comparison by momentum for central Au+Au collision at 10 AGeV. Results for (a) all tracks and (b) secondary tracks.



(a) All tracks



(b) Secondary tracks

Figure 6.5: Minimum bias GNN vs CA reconstruction efficiency comparison by momentum for Au+Au minimum bias collisions at 10 AGeV. Results for (a) all tracks and (b) secondary tracks.

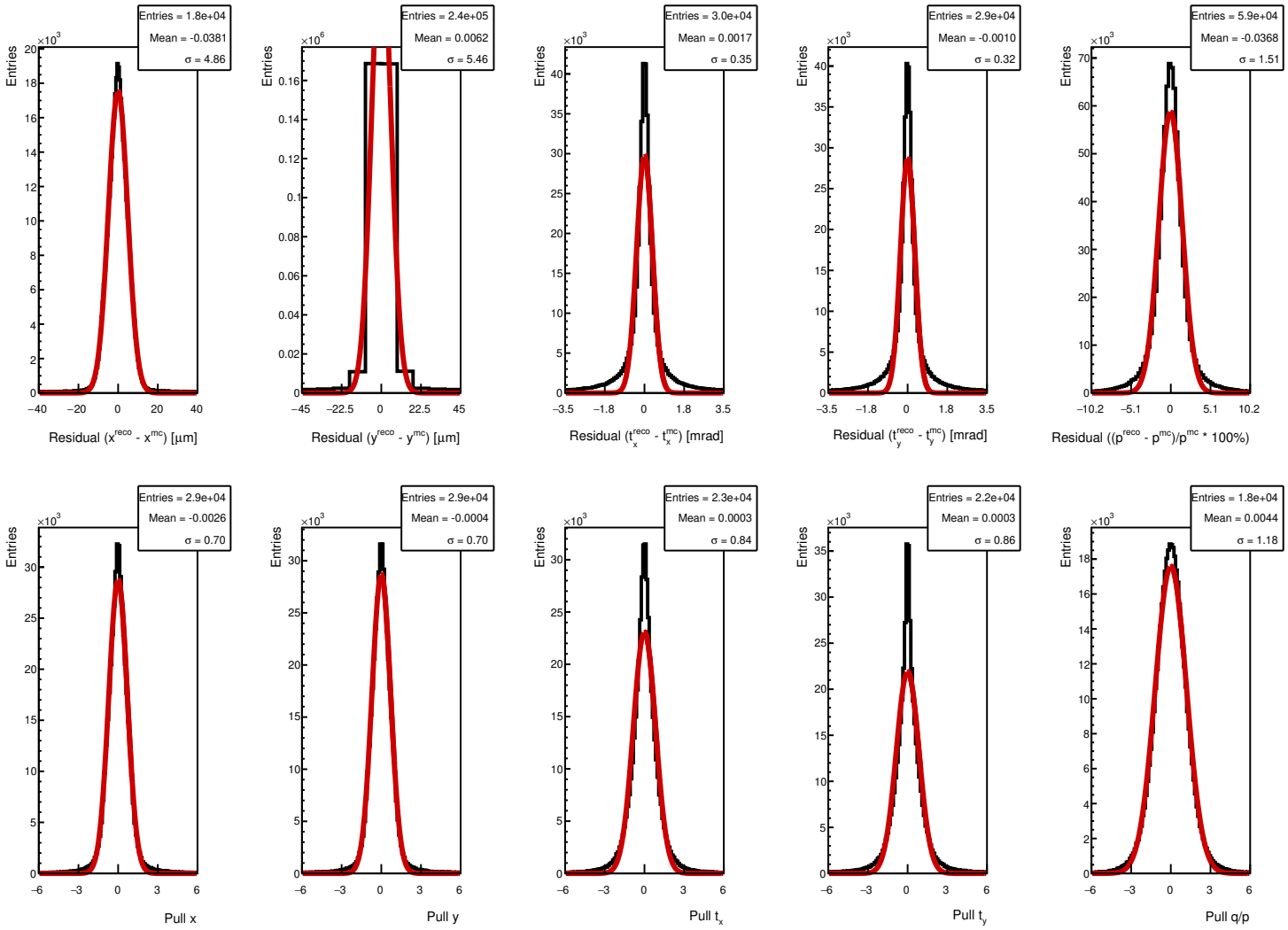


Figure 6.6: First Hit Residual and pull distributions, together with Gaussian fits, for tracks reconstructed by the GNN-based track finder at the first hit position. The first hit of the track is mostly on the MVD detector with spatial resolution of 5 μm. 10 AGeV Au+Au central collisions.

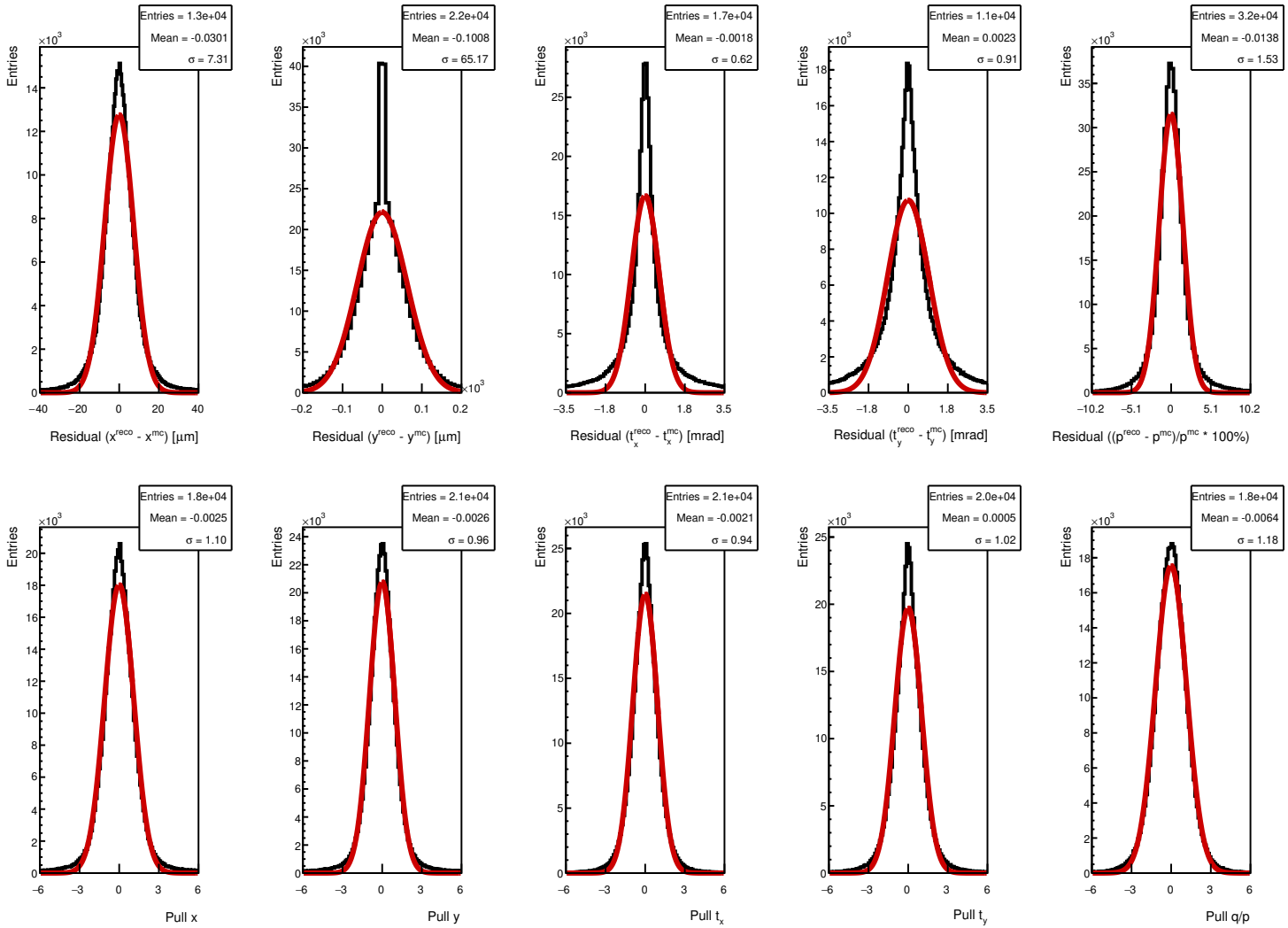


Figure 6.7: Last Hit Residual and pull distributions, together with Gaussian fits, for tracks reconstructed by the GNN-based track finder at the last hit position. The last hit of the track is mostly on the strip-based STS detector. 10 AGeV Au+Au central collisions.

6.4 Track Finder Performance on GPU

To address the high computational cost of the GNN-based approach, a preliminary implementation of the tracking algorithm was developed for GPUs using the `xpu` library (refer to Sec. 3.3 for description). While CPUs are optimized for low-latency serial processing, GPUs are designed for high-throughput parallel computation. Modern GPU architectures employ a Single Instruction, Multiple Thread (SIMT) model, allowing thousands of threads to execute concurrently (see Sec. 3.2). This architecture is particularly advantageous for deep learning applications which rely heavily on dense matrix multiplications and tensor operations that can be effectively parallelized.

Porting legacy reconstruction algorithms from a serial CPU context to a highly parallel GPU environment is a complex engineering challenge. It requires rethinking memory management, thread divergence, and data structures. Consequently, the results presented in this section serve as a *proof of concept*, demonstrating the functional feasibility of running the GNN-based tracker on a GPU. A fully optimized, production-ready implementation remains a subject for future development.

Hardware and Execution Context

The algorithm was benchmarked on an AMD Instinct MI100 accelerator. This GPU is based on the CDNA architecture, featuring 120 Compute Units and high-bandwidth HBM2 memory, designed specifically for high-performance computing (HPC) and machine learning workloads.

The current CBM reconstruction pipeline operates on an event-by-event basis. However, the data volume of a single CBM event (even a central collision) is insufficient to saturate the thousands of cores available on modern GPUs. Consequently, the GPU remains underutilized (low occupancy). Furthermore, in a discrete GPU setup, data must be copied from the host (CPU) memory to the device (GPU) memory via the PCIe bus. In an event-by-event processing model, this high-latency data transfer dominates the total execution time, masking the computational speedup provided by the GPU.

To mitigate this, standard HPC practices involve *batching*: grouping multiple events together to amortize the transfer cost and maximize GPU occupancy. Ideally, performance should be measured in *throughput* (events per second) rather

than latency (time per event). However, as the current framework does not yet support multi-event batching, and the eventual goal is to move the full reconstruction chain to the GPU (eliminating the transfer penalty), the kernel execution time is measured. This metric isolates the time the GPU spends performing actual calculations, excluding memory transfer overhead. The average kernel execution times for the GNN-based tracker are ≈ 30 ms per event for minimum-bias collisions (see Table. 6.2) and ≈ 120 ms per event for central collisions.

Track Category	GNN (GPU) (%)	No. of MC Tracks
All tracks	96.0	93
Primary high-p	99.0	52
Primary low-p	96.7	23
Secondary high-p	95.2	6
Secondary low-p	82.5	12
Clone Rate	6.0	–
Ghost Rate	1.5	–
MC tracks found	89	93
Kernel Time, ms/ev	30	–

Table 6.2: GNN-based track finder efficiencies on GPU across track categories for 10 AGeV minimum bias Au+Au collisions. Results averaged over 1000 events.

Ideally, the physics performance (efficiency and purity) of the GPU implementation should be identical to that of the CPU (Table. 6.1b). However, a slight discrepancy in reconstruction efficiency was observed. This deviation is attributed to the Kalman Filter (KF) triplet fitting stage, specifically when processing jump triplets (hits separated by more than one station) and tracks with high curvature. These instabilities can result in `NaN` (Not a Number) floating-point values during the fit, causing valid triplets to be discarded. This is likely due to differences in floating-point precision handling or intrinsic math library implementations between the host and the device. While a comprehensive debugging of the KF numerical stability on the GPU was outside the scope of this thesis, resolving these precision artifacts is a clearly defined objective for future optimization.

6.5 Yields of Short-Lived Particles

The increase in reconstruction efficiency of low-momentum secondary particles is particularly important because it is expected to enhance the reconstruction efficiency of short-lived particles. Short-lived particles are unstable states that either decay before reaching the first tracking station or do not leave a sufficient number of hits in the tracking system for a direct track reconstruction. Short-lived particles important for the CBM include strange hyperons (Λ, Ω, Ξ), charmed particles (D mesons, J/Ψ), low mass vector mesons (ρ, ϕ, ω) and hyper-nuclei ($^{3}_{\Lambda}$H, $^{4}_{\Lambda}$H, $^{4}_{\Lambda}$He).

Track Category	CA (%)	GNN (%)	Change (%)
K^0_S	82.8	**86.6**	+3.8
Λ	81.1	**84.5**	+3.4

Table 6.3: Comparison of all decay product reconstruction efficiency with CA and GNN-based track finder algorithm in Au+Au central collisions at 10 AGeV. Efficiencies are averaged over 1000 simulated events.

Short-lived particles can only be reconstructed indirectly via their decay products. The Kalman Filter Particle (KF Particle) package [133] has been developed for the complete reconstruction of short-lived particles, providing their energy, momentum, mass, and decay topology. KF Particle requires all decay products (daughters) of a given decay channel to be reconstructed in order to reconstruct the corresponding short-lived particle (mother). Therefore, an increase in the efficiency for the simultaneous reconstruction of all daughter tracks directly translates into an increased reconstruction efficiency for the short-lived particle.

This effect was quantified for two benchmark species, K^0_S and Λ. Table 6.3 shows the fraction of short-lived particles for which all decay products are reconstructed by the two tracking algorithms. Using the GNN-based track finder increases the yield of K^0_S and Λ by 3.8% and 3.4%, respectively.

Summary

The performance of the GNN-based track finder is reported for central and minimum-bias collisions at 10 AGeV and compared with the existing CA-based

track finder. The GNN-based algorithm achieves better overall reconstruction efficiency while maintaining similar clone and ghost rates relative to the CA-based track finder. A substantial improvement in the reconstruction of low-momentum secondary tracks is demonstrated. The correctness of the estimated track parameters is validated through fits to the residual and pull distributions. A proof-of-concept GPU implementation is demonstrated, indicating the potential for efficient execution on accelerator hardware. Finally, it is shown that the improved tracking performance translates into increased yields of indirectly reconstructed short-lived particles, underscoring the physics impact of the GNN-based track finder.

Summary

The Compressed Baryonic Matter (CBM) experiment at the future Facility for Antiproton and Ion Research (FAIR) in Darmstadt, Germany, is designed to study strongly interacting matter under extreme conditions. CBM will explore the high baryon density and moderate temperature region of the QCD phase diagram, where theoretical models predict a first-order phase transition and a critical endpoint between confined hadronic matter and deconfined quark-gluon matter. To investigate these phenomena, CBM will utilize fixed-target heavy-ion collisions to create matter with densities exceeding that of normal nuclear matter. Key observables include dilepton production, charm production, multi-strange particle production, and global observables such as collective flow and event-by-event fluctuations.

To accumulate sufficient statistics for these rare observables, the CBM experiment must operate at unprecedented interaction rates of up to 10^7 collisions/s (10 MHz). This generates raw data volumes on the order of terabytes per second, making full archival of raw data infeasible. Furthermore, the complex topological signatures of the most interesting physics channels, such as multi-strange hyperons and charmed hadrons, preclude the use of simple hardware triggers. Consequently, CBM employs a continuous, free-streaming readout architecture combined with real-time event reconstruction (software trigger) to select interesting events for storage. To achieve the required throughput, track reconstruction must be highly efficient.

The main tracking system of CBM consists of the pixel-based Micro-Vertex Detector (MVD) and the strip-based Silicon Tracking System (STS) which are placed inside a superconducting dipole magnet. The thousands of charged particles produced in collisions create high hit densities which makes track finding challenging due to the large combinatorial background. Over the past two decades, a highly optimized Cellular Automaton (CA)-based track finder has been developed

for the CBM experiment. While the CA-based track finder is fast and robust, its secondary track reconstruction efficiency is low. This is partially due to the use of the primary vertex even in the reconstruction of secondary tracks.

This thesis introduces a novel Graph Neural Network (GNN)-based track finding algorithm for the CBM experiment. Unlike recent end-to-end deep learning approaches, the algorithm developed in this work adopts a hybrid strategy: it integrates the representational power of data-driven deep learning with the physical robustness of Kalman Filter (KF)-based parameter estimation. This approach reduces reliance on neural networks where geometric heuristics suffice, ensuring both accuracy and interpretability.

The algorithm executes in three iterations, targeting: (1) high-momentum primary tracks, (2) all primary tracks, and (3) secondary tracks. In each iteration, hits are grouped into doublets using a k-Nearest Neighbour (k-NN) search within a learned latent space where hits belonging to the same track are clustered. Doublets are filtered using angle cuts relative to the primary vertex in the first two iterations, while a GNN is employed for edge classification in the third iteration to recover secondaries. Doublets sharing a middle hit are combined into triplets, which allows an initial track parameter estimate with the KF. Kinematically compatible overlapping pairs of triplets are then linked into track candidates. Finally, a novel cooperative track selection scheme, which allows the exchange of strip ownership among candidates, is employed to maximize the number of reconstructed tracks.

The GNN-based track finder was integrated into `CBMRoot`, the official software framework of the CBM experiment. This integration, facilitated by the `ANN4FLES` package, allows the algorithm to be executed as a standard module in the reconstruction chain. It enables direct comparison of memory usage, execution time, and physics performance against the CA-based track finder under realistic experimental conditions by including detailed detector geometry, material budgets, magnetic field maps, and detector response.

Table 6.4 summarizes the reconstruction efficiencies for central Au+Au collisions at 10 AGeV. The GNN-based track finder achieves consistently higher efficiencies than the CA algorithm across all track categories. The improvement is most pronounced for low-momentum secondaries ($p < 1\,\mathrm{GeV}/c$) where efficiency increases by approximately 5.7% (from 78.6% to 84.3%) in central collisions and 7.1% (from 80.2% to 87.3%) in minimum-bias collisions.

Importantly, these efficiency gains do not compromise track quality. In central

Track Category	CA (%)	GNN (%)	Change (%)	No. of MC Tracks
All tracks	95.7	**96.9**	+1.2	412
Primary high-p	99.1	**99.2**	+0.1	232
Primary low-p	97.3	**98.2**	+0.9	97
Secondary high-p	95.5	**96.7**	+1.2	29
Secondary low-p	78.6	**84.3**	+5.7	53
Clone Rate	**5.1**	**5.1**	0.0	–
Ghost Rate	**2.2**	2.9	-0.7	–
MC tracks found	394	**399**	+5(+1.2%)	412
Time, s/ev	**0.5**	4.8	–	–

Table 6.4: Comparison of CA and GNN efficiencies across track categories for 10 AGeV Au+Au collisions.

collisions, both algorithms show similar clone and ghost rates. The hit efficiency for central collisions is 86.7% for the GNN-based approach, compared to 83.8% for the CA track finder. Similarly, the hit purity for central collisions is 97.9% for the GNN-based track finder, compared to 98.1% for the CA-based track finder. The GNN-based track finder is significantly more effective at reconstructing short tracks (4 and 5 hits), demonstrating a nearly 10% improvement in efficiency. Notably, the GNN-based algorithm excels in reconstructing challenging tracks, such as those passing close to the beam pipe and those near the acceptance boundaries of the detector.

The track quality of the GNN-based track finder is evaluated using gaussian fits to the residual and pull distributions of the track parameters at the first and last hit positions of each reconstructed track. The distribution means for all parameters are very close to zero, indicating unbiased reconstruction. The width of the residual distributions of track positions x and y correctly reflect the intrinsic detector spatial resolution. The width of the momentum residuals is 1.5% which is close to the requirement of CBM. All the pull widths are close to unity, confirming correctness of the fitting procedure.

The enhanced reconstruction efficiency for low-momentum secondary particles directly translates into increased yields of short-lived particles. Such particles typically decay too quickly to be reconstructed directly; they must instead be

reconstructed indirectly from the trajectories of their decay products. To quantify the physics impact of the improved secondary particle reconstruction, the simultaneous reconstruction efficiency of all daughter tracks was evaluated for two benchmark channels, K_S^0 and Λ. Table 6.5 summarizes the fraction of short-lived particles for which all decay products were successfully reconstructed. These results demonstrate that the improved track finding performance propagates directly to higher yields in key physics channels.

Track Category	CA (%)	GNN (%)	Change (%)
K_S^0	82.8	**86.6**	+3.8
Λ	81.1	**84.5**	+3.4

Table 6.5: Comparison of all decay product reconstruction efficiency with CA and GNN-based track finder algorithm in Au+Au central collisions at 10 AGeV. Results averaged over 1000 events.

The current CPU implementation of the GNN-based track finder is approximately an order of magnitude slower than the highly optimized CA-based algorithm. This latency arises primarily from the relatively poor CPU performance of neural network inference combined with the algorithm's intentionally broader combinatorial exploration, which is required to achieve higher efficiency. However, deep learning workloads are naturally suited to parallel hardware. To investigate the expected performance on accelerators, a proof-of-concept GPU implementation was developed using the `xpu` library, a backend agnostic abstraction layer enabling portability across GPU and other accelerator architectures. Although event-by-event processing incurs significant memory-transfer overheads that limit device occupancy, standalone kernel benchmarks show promising throughput: average execution times of approximately 30 ms per minimum-bias event and 120 ms for central Au+Au events. These measurements indicate that a fully batched, GPU reconstruction workflow could achieve substantial speedups relative to both the CPU implementation and, potentially, even the CA-based track finder.

The generalization capability of the model was assessed across several collision systems: 4 and 20 AGeV Au+Au, as well as 28.3 AGeV p+Au. These span the full range of event complexities anticipated in CBM. Despite being trained exclusively on 10 AGeV central Au+Au collisions, the GNN-based track finder consistently

outperformed the CA-based track finder in all tested systems. This robustness indicates that the network has learned the underlying geometrical and topological properties of particle trajectories in the CBM detector rather than overfitting to a specific collision environment. Further performance gains are expected from targeted hyperparameter tuning for individual collision systems.

A hybrid reconstruction strategy represents another promising application of the developed algorithm. The CA-based track finder, which uses the primary vertex as an initial constraint for KF extrapolation, performs well for primary tracks but is inherently limited for secondaries, whose trajectories do not originate from the primary vertex. In contrast, the GNN-based method, trained directly on data, learns representations of secondary track patterns without relying on vertex constraints. An iterative reconstruction chain in which the CA finder is used for the initial primary track iterations, followed by the GNN-based track finder for secondary-track recovery, could combine the strengths of both approaches.

Even if the GNN-based track finder ultimately does not meet the stringent real-time requirements of CBM, it remains highly valuable as a benchmark tool for algorithm validation and as an analysis-level reconstruction method, where timing constraints are relaxed and even modest efficiency improvements can have significant physics impact.

Conclusion

This thesis introduces a novel GNN-based track finding algorithm that achieves markedly improved reconstruction efficiency, most notably for low-momentum secondary tracks, relative to the standard CA-based track finder. These improvements translate into enhanced yields for short-lived particles, benefiting a broad range of physics analyses. A proof-of-concept GPU implementation demonstrates promising performance characteristics and suggests that the approach is well suited for accelerator-based deployment. Overall, the developed algorithm emerges as a strong candidate for future integration into the CBM reconstruction framework and as a generally applicable algorithm for track finding in high-rate, high-multiplicity experiments.

Appendix A

Performance for different collision systems

This appendix presents supplementary performance results for the Graph Neural Network (GNN)-based track finder evaluated across three collision systems: 4 and 20 AGeV Au+Au, and 28.3 AGeV p+Au collisions. The performance is compared with that of the baseline Cellular Automaton (CA)-based track finder. The selected systems span the range of event complexities expected in the CBM experiment. It should be noted that all the neural networks used in the GNN-based track finder were trained on data from central Au+Au collisions at 10 AGeV. Consequently, the results below demonstrate the ability of the network to generalize to unseen conditions. The hyperparameters were also optimized for central Au+Au collisions at 10 AGeV. Although tuning of the hyperparameters for specific collision systems is expected to improve performance, the present configuration already achieves significant improvements over the baseline.

A.1 4 *A*GeV Au+Au

Collisions at 4 *A*GeV lie near the lower end of the CBM energy range. At this energy, the charged-particle multiplicity is significantly reduced, with an average of 188 reconstructible tracks produced in central Au+Au collisions, compared to 413 tracks at 10 *A*GeV. This lower track density provides a useful test case for evaluating the generalization of the track finding algorithms in low-occupancy environments.

Tables A.1a and A.1b summarize the reconstruction efficiencies obtained for both central and minimum-bias collisions. Across all track categories, the GNN-based track finder achieves consistently higher efficiencies than the CA baseline. The largest improvement is observed for low-momentum secondary tracks: in central collisions the efficiency increases by 9.2% (from 74.7% to 83.9%), while for minimum-bias events it rises by 18.1% (from 63.3% to 81.4%). Even for secondary high-p tracks in minimum-bias events, an improvement of 3.5% is observed.

The clone and ghost rates remain within acceptable limits and are comparable to those from the CA-based algorithm. The GNN-based method exhibits slightly reduced clone rates in central (–0.5%) and minimum-bias (–0.3%) events, accompanied by a minor increase in ghost rates of 0.2% and 0.1%, respectively. These results confirm that the efficiency gain achieved by the GNN does not introduce significant contamination in the reconstructed sample.

Figure A.1 shows the reconstruction efficiency as a function of momentum for central collisions. The GNN algorithm exhibits a marked improvement at low momenta, below 0.7 GeV/c for all tracks and up to 1 GeV/c for secondary tracks, while matching the CA performance in the high-momentum region where performace is already saturated. The trend is even more pronounced for minimum-bias collisions, as illustrated in Figure A.2, where low-momentum tracks achieve efficiency gains exceeding 20%.

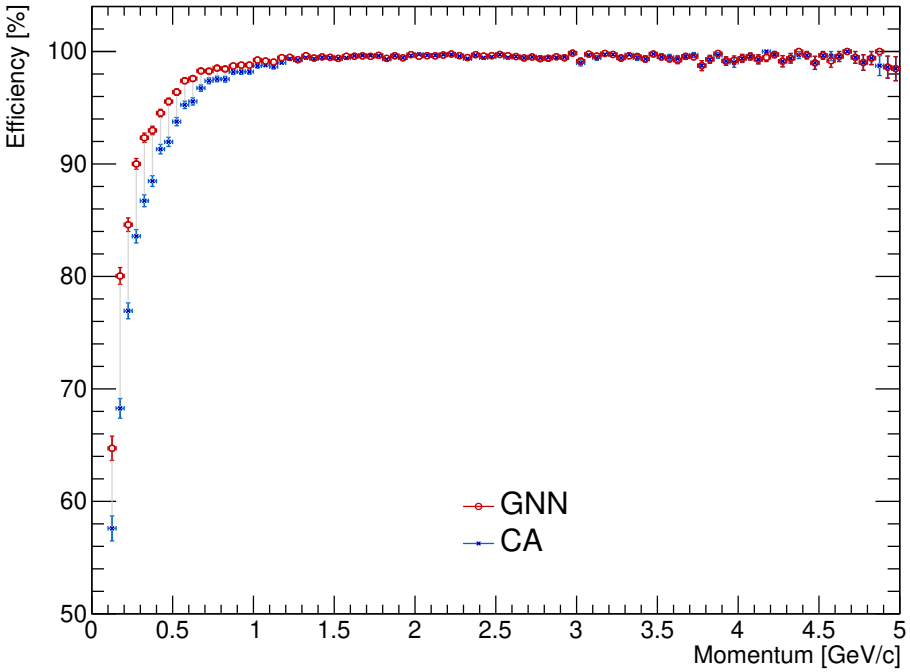
Track Category	CA (%)	GNN (%)	Change (%)	No. of MC Tracks
All tracks	96.5	**97.7**	+1.2	188
Primary high-p	99.6	**99.7**	+0.1	115
Primary low-p	97.3	**98.4**	+1.1	49
Secondary high-p	97.1	**97.8**	+0.7	6
Secondary low-p	74.7	**83.9**	+9.2	18
Clone Rate	4.8	**4.3**	-0.5	–
Ghost Rate	**0.3**	0.5	+0.2	–
MC tracks found	181	**183**	+2	188
Time, s/ev	**0.1**	0.9	–	–

(a) Central collisions

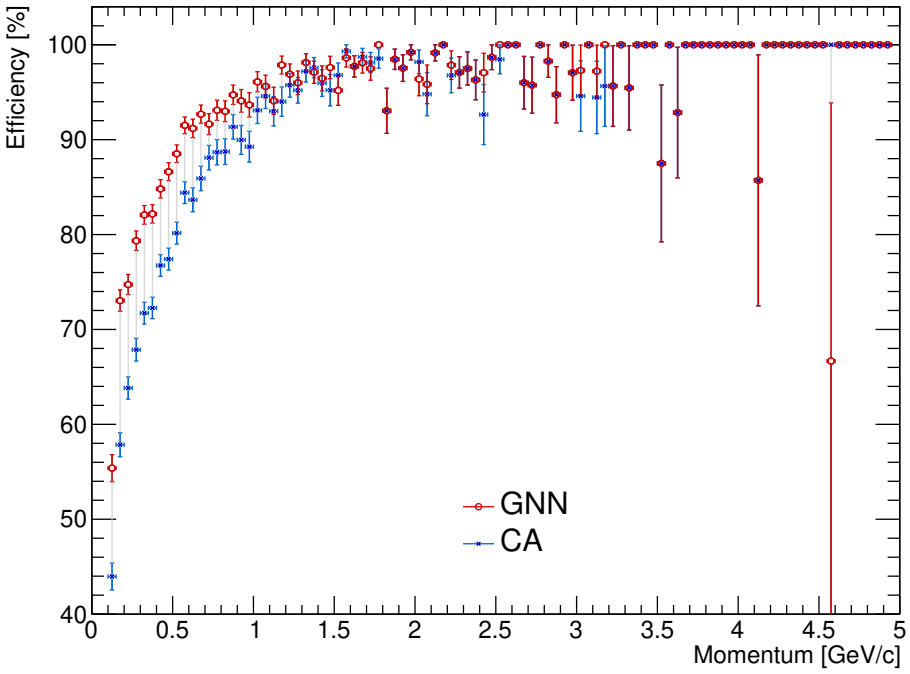
Track Category	CA (%)	GNN (%)	Change (%)	No. of MC Tracks
All tracks	92.5	**95.5**	+3.0	56
Primary high-p	97.8	**97.9**	+0.1	34
Primary low-p	94.4	**96.5**	+2.1	13
Secondary high-p	93.8	**97.3**	+3.5	3
Secondary low-p	63.3	**81.4**	+18.1	7
Clone Rate	4.5	**4.2**	-0.3	–
Ghost Rate	**0.2**	0.3	+0.1	–
MC tracks found	52	**54**	+2	56
Time, s/ev	**0.03**	0.2	–	–

(b) Minimum-bias collisions

Table A.1: Comparison of CA and GNN efficiencies across track categories for 4 AGeV Au+Au collisions. Results averaged over 1000 events.

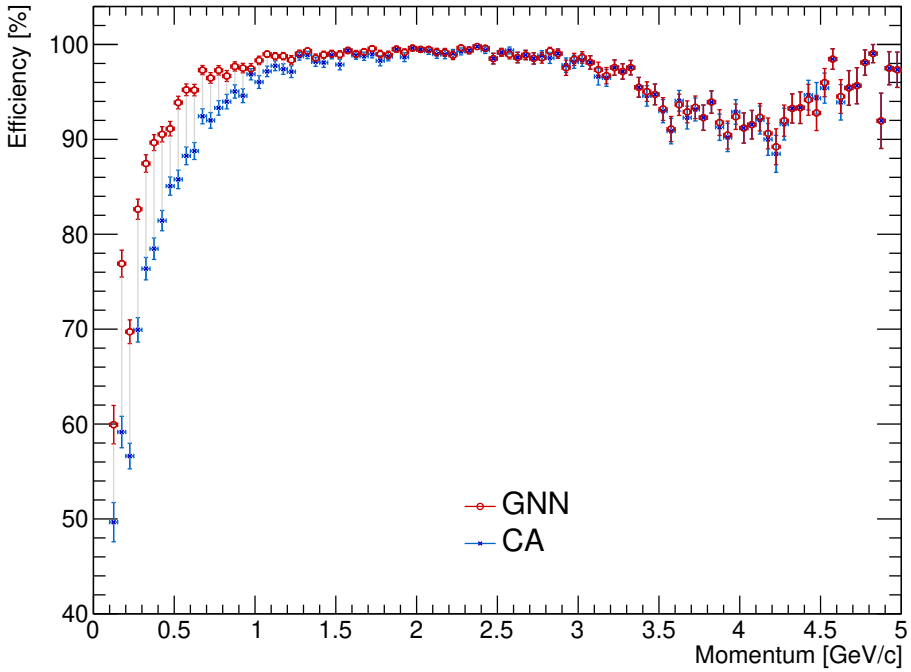


(a) All tracks

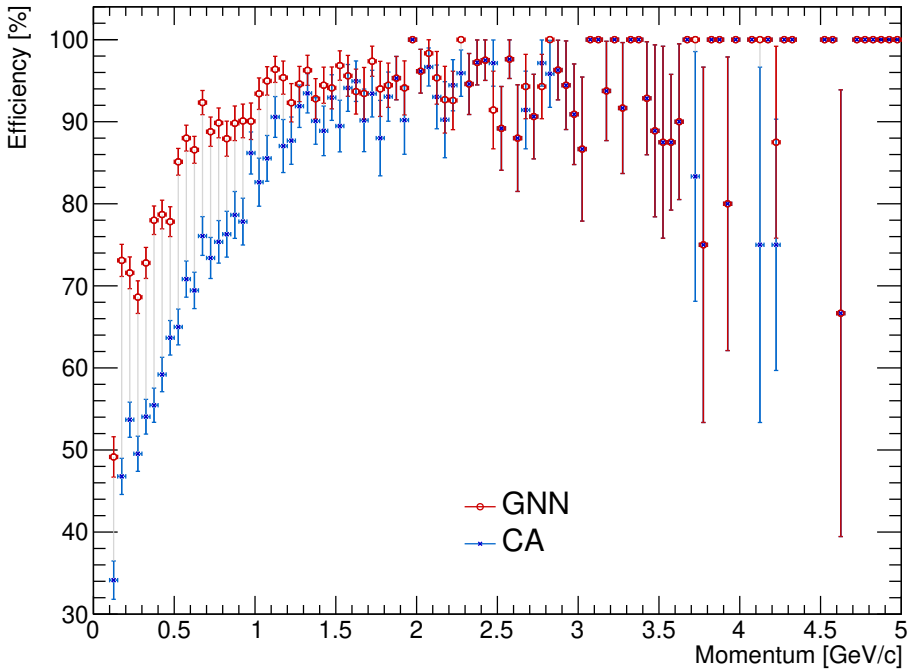


(b) Secondary tracks

Figure A.1: Central GNN vs CA reconstruction efficiency comparison by momentum for Au+Au central collisions at 4 AGeV. Results for (a) all tracks and (b) secondary tracks.



(a) All tracks



(b) Secondary tracks

Figure A.2: Minimum bias GNN vs CA reconstruction efficiency comparison by momentum for Au+Au minimum bias collisions at 4 *A*GeV. Results for (a) all tracks and (b) secondary tracks.

A.2 20 *A*GeV Au+Au

The energy of 20 *A*GeV represents the upper range of Au+Au collision energies investigated by CBM. At this energy, the event multiplicity increases substantially, with approximately 600 reconstructible tracks to be identified among roughly 12,000 detector hits. This high-occupancy environment provides a stringent test of the tracking algorithms under the most challenging CBM conditions.

Tables A.2a and A.2b present the reconstruction results for central and minimum-bias collisions. The GNN-based track finder achieves higher efficiencies than the CA track finder across all track categories, though the improvement is smaller than at lower energies. In central collisions, the overall reconstruction efficiency improves by 0.8%, and by 1.3% in minimum-bias events. Notably, secondary high-p tracks in minimum-bias collisions exhibit a marked efficiency increase of 5.5%. On average, this corresponds to about five additional tracks successfully reconstructed per central collision. Further optimization of the GNN hyperparameters for this energy regime is expected to yield improved performance.

As expected, both clone and ghost rates rise due to higher track densities. The GNN-based method exhibits reduced clone rates of 0.8% in central and 0.6% in minimum-bias collisions relative to the CA baseline. The ghost rate increases by 0.8% in central collisions and remains unchanged for minimum-bias collisions.

Figure A.3 shows the momentum dependence of the reconstruction efficiency for central collisions. The GNN shows enhanced performance at low momenta, below 0.5 GeV/c,while matching the CA efficiency at higher momenta where reconstruction performance is already saturated. These results demonstrate that the GNN algorithm maintains stable reconstruction quality even under high-multiplicity conditions.

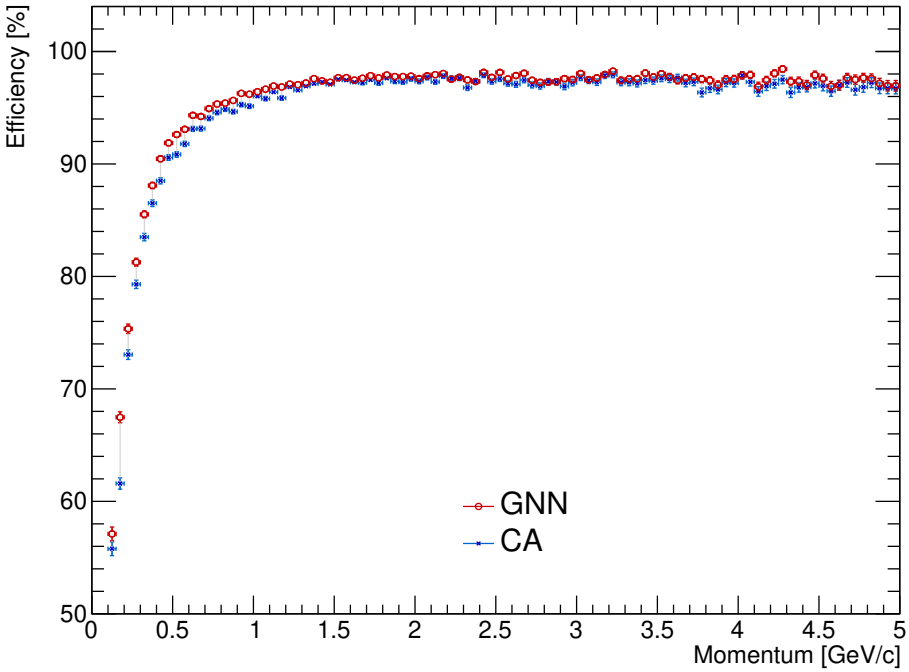
Track Category	CA (%)	GNN (%)	Change (%)	No. of MC Tracks
All tracks	94.0	**94.8**	+0.8	604
Primary high-p	98.1	**98.4**	+0.3	341
Primary low-p	96.7	**97.3**	+0.6	129
Secondary high-p	90.7	**92.2**	+1.5	53
Secondary low-p	74.6	**77.5**	+2.9	81
Clone Rate	6.5	**5.7**	-0.8	–
Ghost Rate	**7.5**	8.3	+0.8	–
MC tracks found	568	**573**	+5	604
Time, s/ev	**1.7**	15	–	–

(a) Central collisions

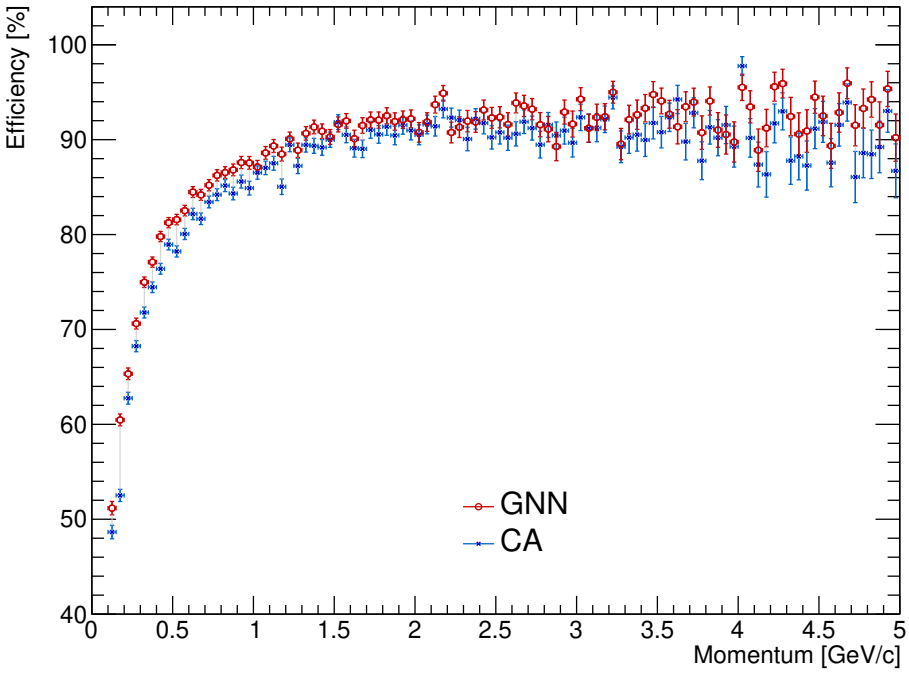
Track Category	CA (%)	GNN (%)	Change (%)	No. of MC Tracks
All tracks	94.9	**96.2**	+1.3	123
Primary high-p	98.5	**98.7**	+0.2	68
Primary low-p	96.3	**97.5**	+1.2	27
Secondary high-p	93.6	**95.3**	+1.7	10
Secondary low-p	79.4	**84.9**	+5.5	17
Clone Rate	5.5	**4.9**	-0.6	–
Ghost Rate	**3.0**	**3.0**	0.0	–
MC tracks found	117	**118**	+1	123
Time, s/ev	**0.16**	1.5	–	–

(b) Minimum-bias collisions

Table A.2: Comparison of CA and GNN efficiencies across track categories for 20 AGeV Au+Au collisions. Results averaged over 1000 events.

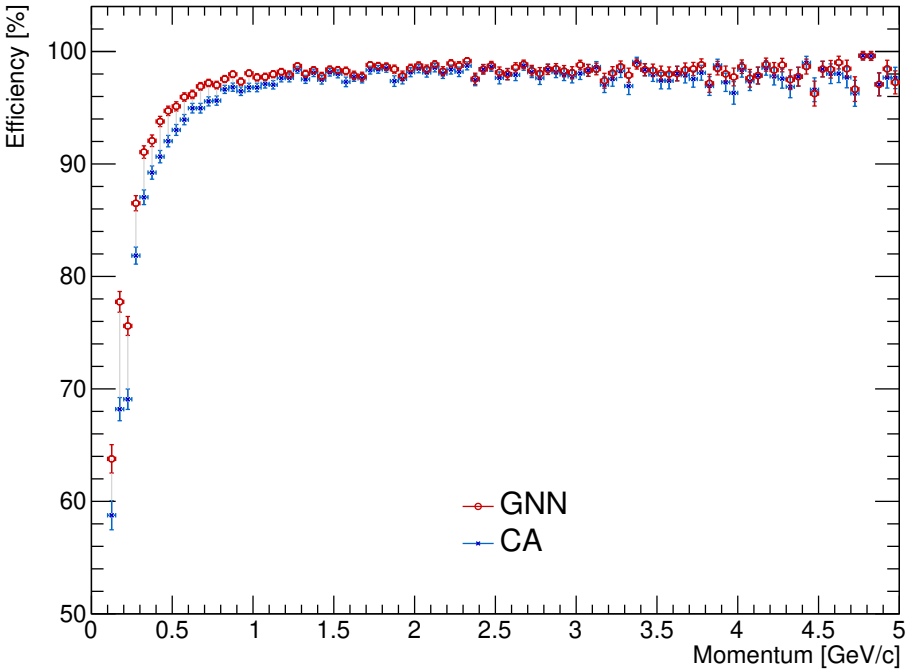


(a) All tracks

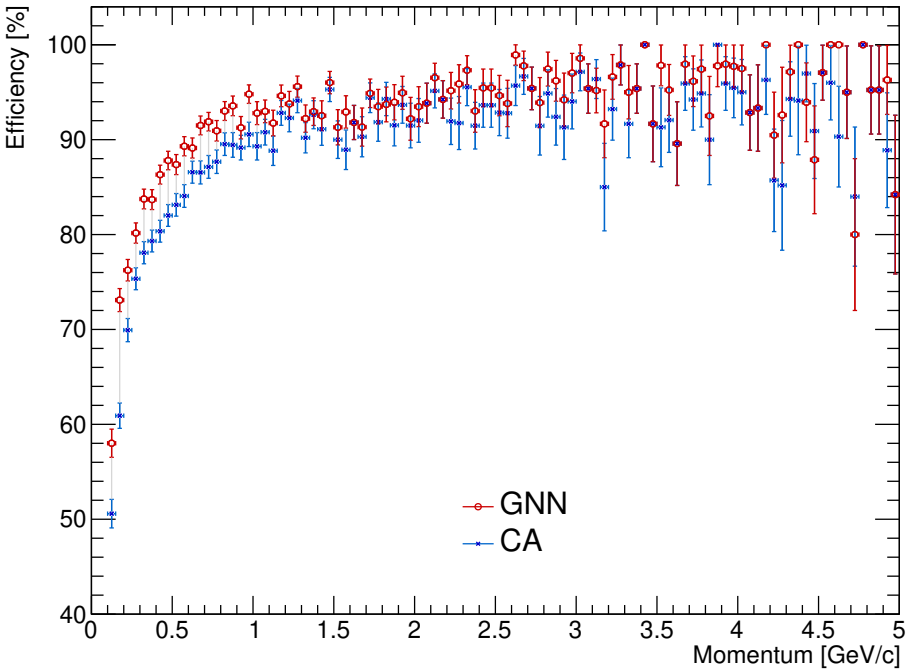


(b) Secondary tracks

Figure A.3: Central GNN vs CA reconstruction efficiency comparison by momentum for Au+Au central collisions at 20 AGeV. Results for (a) all tracks and (b) secondary tracks.



(a) All tracks



(b) Secondary tracks

Figure A.4: Minimum bias GNN vs CA reconstruction efficiency comparison by momentum for Au+Au minimum bias collisions at 20 AGeV. Results for (a) all tracks and (b) secondary tracks.

A.3 28.3 *A*GeV p+Au

To further test the generalizability of the GNN-based track finding algorithm, its performance was evaluated for proton on gold (p+Au) collisions at 28.3 *A*GeV. These collisions produce only a small number of reconstructible tracks, representing an extremely low-occupancy environment.

Tables A.3a and A.3b present the reconstruction performance for central and minimum-bias p+Au collisions. The GNN-based track finder delivers consistent improvements across all track categories. The total reconstruction efficiency rises by 3.5% in central events and by 4.2% in minimum-bias collisions. Notably, the efficiency for low-momentum primary particles increases by 5.9% in central collisions and by 7.4% in minimum-bias conditions.

Because of the extremely low hit density, ghost and clone rates are very low. Both algorithms record an identical ghost rate of 0.1%, while the clone rate remains around 4%. Figures A.5 and A.2 illustrate the momentum dependence of the reconstruction efficiency. Although large statistical fluctuations are visible due to the limited number of tracks, the results clearly indicate that the GNN performs at least as reliably as the CA-based track finder.

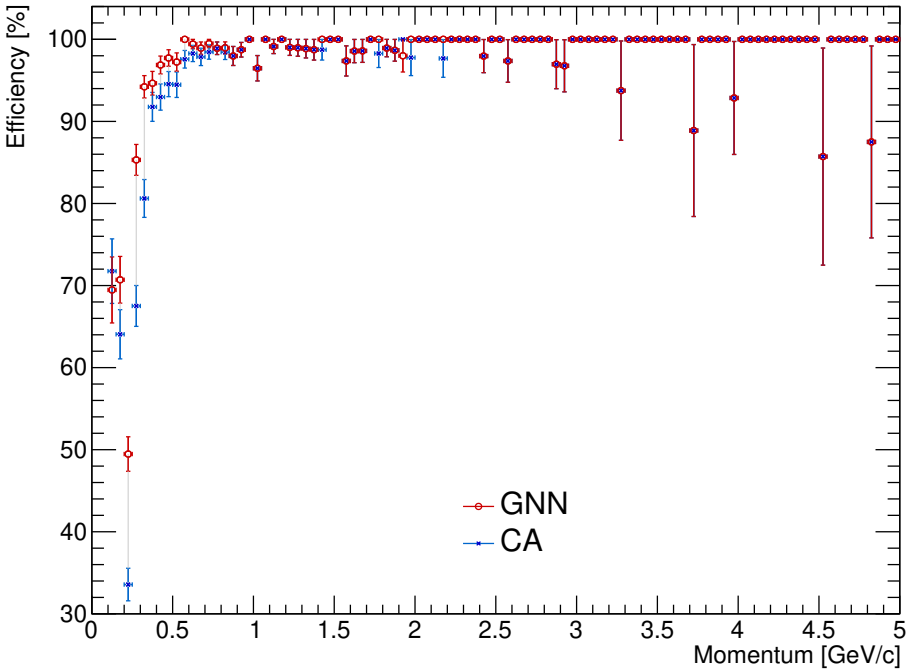
Track Category	CA (%)	GNN (%)	Change (%)	No. of MC Tracks
All tracks	89.4	**92.9**	+3.5	7.4
Primary high-p	**99.2**	**99.2**	0.0	3.0
Primary low-p	81.6	**87.5**	+5.9	3.3
Secondary high-p	97.4	**98.5**	+1.1	0.3
Secondary low-p	82.7	**89.6**	+6.9	0.8
Clone Rate	4.4	**4.3**	-0.1	–
Ghost Rate	**0.1**	**0.1**	0.0	–
MC tracks found	6.6	**6.8**	+0.2	7.4
Time, s/ev	**0.004**	0.03	–	–

(a) Central collisions

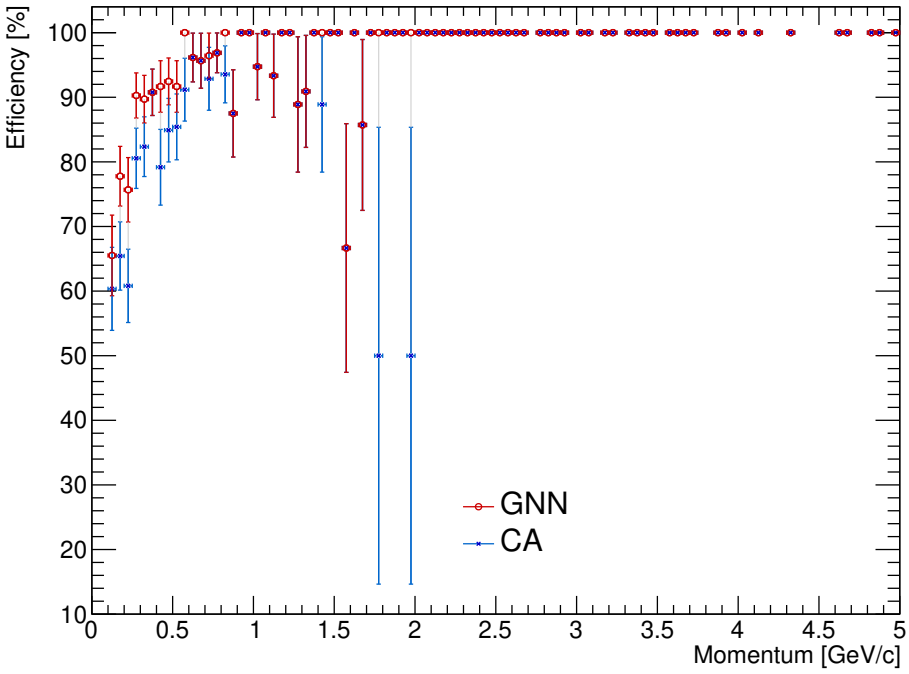
Track Category	CA (%)	GNN (%)	Change (%)	No. of MC Tracks
All tracks	85.9	**90.1**	+4.2	3.7
Primary high-p	99.6	**99.6**	0.0	1.6
Primary low-p	71.7	**79.1**	+7.4	1.5
Secondary high-p	97.3	**98.7**	+1.4	0.1
Secondary low-p	82.5	**91.1**	+8.6	0.4
Clone Rate	**3.7**	3.8	+0.1	–
Ghost Rate	**0.1**	**0.1**	0.0	–
MC tracks found	3.2	**3.3**	+0.1	3.7
Time, s/ev	**0.003**	0.02	–	–

(b) Minimum-bias collisions

Table A.3: Comparison of CA and GNN efficiencies across track categories for 28.3 AGeV p+Au collisions. Results averaged over 1000 events.

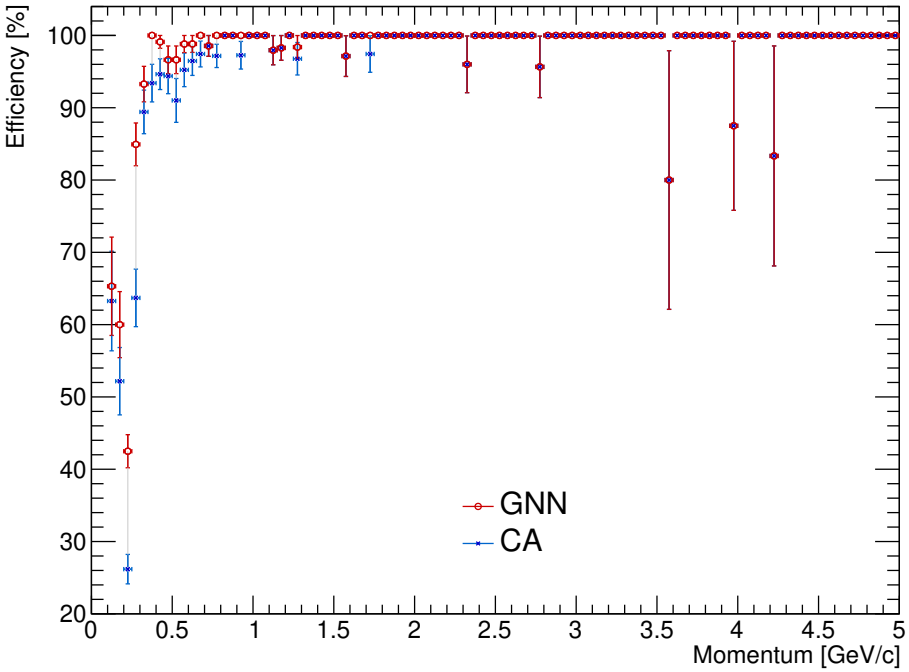


(a) All tracks

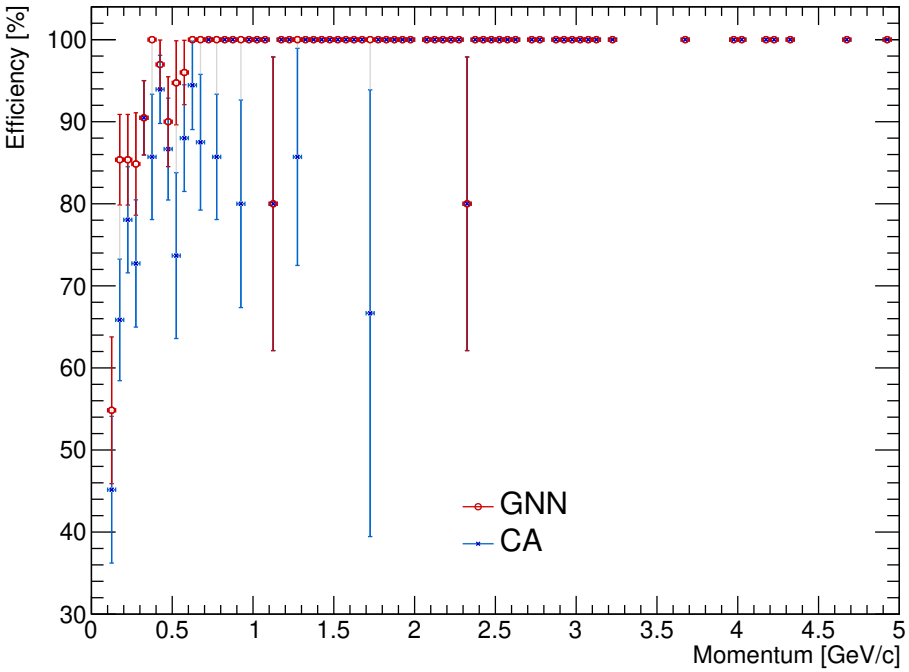


(b) Secondary tracks

Figure A.5: Central GNN vs CA reconstruction efficiency comparison by momentum for p+Au central collisions at 28.3 AGeV. Results for (a) all tracks and (b) secondary tracks.



(a) All tracks



(b) Secondary tracks

Figure A.6: Minimum bias GNN vs CA reconstruction efficiency comparison by momentum for p+Au minimum bias collisions at 28.3 AGeV. Results for (a) all tracks and (b) secondary tracks.

List of Figures

List of Tables

Bibliography

[1] Super-Kamiokande Collaboration. "Evidence for Oscillation of Atmospheric Neutrinos". In: *Physical Review Letters* 81.8 (Aug. 1998), pp. 1562–1567. DOI: 10.1103/PhysRevLett.81.1562. URL: https://link.aps.org/doi/10.1103/PhysRevLett.81.1562.

[2] Cush. *Standard Model of Elementary Particles*. en. 2025. URL: https://en.wikipedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg.

[3] Y. Aoki et al. "The Order of the quantum chromodynamics transition predicted by the standard model of particle physics". In: *Nature* 443 (2006), pp. 675–678. DOI: 10.1038/nature05120. arXiv: hep-lat/0611014.

[4] Szabolcs Borsanyi et al. "Is there still any T_c mystery in lattice QCD? Results with physical masses in the continuum limit III". In: *JHEP* 09 (2010), p. 073. DOI: 10.1007/JHEP09(2010)073. arXiv: 1005.3508 [hep-lat].

[5] A. Bazavov et al. "The chiral and deconfinement aspects of the QCD transition". In: *Phys. Rev. D* 85 (2012), p. 054503. DOI: 10.1103/PhysRevD.85.054503. arXiv: 1111.1710 [hep-lat].

[6] Philippe de Forcrand. "Simulating QCD at finite density". In: *PoS* LAT2009 (2009). Ed. by Chuan Liu and Yu Zhu, p. 010. DOI: 10.22323/1.091.0010. arXiv: 1005.0539 [hep-lat].

[7] GSI. *Hot and dense QCD*. en. 2025. URL: https://www.gsi.de/work/forschung/theorie/theory-new/hot-and-dense-qcd.

[8] J.W. Harris et al. "The STAR experiment at the relativistic heavy ion collider". In: *Nuclear Physics A* 566 (1994), pp. 277–285. ISSN: 0375-9474. DOI: https://doi.org/10.1016/0375-9474(94)90633-5. URL: https://www.sciencedirect.com/science/article/pii/0375947494906335.

[9] D. P. Morrison et al. "The PHENIX experiment at RHIC". In: *Nucl. Phys. A* 638 (1998). Ed. by T. Hatsuda et al., pp. 565–570. DOI: 10.1016/S0375-9474(98)00390-X. arXiv: hep-ex/9804004.

[10] K. Aamodt et al. "The ALICE experiment at the CERN LHC". In: *JINST* 3 (2008), S08002. DOI: 10.1088/1748-0221/3/08/S08002.

[11] Bengt Friman et al., eds. *The CBM physics book: Compressed baryonic matter in laboratory experiments*. Vol. 814. 2011. DOI: 10.1007/978-3-642-13293-3.

[12] V. Lindenstruth and I. Kisel. "Overview of trigger systems". In: *Nucl. Instrum. Meth. A* 535 (2004). Ed. by M. Jeitler et al., pp. 48–56. DOI: 10.1016/j.nima.2004.07.267.

[13] APPA collaboration. *APPA - Atomic, Plasma Physics and Applications*. en. 2025. URL: https://fair-center.eu/user/experiments/appa.

[14] CBM collaboration. *CBM - Compressed Baryonic Matter experiment at FAIR*. en. 2025. URL: https://www.cbm.gsi.de/.

[15] NUSTAR collaboration. *Super-conducting Fragment Separator (Super-FRS)*. en. 2025. URL: https://www.gsi.de/work/gesamtprojektleitung_fair/super_frs.

[16] NUSTAR collaboration. *NUSTAR - Nuclear Structure, Astrophysics and Reactions*. en. 2025. URL: https://fair-center.eu/user/experiments/nustar.

[17] PANDA collaboration. *PANDA - Antiproton Annihilation at Darmstadt*. en. 2025. URL: https://panda.gsi.de/.

[18] H. Gutbrod et al. *FAIR Baseline Technical Report (Volume 1)*. Sept. 2006.

[19] Tetyana Galatyuk. "Future facilities for high μ_B physics". In: *Nucl. Phys. A* 982 (2019). Ed. by Federico Antinori et al., pp. 163–169. DOI: 10.1016/j.nuclphysa.2018.11.025.

[20] A. Andronic, P. Braun-Munzinger, and J. Stachel. "Hadron production in central nucleus-nucleus collisions at chemical freeze-out". In: *Nucl. Phys. A* 772 (2006), pp. 167–199. DOI: 10.1016/j.nuclphysa.2006.03.012. arXiv: nucl-th/0511071.

[21] *Technical Design Report for the CBM Online Systems – Part I, DAQ and FLES Entry Stage*. Tech. rep. -. Darmstadt, 2023, p. 196. DOI: 10.15120/GSI-2023-00739. URL: https://repository.gsi.de/record/340597.

[22] Artemiy Belousov. "A QGP trigger based on convolutional neural network for the CBM experiment". PhD thesis. Goethe U., Frankfurt (main), 2025. DOI: 10.21248/gups.95440.

[23] Adrian Meyer-Ahrens. "Dielectron Performance of the Compressed Baryonic Matter Experiment". en. PhD thesis. University of Munster, 2025. URL: https://www.uni-muenster.de/imperia/md/content/physik_kp/meyerahrens2025.pdf.

[24] Alexander Malakhov and Alexey Shabunov, eds. *Technical Design Report for the CBM Superconducting Dipole Magnet*. Darmstadt: GSI, 2013, 80 S. URL: https://repository.gsi.de/record/109025.

[25] J. Stroth and M. Deveaux. *Technical Design Report for the CBM: Micro Vertex Detector (MVD)*. Tech. rep. 1. 2022, 157 p. URL: https://repository.gsi.de/record/246516.

[26] Johann Heuser et al., eds. *[GSI Report 2013-4] Technical Design Report for the CBM Silicon Tracking System (STS)*. Darmstadt: GSI, 2013, 167 p. URL: https://repository.gsi.de/record/54798.

[27] Shaifali Mehta. "Investigation of thermal and structural integrity of modules and ladders of Silicon Tracking System of the CBM experiment". PhD thesis. U. Tubingen, 2025. DOI: 10.15496/publikation-104689.

[28] Minni Singla. "The Silicon Tracking System of the CBM experiment at FAIR: Development of microstrip sensors and signal transmission lines for a low-mass, low-noise system". PhD thesis. Frankfurt U., 2014.

[29] Valentina Akishina. "Four-dimensional event reconstruction in the CBM experiment". PhD thesis. Goethe U., Frankfurt (main), 2017.

[30] John David Jackson. *Classical Electrodynamics*. Wiley, 1998. ISBN: 978-0-471-30932-1.

[31] *Technical Design Report for the CBM Ring Imaging Cherenkov Detector*. Tech. rep. 2013, 215 p. URL: https://repository.gsi.de/record/65526.

[32] Subhasis Chattopadhyay et al., eds. *Technical Design Report for the CBM : Muon Chambers (MuCh)*. Darmstadt: GSI, 2015, 190 S. URL: https://repository.gsi.de/record/161297.

[33] V. L. Ginzburg and I. M. Frank. "Radiation of a uniformly moving electron due to its transition from one medium into another". In: *J. Phys. (USSR)* 9 (1945), pp. 353–362.

[34] G M Garibyan. "Contribution to the theory of transition radiation". In: *Zhur. Eksptl'. i Teoret. Fiz.* Vol: 33 (Nov. 1957). URL: https://www.osti.gov/biblio/4331529.

[35] *The Transition Radiation Detector of the CBM Experiment at FAIR : Technical Design Report for the CBM Transition Radiation Detector (TRD)*. Tech. rep. FAIR Technical Design Report. Darmstadt, 2018, 165 p. DOI: 10.15120/GSI-2018-01091. URL: https://repository.gsi.de/record/217478.

[36] V. V. Gligorov and V. Reković. "Review of real-time data processing for collider experiments". In: *The European Physical Journal Plus* 138.11 (Nov. 2023). ISSN: 2190-5444. DOI: 10.1140/epjp/s13360-023-04599-6. URL: http://dx.doi.org/10.1140/epjp/s13360-023-04599-6.

[37] GSI. *Green IT Cube: Supercomputing center for GSI and FAIR*. en. 2025. URL: https://www.gsi.de/en/researchaccelerators/research_an_overview/green-it-cube.

[38] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach*. Fourth. Morgan Kaufmann, 2007.

[39] William Gropp. *Designing and Building Applications for Extreme Scale Systems*. 2015. URL: https://wgropp.cs.illinois.edu/courses/cs598-s15/.

[40] Michael J. Flynn. "Some Computer Organizations and Their Effectiveness". In: *IEEE Transactions on Computers* C-21.9 (1972), pp. 948–960. DOI: 10.1109/TC.1972.5009071.

[41] Cburnett. *SISD*. en. 2007. URL: https://commons.wikimedia.org/wiki/File:SISD.svg.

[42] Cburnett. *MISD*. en. 2007. URL: https://commons.wikimedia.org/wiki/File:MISD.svg.

[43] Cburnett. *SIMD*. en. 2007. URL: https://commons.wikimedia.org/wiki/File:SIMD.svg.

[44] Cburnett. *MIMD*. en. 2007. URL: https://commons.wikimedia.org/wiki/File:MIMD.svg.

[45] LLVM project. *Auto-Vectorization in LLVM*. 2025. URL: https://llvm.org/docs/Vectorizers.html.

[46] FIAS Kisel Group. *High Performance Computing - Practical Course, Göethe University Frankfurt*. 2016.

[47] Alfred Spector and David Gifford. "The Space Shuttle Primary Computer System". In: *Commun. ACM* 27.9 (1984), pp. 872–900. ISSN: 0001-0782. DOI: 10.1145/358234.358246. URL: https://doi.org/10.1145/358234.358246.

[48] Kaz Sato and Cliff Young. *An in-depth look at Google's first Tensor Processing Unit (TPU)*. en. 2017. URL: https://cloud.google.com/blog/products/ai-machine-learning/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu.

[49] Leonardo Dagum and Ramesh Menon. "OpenMP: an industry standard API for shared-memory programming". In: *Computational Science & Engineering, IEEE* 5.1 (1998), pp. 46–55.

[50] OpenMP. *OpenMP - The OpenMP API specification for parallel programming*. 2025. URL: https://www.openmp.org/.

[51] Gene M. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities". In: AFIPS '67 (Spring). New York, NY, USA: Association for Computing Machinery, 1967, pp. 483–485. ISBN: 9781450378956. DOI: 10.1145/1465482.1465560. URL: https://doi.org/10.1145/1465482.1465560.

[52] John L. Gustafson. "Reevaluating Amdahl's Law". In: *Commun. ACM* 31.5 (1988), pp. 532–533. ISSN: 0001-0782. DOI: 10.1145/42411.42415. URL: https://doi.org/10.1145/42411.42415.

[53] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. en. Addison-Wesley Professional, July 2010. ISBN: 978-0-13-218013-9. URL: https://books.google.fr/books?id=49OmnOmTEtQC.

[54] David Rohr et al. “ALICE HLT TPC tracking of Pb-Pb Events on GPUs”. In: *J. Phys. Conf. Ser.* 396 (2012), p. 012044. DOI: 10.1088/1742-6596/396/1/012044. arXiv: 1712.09407 [physics.ins-det].

[55] Daniel Funke et al. “Parallel track reconstruction in CMS using the cellular automaton approach”. In: *J. Phys. Conf. Ser.* 513 (2014). Ed. by D. L. Groep and D. Bonacorsi, p. 052010. DOI: 10.1088/1742-6596/513/5/052010.

[56] V. Singhal, S. Chattopadhyay, and V. Friese. “Investigation of heterogeneous computing platforms for real-time data analysis in the CBM experiment”. In: *Comput. Phys. Commun.* 253 (2020), p. 107190. DOI: 10.1016/j.cpc.2020.107190. arXiv: 1810.11966 [physics.comp-ph].

[57] NVIDIA. *CUDA C++ Programming Guide*. 2025. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[58] Felix Wiglhofer. “GPU-based Online Processing for the ALICE and CBM Experiments”. PhD thesis. Goethe U., Frankfurt (main), 2025.

[59] Felix Weiglhofer. *xpu*. URL: https://github.com/fweig/xpu.

[60] *mCBM@SIS18*. Tech. rep. CBM. Darmstadt, 2017, 58 S. DOI: 10.15120/GSI-2019-00977. URL: https://repository.gsi.de/record/220072.

[61] Norbert Herrmann and Christian Sturm. *mCBM@SIS18 2023/24*. Tech. rep. 2022. Darmstadt, 2025, 13 p. DOI: 10.15120/GSI-2025-00774. URL: https://repository.gsi.de/record/359718.

[62] Grigory Kozlov. *Porting the CBM CA Track Finder to GPU*. 2025. URL: https://indico.gsi.de/event/20881/contributions/92733/attachments/53218/80105/G.Kozlov_GPU_CA_tracking.pdf.

[63] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[64] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[65] J. Jumper, R. Evans, A. Pritzel, et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596 (Aug. 2021), pp. 583–589. DOI: 10.1038/s41586-021-03819-2.

[66] Warren S. McCulloch and Walter Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity". en. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[67] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer Feedforward Networks Are Universal Approximators". In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[68] Zhou Lu et al. "The Expressive Power of Neural Networks: A View from the Width". In: *CoRR* abs/1709.02540 (2017). arXiv: 1709.02540. URL: http://arxiv.org/abs/1709.02540.

[69] Patrick Kidger and Terry J. Lyons. "Universal Approximation with Deep Narrow Networks". In: *CoRR* abs/1905.08539 (2019). arXiv: 1905.08539. URL: http://arxiv.org/abs/1905.08539.

[70] Namig J Guliyev and Vugar E Ismailov. "Approximation capability of two hidden layer feedforward neural networks with fixed weights". In: *Neurocomputing* 316 (2018), pp. 262–269.

[71] Zuowei Shen, Haizhao Yang, and Shijun Zhang. "Optimal approximation rate of ReLU networks in terms of width and depth". In: *Journal de Mathématiques Pures et Appliquées* 157 (2022), pp. 101–135.

[72] Frank Rosenblatt. "The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain". eng. In: *Psychological Review* 65.6 (Nov. 1958), pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519.

[73] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (Sept. 1951), pp. 400–407. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177729586. URL: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-3/A-Stochastic-Approximation-Method/10.1214/aoms/1177729586.full.

[74] Shunichi Amari. "A Theory of Adaptive Pattern Classifiers". In: *IEEE Transactions on Electronic Computers* EC-16.3 (June 1967), pp. 299–307. ISSN: 0367-7508. DOI: 10.1109/PGEC.1967.264666. URL: https://ieeexplore.ieee.org/document/4039068.

[75] Seppo Linnainmaa. "The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors". fin. PhD thesis. 1970. URL: http://hdl.handle.net/10138/316565.

[76] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0.

[77] Yann LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791. URL: https://ieeexplore.ieee.org/document/726791.

[78] Marta R Costa-Jussà et al. "No language left behind: Scaling human-centered machine translation". In: *arXiv preprint arXiv:2207.04672* (2022).

[79] Awni Hannun et al. *Deep Speech: Scaling up end-to-end speech recognition*. 2014. arXiv: 1412.5567 [cs.CL]. URL: https://arxiv.org/abs/1412.5567.

[80] OpenAI. *Introducing ChatGPT*. en-US. 2022. URL: https://openai.com/index/chatgpt/.

[81] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV]. URL: https://arxiv.org/abs/2112.10752.

[82] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[83] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. DOI: 10.1214/aoms/1177703732. URL: https://doi.org/10.1214/aoms/1177703732.

[84] Nadav Cohen. *Understanding Optimization in Deep Learning by Analyzing Trajectories of Gradient Descent*. Nov. 2018. URL: http://offconvex.github.io/2018/11/07/optimization-beyond-landscape/.

[85] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980.

[86] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[87] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., Dec. 2019, pp. 8026–8037.

[88] Martín Abadi et al. "TensorFlow: A System for Large-Scale Machine Learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16 (Nov. 2016), pp. 265–283.

[89] Bigbossfarin. *Bias and variance contributing to total error*. en. 2021. URL: https://commons.wikimedia.org/wiki/File:Bias_and_variance_contributing_to_total_error.svg.

[90] Izaak Neutelings. *Neural Networks*. en-US. Apr. 2024. URL: https://tikz.net/neural_networks/.

[91] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285 [stat.ML]. URL: https://arxiv.org/abs/1603.07285.

[92] Reza Kalantar. *Receptive Field in Deep Convolutional Networks*. en. 2023. URL: https://medium.com/@rekalantar/receptive-fields-in-deep-convolutional-networks-43871d2ef2e9.

[93] William L. Hamilton. *Graph Representation Learning*. Morgan & Claypool Publishers, 2020. ISBN: 978-1-68173-964-9. URL: https://ieeexplore.ieee.org/book/9205745.

[94] Davide Bacciu et al. “A Gentle Introduction to Deep Learning for Graphs”. In: *Neural Networks* 129 (Sept. 2020), pp. 203–221. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2020.06.006. URL: https://www.sciencedirect.com/science/article/pii/S0893608020302197.

[95] M. Gori, G. Monfardini, and F. Scarselli. “A new model for learning in graph domains”. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. 2005, pp. 729–734. DOI: 10.1109/IJCNN.2005.1555942.

[96] Franco Scarselli et al. “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1 (Jan. 2009), pp. 61–80. ISSN: 1941-0093. DOI: 10.1109/TNN.2008.2005605. URL: https://ieeexplore.ieee.org/document/4700287.

[97] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. en. Sept. 2016. URL: https://arxiv.org/abs/1609.02907v4.

[98] Peter W. Battaglia et al. *Relational Inductive Biases, Deep Learning, and Graph Networks*. Oct. 2018. DOI: 10.48550/arXiv.1806.01261. URL: http://arxiv.org/abs/1806.01261.

[99] Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal. “Understanding Convolutions on Graphs”. In: *Distill* (2021). DOI: 10.23915/distill.00032.

[100] Benjamin Sanchez-Lengeling et al. “A Gentle Introduction to Graph Neural Networks”. In: *Distill* (2021). DOI: 10.23915/distill.00033.

[101] Jure Leskovec. *CS224W: Machine Learning with Graphs*. URL: https://web.stanford.edu/class/cs224w/.

[102] Justin Gilmer et al. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG]. URL: https://arxiv.org/abs/1704.01212.

[103] Peter Senger and Volker Friese. *CBM Progress Report 2022*. Tech. rep. CBM Progress Report 2022. Darmstadt, 2022, p. 226. DOI: 10.15120/GSI-2023-00384. URL: https://repository.gsi.de/record/336786.

[104] Fedor Sergeev et al. "Deep learning for quark–gluon plasma detection in the CBM experiment". In: *Int. J. Mod. Phys. A* 35.33 (2020), p. 2043002. DOI: 10.1142/S0217751X20430022.

[105] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[106] R. Fruhwirth. "Application of Kalman filtering to track and vertex fitting". In: *Nucl. Instrum. Meth. A* 262 (1987), pp. 444–450. DOI: 10.1016/0168-9002(87)90887-4.

[107] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[108] A. Glazov et al. "Filtering tracks in discrete detectors using a cellular automaton". In: *Nucl. Instrum. Meth. A* 329 (1993), pp. 262–268. DOI: 10.1016/0168-9002(93)90945-E.

[109] M. P. Bussa et al. "Application of a cellular automaton for recognition of straight tracks in the spectrometer DISTO". In: (Jan. 1995).

[110] I. Kisel et al. "Cellular automaton and elastic net for event reconstruction in the NEMO-2 experiment". In: *Nucl. Instrum. Meth. A* 387 (1997), pp. 433–442. DOI: 10.1016/S0168-9002(97)00097-1.

[111] I. Abt et al. "CATS: A cellular automaton for tracking in silicon for the HERA-B vertex detector". In: *Nucl. Instrum. Meth. A* 489 (2002), pp. 389–405. DOI: 10.1016/S0168-9002(02)00790-8.

[112] S. Gorbunov and I. Kisel. "Analytic formula for track extrapolation in non-homogeneous magnetic field". In: *Nucl. Instrum. Meth. A* 559 (2006). Ed. by J. Blumlein et al., pp. 148–152. DOI: 10.1016/j.nima.2005.11.133.

[113] S. Gorbunov et al. "Fast SIMDized Kalman filter based track fit". In: *Computer Physics Communications* 178.5 (Mar. 2008), pp. 374–383. DOI: 10.1016/j.cpc.2007.10.001.

[114] Sergey Gorbunov. "On-line reconstruction algorithms for the CBM and ALICE experiments". PhD thesis. Johann Wolfgang Goethe-Universität Frankfurt, 2013, 104, [9] S. : Ill., graph. Darst., 30 cm. URL: https://repository.gsi.de/record/206421.

[115] Bruce H. Denby. "Neural Networks and Cellular Automata in Experimental High-energy Physics". In: *Comput. Phys. Commun.* 49 (1988), pp. 429–448. DOI: 10.1016/0010-4655(88)90004-5.

[116] Miklos Gyulassy and Magnus Harlander. "Elastic tracking and neural network algorithms for complex pattern recognition". In: *Comput. Phys. Commun.* 66 (1991), pp. 31–46. DOI: 10.1016/0010-4655(91)90005-6.

[117] Valerio Bertacchi. *DeepCore: Convolutional Neural Network for high p_T jet tracking*. 2020. arXiv: 1910.08058 [physics.ins-det]. URL: https://arxiv.org/abs/1910.08058.

[118] S. Chatrchyan et al. "The CMS Experiment at the CERN LHC". In: *JINST* 3 (2008), S08004. DOI: 10.1088/1748-0221/3/08/S08004.

[119] Steven Farrell et al. *Novel deep learning methods for track reconstruction*. 2018. arXiv: 1810.06111 [hep-ex]. URL: https://arxiv.org/abs/1810.06111.

[120] Steven Farrell et al. "The HEP.TrkX Project: deep neural networks for HL-LHC online and offline tracking". In: *EPJ Web Conf.* 150 (2017). Ed. by C. Germain et al., p. 00003. DOI: 10.1051/epjconf/201715000003.

[121] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[122] Sascha Caron et al. "Trackformers: in search of transformer-based particle tracking for the high-luminosity LHC era". In: *Eur. Phys. J. C* 85.4 (2025), p. 460. DOI: 10.1140/epjc/s10052-025-14156-3. arXiv: 2407.07179 [hep-ex].

[123] Moritz Kiehn et al. "The TrackML high-energy physics tracking challenge on Kaggle". In: *EPJ Web Conf.* 214 (2019). Ed. by A. Forti et al., p. 06037. DOI: 10.1051/epjconf/201921406037.

[124] Xiangyang Ju et al. "Performance of a geometric deep learning pipeline for HL-LHC particle tracking". In: *Eur. Phys. J. C* 81.10 (2021), p. 876. DOI: 10.1140/epjc/s10052-021-09675-8. arXiv: 2103.06995 [physics.data-an].

[125] Michael M. Bronstein et al. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42. DOI: 10.1109/MSP.2017.2693418.

[126] Sabrina Amrouche et al. "The Tracking Machine Learning challenge : Accuracy phase". In: *The NeurIPS '18 Competition: From Machine Learning to Intelligent Conversations*. Apr. 2019. DOI: 10.1007/978-3-030-29135-8_9. arXiv: 1904.06778 [hep-ex].

[127] Anthony Correia et al. "Graph Neural Network-based track finding in the LHCb vertex detector". In: *JINST* 19.12 (2024), P12022. DOI: 10.1088/1748-0221/19/12/P12022. arXiv: 2407.12119 [physics.ins-det].

[128] I. Kisel. "Event reconstruction in the CBM experiment". In: *Nucl. Instrum. Meth. A* 566 (2006). Ed. by R. Bernhard et al., pp. 85–88. DOI: 10.1016/j.nima.2006.05.040.

[129] Valentina Akishina and Ivan Kisel. "Time-based Cellular Automaton track finder for the CBM experiment". In: *J. Phys. Conf. Ser.* 599.1 (2015). Ed. by Marco Destefanis et al., p. 012024. DOI: 10.1088/1742-6596/599/1/012024.

[130] *CBM Progress Report 2023*. Tech. rep. CBM Progress Report 2023. 2024, 240 p. DOI: 10.15120/GSI-2024-00765. URL: https://repository.gsi.de/record/352779.

[131] Alexander LeNail. "NN-SVG: Publication-Ready Neural Network Architecture Schematics". In: *Journal of Open Source Software* 4.33 (2019), p. 747. DOI: 10.21105/joss.00747. URL: https://doi.org/10.21105/joss.00747.

[132] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.2 (Feb. 2020), pp. 318–327. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2018.2858826. URL: https://ieeexplore.ieee.org/document/8417976.

[133] Maksym Thesis. "Online selection of short-lived particles on many-core computer architectures in the CBM experiment at FAIR". PhD thesis. Johann Wolfgang Goethe-Universität Frankfurt, 2015. URL: https://publikationen.ub.uni-frankfurt.de/opus4/frontdoor/deliver/index/docId/41428/file/Maksym_Zyzak_thesis.pdf.

[134] ROOT CERN. *TMath:Prob*. URL: https://root.cern/root/html602/TMath.html#TMath:Prob.

[135] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[136] Geoffrey E Hinton and Sam Roweis. "Stochastic Neighbor Embedding". In: *Advances in Neural Information Processing Systems*. Vol. 15. MIT Press, 2002. URL: https://proceedings.neurips.cc/paper_files/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf.

[137] CBM collaboration. *CbmRoot*. 2020. URL: https://git.cbm.gsi.de/computing/cbmroot.

[138] M. Al-Turany et al. "The FairRoot framework". In: *J. Phys. Conf. Ser.* 396 (2012). Ed. by Michael Ernst et al., p. 022001. DOI: 10.1088/1742-6596/396/2/022001.

[139] Mohammad Al-Turany et al. *FairRoot*. URL: https://fairroot.gsi.de/.

[140] Rene Brun and Fons Rademakers. *Root: Data Analysis Framework*. URL: https://root.cern/.

[141] S. A. Bass et al. "Microscopic models for ultrarelativistic heavy ion collisions". In: *Prog. Part. Nucl. Phys.* 41 (1998), pp. 255–369. DOI: 10.1016/S0146-6410(98)00058-1. arXiv: nucl-th/9803035.

[142] M. Bleicher et al. "Relativistic hadron hadron collisions in the ultrarelativistic quantum molecular dynamics model". In: *J. Phys. G* 25 (1999), pp. 1859–1896. DOI: 10.1088/0954-3899/25/9/308. arXiv: hep-ph/9909407.

[143] S Agostinelli et al. "GEANT4–a simulation toolkikt". In: *Nucl. Instrum. Methods Phys. Res., A* 506.3 (2003), pp. 250–303. DOI: 10.1016/S0168-9002(03)01368-8. URL: https://cds.cern.ch/record/602040.

[144] John Allison et al. "Geant4 developments and applications". In: *IEEE Trans. Nucl. Sci.* 53 (2006), p. 270. DOI: 10.1109/TNS.2006.869826.

Zusammenfassung

Das Compressed Baryonic Matter (CBM) Experiment an der zukünftigen Facility for Antiproton and Ion Research (FAIR) in Darmstadt, Deutschland, wurde konzipiert, um stark wechselwirkende Materie unter extremen Bedingungen zu untersuchen. CBM wird den Bereich hoher Baryonendichten und moderater Temperaturen des QCD-Phasendiagramms erforschen, in dem theoretische Modelle einen Phasenübergang erster Ordnung sowie einen kritischen Endpunkt zwischen gebundener hadronischer Materie und entbundenem Quark-Gluon-Plasma vorhersagen. Um diese Phänomene zu untersuchen, wird CBM Schwerionenkollisionen an einem festen Target nutzen, um Materie mit Dichten zu erzeugen, die jene normaler Kernmaterie übersteigen. Zu den wichtigsten Observablen gehören die Dileptonenproduktion, die Produktion von Charm-Quarks und Teilchen mit vielfacher Seltsamkeit (Multi-Strange Particles) sowie globale Observablen wie der kollektive Fluss und Ereignis-zu-Ereignis-Fluktuationen.

Um für diese seltenen Sonden eine ausreichende Statistik zu akkumulieren, muss das CBM-Experiment bei beispiellosen Interaktionsraten von bis zu 10^7 Kollisionen/s (10 MHz) operieren. Dies erzeugt Rohdatenvolumina in der Größenordnung von Terabytes pro Sekunde, was eine vollständige Archivierung der Rohdaten unmöglich macht. Darüber hinaus schließen die komplexen topologischen Signaturen der physikalisch interessantesten Kanäle, wie etwa Hyperonen mit vielfacher Seltsamkeit und Charmed Hadrons, die Verwendung einfacher Hardware-Trigger aus. Folglich setzt CBM eine kontinuierliche, freilaufende Auslesearchitektur (Free-Streaming Readout) in Kombination mit einer Echtzeit-Ereignisrekonstruktion (Software-Trigger) ein, um interessante Ereignisse vor der Speicherung zu selektieren. Um den erforderlichen Datendurchsatz zu erreichen, muss die Spurrekonstruktion hochgradig effizient sein.

Das Hauptspursystem von CBM besteht aus dem pixelbasierten Micro-Vertex Detector (MVD) und dem streifenbasierten Silicon Tracking System (STS), die

sich innerhalb eines supraleitenden Dipolmagneten befinden. Die tausenden geladenen Teilchen, die in den Kollisionen entstehen, erzeugen hohe Trefferdichten auf den Tracking-Stationen. Die Spurfindung ist unter diesen Bedingungen aufgrund des großen kombinatorischen Untergrunds eine Herausforderung. In den letzten zwei Jahrzehnten wurde ein hochoptimierter, auf zellulären Automaten (CA) basierender Spurfinder für das CBM-Experiment entwickelt. Während der CA-basierte Spurfinder schnell und robust ist, weist er eine geringe Rekonstruktionseffizienz für Sekundärspuren auf. Diese niedrige Effizienz ist teilweise darauf zurückzuführen, dass auch bei der Rekonstruktion von Sekundärspuren der primäre Vertex verwendet wird.

Diese Arbeit stellt einen neuartigen, auf Graph Neural Networks (GNN) basierenden Algorithmus zur Spurfindung für das CBM-Experiment vor. Im Gegensatz zu aktuellen „End-to-End“-Deep-Learning-Ansätzen verfolgt der in dieser Arbeit entwickelte Algorithmus eine hybride Strategie: Er integriert die Repräsentationskraft des datengetriebenen Deep Learning mit der physikalischen Robustheit der Parameterschätzung mittels Kalman-Filter (KF). Dieser Ansatz reduziert die Abhängigkeit von neuronalen Netzen dort, wo geometrische Heuristiken ausreichen, und gewährleistet so sowohl Genauigkeit als auch Interpretierbarkeit.

Der Algorithmus läuft in drei Iterationen ab, die auf folgende Kategorien abzielen: (1) Primärspuren mit hohem Impuls, (2) alle Primärspuren und (3) Sekundärspuren. In jeder Iteration werden Treffer (Hits) mithilfe einer k-Nearest Neighbour (k-NN) Suche innerhalb eines gelernten latenten Raums, in dem zum gleichen Track gehörende Treffer geclustert sind, zu Paaren (Doublets) gruppiert. In den ersten beiden Iterationen werden Doublets mittels Winkelschnitten relativ zum Primärvertex gefiltert, während in der dritten Iteration ein GNN zur Kantenklassifizierung eingesetzt wird, um Sekundärteilchen wiederherzustellen. Doublets, die einen mittleren Treffer teilen, werden zu Triplets kombiniert, was eine erste Schätzung der Spurparameter mittels eines Kalman-Filters (KF) ermöglicht. Kinematisch kompatible, überlappende Paare von Triplets werden anschließend zu Spurkandidaten verknüpft. Schließlich wird ein neuartiges kooperatives Spurauswahlschema angewendet, das den Austausch von Trefferzugehörigkeiten zwischen Kandidaten ermöglicht, um die Anzahl der rekonstruierten Spuren zu maximieren.

Der GNN-basierte Spurfinder wurde in `CBMRoot`, das offizielle Software-Framework des CBM-Experiments, integriert. Diese Integration, unterstützt

Spurkategorie	CA (%)	GNN (%)	Änderung (%)	Anz. MC Spuren
Alle Spuren	95,7	**96,9**	+1,2	412
Primär hoch-p	99,1	**99,2**	+0,1	232
Primär niedrig-p	97,3	**98,2**	+0,9	97
Sekundär hoch-p	95,5	**96,7**	+1,2	29
Sekundär niedrig-p	78,6	**84,3**	+5,7	53
Klon-Rate	**5,1**	**5,1**	0,0	–
Ghost-Rate	**2,2**	2,9	-0,7	–
Gefundene MC Spuren	394	**399**	+5(+1,2%)	412
Zeit, s/Evt	**0,5**	4,8	–	–

Table A.4: Vergleich der Effizienzen von CA und GNN über verschiedene Spurkategorien für 10 AGeV Au+Au Kollisionen. Die Ergebnisse sind über 1000 Ereignisse gemittelt.

durch das `ANN4FLES`-Paket, ermöglicht die Ausführung des Algorithmus als Standardmodul in der Rekonstruktionskette. Sie erlaubt einen direkten Vergleich von Speichernutzung, Ausführungszeit und physikalischer Leistungsfähigkeit mit dem CA-basierten Spurfinder unter Verwendung realistischer Simulationen, die detaillierte Detektorgeometrie, Materialbudgets, Magnetfeldkarten und Detektorantworten beinhalten.

Tabelle A.4 fasst die Rekonstruktionseffizienzen für zentrale Au+Au-Kollisionen bei 10 AGeV zusammen. Der GNN-basierte Spurfinder erzielt über alle Spurkategorien hinweg konsistent höhere Effizienzen als der CA-Algorithmus. Die Verbesserung ist bei Sekundärteilchen mit niedrigem Impuls ($p < 1\,\mathrm{GeV}/c$) am stärksten ausgeprägt; hier steigt die Effizienz um etwa 5,7% (von 78,6% auf 84,3%) in zentralen Kollisionen und um 7,1% (von 80,2% auf 87,3%) in Minimum-Bias-Kollisionen.

Wichtig ist, dass diese Effizienzgewinne nicht die Spurqualität beeinträchtigen. In zentralen Kollisionen zeigen beide Algorithmen ähnliche Klon- und Ghost-Raten. Die Treffereffizienz (Hit Efficiency) für zentrale Kollisionen beträgt 86,7% für den GNN-basierten Ansatz, verglichen mit 83,8% für den CA-Spurfinder. Ebenso liegt die Trefferreinheit (Hit Purity) für zentrale Kollisionen beim GNN-basierten Spurfinder bei 97,9%, verglichen mit 98,1% beim CA-

basierten Spurfinder. Der GNN-basierte Spurfinder ist signifikant effektiver bei der Rekonstruktion kurzer Spuren (4 und 5 Treffer) und zeigt hier eine fast 10%ige Verbesserung der Effizienz. Bemerkenswert ist, dass der GNN-basierte Algorithmus besonders bei der Rekonstruktion schwieriger Spuren, wie solchen, die nahe am Strahlrohr verlaufen oder sich in der Nähe der Akzeptanzgrenzen des Detektors befinden, hervorragende Leistungen erbringt.

Die Spurqualität des GNN-basierten Spurfinders wird anhand von Residuen- und Pull-Verteilungen der Spurparameter an den Positionen des ersten und letzten Treffers jeder rekonstruierten Spur bewertet. Die Mittelwerte der Verteilungen liegen für alle Parameter sehr nahe bei Null, was auf eine unverzerrte (unbiased) Rekonstruktion hinweist. Die Breite der Residuenverteilungen der Spurpositionen x und y spiegelt korrekt die intrinsische räumliche Auflösung des Detektors wider. Die Breite der Impulsresiduen beträgt $1,5\%$, was nahe an den Anforderungen von CBM liegt. Alle Breiten der Pull-Verteilungen liegen nahe bei Eins, was die Korrektheit des Fit-Verfahrens bestätigt.

Die gesteigerte Rekonstruktionseffizienz für Sekundärteilchen mit niedrigem Impuls überträgt sich direkt in erhöhte Ausbeuten (Yields) von kurzlebigen Teilchen. Solche Teilchen zerfallen typischerweise zu schnell, um direkt rekonstruiert zu werden; stattdessen müssen sie indirekt über die Trajektorien ihrer Zerfallsprodukte rekonstruiert werden. Um den physikalischen Einfluss der verbesserten Sekundärteilchenrekonstruktion zu quantifizieren, wurde die simultane Rekonstruktionseffizienz aller Tochterspuren für zwei Benchmark-Kanäle, K^0_S und Λ, evaluiert. Tabelle A.5 fasst den Anteil der kurzlebigen Teilchen zusammen, für die alle Zerfallsprodukte erfolgreich rekonstruiert wurden. Diese Ergebnisse zeigen, dass sich die verbesserte Spurfindungsleistung direkt auf höhere Ausbeuten in zentralen Physikkanälen auswirkt.

Spurkategorie	CA (%)	GNN (%)	Änderung (%)
K^0_S	82,8	**86,6**	+3,8
Λ	81,1	**84,5**	+3,4

Table A.5: Vergleich der Rekonstruktionseffizienz aller Zerfallsprodukte mit CA- und GNN-basiertem Spurfindungsalgorithmus in zentralen Au+Au Kollisionen bei 10 AGeV. Die Effizienzen sind über 1000 simulierte Ereignisse gemittelt.

Die aktuelle CPU-Implementierung des GNN-Trackers ist etwa eine

Größenordnung langsamer als der hochoptimierte CA-Tracker. Diese Latenz resultiert primär aus der relativ schlechten CPU-Leistung bei der Inferenz neuronaler Netze in Kombination mit der absichtlich breiter angelegten kombinatorischen Exploration des Algorithmus, die für die Erzielung der höheren Effizienz erforderlich ist. Deep-Learning-Workloads sind jedoch von Natur aus für parallele Hardware geeignet. Um die zu erwartende Leistung auf Beschleunigern zu untersuchen, wurde eine Proof-of-Concept GPU-Implementierung unter Verwendung der `xpu`-Bibliothek entwickelt, einer Backend-agnostischen Abstraktionsschicht, die Portabilität zwischen GPU- und anderen Beschleunigerarchitekturen ermöglicht. Obwohl die ereignisweise Verarbeitung signifikante Overheads beim Datentransfer verursacht, die die Auslastung (Occupancy) des Geräts einschränken, zeigen alleinstehende Kernel-Benchmarks einen vielversprechenden Durchsatz: durchschnittliche Ausführungszeiten von etwa 30 ms pro Minimum-Bias-Ereignis und 120 ms für zentrale Au+Au-Ereignisse. Diese Messungen deuten darauf hin, dass ein vollständig gebatchter, GPU-residenter Rekonstruktionsworkflow beträchtliche Geschwindigkeitssteigerungen gegenüber der CPU-Implementierung und potenziell sogar gegenüber dem CA-basierten Spurfinder erzielen könnte.

Die Generalisierungsfähigkeit des Modells wurde über mehrere Kollisionssysteme hinweg bewertet: 4 und 20 AGeV Au+Au sowie 28,3 AGeV p+Au. Diese decken den gesamten Bereich der für CBM erwarteten Ereigniskomplexitäten ab. Obwohl der GNN-Tracker ausschließlich auf zentralen 10 AGeV Au+Au Kollisionen trainiert wurde, übertraf er den CA-Finder in allen getesteten Systemen konsistent. Diese Robustheit deutet darauf hin, dass das Netzwerk die zugrundeliegenden geometrischen und topologischen Eigenschaften von Teilchentrajektorien im CBM-Detektor gelernt hat, anstatt sich auf eine spezifische Kollisionsumgebung zu überanpassen (Overfitting). Weitere Leistungssteigerungen werden durch gezieltes Hyperparameter-Tuning für einzelne Kollisionssysteme erwartet.

Eine hybride Rekonstruktionsstrategie stellt eine weitere vielversprechende Anwendung des entwickelten Algorithmus dar. Der CA-basierte Spurfinder, der den Primärvertex als anfängliche Randbedingung für die Kalman-Filter-Extrapolation nutzt, funktioniert gut für Primärspuren, ist jedoch inhärent limitiert bei Sekundärspuren, deren Trajektorien nicht vom Primärvertex ausgehen. Im Gegensatz dazu lernt die GNN-basierte Methode, die direkt auf Daten trainiert wird, Repräsentationen von Sekundärspurenmustern, ohne auf Vertex-

Randbedingungen angewiesen zu sein. Eine iterative Rekonstruktionskette, in der der CA-Finder für die anfänglichen Iterationen der Primärspuren und anschließend der GNN-Tracker für die Wiederherstellung der Sekundärspuren verwendet wird, könnte die Stärken beider Ansätze vereinen.

Eine höhere Rekonstruktionseffizienz wird die Triggerentscheidungen direkt verbessern. Selbst wenn der GNN-Tracker letztendlich nicht die strengen Echtzeitanforderungen von CBM erfüllen sollte, bleibt er als Benchmark-Tool für die Algorithmenvalidierung und als Rekonstruktionsmethode auf Analyseebene, wo Zeitbeschränkungen entspannter sind und selbst moderate Effizienzverbesserungen signifikante physikalische Auswirkungen haben können, äußerst wertvoll.

Fazit

Diese Arbeit stellt einen neuartigen GNN-basierten Spurfindungsalgorithmus vor, der eine deutlich verbesserte Rekonstruktionseffizienz – insbesondere für Sekundärspuren mit niedrigem Impuls – im Vergleich zum standardmäßigen CA-basierten Ansatz erzielt. Diese Verbesserungen führen zu erhöhten Ausbeuten für kurzlebige Teilchen, was einem breiten Spektrum physikalischer Analysen zugutekommt. Eine Proof-of-Concept GPU-Implementierung zeigt vielversprechende Leistungsmerkmale und legt nahe, dass der Ansatz gut für den Einsatz auf Beschleunigern geeignet ist. Insgesamt erweist sich der entwickelte Algorithmus als starker Kandidat für die zukünftige Integration in das CBM-Rekonstruktionsframework und als allgemein anwendbare Strategie für die Spurfindung in Experimenten mit hohen Raten und hohen Multiplizitäten.