INAUGURAL-DISSERTATION

submitted to the

Combined Faculty of Mathematics, Engineering and Natural Sciences

of the

Ruprecht–Karls-University
Heidelberg

for the Degree of

Doctor of Natural Sciences

put forward by

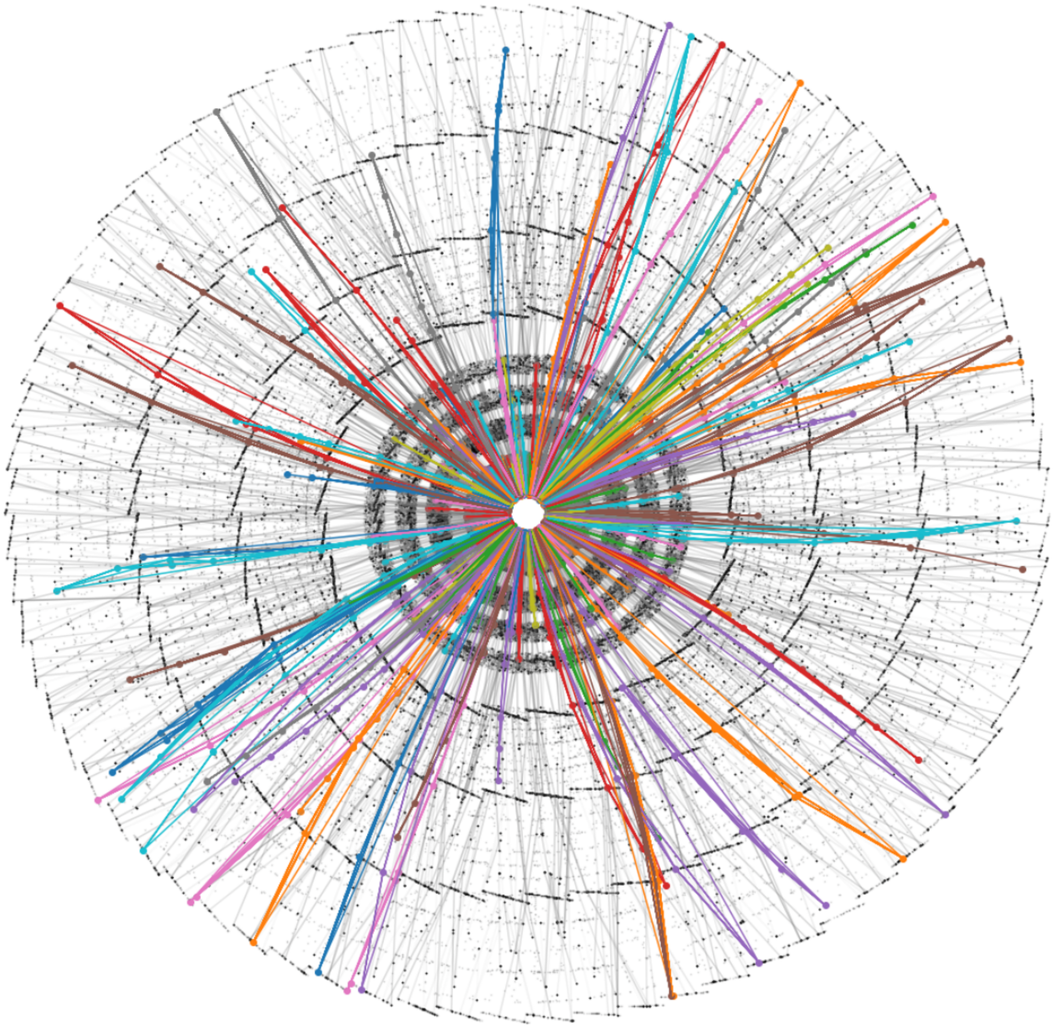
Aleksandra Wiktoria Poreba, M.Sc.

from Nowy Sacz, Poland

Day of the oral exam: ....

How to connect the dots very fast?

High-Performance Heterogeneous Particle Track Reconstruction for the ATLAS Phase-II High Level Trigger



Supervisor: Professor Dr. Holger Fröning

Abstract

The ATLAS experiment is a key project in high-energy particle physics exploring particle collisions at unprecedented energies to recreate early-universe conditions and probe phenomena that occur at extreme scales. The upcoming high luminosity upgrade of the LHC will significantly increase the collision rate and energy; the higher data volume and collision complexity necessitate a major upgrade of the ATLAS detector to consist of a more granular tracking system. Due to much higher data rate and processing complexity, it is crucial to optimise the collision selection system (trigger system) and its most computationally expensive component - the track reconstruction algorithms. This work explores advanced optimisation methods, including graphics card acceleration and machine learning, to enhance computational efficiency and effectively manage the increased data throughput and complexity of the upgraded detector.

The first considered approach optimises the track reconstruction algorithm used in the ATLAS trigger system. By employing the track seeding on the graphic card accelerator and adjusting the track seed selection criteria, the final performance was improved by 95%, achieving an average processing time per event of 1.16 s. The performance was evaluated on different graphics cards, considering their limitations, with NVIDIA RTX 5000 Ada achieving the best results due to its exceptionally high number of processing cores.

The second part of this work focuses on the application of machine learning techniques to particle track reconstruction. A novel Interaction Graph Neural Network (IGNN) demonstrates competitive reconstruction accuracy; however, it is known to be resource-consuming. To address these computational challenges, two optimisation strategies are proposed, aimed at reducing both memory consumption and inference time without compromising model performance.

The instantaneous memory footprint of the model was reduced by partial processing (substepping). Memory consumption can be decreased by approximately 30% without an increase in processing time. Further memory reductions are achievable by adjusting the size of partitions, enabling the deployment of the IGNN on memory-constrained GPUs and allowing parallel processing, depending on the available hardware resources.

The second discussed compression technique is structured pruning of IGNN, where by removing the least important groups of parameters, the model size is reduced. A selection of pruning techniques applied to Graph Neural Networks (GNN) was analysed to determine the most effective methodology for GNN compression. The final pruning configuration achieves up to 20% improvement in computational performance without compromising model accuracy. Furthermore, per-layer sensitivity was analysed and incorporated in the pruning strategy to guide layer-wise pruning aggressiveness, enabling further model size reduction by 20% while maintaining reconstruction accuracy. The pruning strategy was evaluated on the standard GNN benchmark models, demonstrating satisfying performance gains. The performance of IGNN was evaluated on different graphics cards, considering their limitations, with NVIDIA RTX A100 achieving the best results due to its highly efficient memory throughput.

Zusammenfassung

Das ATLAS Experiment ist ein Schlüsselprojekt der Hochenergie-Teilchenphysik, das Teilchenkollisionen bei bislang unerreichten Energien untersucht, um Bedingungen des frühen Universums nachzubilden und Phänomene zu erforschen, die bei extremen Skalen auftreten. Das bevorstehende Hochluminositätsupgrade des LHC wird die Kollisionsrate und -energie signifikant erhöhen. Das dadurch entstehende größere Datenvolumen und die erhöhte Komplexität der Kollisionen erfordern eine umfassende Verbesserung des ATLAS-Detektors, insbesondere durch ein hochgranulares Spurdetektorsystem. Aufgrund der stark gesteigerten Datenrate und Verarbeitungskomplexität ist die Optimierung des Triggersystems und dessen rechenintensivsten Bestandteilen – den Spurrekonstruktionsalgorithmen – von zentraler Bedeutung. Die vorgestellte Arbeit untersucht anspruchsvolle Optimierungsmethoden, darunter die Beschleunigung durch Grafikkarten sowie den Einsatz von maschinellen Lernens, um die Recheneffizienz zu steigern und den erhöhten Datendurchsatz sowie die gesteigerte Komplexität des verbesserten Detektors effektiv zu bewältigen.

Der erste betrachtete Ansatz optimiert den im ATLAS-Triggersystem verwendeten Spurrekonstruktionsalgorithmus. Durch die Implementierung des Track Seedings auf einem GPU Beschleuniger und die Anpassung der Auswahlkriterien des Track Seedings konnte die Leistung um 95% verbessert werden, bei einer durchschnittlichen Verarbeitungszeit von 1.16 s pro Ereignis. Die Leistung wurde auf verschiedenen GPUs evaluiert, wobei deren Einschränkungen berücksichtigt wurden; die NVIDIA RTX 5000 Ada erzielte aufgrund ihrer außergewöhnlich hohen Anzahl an Verarbeitungskernen die besten Ergebnisse.

Der zweite Teil dieser Arbeit befasst sich mit dem Einsatz von Techniken des maschinellen Lernens zur Teilchenspurrekonstruktion. Ein neuartiges Interaction Graph Neural Network (IGNN) zeigt eine konkurrenzfähige Rekonstruktionsgenauigkeit, ist jedoch sehr ressourcenintensiv. Zur Bewältigung dieser rechentechnischen Herausforderungen werden zwei Optimierungsstrategien vorgeschlagen, die sowohl den Speicherbedarf als auch die Inferenzzeit reduzieren, ohne die Modellleistung zu beeinträchtigen.

Der momentane Speicherbedarf des Modells wurde durch partielle Verarbeitung (Substepping) verringert. Dadurch kann der Speicherverbrauch um etwa 30% gesenkt werden, ohne die Verarbeitungszeit zu erhöhen. Weitere Reduktionen sind durch Anpassung der Partitionsgröße möglich, was den Einsatz des IGNN auf speicherbeschränkten GPUs ermöglicht und – abhängig von den verfügbaren Hardwareressourcen – parallele Verarbeitung erlaubt.

Die zweite untersuchte Kompressionstechnik ist das strukturierte Pruning des IGNN, bei dem durch Entfernen der am wenigsten signifikanten Parametergruppen die Modellgröße reduziert wird. Eine Auswahl an Pruning-Methoden für Graph Neural Networks (GNNs) wurde analysiert, um die effektivste Methode zur GNN-Kompression zu ermitteln. Die finale Pruning-Konfiguration erreicht eine Leistungssteigerung von bis zu 20% bei unveränderter Modellgenauigkeit. Darüber hinaus wurde die schichtweise Sensitivität analysiert und in die Pruning-Strategie integriert, um die Aggressivität des Prunings pro Schicht zu steuern. Dies ermöglichte eine zusätzliche Reduktion der Modellgröße um 20% bei gleichbleibender Rekonstruktionsgenauigkeit. Die Pruning-Strategie wurde anhand standardisierter GNN Benchmarks evaluiert und zeigte signifikant Leistungsgewinne. Die Leistung des IGNN wurde auf verschiedenen Grafikkarten untersucht; die NVIDIA RTX A100 erzielte aufgrund ihres hoch effizienten Speicherdurchsatzes die besten Ergebnisse.

Acknowledgments

First and foremost, I would like to express my gratitude to my advisor, Prof. Dr. Holger Fröning. Thank you for taking the risk of supervising this project in collaboration with CERN. Over the years, your guidance and insightful feedback have provided me with a fresh perspective on the challenges within the ATLAS environment and opened doors for collaboration outside of ATLAS.

I would also like to thank my co-advisor, Dr. Brian Petersen, for working with me on the fifth floor at CERN. You always had time to discuss my work, offer me a broader perspective and read my thesis over and over. Thank you for believing in me enough to accept me into your group, even though I am not a standard experimental particle physics student.

I cannot forget about Dr. Mark Stockton, who supervised me in the first year of my doctoral studies. You were working with me since I first arrived at CERN, and despite my initial worries after your leaving, I managed to finish this work! Thank you for passing me your trigger expertise and helping to shape me as a scientist.

Furthermore, I would like to thank the members of the teams working with me on the G-300 and G-400 EF-Tracking pipelines. I would like to particularly mention Dr. Dmitry Emeliyanov for revealing the mysteries of the track seeding, and Dr. Benjamin Huth, Prof. Dr. Alina Lazar and Dr. Santosh Parajuli for help and guidance with the IGNN.

I must also thank my colleagues, especially Stefanie, Marco and Stef for all the lunches, coffees, beers and lifts over the last years, and my dear friend Janina for always being there for me. Thank you, John, for explaining to me all the mysterious particle physics I asked about and for believing in me. I have to mention the "generations" of Girls Office as well - every single one of you is a different scientist, inspiring me every day. I cannot forget about my friends from Poland: Zuza, Marysia, Gosia and others - thank you for staying with me throughout those years, even with the distance between us. And a huge thank you to my dear family - I hope you forgive me for staying this far and not visiting you often enough. I am very grateful for your support over the years, always motivating me to pursue knowledge.

Last, but definitely not least, I would like to acknowledge the funding I received, without which this work would not have been possible:

This work has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research (grant no. 13E18CHA)

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a Department of Energy User Facility under Contracts No. DE-AC02-05CH11231 (CompHEP Exa.TrkX)

Credits for the illustration on the title page to Dr. Daniel Murnane

Dla dziadka Kazia

Table of contents

Chapter 1

Introduction

Since the mid-20th century, computers have been assisting humans with algorithm executions. With the technological advancements over the years, microprocessors have become progressively smaller and more cost-effective, enabling significantly more complex computations to be performed at higher speeds—an evolution historically described by Moore's Law. However, this trend is approaching its physical limits, as further miniaturisation of transistors is constrained due to quantum effects and heat dissipation challenges. Initially, the performance was improved by task parallelisation; however, the speedup of a program is limited by the fraction of the code that must be executed sequentially, as described by Amdahl's Law.

To address the limitations of the CPU-based processing, researchers and engineers have increasingly turned to specialised hardware accelerators, such as Graphics Processing Units (GPUs). Unlike traditional CPUs, GPUs are designed to exploit massive parallelism, enabling them to process large volumes of data simultaneously. Prioritising throughput and parallel computation over complex control logic has proven to be an effective strategy with significant performance improvements across various computational tasks.

The second approach to improve the computing performance involves exploring alternatives to traditional algorithms, such as machine learning. These methods can approximate complex functions often more efficiently than conventional rule-based algorithms while preserving accuracy. The combination of advanced algorithms and high-performance computing architectures has demonstrated significant improvements in both execution time and accuracy across a variety of scientific and engineering applications.

One of the key projects facing the computing challenge is the ATLAS experiment [1], [2] at CERN, focusing on the general high-energy particle physics research. The collisions of particles in the centre of ATLAS reach the highest collision energies ever achieved, enabling recreation of early-universe conditions and revealing processes that only emerge at extreme scales. The data recorded by the detector for each particle collision is processed in real time in order to select only the most interesting collisions for further analysis. Until now, this processing has been performed by using a CPU computing farm. Since the beginning of the experiment, significant efforts have been made to optimise computational performance by improving the algorithms and adoption of multi-threading parallelism.

However, during the years 2026-2030, the Large Hadron Collider (LHC) [3], the accelerator providing the particle collisions in ATLAS, will be upgraded and able to deliver many more particle collisions with higher energy (High-Luminosity LHC [4]). Moreover, to accommodate the increased collision rate, the ATLAS detector will undergo a significant upgrade that will increase its granularity, significantly increasing the data volume and processing complexity.

The real-time data processing in the ATLAS experiment has its processing constraints. With the available computing resources, planned to be expended for the future data taking, after applying low-latency filtering, the recorded collisions must be processed at a rate of 1 MHz to prevent data loss. In order to achieve this limit after the incoming upgrades, the processing algorithms need to be significantly optimised. The most time-consuming part is the reconstruction of the trajectory of the particles passing through the detector. This task involves reconstructing the track left by each particle based on the detector readouts, while accounting for the approximate shape of the tracks and relevant experimental conditions. The performance of the current algorithm, fast track reconstruction [5], scales non-linearly with the increasing number of collisions. The increased granularity of the detector introduces additional complexity to the data processing.

To improve the computing performance of this algorithm while achieving satisfactory accuracy, the ATLAS Collaboration is considering several approaches, including new algorithms and hardware acceleration techniques. The different pipelines are prepared and optimised for the final technology choice, planned to take place in December 2025. This work focuses on two of the proposed solutions, each leveraging the massive parallelism offered by GPUs. The first approach enhances the computing performance of the traditional algorithm by offloading a portion of its execution to the GPU. The second solution proposes a machine learning-based algorithm, with a complete deployment on the GPU. Both pipelines face their challenges, which include suboptimal accuracy, high memory footprint, and extended processing times.

The work will focus on achieving and preserving the required accuracy while applying extensive performance optimisations specifically tailored for GPU accelerators. Both high-level and low-level optimisation strategies will be explored across multiple software frameworks. In addition to GPU-specific performance enhancements, machine learning optimisation techniques adapted for GPU architectures will also be studied. The proposed methods will be evaluated in the context of ATLAS reconstruction as well as on standard benchmark problems to examine their robustness and generalisation. A detailed performance study will be conducted, with attention to the specific features of the GPU as well as variations across different hardware platforms.

Contributions

In this manuscript, the following main contributions will be presented:

1. Analysis of GPU-accelerated Track Seeding—a component of the ATLAS track reconstruction algorithm—focusing on limitations and required optimisations to meet High-Luminosity LHC workloads. Performance is evaluated across diverse hardware architectures with optimisations targeting massively parallel processors. (Chapter 3)

2. Memory footprint reduction for the Interaction Graph Neural Network via iterative partial processing, achieving lower memory consumption without compromising model accuracy; execution time penalties are mitigated depending on step size. (Chapter 4)

3. Search for optimal structured pruning configurations for Graph Neural Networks, evaluated on both benchmark models and the Interaction Graph Neural Network. The study measures the impact of pruning on inference performance across multiple hardware platforms and provides an in-depth analysis of the observed limitations. (Chapter 5)

Preview

This work is organised into six chapters. The current, introductory Chapter 1 describes the motivation of this work and its context. It is followed by a Background Chapter 2, introducing the reader to the ATLAS experiment, its upgrade and the upcoming challenges. The particle track reconstruction task is introduced, as well as the Graph Neural Networks. The two auxiliary chapters are followed by three core Chapters 3-5, each presenting a separate track reconstruction algorithm optimisation. Each of the core chapters consists of a short introduction, related works, method description, experiments and discussion of the results. The final Chapter 6 summarises the presented work.

Contribution 1: GPU-accelerated Track Seeding Optimisation

The first contribution focuses on upgrading the ATLAS track reconstruction algorithm. The selection criteria, which enable rejection of incorrect particle trajectories, are adjusted to accommodate the higher granularity of the upgraded detector and accept only the tracks that target particles can create. By reducing the combinatorial complexity by rejection, the accuracy and the computational performance of the algorithm are substantially improved. The algorithm is additionally analysed and fine-tuned to maximise utilisation of the GPU's computational capabilities. The search for the optimal GPU configuration for such an algorithm is conducted, with attention to present bottlenecks. The performance across different hardware platforms is analysed, taking into account the architectural characteristics and capabilities of the available accelerator cards.

Contribution 2: Memory Footprint Optimisation of Interaction Graph Neural Network

The focal point of the second contribution is the reduction of the memory footprint of the machine learning based trajectory reconstruction algorithm, Interaction Graph Neural Network (IGNN). The memory consumption is reduced by explicitly splitting the most memory-consuming tensors into batches of optimal size (substeps). Since only the temporary data structure is divided, the accuracy is not affected. To study the impact of this optimisation on the computing performance, a search for the optimal substep size is conducted, considering different aspects of the resource utilisation of the algorithm. The performance is studied on hardware with different on-chip memory capacities to ensure that the algorithm is available even on the cost-efficient units.

Contribution 3: Inference Optimisation of Interaction Graph Neural Network

The last contribution concentrates on improving the inference time and model size of the IGNN algorithm on GPU by applying a structured pruning model compression method. By removing the least essential parameters of the model, the number of calculations to perform is reduced. Various configurations of the pruning procedure are studied to identify the setting that achieves the highest compression while preserving accuracy. Additionally, a sensitivity-aware structured pruning is introduced and evaluated against traditional structured pruning. The impact of the compression method on the computing performance is considered on the selection of different hardware platforms. The pruning configuration achieving the best results is evaluated on benchmark models to assess the generalisation of the study.

Publications

The following works were published related to the presented studies, exploring the heterogeneous track reconstruction as well as its context - the ATLAS trigger

- Poreba A. on behalf of the ATLAS Collaboration, *Operational experience with the new ATLAS HLT framework for LHC Run 3*, EPJ Web of Conf. Volume 295 (2024), Proceedings of 26th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2023)
- The ATLAS Collaboration, *The ATLAS Trigger System for LHC Run 3 and Trigger performance in 2022*, Journal of Instrumentation 19 (2024), contributed to the HLT Performance chapter
- Poreba A., Fröning H. on behalf of the ATLAS TDAQ Collaboration, *Performance of the ATLAS GNN4ITk Particle Track Reconstruction GPU pipeline*, Proceedings of 27th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2024)
- Poreba A., Barley D., Petersen B., Fröning H., *Accelerating Interaction Graph Neural Network Inference on Massively Parallel Accelerators by Sensitivity-aware Structured Pruning*, (under a journal review)
- The GNN4ITk group, *Improving Computational Performance of ATLAS GNN4ITk Track Reconstruction Pipeline* (in preparation, contributing to the computing performance chapter)

Chapter 2

Background

2.1 ATLAS Experiment

The ATLAS experiment [1], [2] at the European Organisation for Nuclear Research (CERN) is one of four major experiments built on the Large Hadron Collider (LHC) accelerator [3] to study and expand the understanding of particle physics. It is a general-purpose detector (as well as the CMS [6] experiment), focused on the Standard Model studies and new particle searches.

2.1.1 The Large Hadron Collider

The LHC is the world's largest hadron collider, able to reach the world's highest centre of mass energy of the collision of 13.6 TeV. The high energy allows for the study of processes that were not possible before, including the process of Higgs Boson production, discovered in 2012 [7].

The LHC is capable of producing proton-proton collisions as well as heavy ion collisions, including lead, oxygen, or neon. The particles arranged in beams follow a series of accelerators, gradually increasing their energy. Before entering the LHC, the beams are split into two and circulated in opposite directions, to eventually collide in the centre of the experiments (called *interaction points*, IPs or *beam spots* referring to the interaction regions) every 25 ns (for proton-proton program, 50 ns for heavy ion). The beams are steered by a series of magnets: superconducting dipole magnets responsible for bending the beams and quadrupole magnets used to focus the beams.

To quantify the collision rate and the amount of collected data by the experiments, a parameter called *luminosity* reflects how many particles are delivered to the interaction point per unit area per unit time. The higher the luminosity, the higher the chances of observing rare physics processes.

In the accelerator, the particles are grouped in *bunches*, circulated, and accelerated together. Depending on the number of particles within a bunch, the number of bunches, and the beam shape, more particles can collide in each crossing. The quantity describing the number of interactions occurring in a single bunch crossing is called *pile-up*.

The LHC and its experiments operate in multi-year-long periods called *"Runs"*, lasting a few years and separated by breaks focused on the upgrades. During the time of writing this thesis (June 2025), Run 3 is approaching its final phase. Two more "Runs" are planned for LHC: Run 4 (2030-2033) and Run 5 (2036 - 2041).

The HL-LHC Upgrade

To push the particle physics knowledge boundaries even further by increasing the luminosity and the centre of mass energy of 14 TeV, a LHC upgrade was proposed, called High Luminosity LHC (HL-LHC) [4]. By upgrading the existing technology, the number of proton-proton interactions per bunch crossing will increase to 140 (or 200 in Run 5).

The substantial rise in collision rates must be processed by the LHC experiments, necessitating significant upgrades to the detector systems and data processing infrastructure.

2.1.2 ATLAS Detector

The ATLAS experiment is a multipurpose particle detector, illustrated in Figure 2.1, consisting of a series of subdetectors to record the properties of the particle collision products.

The ATLAS detector has a cylindrical shape divided into a central Barrel region and two EndCaps at the ends of the Barrel. This layout was optimised to maximise the detector coverage and reconstruction efficiency for the particles produced at LHC. The innermost part of ATLAS includes the Inner tracking Detector (ID) surrounded by a thin superconducting solenoid providing a 2 T axial magnetic field. It consists of silicon pixels, silicon microstrip, and transition

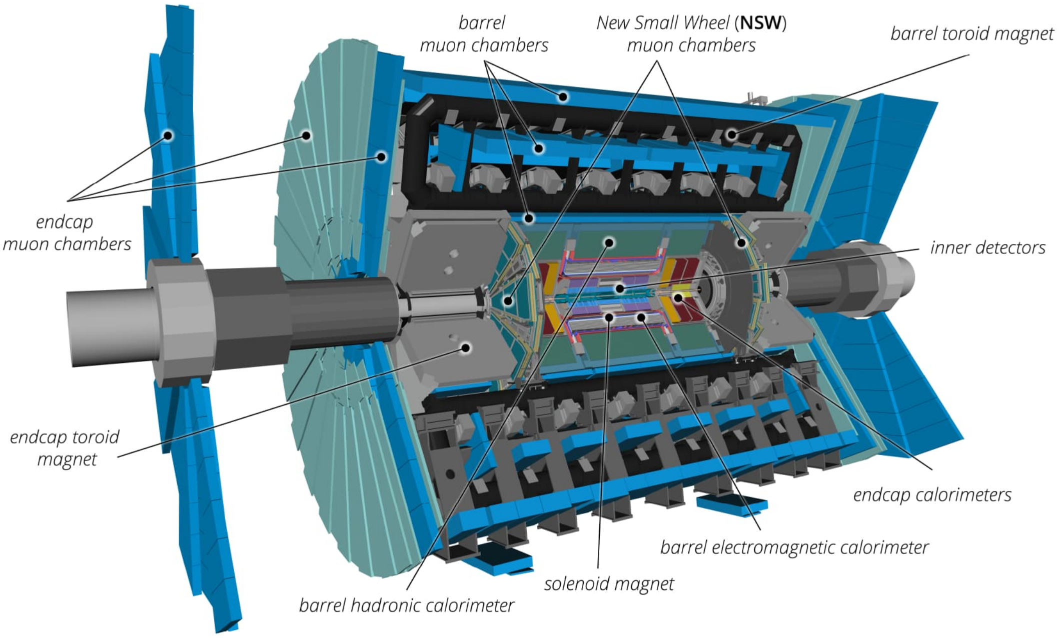


Fig. 2.1 Schematic overview of the ATLAS Detector for Run 3 with highlighted locations of the larger detector sub-systems. [2]

radiation tracking detectors, able to interact with charged particles passing through them. Usually, a particle leaves a maximum of twelve measurements on the silicon detectors, which are used to reconstruct the trajectory of the particle. The solenoidal magnetic field enables the reconstruction of the particle's charge through its effect on the particle's trajectory.

The inner detector is surrounded by calorimeters: electromagnetic and hadronic, enabling the measurement of the energy of a passing particle. The outermost layer of the ATLAS contains a muon spectrometer based on three large superconducting toroidal magnets with eight coils each. The spectrometer allows identification and precise tracking of muons passing through the calorimeters.

The ATLAS Tracker Upgrade

To manage the particle reconstruction under the high-luminosity conditions of the HL-LHC, the ATLAS collaboration initiated an upgrade of the existing tracking detector. The increased number of modules will allow the reconstruction of passing particles on a broader region than before, increasing the physics potential of the ATLAS experiment. Moreover, each layer will contain more sensors than before. Apart from the higher granularity, the new detector will provide higher radiation hardness, read-out speed, and lower maintenance costs.

The planned Inner Tracker (ITk) consists of two technologies: ITk-Pixel [8] and ITk-Strip [9] detectors, constructed entirely with silicon sensors. The layout of the ITk detector is illustrated in Figure 2.2.

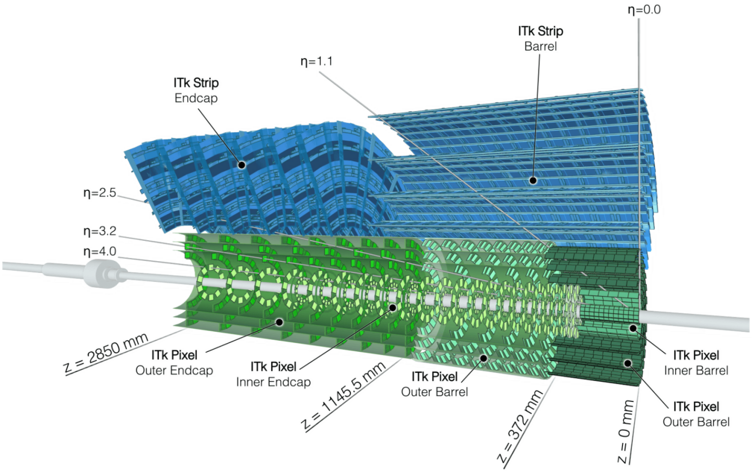


Fig. 2.2 Schematic overview of the ATLAS ITk Detector. [10]

The Pixel detector, the closest one to the LHC beam pipe, is designed to consist of approximately 100 million read-out channels with pixels of dimension $50 \times 50 \mu m^2$ (or $25 \times 100 \mu m^2$ in the Barrel region). This fine granularity enables a precise measurement of the position where the charged particle interacted with the detector (Figure 2.3). The exact location can be reconstructed based on the known spatial coordinates of the activated sensor elements. Additionally, the deposited charge of the particle can be measured by quantifying the electric signal caused by ionisation within the silicon substrate.

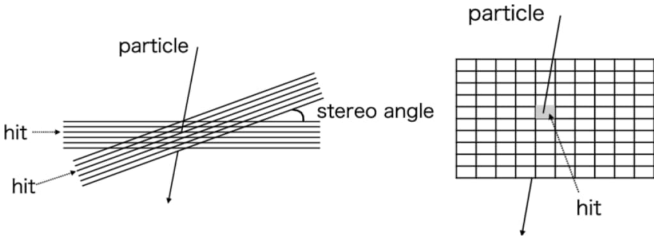


Fig. 2.3 Illustration of interaction of a particle with the ITk-Strip detector (left), leaving two one-dimensional measurements, and the ITk-Pixel (right), leaving two-dimensional measurements [11].

The ITk-Strip detector, located outside of the ITk-Pixel, consists of long, narrow silicon strips rather than small pixels. The detector is designed to consist of 60 million strip channels, each with a width of approximately 75 µm and a length spanning over several centimetres. Similarly to pixel technology, passing charged particles ionise the silicon atoms, creating multiple electron-hole pairs in one strip. The measurement is only one-dimensional; to identify the interaction point between the detector and a particle, two strips need to be considered. As illustrated in Figure 2.3, the strip pairs are tilted by a small stereo angle of 40 mrad, and the intersection of them gives a precise two-dimensional measurement in the module plane, which, combined with the known detector geometry, allows reconstruction of the exact spatial location. In contrast to pixels, strips do not read out the charge of the particle.

Although the strip technology offers lower spatial resolution compared to pixel detectors, it is more cost-effective for covering large surface areas. It requires less material, lighter cooling infrastructure, and a reduced number of read-out channels per unit, reducing the power demand.

In comparison to the ATLAS Inner Detector, the ITk extends the coverage of the ATLAS detector to include more forward tracks, close to parallel to the beam axis. Moreover, the position of the EndCap modules was fine-tuned in comparison to the ID; instead of disks, each layer consists of rings. The detector coverage improvement is illustrated in Figure 2.4, the ITk surface will cover 178 m^2. The number of modules will be increased to 28 000 from the 6000 installed in ID; the comparison of the composition of layers is summarised in Table 2.1. The new design allows for more efficient track reconstruction, with at least nine expected silicon hits per track, compared to seven [12], enabling the detection of particles from high-luminosity collisions.

2.1.3 ATLAS Trigger

The bunch crossings within the ATLAS detector are happening at a rate reaching 40 MHz for the proton-proton program, with the ATLAS detector sampling every 25 ns, synchronously with the beam. However, not all of the collisions contain rare processes - the objects of interest for particle physicists. To reduce the amount of data saved for further studies, a trigger system is used, filtering the signals recorded by ATLAS. The collisions, called *events*, are partially reconstructed to

Table 2.1 Comparison of number of layers of Pixel and rings of Strips for Inner Detector (Run 3) [13] and Inner Tracker (Run 4 and Run 5) [8]. ITk-Pixel EndCap includes inclined Barrel layers.

Number of layers /average rings per layer	Inner Detector (Run 2 and 3)	Inner Tracker (Run 4 and 5)
Pixel Barrel	4	5
Pixel EndCap	3	17
SCT/Strip Barrel	4	4
SCT/Strip EndCap	9	6

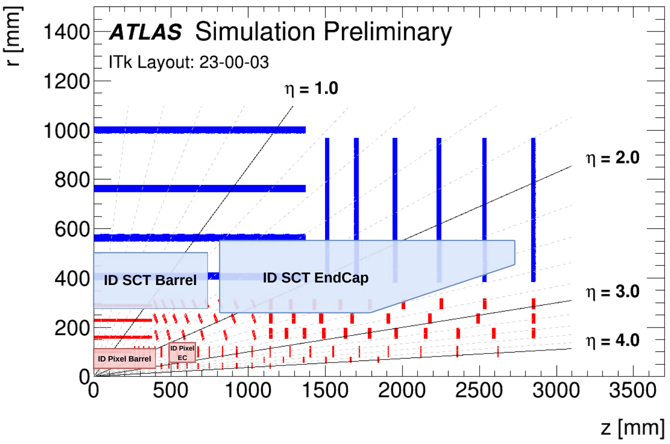


Fig. 2.4 Layout of the Inner Tracker (Run 4 and Run 5) and overlayed geometry of the Inner Detector (Run 3) [14]. The red shapes represent the ID Pixel coverage, and the blue shapes - ID SCT coverage.

decide if they should be saved or discarded.

The ATLAS trigger consists of two levels: the Level 1 (L1), a hardware trigger, and the High Level Trigger (HLT), a software trigger (schematic on Figure 2.5). The L1 is based on coarse data from Muon Spectrometers (L1Muon) and Calorimeters (L1Calo). For every event, the L1 system attempts to verify the 512 physics signatures. If the L1 Trigger passes a collision, it is processed by the HLT, performing a more detailed analysis, for example, including the tracking detector read-out for the reconstruction of the particle path. The collision products are reconstructed, identified, and tested against predefined requirements. The reconstruction-hypothesis steps are organised in a way that the most computationally expensive steps are executed only on a subset of events in order to save resources. An event passing at least one whole chain of the hypothesis is saved for future physics studies.

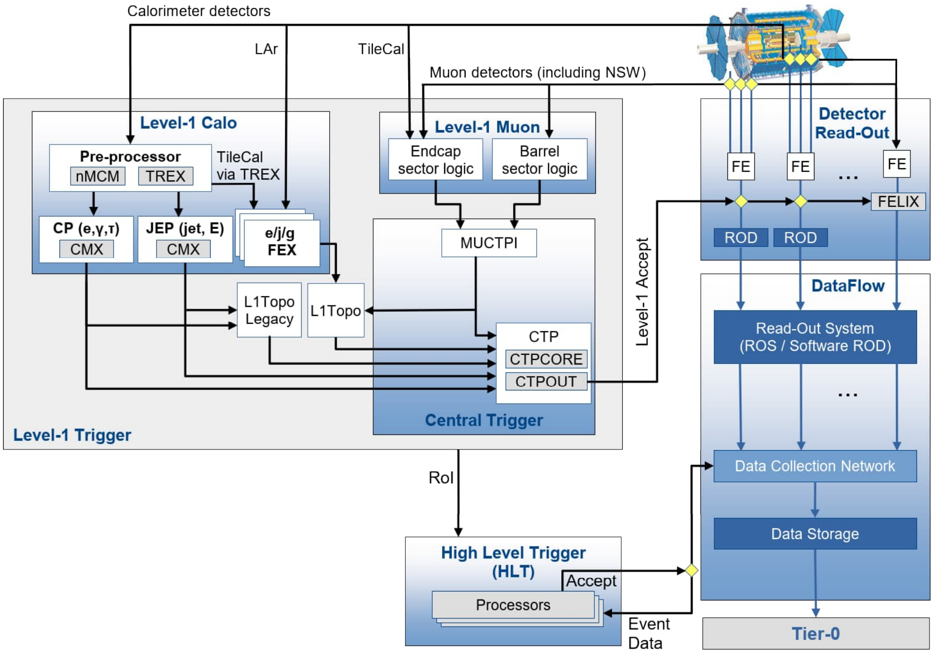


Fig. 2.5 The ATLAS TDAQ system in Run 3 with emphasis on the components relevant for triggering as well as the detector read-out and data flow. [15]

The L1 Trigger, synchronous with the ATLAS Detector read-out, is processing the signals every 25 ns with a latency of 2.5 µs. It is operating on custom-built Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs), creating object candidates and applying predefined selections. The servers are located close to the detector itself to ensure low data transfer latency. After an event is accepted, the data from the L1 trigger and detector read-out is transferred to the HLT computing farm with approximately 60 000 CPU cores, upgraded for Run 3 with dual-processor servers with AMD EPYC 7302 CPUs. The total farm performance improved from 1.2×10^6 HS06 [1] at the end of Run 2 to 2×10^6 HS06 since 2023.

The reconstruction of an event takes place twice in the ATLAS event processing pipeline: first, during the event selection by the trigger (online) and later, during the detailed reconstruction for physics analysis purposes (offline). Since

[1] HS06 - HEP-SPEC06 benchmark [16], a measure of the CPU performance.

Run 2, the code of reconstruction algorithms is shared between online and offline in the software package called Athena [17]; however, the settings on which and how the algorithms are executed can differ. One of the differences is the regional reconstruction in the online trigger. To reduce the data volume necessary to process, the reconstruction is restricted to small regions guided by the L1 trigger, called Regions-of-Interest (RoIs) [18].

Online event reconstruction has specific performance requirements regarding its throughput. The L1 trigger, by applying its selection, reduces the output rate to 100 kHz, which constitutes the input of the HLT. To process all the incoming events, the average reconstruction time of one event cannot exceed 500 ms, depending on the available CPU farm. To maximise the computing performance, the executed reconstruction algorithms need to be analysed in detail and optimised. Table 2.2 contains a summary of the processing time spent in different components of the HLT. Most of the time (59%) is spent on the reconstruction of the particle path.

Table 2.2 Average distribution of HLT processing time per event in 2022 [15].

HLT Component	Total time [%]
ID reconstruction	59
Muon reconstruction	14
Calorimeter reconstruction	11
Combined reconstruction and hypothesis algorithms	8
Trigger infrastructure	2
Other	6

The ATLAS Trigger Upgrade

The High Luminosity upgrade will result in a significantly higher number of particles produced per collision (an event with pile-up of 200 is illustrated in Figure 2.8), thereby increasing the volume of data recorded by the ATLAS detector. The dataload will directly impact the performance of the algorithms reconstructing the event, growing exponentially with the pile-up (Figure 2.6). The track reconstruction algorithm will have to disentangle the trajectories of approximately 1500 particles per event (for pile-up 200, compared to 800 for pile-up 70), as presented in Figure 2.7, and utilise significantly more read-out data as a result of the Inner Tracker upgrade. Moreover, the computing requirements are not eased - the Event Filter (EF) software trigger, replacing the HLT, must operate at a rate of 1 MHz - over ten times higher than in Run 3 HLT, and the EF track reconstruction algorithms must execute at a rate of 200 kHz.

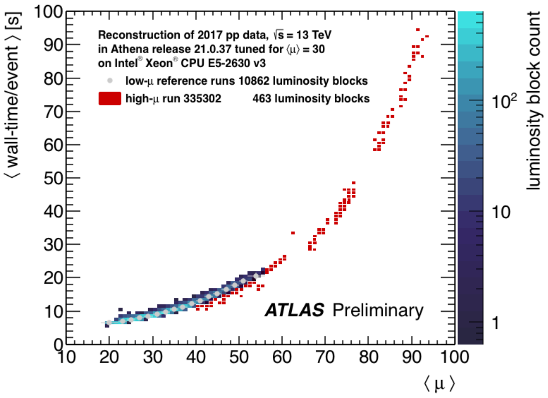


Fig. 2.6 The dependency of reconstruction wall time per event on the average number of interactions per bunch crossing ($< \mu >$) is shown for the Run 2 Inner Detector reconstruction with default tracking cuts. [19]

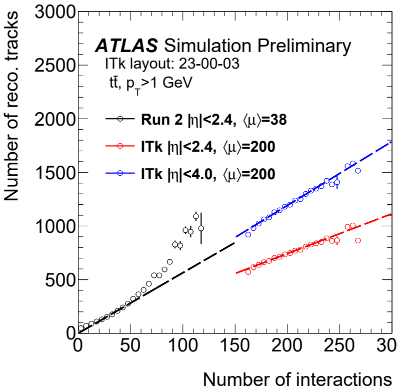


Fig. 2.7 Number of reconstructed tracks per event with $p_T > 1\,\text{GeV}$ for $t\bar{t}$ events at $< \mu >= 200$ with the updated ITk layout compared with the Run 2 detector at $< \mu >= 38$ [14].

In order to reach the expected computing capabilities of EF, the ATLAS software trigger requires a significant upgrade. Both new algorithms and hardware acceleration techniques are considered promising solutions for efficient processing, as described in this work.

2.1.4 ATLAS Upgrade Input Data

To prepare and validate the reconstruction software ahead of Run 4, featuring an improved tracking detector and substantially higher luminosity, simulated datasets reflecting the anticipated future conditions are essential.

One of the widely used data samples is simulated top quark pair production $t\bar{t}$, one of the most interesting outcomes of proton-proton collision for ATLAS researchers. It allows the study of the top quark, the heaviest particle of the Standard Model, as well as searches for new physics beyond the Standard Model. From the perspective of the track reconstruction, the decay of a top quark pair

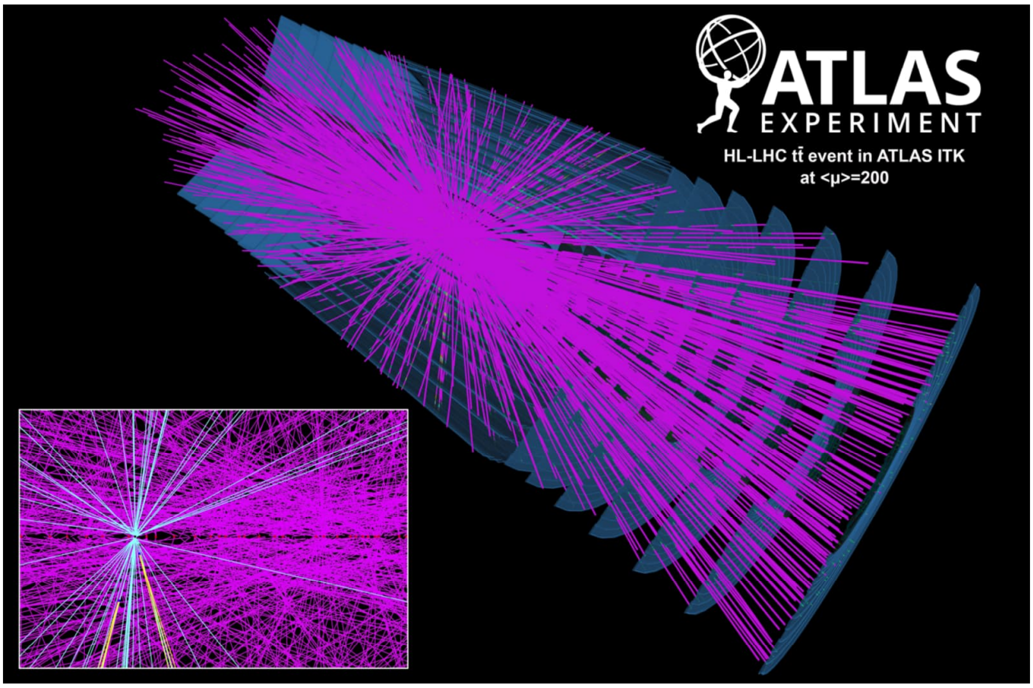


Fig. 2.8 A simulated proton-proton collision event at average pile-up of 200 collisions per bunch crossing, with an ITk layout including the very forward extension. The bottom-left inset is a 2D $z - r$ view of the interaction region. The vertical scale is 2.5mm and the horizontal one 12cm. All reconstructed particle tracks have pT>1 GeV. The tracks coming from the primary vertex are coloured in cyan. Two secondary vertices can be reconstructed, and the tracks coming from them are highlighted in yellow. [20]

produces a large number of charged particles that interact with the tracking detector. This way, the accuracy and computing efficiency can be evaluated under high-load conditions.

However, the reconstruction of such a task is time-consuming and highly complex. For the research and development of the new algorithms, a simplified dataset is available. Initially, it was intended for the TrackML challenge [21], aiming to find the most efficient method for track reconstruction. The main difference with respect to the ATLAS dataset is that the number of hits in one event is an order of magnitude smaller than ATLAS, the tracking detector read-out data is replaced by partially reconstructed information, and the detector geometry is simplified. One event of the TrackML dataset contains 1150 particles per event to reconstruct from 1.2×10^4 hits, compared to 1500 tracks from 3.5×10^5 hits in an average $t\bar{t}$ sample with pile-up $< \mu >= 200$.

In order to validate the accuracy of the event reconstruction, a dedicated sample is used, consisting of events with single muons passing the ATLAS detector. The primary objective is to ensure the correct reconstruction of individual particle trajectories prior to evaluating the overall throughput efficiency. The samples are available with different momenta of the muon, impacting the trajectory of a track. The samples will still contain detector noise and inefficiencies.

2.2 Track Reconstruction

The particle track reconstruction, commonly known as *tracking*, is the most expensive of the algorithms in the online event reconstruction (Table 2.2). From the clusters of hits from the tracking detectors left by charged particles after the collision, single measurement spatial points are constructed in three-dimensional space called *SpacePoints* (SPs) with dedicated clustering algorithms. From the SpacePoint cloud, the tracking algorithm is connecting SpacePoints to create paths reflecting the paths of the passed particles (*tracks*). The paths are constructed from *track segments*, connecting the subsequent SpacePoints. The paths are restricted by physical constraints, related to the detector geometry, physics of particle path, as well as requirements motivated by the physics program of the experiment. Last, the parameters of the particle track must be estimated, serving as the input to further event reconstruction and hypothesis testing.

2.2.1 Interaction of Particles with Matter

The charged particles passing the detector interact with the atoms of the material; the effects can change the trajectory of the particle. Moreover, the particles are also influenced by the surrounding conditions, such as the magnetic field. In order to achieve a high quality of track reconstruction, these effects need to be considered.

Scattering of Particles on Material

When charged particles pass through a material, they experience small-angle deflections due to repeated interaction with the electric fields of nuclei. This phenomenon is called Multiple Coulomb Scattering (MCS). The scattering angles have a Gaussian distribution and can be approximated by the Equation 2.1 proposed by Lynch & Dahl [22].

$$\theta_0 = \frac{13.6\,\mathrm{MeV}}{p_T}\sqrt{\frac{X}{X_0}} \tag{2.1}$$

where:

θ_0 = angular deviation

$13.6\,\mathrm{MeV}$ = empirical constant

$\sqrt{\frac{X}{X_0}}$ = thickness of the scattering medium in radiation lengths, ($\sqrt{\frac{X}{X_0}} = 0.036$)

p_T = transverse momentum of the particle

Magnetic Field

In the ATLAS detector, the tracking sub-systems are surrounded by a solenoid magnet, enabling reconstruction of the charge of the particles. In a constant magnetic field, a charged particle is under the magnetic field force (Equation 2.2 derived from the Lorentz force law), resulting in a helix-shaped particle trajectory.

$$F = qv_{\perp}B \tag{2.2}$$

where:

F = magnetic force

q = electric charge of the particle

$v_{\perp}$ = instantaneous velocity perpendicular to magnetic field

B = magnetic field strength

The circular motion of the particle can be described with (Equation 2.3).

$$F = ma = \frac{mv^2}{\rho} = \frac{pv}{\rho} \tag{2.3}$$

where:

F = kinematic force

m = mass of the particle

a = centripetal acceleration $a = \frac{v^2}{\rho}$

v = instantaneous velocity

ρ = radius

p = momentum

The equation describing particle momentum can be derived from those two forces (Equation 2.4).

$$p = qB\rho \tag{2.4}$$

2.2.2 Geometry of Tracking

The tracking task involves pattern recognition for reconstructing particle trajectories and estimating track parameters. Those tasks are strongly related to the geometrical configuration of the tracking detectors, as well as to the kinematic constraints dictated by particle physics models.

After the initial preprocessing of clusterisation and SpacePoint formation, only the position of the SpacePoint in Cartesian coordinates (x, y, z) is known, and its conversion to the cylindrical coordinate system (r, ϕ, θ). The reconstructed parameters describing the particle trajectory consist of five parameters, expressed in Equation 2.5 and illustrated in Figure 2.9.

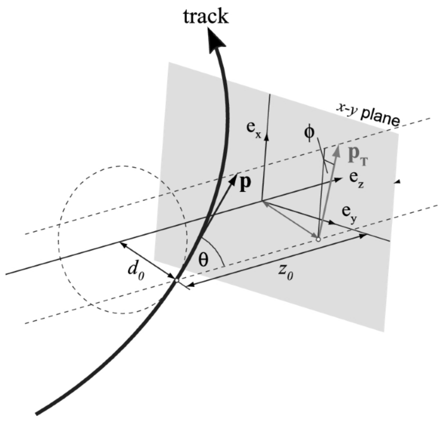


Fig. 2.9 Illustration of the global track parameters, defined relative to a reference point, the perigee, or point of closest approach to the beam line. [23]

The parametrisation is defined with respect to a reference point (perigee). It includes the impact parameters: the distance of the reference point to the IP d_0 and longitudinal distance z_0, as well as azimuthal angle ϕ, polar angle θ of the track at the reference point, and ratio of the charge divided by the momentum of the reconstructed track, reflecting the its curvature $\frac{q}{p}$. The default reference is the beam line centred at the beam spot.

$$h \equiv \begin{pmatrix} d_0 \\ z_0 \\ \phi_0 \\ \theta \\ \frac{q}{p} \end{pmatrix} = \begin{pmatrix} \sqrt{x^2+y^2} \\ d_0 \frac{p_z}{p_T} \\ \arctan\left(\frac{p_y}{p_x}\right) \\ \arccos\left(\frac{p_z}{p_T}\right) \\ \frac{q}{|\mathbf{p}|} \end{pmatrix} \tag{2.5}$$

where:

$\mathbf{x}$ = global position of the reference point, $\mathbf{x} = (x, y, z)$

$\mathbf{p}$ = momentum of the particle, $\mathbf{p} = (p_x, p_y, p_z)$

p_T = transverse momentum $p_T = \sqrt{p_x^2 + p_y^2}$

q = electric charge of the particle

Selected geometry parameters in the transverse $(x - y)$ and longitudinal $(z - r)$ planes, used often for visualisation of the track parameters, are illustrated in Figure 2.10.

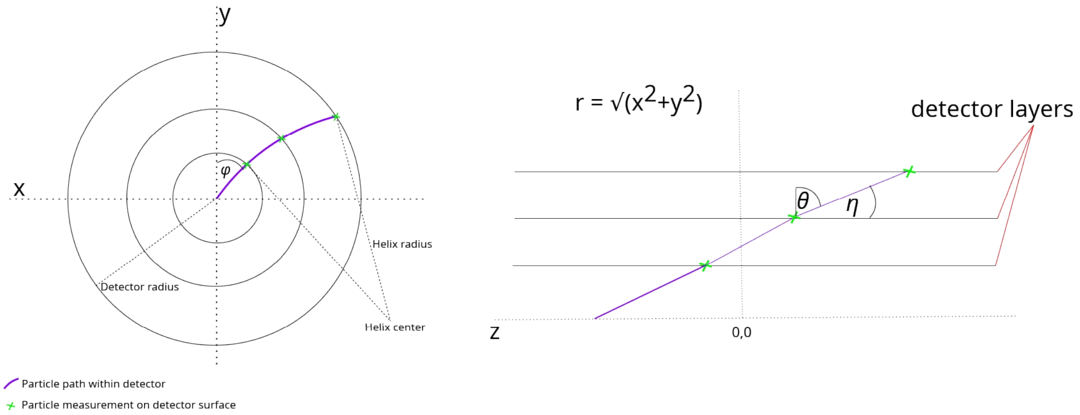


Fig. 2.10 Schematic overview of the ATLAS Detector geometry and the track characteristics in the longitudinal and transverse planes. The most important descriptors used for tracking were highlighted. [24]

Beam Spot

During a proton-proton collision, apart from the particle tracks originating from the primary physics process, there are also secondary tracks caused by pile-up interactions, which are unrelated to the primary event. The tracks recorded by the ATLAS detector can come from the products of proton-proton collisions or secondary decays.

The online event selection focuses on the tracks originating directly from the proton-proton collision. They can be identified by the fact that they start from the luminous region of ATLAS, where the collisions take place, referred to as *beam spot*. Therefore, the reconstructed track or its segment can be classified as primary only after considering the two parameters: the extrapolated crossing of the track with the z-axis, z_0, and the distance in the x-y plane d_0. The z-distance extrapolation is performed in the $z-r$ plane, where the track is approximated by a straight line (Figure 2.10).

Curvature and Transverse Momentum Estimation

The track curvature is used to approximate the transverse momentum of the particle, that is, the amount of a particle's momentum perpendicular to the beam direction. It is one of the most important properties calculated during the track reconstruction, used for the trigger selection and later for physics analysis. The curvature and transverse momentum estimations (Equation 2.6) are derived from charged particle motion in a magnetic field (Equation 2.4).

$$k = \frac{1}{\rho}, \qquad p_T = \frac{qB}{k} \tag{2.6}$$

where:

k = curvature

ρ = radius of curvature of a charged particle's path in a magnetic field

p_T = transverse momentum

q = particle's charge

B = magnetic field strength

Pseudorapidity

Apart from the perigee parameters, described in this subchapter, a spatial coordinate called pseudorapidity (η) is calculated for every track, defined by Equation 2.7. It describes the angle of a particle relative to the beam axis. For the tracks perpendicular to it $\eta = 0$, for tracks along the beam axis, η tends towards infinity.

$$\eta = -ln\left(tan\left(\frac{\theta}{2}\right)\right) \tag{2.7}$$

This description of the angle of particle trajectory is used in the event reconstruction, the hypothesis testing, and the analysis. The pseudorapidity is a preferred parameter over the polar angle θ, because differences in pseudorapidity $\delta\eta$ remain approximately invariant under Lorentz boosts along the beam axis.

2.2.3 The Tracking Challenge

The computational complexity of tracking is very high, involving hits matching, rejecting the false positive tracks, and track fitting, all within the online trigger throughput requirements. In an average bunch crossing in Run 3, the algorithm needs to process up to 15 000 silicon hits, comprising up to 800 final tracks (for pile-up $<\mu>= 70$) [25].

Additionally, the algorithm needs to consider interactions of particles with matter (multiple scattering) that impact the trajectory, as well as the impact of the magnetic field bending the tracks, depending on their momentum. Moreover, not all the hits can be associated with particles; some of the hits come from the detector noise. Not all the possible hits of a track are present in the read-out - a layer can be missed or not reconstructed because of the detector inefficiencies (called *holes*). If the hits were close enough, distinguishing them during the hit formation can be challenging.

The quality of the track reconstruction can be assessed based on the simulated data samples, where the particle paths are known (called *truth*). The reconstructed tracks can be classified in three categories: *true* tracks, corresponding to the simulated tracks, *fake* tracks, not corresponding to any of the simulated tracks, or to the tracks that are not the target of the reconstruction, based on the event reconstruction parameters (false positive tracks). The third category, *duplicate* tracks, corresponds to the tracks that overlap with already reconstructed true tracks. The track reconstruction is quantified as the fraction of successfully reconstructed target tracks. However, the number of fake and duplicate tracks should also be considered, as it impacts the computing performance of the algorithms. With increasing pile-up, the quality of tracking decreases, as more available clusters result in more opportunities for incorrectly associated hits, resulting in fake track reconstruction.

2.2.4 Track Reconstruction Requirements

To ensure that only desired tracks are reconstructed, a series of criteria is applied during reconstruction to reject tracks that are not of interest from a physics point of view. The settings can be adjusted to accommodate particular needs; this subchapter presents the requirements from the EF Tracking requirements document [26], describing requirements for the online track reconstruction for the ATLAS trigger for Run 4 and Run 5. The criteria are based on the prepared set of physics signatures that the online trigger should efficiently reconstruct.

The criteria summarised in Table 2.3 ensure that the reconstructed tracks are within ITk detector coverage $|\eta| < 4$ and expected beam spot $|z_0| < 200\,\text{mm}$. The tracking must cover all the particles with $p_T > 1\,\text{GeV}$ for full-scan tracking and $p_T > 2\,\text{GeV}$ for tracking within RoIs. The minimum tracking efficiency for high-p_T tracks ($p_T > 10\,\text{GeV}$) is 98% and for low-p_T tracks ($1\,\text{GeV} < p_T < 10\,\text{GeV}$) is 95% with respect to tracks reconstructed by the offline reconstruction algorithms. The maximum rate of fake and duplicate tracks cannot exceed 1% at pile-up $< \mu >= 200$.

Table 2.3 EF Tracking track selection requirements as a function of pseudorapidity [26].

Requirements	Pseudorapidity Interval		
	$\|\eta\| < 2.0$	$2.0 < \|\eta\| < 2.6$	$2.6 < \|\eta\| < 4.0$
pixel + strip hits	≥ 9	≥ 8	≥ 7
pixel hits	≥ 1	≥ 1	≥ 1
holes	≤ 2	≤ 2	≤ 2
p_T [MeV]	> 900	> 400	> 400
$\|d_0\|$ [mm]	≤ 2.0	≤ 2.0	≤ 10.0
$\|z_0\|$ [cm]	≤ 20.0	≤ 20.0	≤ 20.0

Additionally, a series of requirements can be derived from the presented requirements and the detector geometry, described in detail in the track seeding-related chapter.

2.3 Graph Neural Networks

Neural Networks (NNs) are powerful tools for solving complex problems by transforming input data through weighted connections and non-linear activation functions. By adjusting the learnable parameters during training, NNs learn to approximate complex functions, enabling them to perform tasks such as image recognition, natural language processing, and time-series forecasting.

Perceptron

One of the first and simplest neural networks is a *perceptron* [27], capable of making predictions for linearly separable data, usually in binary classification tasks.

The functions are modelled by transforming the input vector $\mathbf{x}$ with n features using the network parameters: weight vector $\mathbf{w}$, and a bias b. The weights and biases are adjusted during the learning process of the NN to minimise the error between the network's predictions and the expected values.

During evaluation, the feature vector is combined with the corresponding set of weights to compute a weighted sum, which is then transformed by a non-linear activation function f. The step activation function, configured for a given threshold, allows the decision-making process by distinguishing between two classes. The result from the activation is the final output of the perceptron. The evaluation process was illustrated in Figure 2.11 and can be described with Equation 2.8.

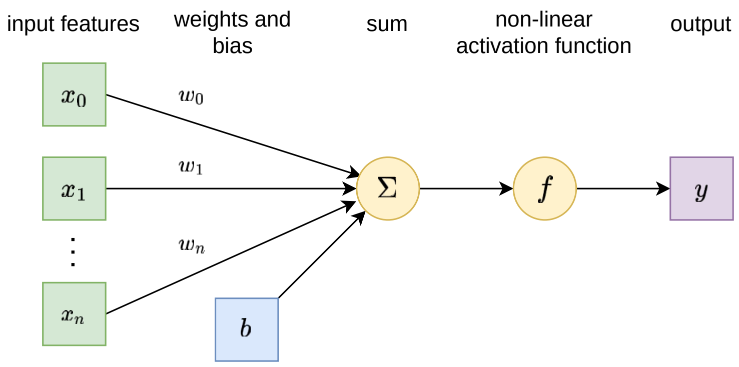


Fig. 2.11 Schematic overview of a perceptron neural network.

$$y = f\left(b + \sum_{i=0}^{n} x_i \cdot w_i\right) \tag{2.8}$$

The model can be extended to result in an output vector instead of a single value, to model more complex non-linear functions, or facilitate multiclass classification. In this case, each output feature will have a separate set of weights and biases.

Before the evaluation, the NN must undergo a training process to learn how to reflect the expected output by transforming the input features. The process of minimising the difference between predicted and expected output is guided by a *loss function*, which quantifies this difference. The training process where the expected output is known is called supervised learning. During the training, the dataset that the network is supposed to model should be split into three parts: training set, validation set and testing set - the usual proportions are 80-10-10. The training set is used to fit the parameters of the network during training, and validation is used to give an unbiased evaluation of the model fit during training, for example, for early stopping of the training to prevent overfitting (described later in this subchapter). The test set is used for unbiased evaluation of the final model fit. In the beginning, the weights and biases are initialised with random values. In each training iteration, called an *epoch*, the following steps are taken:

1. Calculate the output of the NN y. (Equation 2.8)

2. Calculate the loss l, quantifying the difference between the NN output and the expected value d. For a simple network like a perceptron, loss can be defined as a simple difference with Equation 2.9.

$$l = y - d \tag{2.9}$$

3. Update the weights and bias values based on the loss value and the learning rate parameter r, controlling the volatility in the weight changes. The parameter adjustment is described by Equation 2.10.

$$w_i = w_i + r \cdot l \cdot x_i \tag{2.10}$$

Defining an appropriate stopping criterion for the training is critically important. The process is usually repeated until the desired minimal loss is achieved or the maximum number of epochs is reached. However, excessive training duration frequently leads to overfitting by causing the model to align too closely with the training data, and therefore a deterioration in generalisation performance.

Multi-Layer Perceptron

The simplicity of the perceptron struggles to model problems where the data is not linearly separable. To mitigate this issue, a Multi-Layer Perceptron (MLP) [27] allows modelling of complex problems. It consists of an input layer, one or more hidden layers, and an output layer. Each layer consists of neurons, transforming its input through a linear operation followed by a non-linear activation function, in the same way as the perceptron. Output of a single layer can be described by Equation 2.11.

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.11}$$

where:

$\mathbf{h}$ = the output tensor of a single hidden layer

σ = activation function [2]

$\mathbf{W}$ = weight matrix

$\mathbf{x}$ = input vector

$\mathbf{b}$ = bias vector

The neurons are fully connected between neighbouring layers. A pass through all L layers of MLP can be described by Equation 2.12.

$$\begin{aligned} \mathbf{h}^{(1)} &= \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}), \\ \mathbf{h}^{(2)} &= \sigma(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}), \\ &\cdots \\ \mathbf{y} &= \sigma(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}) \end{aligned} \tag{2.12}$$

In comparison to the training process described for the perceptron, there are a few changes applied to the training of MLP. The process of updating the parameters based on the loss is done through backpropagation. After the forward pass, a gradient of the loss function with respect to the parameters is calculated

[2] To simplify the notation, σ is used to denote both the summation operation and the activation function f

(Equation 2.13), representing how much the parameter impacts the loss and how to adjust it to minimise the model error.

$$\frac{\partial l}{\partial \mathbf{W}} = \frac{\partial l}{\partial y} \cdot \frac{\partial y}{\partial \mathbf{W}} \tag{2.13}$$

The parameters are updated using this information with optimisation algorithms, for example, the Gradient Descent. The parameters will be adjusted in the gradient direction that minimises the loss function, corrected by the learning rate r, as presented in Equation 2.14.

$$\mathbf{W} = \mathbf{W} - r \cdot \frac{\partial l}{\partial \mathbf{W}} \tag{2.14}$$

Despite their simplicity, MLPs are widely used across a variety of tasks as effective function approximators. Moreover, they are often used as a building block for more complicated networks, including Graph Neural Networks.

2.3.1 Graph Neural Networks

Graph Data Structure

Despite being the most popular data structure, the tabular format, which organises the data in rows and columns, exhibits limitations when modelling complex relationships between entities or dynamic structures. Such data representation often requires extensive restructuring or introducing multiple relational tables to capture the interdependencies. Graphs offer a more flexible format, consisting of nodes and connections between them (edges) that capture complex and dynamic interactions. An illustrative example of a simple graph structure is presented in Figure 2.12.

In the graph data format, both nodes and edges can have features; moreover, they do not have to be uniform in the graph (*heterogeneous graphs*). Not all nodes have to be connected via edges - the relations between nodes are encoded in different formats, for example, an adjacency matrix A. An example representation for a graph from Figure 2.12 is shown in Equation 2.16, where n represents the graph nodes.

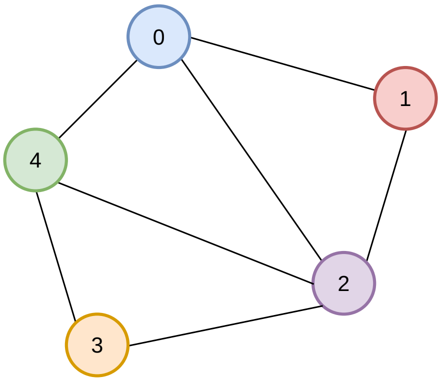


Fig. 2.12 An example graph visualisation, consisting of 5 nodes and connections (edges) between them.

$$n = [0,1,2,3,4], \quad A = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{2.15}$$

Another widely-used format is the edge list (Equation 2.15), where the edges are represented by a list of their source E_{src} and destination E_{dst} nodes. The choice of the format should depend on the sparsity of the graph - for graphs with very few connections, the edge list representation is more beneficial as it is much less memory-consuming.

$$n = [0,1,2,3,4], \quad E_{src} = [0,0,0,1,2,2,3], \quad E_{dst} = [1,2,4,2,3,4,4] \tag{2.16}$$

The graphs can be directed, which means that an edge (a,b) leads from a to b but not the other way. Sequences of distinct edges joined by nodes are called *paths*. A relation between any two nodes in the graph can be described by a *distance*, defined as the number of edges in the shortest path connecting the nodes.

For each node (and edge), a *neighbourhood* N can be defined, consisting of all the nodes connected to the node by an edge. A neighbourhood of distance d includes all the nodes in the graph to which the distance is less than or equal to d.

Graph Neural Networks

Graph Neural Networks (GNNs) [28] are a special type of NNs focused on tasks related to graph-structured data. Unlike the traditional NNs, instead of a simple input vector of tabular-format data, GNNs are well-suited for domains where relationships between entities are crucial, such as social networks or molecular structures. By using both node-level information and the connections between them, the model can capture complex dependencies within the graph.

The key element of the GNNs is the *message passing* mechanism, which aggregates information through the graph, allowing them to capture both local and global information. Both node- and edge-related features can be propagated, depending on the task. This process has two steps, illustrated in Figure 2.13. First, for each item (node or edge), the features of the defined neighbourhood are gathered together in a vector or matrix. Next, the gathered features are aggregated, creating an updated feature value for the given item. The aggregation can be done by a simple sum or a more advanced function. The process can be repeated to aggregate the messages from further neighbours.

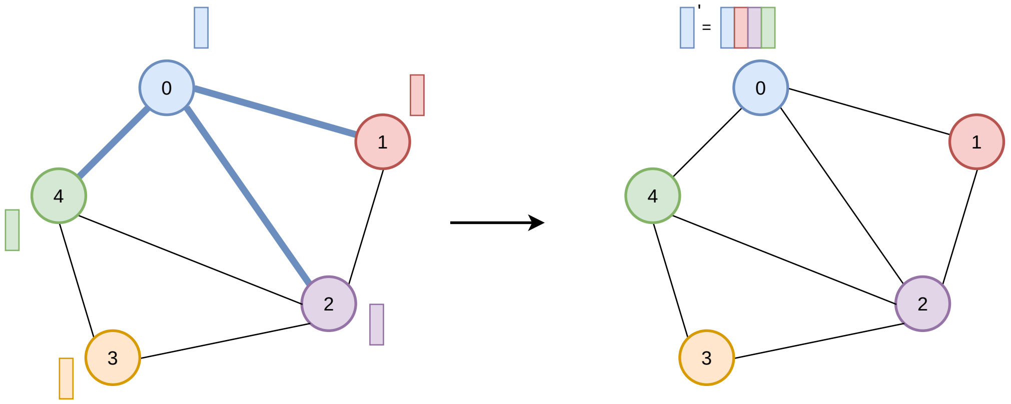


Fig. 2.13 A diagram illustrating the message passing mechanism in GNNs inspired by [29]. The left side presents the initial state of the graph with five nodes, not fully connected. Each node has its feature vector. After the message passing step for node 0, the new feature vector consists of the combined features of neighbouring nodes and the old node feature.

The most popular tasks involving GNNs are node classification, link prediction and graph classification. To solve these problems, many different types of GNNs are available, including Graph Attention Networks [30], Graph Convolutional

Networks [31], Graph Auto-Encoder Networks [32], or Interaction Graph Neural Networks.

Graph Attention Networks

Graph Attention Networks (GATs) are a special type of GNNs, introducing the attention mechanism in the message passing step. Attention heads can indicate how relevant the information from each neighbouring node is by attention coefficients (acting as weights). As a result, GATs are especially powerful for processing noisy graphs, where not all connections contribute equally to the prediction accuracy. The model can employ multiple attention heads, learning different attention coefficients in parallel, and capture diverse patterns in the graph. GATs are successfully applied in different tasks, including bioinformatics [33], traffic networks [34], and cybersecurity [35].

Graph Convolutional Networks

Graph Convolutional Networks (GCNs) are a type of GNNs, enabling convolution on graph-represented data. Convolution mechanism, widely used in the machine learning community via Convolutional Neural Networks (CNNs), acts as a filter over structured input data, such as grids or images, to extract localised functions. This operation enables the network to capture local patterns and reduce the dimensionality of the input while preserving essential information. Graph convolution, instead of filtering over a grid-shaped space, works by aggregating the features from the neighbours. GCNs are commonly applied to tasks related to recommendation systems [36], drug discovery [37], or social network analysis [31].

Interaction Graph Neural Network

One of the special types of GNNs is the Interaction Graph Neural Network (IGNN), based on the Interaction Neural Network concept [38]. They are designed to model the interaction between the nodes in the graph through the message passing mechanism and feature aggregation. Through the ability of learning from both node attributes and the interactions between them, IGNNs can capture dynamic higher-order relationships between nodes. IGNNs are widely used in domains such as molecular modeling [33], [39], physics simulations [40], and social network analysis [41].

2.4 GPU Programming

Graphics cards implement a massively parallel architecture optimised for handling large volumes of data, initially designed for graphics rendering. GPU programming leverages this architecture to accelerate a broad range of computational tasks. Unlike CPUs, which typically have a few cores optimised for sequential processing, GPUs contain thousands of smaller, efficient cores designed for parallel execution of many threads simultaneously (Figure 2.14). However, in the single-thread performance, CPU processing is always more efficient. The GPUs are widely applied in fields requiring high computational throughput, including machine learning, simulations, and real-time data processing.

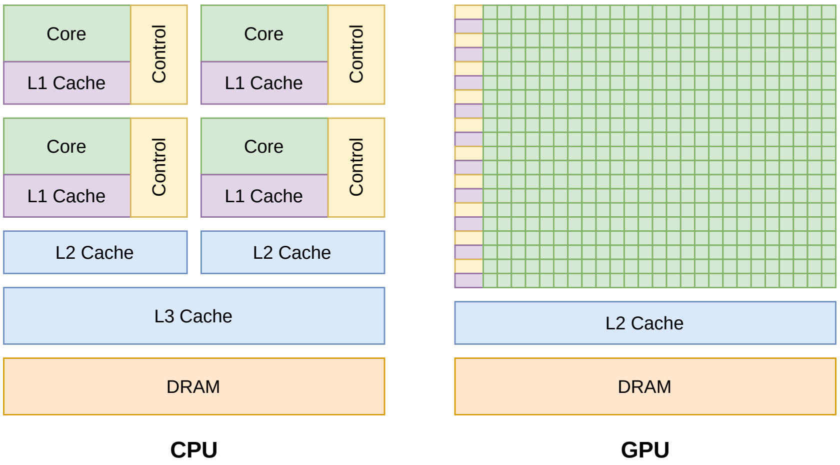


Fig. 2.14 Comparison of CPU and GPU architecture [42]. The CPU architecture consists of a few powerful cores performing basic arithmetic and logical operations, which are supported by large caches and advanced control. In contrast, a GPU consists of thousands of simpler cores with minimal control logic, providing high throughput for parallel applications.

The programming of graphics processors requires specialised frameworks that provide APIs and language extensions to offload code to the hardware. They are leveraging the parallelism at different granularities and providing various levels of user customisation and interaction with the GPU. One of the most popular platforms is CUDA [43], providing a software layer to access the NVIDIA GPUs directly. The user can directly implement functions called *kernels* and control their runtime, requiring advanced programming skills. The second framework used in this work is PyTorch [44], providing seamless CUDA integration for

Python applications. Users can leverage GPU acceleration with minimal code modification; moreover, it is well integrated with ML tools available in Python. Custom CUDA extensions can be added for low-level GPU optimisation.

In the CUDA programming model [45], the parallelisation is based on a hierarchy that efficiently divides the workload between the processing units. A thread *block* collects *threads* executing the same kernel function on the same Streaming Multi-processor (SM), which gives them access to low-latency shared memory. The number of threads can be configured during the kernel launch, and exposed variables can address them. Each thread can be uniquely identified by its index within a block `threadIdx` and the x, y, z dimension, as thread blocks can be defined in up to three dimensions. The thread blocks are organised into a *grid*, which can also be multidimensional, collectively executing a given kernel. Each block can be referred to, similarly to threads, by the `blockIdx` and its position in the grid dimensions. Additionally, the CUDA interface provides the block and thread dimensions as well: `blockDim` and `threadDim` respectively. Blocks are independent - they do not share memory or synchronise. The described hierarchy is illustrated in Figure 2.15.

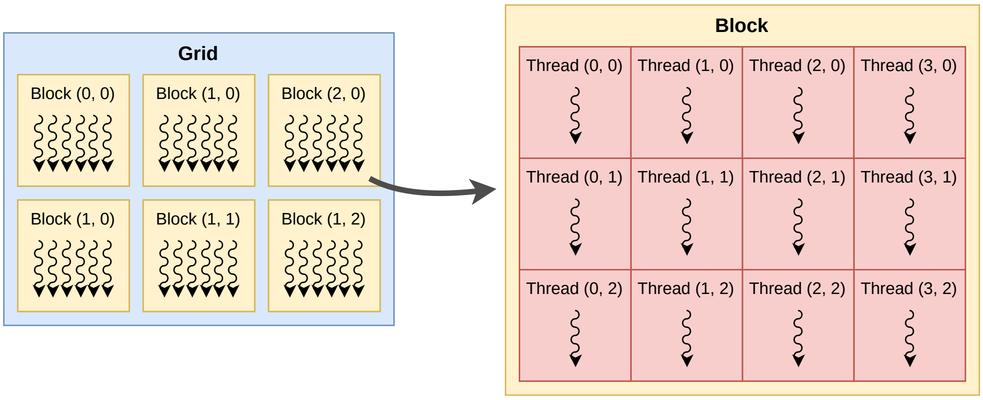


Fig. 2.15 CUDA virtual thread-block-grid organization [42]. The illustration shows two-dimensional grid of size $(2, 3)$ with two-dimensional blocks, each of a size $(3, 4)$.

The threads are grouped within a block by the configuration of the user; however, they are scheduled and executed on the GPU in groups of 32 called *warp*, partitioned by the hardware. The hardware does the grouping, and each warp executes the same instruction at a time, but on different data (Single Instruction, Multiple Thread). The warp mechanism should be considered during the code optimisation; the conditional branching within a warp can lead to reduced performance, as divergent paths are serialised. To improve the throughput, aligning memory accesses among threads in a warp should be optimised to exploit coalesced memory access.

Apart from the mentioned shared memory (accessible within a thread block), the GPU structure offers global off-chip dynamic random-access memory (DRAM). It is accessible to all the threads and blocks with high latency.

2.4.1 Available Hardware

Graphics cards available on the market have fine-tuned features for different purposes, prioritising the optimal memory access, precision, or concurrency. To identify the best hardware for each algorithm presented in this work, the performance will be evaluated on the following GPUs:

- NVIDIA Tesla T4 (released 2018),
- NVIDIA RTX A5000 (released 2021),
- NVIDIA RTX 5000 Ada (released 2023),
- NVIDIA A100 (released 2020).

In Table 2.4, chosen properties of the cards are listed, which can noticeably impact the performance of the algorithm. The details of the differences will be described in this subchapter.

The memory amount on a card can be limiting for memory-consuming algorithms, including NNs. The high memory consumption can arise from the model size (for example, a large number of parameters in Convolutional Neural Networks) or from large model input (often encountered with Graph Neural Networks). The DRAM type can also be a limiting component. The considered GPUs have two different types: GDDR and HBM. Graphics Double Data Rate 6 (GDDR) [46] is a standard of synchronous random access memory implementing data transfer on both the rising and falling edges of the clock cycle (Double Data Rate - DDR) and doubling the data transfer rate. Each memory chip has two independent 16-bit channels, doubling the throughput in comparison to the previous technology. The second memory architecture, High Bandwidth Memory (HBM) [47] instead of the traditional two-dimensional model, relies

Table 2.4 Overview of chosen properties of GPU cards used for particle track reconstruction evaluation.

Property	NVIDIA Tesla T4	NVIDIA RTX A5000	NVIDIA RTX 5000 Ada	NVIDIA A100
Architecture	Turing	Ampere	Ada	Ampere
Memory [GB]	16	24	32	40
Memory type	GDDR6	GDDR6	GDDR6	HBM2e
Memory bandwidth [GB/s]	320	768	576	1555
Number of SM	40	64	100	108
Number of CUDA cores per SM	64	128	128	64
Maximum number of threads per SM	1024	1536	1536	2048
Thermal Design Power [2] [W]	70	230	250	250
Market price k$ [3]	0.8-1.1	3-4	4.5-5.5	8-10

on three-dimensional stacking of memory dice, providing high bandwidth for memory access. The dice are in physical proximity to the processors; therefore, a wide data bus and low latency are possible. This memory type is popular in bandwidth-constrained environments, such as high-performance computing or AI training.

The next property to consider when choosing GPU hardware is its processing power. The more SMs a GPU has, the more independent parallel blocks can execute at the same time, improving scalability for large workloads. More CUDA cores per SM provide higher throughput within SM, allowing, for example, more simultaneous threads.

The last criterion to consider is the price and the availability of the hardware. The cutting-edge generation of graphics cards, including NVIDIA Hopper or Blackwell, is subject to prolonged delivery times. Moreover, their market-driven cost is considerably high (NVIDIA H100 80GB costs 25 k$-35 k$ per unit, June 2025). Making optimal decisions for the event processing farm requires selecting hardware that achieves a balance between cost and performance, matched with

[2] Thermal Design Power (TDP) - the power consumption under the maximum theoretical load

[3] June 2025

an optimised algorithm to ensure cost efficiency. The power consumption of the card should be taken into account in long-term cost planning.

2.4.2 Profiling

To fully leverage the computing power provided by GPU parallel processing, the code can be analysed and fine-tuned with dedicated tools. This subchapter provides an overview of profiling programs used in this work.

CUDA

Along with the CUDA platform, NVIDIA provides advanced tools for GPU program analysis. In this work, NVIDIA Nsight Compute was used to measure the resource utilisation during the inference. This tool offers easy access to the execution time of kernels, memory consumption, throughput, scheduling, and many other characteristics.

One of the important metrics is the number of Floating Point Operations (FLOPs), necessary to estimate the computational demand placed on the processor and measure how intensively a model utilises the resources. As the subtotals of fused multiply-add (FMA), addition (ADD), and multiplication (MUL) (floating point) operations are not directly comparable due to different hardware complexity, we calculate the total number of FLOPs as follows to approximate underlying hardware complexity:

$$\mathrm{FLOP} = 2 \cdot \mathrm{FMA} + \mathrm{ADD} + \mathrm{MUL} \tag{2.17}$$

Another metric measured with the NVIDIA tools is the instantaneous power consumption. The measurements are done every 1 ms during the algorithm's execution. The granularity can be adjusted according to the walltime of the algorithm.

The CUDA code used in this work is a part of the Athena framework. Internal Athena time monitoring was used to measure the wall-clock time of different components of the tracking algorithm.

PyTorch

To analyse the PyTorch code, the same NVIDIA profiling tools can be used; however, the PyTorch framework provides a set of profiling tools fine-tuned for its algorithms.

In this work, two built-in PyTorch tools are used to measure the memory consumption of the algorithm. Both require enabling the recording of memory history [4]. The recorded allocations can be dumped into a snapshot [5] and later analysed with the visualisation tool called `memory_viz` [6]. The allocations are annotated with relevant information such as size, duration, and code source. Alternatively, to measure the peak memory per input, a dedicated built-in tool [7] can be used. Throughout the measurements summarised in this work, the mentioned tool was integrated into the PyTorch Lightning Callback [8] to measure the peak memory consumption during the inference per the model input.

In order to measure the processing time of the given input, the `torch.event` package [9] was used. It provides a precise measurement, but it requires synchronisation; therefore, the measurements cannot be nested, and the total walltime is impacted. The PyTorch Lightning Advanced Profiler [10] is implemented using the `cProfile` profiler [11], providing a generic overview of the most computationally expensive functions on both CPU and GPU. By inserting hooks and callbacks, all the operations are recorded; however, the measurements are not as precise as `torch.event` package, using native CUDA timing mechanisms with nanosecond resolution. Moreover, by profiling the nested operations, the overhead from measuring them can be observed.

[4] https://docs.pytorch.org/docs/stable/torch_cuda_memory.html#torch.cuda.memory._record_memory_history

[5] https://docs.pytorch.org/docs/stable/generated/torch.cuda.memory_snapshot.html#torch.cuda.memory_snapshot

[6] https://pytorch.org/memory_viz

[7] https://docs.pytorch.org/docs/stable/generated/torch.cuda.max_memory_allocated.html

[8] https://lightning.ai/docs/pytorch/stable/extensions/callbacks.html

[9] https://docs.pytorch.org/docs/stable/generated/torch.cuda.Event.html

[10] https://lightning.ai/docs/pytorch/stable/api/lightning.pytorch.profilers.AdvancedProfiler.html

[11] https://docs.python.org/3/library/profile.html#module-cProfile

Chapter 3

GPU-accelerated Track Seeding for ATLAS Phase-II Trigger

Particle track reconstruction remains the most computationally demanding task of the ATLAS online trigger (the details of the challenge of event reconstruction are presented in Subchapter 2.2.3). The challenges are further amplified in the context of the High-Luminosity LHC (Subchapter 2.1.1), and the enhanced complexity introduced by the upgraded ATLAS detector (Subchapter 2.1.2). This chapter provides an overview of the tracking algorithm employed during Run 2 and Run 3 of the ATLAS experiment, highlighting the limitations of the current approach and the necessary upgrades for the expected Run 4 conditions. To further improve the computational performance, a GPU-accelerated solution is proposed, exploiting the parallelism of this algorithm. The proposed optimisations and their effects on the accuracy and performance are discussed in detail.

3.1 Fast Track Reconstruction Algorithm

This subchapter provides an overview of the fast tracking algorithm (FTF) [5], used by the ATLAS Collaboration as an online trigger track reconstruction algorithm. The GPU-accelerated track seeding [48] will be described, currently not used for the online trigger, but prepared as a proof of concept for early studies of accelerators in ATLAS event reconstruction.

3.1.1 Algorithm Overview

Clustering and SpacePoint Formation

Before the algorithm matching the hits together into particle tracks can be executed, first, the raw detector data is transformed into SpacePoint representation in a process called *clustering* [49].

The SpacePoints are constructed from *clusters*, created directly from the detector read-out of the charge deposit as well as the read-out module position. A clustering algorithm groups together signals left in adjacent channels, representing the intersection of a passing charged particle with detector material. A visualisation of a particle interacting with detector cells is presented in Figure 3.1.

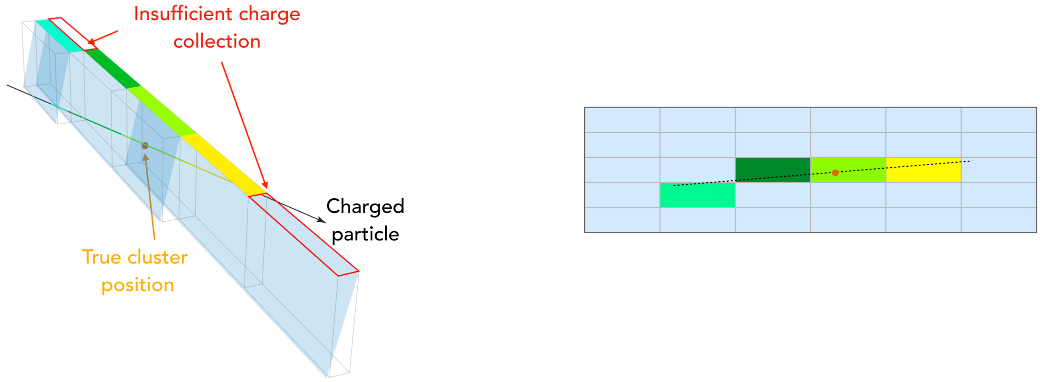


Fig. 3.1 Illustration of a particle interacting with detector cells: three-dimensional view (left) and local cluster two-dimensional view (right).

Based on the charge read-outs q and positions $\mathbf{l}$ of N cells with recorded signal, the final measurement of the cluster centre $\mathbf{m}$ can be calculated with a charge-weighted approach, described by Equation 3.1.

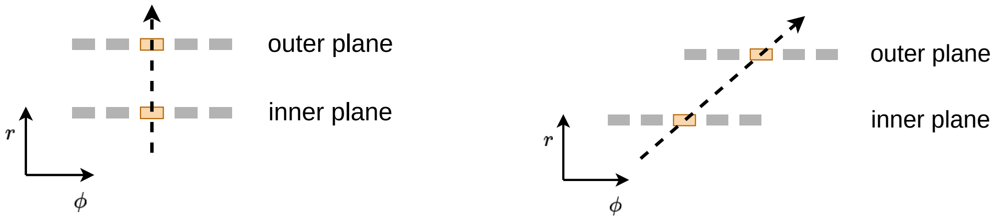
$$\mathbf{m} = \frac{1}{\sum_{i=1}^{N} q_i} \sum_{i=1}^{N} q_i \mathbf{l_i} \tag{3.1}$$

The process of clusterisation requires sophisticated algorithms to disentangle close-by or joint clusters. Based on the formed clusters and their geometry: location and rotation, SpacePoints can be created as three-dimensional points in space. Each SpacePoint is represented by one cluster from the ITk-Pixel detector or two clusters from the ITk-Strips, illustrated in Figure 2.3.

To correctly reconstruct the track, the uncertainty of the measurement needs to be considered. Although the modules are small and highly precise — the pixels from ITk detector are approximately five times smaller than those in the Inner Detector Pixel — they do not provide the exact location on the module's surface where the interaction took place. To account for this uncertainty in the measurement, each SpacePoint has an associated error, calculated based on the dimensions of the cluster surface. Afterwards, the error on the cluster needs to be transformed according to its position in the detector. The active size of a single read-out pixel chip module is $19.2 \times 20\text{mm}^2$ (384 × 400 pixels, each pixel sensor of size $50 \times 50\text{µm}^2$), and the depth is estimated to be $100 - 150\text{µm}^2$ [8].

Similar constraints regarding the measurement uncertainty arise for the ITk-Strip detector. The created clusters are combinations of two one-dimensional read-outs. The layers of strip modules are tilted by a stereo angle with respect to each other as presented in Figure 2.3. The position of the SpacePoint is determined by the geometric intersection of two strip modules.

However, the sensors are not directly above each other, but separated by a gap. Therefore, a particle passing through the detector with a bent track due to the magnetic field could hit the outer sensor, not directly above the inner plane. The cases are illustrated in Figure 3.2. By default, the hits from the ITk-Strip detector are created assuming the particle travelled straight from the IP; however, it's only true for the high p_T tracks.



(a) Particle passing in a direction perpendicular to the module plane.

(b) Particle passing in a direction leaning towards right wrt the module plane.

Fig. 3.2 Illustration of particle passing through strips modules in different directions based on illustrations from [50].

Final SpacePoint contains information about its global position and derived parameters, the ID of the detector module that it interacted with, and a list of clusters that it was created from.

Track Finding

The second step of the fast track reconstruction algorithm, track finding, consists of two parts: track seeding and track following. Track seeding searches for combinations of three SpacePoints (called *triplets*) that are promising to be part of a track corresponding to a target particle (true tracks). Seeds are filtered by applying selection criteria to reject the seeds that are not likely to be part of the true tracks. A typical track finding algorithm in ATLAS and its selection cuts will be described in detail in Subchapter 3.1.2.

Track seeding is followed by track finding, where the track candidates are built from the seeds. There are several algorithms available, including the combinatorial Kalman Filter [51] (CKF) used for ATLAS online track reconstruction. The track finding algorithm should maximise the number of true tracks and minimise the number of fake and duplicate tracks.

The ATLAS online track following algorithm is illustrated in Figure 3.3. First, based on the identified seeds, *search roads* are created through the remaining layers of the detector based on the estimated track trajectory from the seed. The roads consist of detector modules that are expected to contain clusters compatible with the considered seed. Within the roads, the track is extended from the seed through a CKF. First, the track is extrapolated to find the location of the best-fitting next hit. Next, the most compatible seeds are selected as candidates for the extrapolation - this allows branching of the fit, in contrast to the standard Kalman Filter approach. This way, even if the true hit is not the best candidate due to detector inefficiencies or noise, it will still be considered to be a part of a track. At each step, the procedure of refitting, extrapolation, and selection is repeated until there are no hits available, and the set of track candidates is selected. The ambiguity resolution algorithm selects the best candidate for a seed based on the reconstructed parameters of tracks.

Track Fitting

The last step of the track reconstruction is track fitting, which aims to estimate the parameters of the created track.

The fast tracking algorithm uses the Kalman filter track fitter [52], which is much more computationally efficient compared to the χ^2 method [53] used

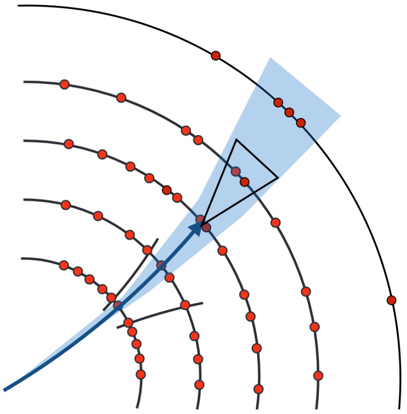


Fig. 3.3 Illustration of track finding with CKF algorithm. The arches represent detector layers, and red circles - identified SpacePoints. The search road is marked in blue, and the current track candidate is indicated by a blue arrow. The triangle represents the search area for the next hit, extrapolating the track.

for precision tracking. This strategy takes into account uncertainties caused by particles interacting with matter.

3.1.2 Track Seeding

The first part of the track finding, the track seeding, is a crucial step, allowing the start of the search for the track candidates. The algorithm first creates pairs of hits (called *doublets*) prefiltered by their geometric properties, which are later matched into track seed candidates (*triplets*). The two doublets creating a triplet must share one of their points (the middle SpacePoint of the seed). To reject the fake seeds, a series of selection cuts is applied, closely related to the position of the hits, the detector geometry and parameters of the tracks to reconstruct. The selections are scheduled in a way that the fastest and the most rejecting cuts are applied first, to reject as many fake seeds as soon as possible and accelerate the algorithm execution.

The GPU-accelerated track seeding was implemented by three kernels; their workflow is illustrated in Figure 3.4. The first two kernels, doublet counting and doublet making, are responsible for the creation of SpacePoint pairs, which will later construct a track seed. The same selection code is executed twice to distribute the memory along the preallocated data structures efficiently. The size of the data structure corresponds to the expected maximum number of doublets. First, the number of doublets for each middle SpacePoint is counted, and later, the doublets are saved; this way, continuous memory to store all doublets can be used. If the first step was skipped, and for each middle SpacePoint, the maximum possible doublet size was allocated, the available memory on the hardware would be exceeded.

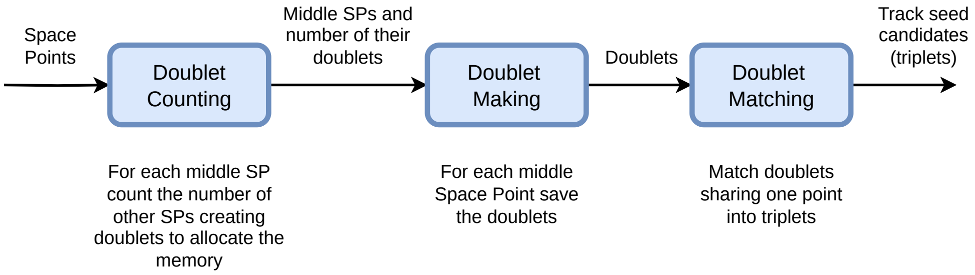

Space Points
Doublet Counting
Middle SPs and number of their doublets
Doublet Making
Doublets
Doublet Matching
Track seed candidates (triplets)
For each middle SP count the number of other SPs creating doublets to allocate the memory
For each middle Space Point save the doublets
Match doublets sharing one point into triplets

Fig. 3.4 Illustration of the flow between the kernels implementing the GPU-accelerated track seeding.

The GPU acceleration of the track seeding is possible because the seeds can only be created from points within a defined neighbourhood. The search for the seeds is performed in sectors of the detector, illustrated in Figure 3.5, restricted by azimuthal angle ϕ and pseudorapidity η. The SpacePoints from the Barrel are divided into 30 equally sized sectors in ϕ and 13 in η, and SpacePoints from the EndCap into 73 sectors in ϕ and 2 in η.

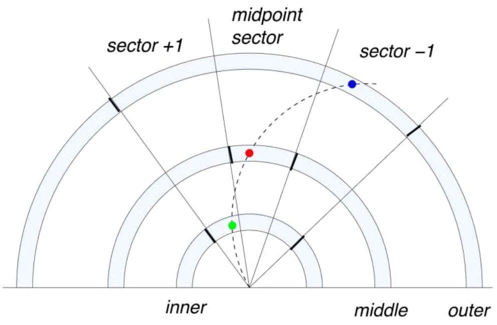


Fig. 3.5 Illustration of sectors considered for track seeding. During the triplet search, the currently processed sector is considered as well as the adjacent ones [49].

Doublet Selection Cuts

The filtering in search for track seed candidates begins with the selection of layers within the considered sector and its two closest neighbours, which are considered for a given point to contain a corresponding point that can form a doublet.

The layer selection process involves projecting a straight line in the z-r plane from the currently processed SpacePoint (further called SP_m) to the boundaries of the layer under consideration; the projected range must overlap with the beam spot boundaries to accept this layer. This way, the criterion described in Subchapter 2.2.2 to reconstruct only primary tracks originating from the beam spot is fulfilled. The projection is illustrated in Figure 3.6, where z_{lmin} and z_{lmax} mark the projected range.

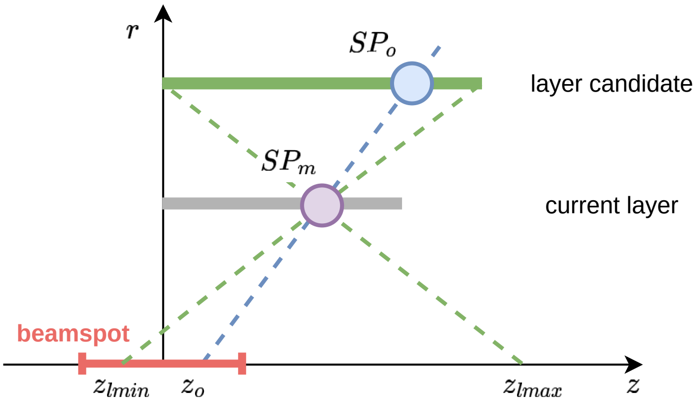


Fig. 3.6 Illustration of a layer and SpacePoint projection onto the beam spot used as a selection criterion for track seeding. The purple circle symbolises the currently considered SpacePoint, SP_m, and the red line at $r = 0$ - beam spot limits. The layer candidate for containing possible hits to form a seed and its projection over the z-axis is marked with a green colour. A hit belonging to the layer candidate SP_o and its projection are marked in blue.

Additionally, only detector layers located within a specified distance l from the SpacePoint SP_m should be accepted. This distance is evaluated from the geometric centre of each layer and is defined to ensure the inclusion of both the nearest and the most distant successive layers. Considering the doublets created from distant layers lead to the creation of duplicate seeds or even fake seeds. In the current configuration, the parameter l ranges from 2 to 150mm.

After accepting the layer for further processing, each of the SpacePoints belonging to it is (further called SP_o). The distance selection cut is applied again for SP_o, followed by a cut on the pseudorapidity of the doublet - the maximum $\eta \in [-4, 4]$ of the track seed candidate, described by Equation 2.7, cannot be exceeded.

Last but not least, a projection is made, similar to the layer boundaries described earlier in this subchapter, if the track created from this seed could originate from the beam spot, as illustrated in Figure 3.6.

If a doublet passes all the selection requirements, it is saved for further processing by triplet formation.

Doublet Selection Implementation

A block diagram of the doublet selection algorithm is presented in Figure 3.8. These kernels are leveraging the parallel processing by utilising a two-dimensional configuration of thread blocks and grid, illustrated in Figure 3.7, enabling efficient computation across spatially organised data. Each block is assigned to process a separate part of the detector consisting of three adjacent sectors containing points able to create a seed (Figure 3.5). The results from different sectors are independent of each other, enabling parallelism of the algorithm. If the number of scheduled blocks exceeds the available GPU resources, they will be processed iteratively.

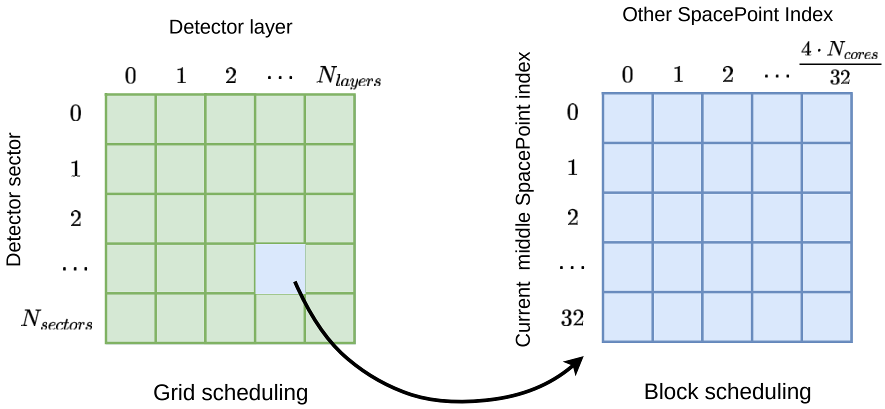


Fig. 3.7 Illustration of the grid and block configuration for the doublet counting and making kernels.

Each thread assigned to process the data from a three-sector window evaluates points from the current layer to determine whether a doublet can be created based on predefined selection criteria. Within each block, the threads with the same x-index are processing the same middle SpacePoint. A maximum of 32 middle SpacePoints can be processed in parallel. Threads sharing the same y-index test whether the SpacePoint with a given index quantifies as a doublet candidate. Each middle SpacePoint is assigned a multiple of four threads to process potential partner SpacePoints, depending on the number of cores on the underlying hardware.

Triplet Selection Cuts

Each triplet considered consists of a pair of doublets sharing a middle point; therefore, the filtering is based on the kinematic and geometric properties of these doublets. The selection cuts described in this subchapter are designed to assess whether the properties of the seed candidate are consistent with those of a track candidate seed.

The first applied selection evaluates if the polar angle θ, therefore, the inclination in the z-r plane of the doublets is similar enough to form a triplet, expressed in Equation 3.2. The track seed in this plane should form approximately a straight line.

$$\left|\frac{\Delta z_1}{\Delta r_1} - \frac{\Delta z_2}{\Delta r_2}\right| = |\cos\theta_1 - \cos\theta_2| < 0.25 \tag{3.2}$$

The next selection fine-tunes the z-r plane inclination cut even more, estimating whether the triplet forms an approximately linear trajectory in the z-r plane, unaffected by the magnetic field (Figure 2.10), after taking into consideration the Multiple Coulomb Scattering (MCS) and pixel cluster measurement error.

The effects of Multiple Coulomb Scattering (MCS), described in Subchapter 2.2.1, are corrected by including a variance factor for the possible scattering angles. They are estimated by Equation 2.1. In order to calculate the maximum variance that can occur, the minimum $p_T = 900\,\mathrm{GeV}$ is used for the calculations.

The cluster measurement error is determined by the detector resolution, for the Inner Detector $err_z = 0.01\,\mathrm{mm}, err_r = 0.13\,\mathrm{mm}$ for the Barrel region and $err_z = 0.13\,\mathrm{mm}, err_r = 0.01\,\mathrm{mm}$ for the EndCap.

Next, the transverse momentum p_T is estimated based on the track curvature (Equation 2.6). The calculation is done by transforming the transverse plane (x, y) coordinates onto a new plane (conformal mapping) to find the linear equation describing the circle. The transformation is illustrated in Figure 3.10.

First, each doublet (SP_m, SP_o), where each SpacePoint is represented by coordinates (x, y, z) and their polar representation (r, ϕ, θ), undergoes a transformation to two-dimensional plane, where the middle point of the triplet is the

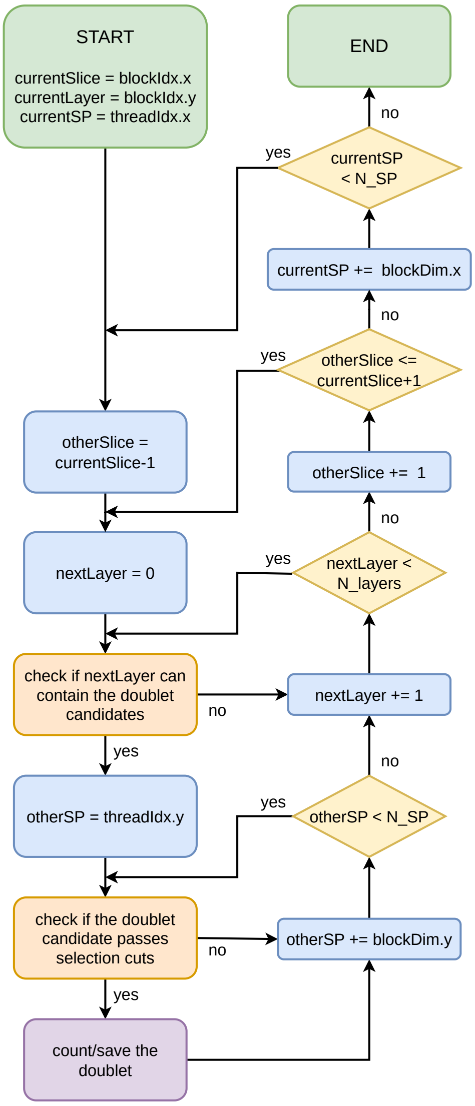


Fig. 3.8 Block diagram representing the operations flow in the doublet counting and making algorithms. Block and thread indices identify the current block and thread executing the kernel.

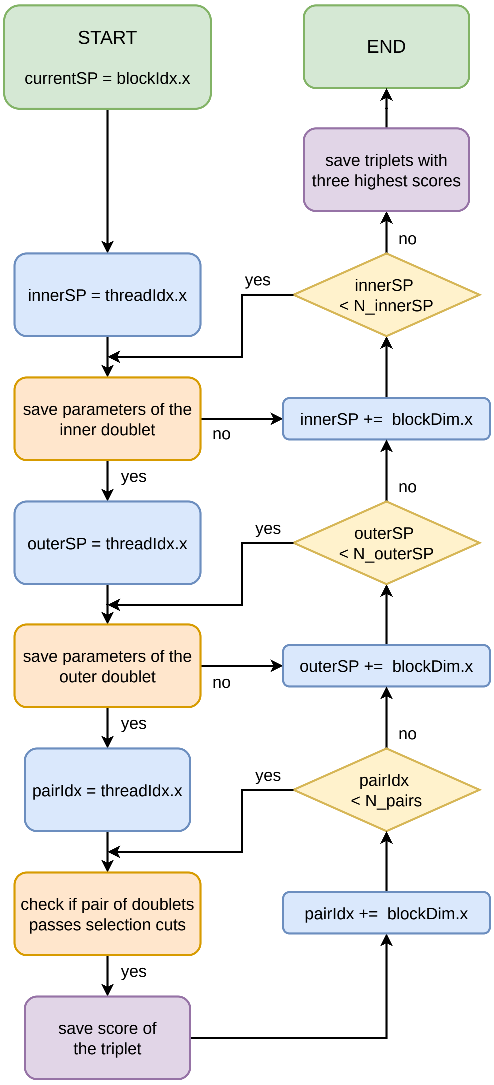


Fig. 3.9 Block diagram representing the operations flow in the doublet matching algorithm. Block and thread indices identify the current block and thread executing the kernel.

center of the plane to simplify the further calculations (Equation 3.3). This way, one of the points of the triplet belongs to the circular path. After the transformation, the SpacePoints are represented by a vector (u, v). The circular shape of the track is in the x-y plane; therefore, the z coordinate is not considered here.

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.3}$$

Next, conformal mapping to a plane (ξ, η), where a straight line represents a circle, is applied to each SpacePoint SP_i. The transformation is described by Equation 3.4, where ρ is the distance of the point from the centre of the (u, v) plane.

$$\xi_i = \frac{u_i}{\rho_i^2}, \qquad \eta_i = \frac{v_i}{\rho_i^2} \tag{3.4}$$

If the inner and outer SpacePoint of the track belong on the same line in the new space (ξ, η), they are part of the same circle and can represent a true track seed.

The last cut is related to the distance of the middle of the seed from the IP, $d_0 < 4\,\text{mm}$. The triplets closest to the IP are chosen to avoid the duplicates.

Seed Selection

For each middle space point, only a selected number of them (the default value is 3) are saved for further track reconstruction. The selection is based on the distance from IP d_0. This way, the amount of duplicate seeds, related to the same track, is reduced, and seeds that are a part of the same track are not separately processed - only the seed closest to the IP is selected. However, the seeds of a true track can also be rejected in favour of fake track seeds very close to IP but not corresponding to a real track.

Triplet Selection Implementation

In contrast to the doublet selection, only one dimension of the grid and thread block is used in the doublet matching implementation (illustrated in Figure 3.11). Each block of the grid processes a separate middle SpacePoint, shared between a

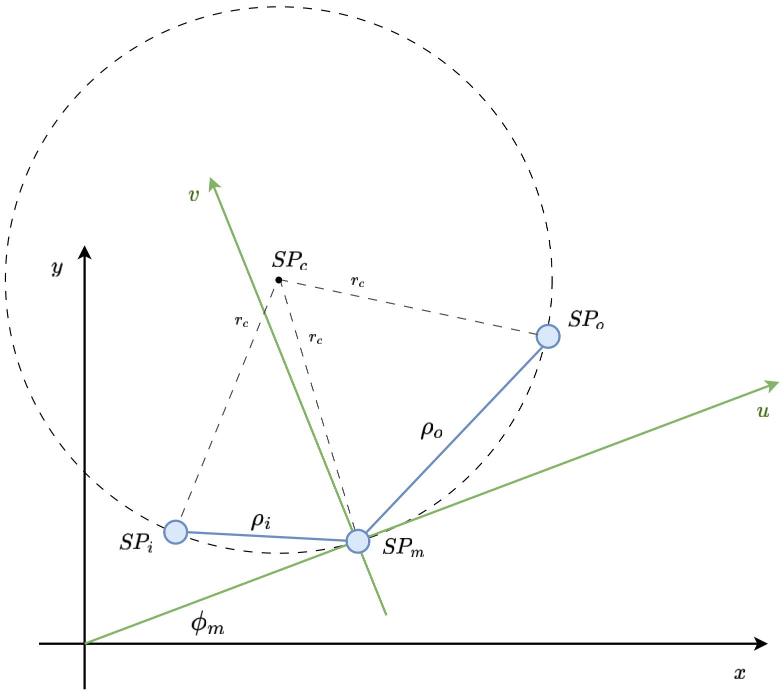


Fig. 3.10 Illustration of a conformal mapping of three SpacePoints. The initial three considered points: inner SP_i, middle SP_m and outer SP_o are transformed from the $x-y$ plane (black axes), to $u-v$ (green axes), where SP_m is the origin of the plane.

doublet in a triplet candidate. As illustrated in the block diagram of this algorithm in Figure 3.9, first, the doublet parameters are calculated and saved, split into inner doublet and outer doublet categories; the classification is done based on the distance from the beam spot. If the distance of the middle SpacePoint (SP_m) is higher than that of the other SpacePoint (SP_o), the doublet is considered as inner; otherwise, it is considered as outer. Next, all the combinations of inner and outer doublet pairs are evaluated to determine which pairs can become track seed candidates. If all selection criteria are passed, a score for the pair is saved, based on the estimated d_0 distance from the IP. Lastly, for each processed middle SpacePoint, the three best candidates are saved as track seed candidates (the number of the best candidates is an adjustable parameter).

A maximum of 1024 middle SpacePoints are processed in parallel by different blocks. The threads are responsible for data preparation and selection of the doublet pairs, associated with the middle SpacePoint, a maximum of 1024 in parallel, which is the maximum number of threads per block possible.

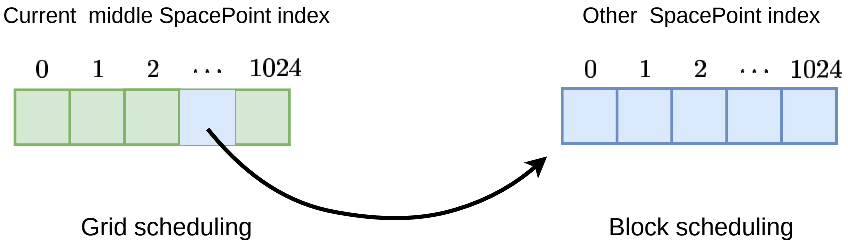


Fig. 3.11 Illustration of the grid and block configuration for the doublet matching kernel.

3.1.3 Performance of Tracking

Throughput

To understand the correctness of the algorithm and identify the inefficient areas, the distribution of the output seeds is analysed. The seeds are divided into three categories:

- true seeds, which are reconstructed into a track in the following steps,
- duplicate seeds, which are a part of an already reconstructed track from another seed,
- fake seeds, that are rejected during the track extension or track fitting stage.

To minimise the memory consumption, the number of duplicate seeds should be reduced, as they are unnecessary for a successful reconstruction yet occupy valuable memory space. To minimise the overall time consumption of the track reconstruction algorithm, the number of fake seeds should be minimised, so that the further steps are not unnecessarily executed. The number of true tracks should remain as high as possible.

Efficiency

The accuracy of the online track reconstruction is measured by considering how many of the true tracks were reconstructed. This measurement can be done with respect to the true information; in the case of the MC simulated data, where the information which the tracks were simulated is preserved. However, considering, for example, the data recorded by the ATLAS detector, the truth particle labels are not available. In this case, the performance is often compared to the track reconstruction accuracy of the offline precision tracking algorithm.

To measure the physics efficiency of the algorithm, an Athena tool IDPVM was used, prepared by the EF Tracking group, to compare the performance of different track reconstruction workflows. There are multiple different measurements provided; however, during evaluation, the analysis in this work focused on the η and p_T distributions of the efficiency. The first one ensures the algorithm correctly reconstructs the tracks from all the regions of the detector, both the forward detector and the Barrel tracks. The second ensures that particles with different traverse momenta, therefore, different track shapes are reconstructed in the event.

3.2 Fast Track Reconstruction Upgrade

The fast track reconstruction algorithm was successfully used during Run 2 and Run 3. However, with the upcoming upgrade of the ATLAS detector and LHC, the computing performance of this algorithm is expected to worsen drastically, based on the experiments with simulated data. The cost of the fast tracking algorithm scales exponentially with pile-up, as shown in Figure 2.6. With expected conditions of pile-up of 140 during Run 4 and 200 during Run 5, the track reconstruction algorithm requires optimisation.

The proposed GPU-accelerated version of the fast track reconstruction [48] is a promising candidate for the future ATLAS online track reconstruction algorithm. The track seeding phase involves the largest volume of data to process (3.5×10^5 SpacePoints on average [1]); moreover, the data processing can be naturally parallelised. This stage directly influences the performance of the subsequent track reconstruction steps, as all created seeds will be extended to a track using a computationally expensive Kalman filter algorithm.

Before the future integration of this algorithm as a part of the online trigger, several upgrades are necessary. The algorithm was configured to support only the Inner Detector geometry, containing fewer layers than the ITk detector and covering a much smaller area (Subchapter 2.1.2), and the expected Run 3 workload (Subchapter 2.1.3). The initial selections for track seeding prepared for

[1] Average value of SpacePoints from ITk-Pixel and ITk-Strips, from simulated ATLAS $t\bar{t}$ events with $<\mu>= 200$

Run 2 and Run 3 require extensive tuning to adapt to the new geometry and to reduce the combinatorics of the fast track reconstruction algorithm. The existing selection cuts should be studied in detail and fine-tuned to reflect the new ATLAS detector. After improving the performance of the complete tracking pipeline to match the accuracy and computing performance of the CPU approach, the track seeding part should be optimised further to fully leverage the parallelism offered by the GPU.

3.3 Related Work

To replace the track seeding approach used in the fast track reconstruction algorithm, a Graph-Based Track Seeding (GBTS) was proposed, also known as FASTrack [54]. The SpacePoints are connected by an edge, within restrictions imposed by the geometry, and sharing similar parameters (similarly to doublet creation in FTF track seeding, Subchapter 3.1.2). The connected edges create a graph - by walking through it with a Kalman filter, pure seeds can be found. At the start of that work (July 2023), the GBTS was in development for the CPU track reconstruction. A GPU-accelerated version is planned as well, available in Athena at the time of writing this work (July 2025). The traditional track seeding approach described in this work was prepared as a baseline for the GBTS implementation and a proof of concept for GPU acceleration for ITk seeding.

As an alternative to the traditional track reconstruction algorithms with track seeding, various machine learning models are proposed. One of them, the Interaction Graph Neural Network, is described in detail in Subchapter 4.1. Similarly to GBTS, the ITk detector readouts are represented as a graph, with nodes representing hits, and edges representing track segment candidates.

Alternatively, a project called TrackFormers [55] proposes the application of transformer networks [56] to solve the tracking challenge. Inspired by the large language models, in one inference step, each hit is classified as a part of a specific track. The process consists of three main stages: embedding hits to a multi-dimensional latent space, a transformer encoder with multi-layer self-attention capturing the global relations between hits, and a classifier assigning each hit a probability distribution over predefined track classes. The project is reported to be in an early stage of development, only available with a simplified dataset, TrackML, not with the ATLAS simulated events. Moreover, the transformer networks have proven to be highly resource-consuming solutions.

3.4 Track Seeding Improvements

This subchapter describes improvements to the GPU-accelerated track seeding algorithm to process the simulated HL-LHC data efficiently.

After applying necessary adjustments in the configuration to support the ITktracking, including using correct tools to extract the detector geometry, scaling the data structures for the expected workload, and expanding the pseudorapidity coverage, the average efficiency of the track reconstruction was 73%. Based on its distribution as a function of η, presented in Figure 3.12, specifically, the Barrel region is the area requiring improvement. The low efficiency will be addressed in this subchapter and improved by relevant selection cuts.

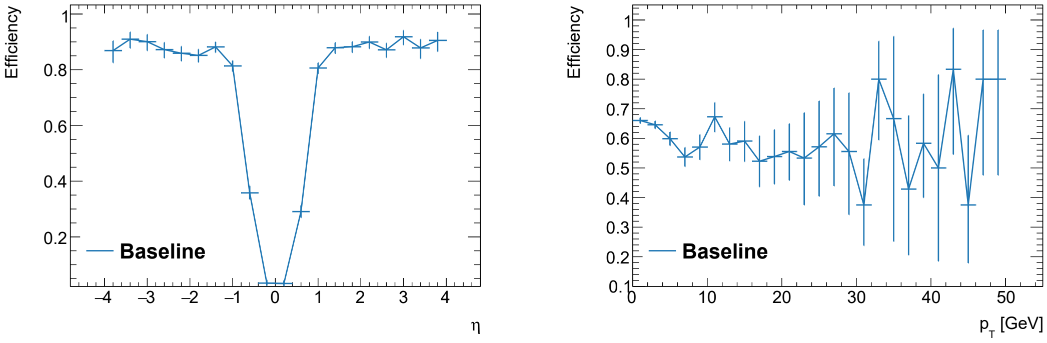


Fig. 3.12 Initial tracking efficiency as a function of pseudorapidity η and transverse momentum p_T for a 100 simulated ATLAS $t\bar{t}$ events with $<\mu>= 200$.

The inefficiency in the Barrel region is not a result of incorrect selection cuts, but rather a consequence of high combinatorial complexity rejecting the correct seeds. Tests with a Single Muon sample (Figure 3.13) show that the efficiency of reconstruction particles is high, just not with the high workload.

The computing performance of the algorithm needs improvement as well. The target performance should match the offline tracking, where the average time of processing one event is 1.52 s; while the current performance of the FTF with GPU-accelerated track seeding on average is 26.7 s. The combinatorial tracking drives the execution time of the algorithm. If the number of input seeds is decreased (in particular, the fake input seeds), the track finding step will not be unnecessarily executed, and resources will be saved.

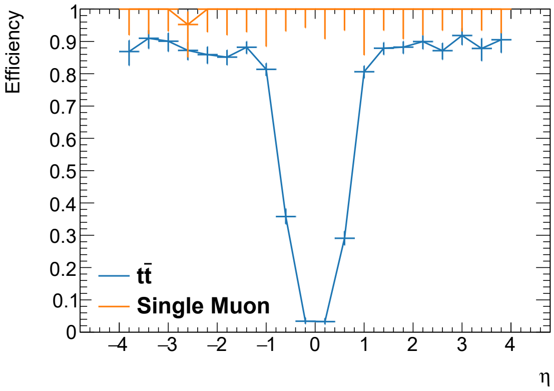
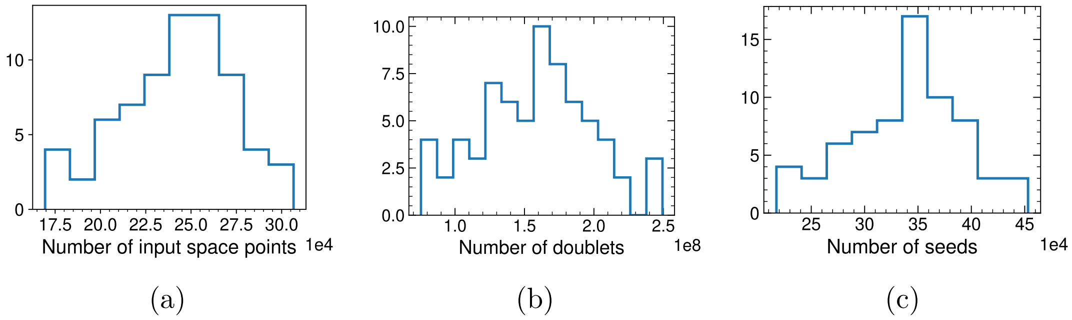


Fig. 3.13 Tracking efficiency as a function of pseudorapidity η 100 simulated ATLAS $t\bar{t}$ events with $<\mu>= 200$ and a Single Muon sample of $p_T = 100\,\mathrm{GeV}$. The high uncertainty for the single muon sample occurs due to the low number of reconstructed tracks - one per event, compared to 1500 per event for $t\bar{t}$ events.

The peak memory consumption depends on the number of SpacePoints, and it is not a bottleneck for the online FTF algorithm - the maximum observed value was 2.5 GB[2]. In this algorithm, the memory is statically allocated to fit the largest expected workload. In the initial state, the maximum number of SpacePoints is 3.5×10^5, the number of created doublets is 5×10^8, and the number of seed candidates is 5×10^5. The numbers were chosen based on the experiments; the distribution of the input (SpacePoints), throughput (doublets) and output (seeds) of the track seeding algorithm is presented in Figure 3.14. The consumption will naturally decrease as the number of processed points is reduced through execution time optimisation.



(a)

(b)

(c)

Fig. 3.14 Number of input SpacePoints (3.14a), created doublets (3.14b) and selected triplets (3.14c) for 100 simulated ATLAS $t\bar{t}$ events with $<\mu>= 200$.

[2] Measured on NVIDIA RTX A5000 24GB

3.4.1 Input Prefiltering

To minimise the volume of data handled by the algorithm, the area of the seed finding was restricted to only the ITk-Pixel detector. In comparison to the geometry of the ATLAS Inner Detector, the number of layers and their range within ITk-Pixel are enough to form seeds for all physics-interesting tracks efficiently (Subchapter 2.1.2).

After the restriction, the number of doublet candidates decreased from over 3.95×10^9 to 1.3×10^8 on average. The number of SpacePoints from the ITk-Strip detector stands for 20% of the total seed number; and the accuracy of track finding is not significantly affected by this change. To further decrease the number of track seed candidates, the middle SpacePoints were restricted to only selected layers, based on the detector geometry, illustrated in Figure 3.15. Each track originating from the IP should interact with one of them, even if one of the layers is missed.

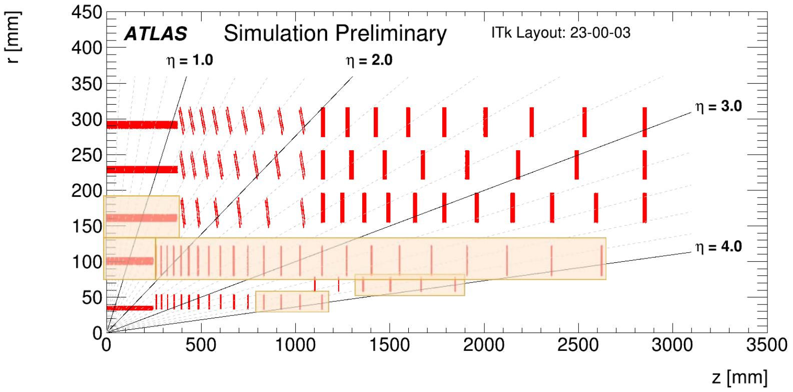


Fig. 3.15 Layout of the ITk-Pixel detector [14] with layers marked in orange colour, that are candidates for containing the middle SpacePoint of the track seed.

Impact on Performance

The impact of the introduced changes on computing performance is presented in Table 3.1. It is not possible with available resources to measure the track seeding algorithm with both ITk-Pixel and ITk-Strips - the ITk-Pixel only version will represent a baseline in this chapter. With respect to the initial data volume, the number of doublet candidates was reduced by 37%, reducing the track seeding time by 18%. The number of the final track seed candidates decreased by 56%, improving the track seeding time by 42% due to processing significantly fewer doublets.

Table 3.1 Summary of the impact of the track seeding input preparation on the number of track seed candidates and the track seeding time.

Optimization	Average number of seeds	Average number of true seeds	Seeding time per event [ms]	Tracking time per event [s]
Restriction to only ITk-Pixel SPs	3.35×10^5	9.46×10^3	1219 ± 43	26.7 ± 0.5
Restriction to only selected layers	1.91×10^5	6.07×10^3	1000 ± 36	15.6 ± 0.3

The accuracy after reducing the input data is presented in Figure 3.16. The efficiency overall decreased; however, the slight loss of efficiency is not significant enough to discard all the improvements with regard to the processing time. The average execution time per event improved by 42% with a loss of 56% of true seeds. The low efficiency will be addressed in further steps.

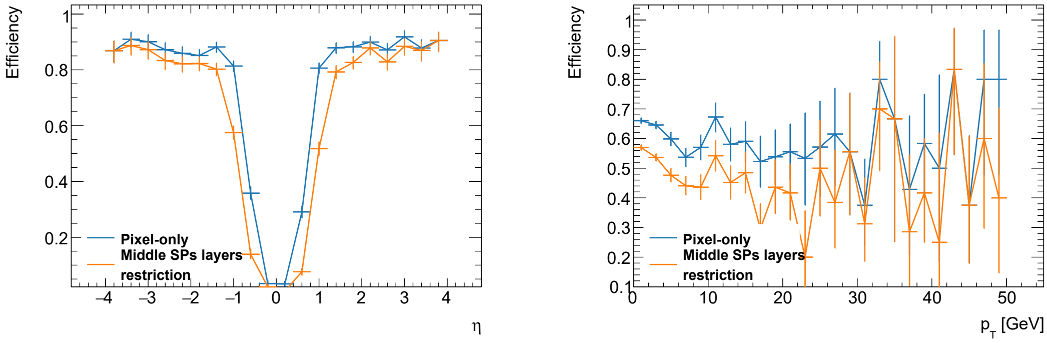


Fig. 3.16 Tracking efficiency after applying selection cuts related to the track seeding input.

3.4.2 Improvements to Doublet Selection

The next series of optimisations added to the algorithm are related to the doublet selection, applied during doublet counting and doublet making.

Doublet Length

The first considered selection focuses on the length of the doublet. This parameter is calculated with Equation 3.5, and is dependent on the distance between detector layers to which the considered SpacePoints belong; therefore, the expected values can be easily restricted by a selection cut. The doublet length value will be different for doublets very close to the IP, where the distance of the layers is significantly smaller than in the outer layers of the detector, and for particles with varying angles of the particle path, dependent on the transverse momentum p_T of the particle.

$$\Delta l = \sqrt{(r_m - r_o)^2 + (z_m - z_o)^2} \tag{3.5}$$

The measured doublet length for true tracks is presented in Figure 3.17.

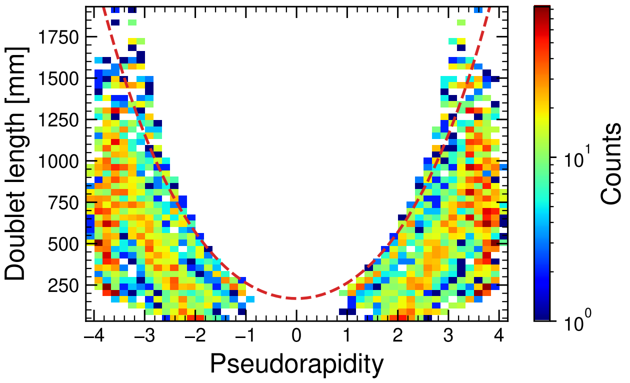


Fig. 3.17 Distribution of the doublet length for different pseudorapidity η regions. The applied doublet length cut is marked with a red dashed line.

The doublet length selection cut was described by a single value in Run 3. To fine-tune this criterion, instead of a single value, a new η-dependent selection cut was applied, restricting the doublet length by a combination of linear (defining maximum value in high-η region) and quadratic functions. The new cut eliminated doublets between layers that are too distant from each other to create a promising track seed candidate.

Cluster Width Uncertainty

The next candidate considered for fine-tuning is the selection cut on the polar angle of the doublet. The maximum and minimum value, determined by the parameters of target tracks, needs to be corrected by the uncertainty on the SpacePoint position measurement, dictated by the width of the clusters creating the point. While in the EndCap region, the paths of the particles cross the sensors nearly perpendicularly, in the Barrel region, the hits are left by both high-p_T, perpendicular tracks as well as low-p_T tracks, almost parallel to the z-axis. This results in both small and large clusters left in the Barrel region, while EndCap only contains small-sized clusters (Figure 3.18).

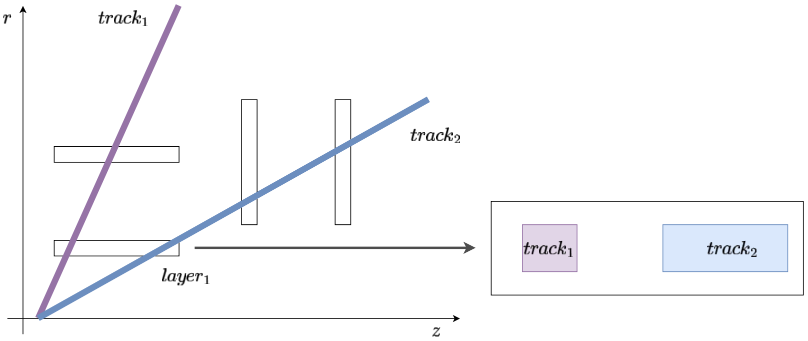


Fig. 3.18 Illustration of the cluster sizes for two tracks: track 1, passing only through the Barrel region, and track 2, with higher pseudorapidity passing through both Barrel and EndCap regions. The clusters left in the Barrel region will consist of small and larger clusters, depending on the polar angle of the passing particle; the EndCap will only contain small-sized clusters.

To fine-tune the doublet selection, the uncertainty related to the cluster size was added to the polar angle selection cut. The smaller the polar angle, the more forward the track is, therefore the bigger the cluster leaving the track. The selection should only accept the points that left clusters with expected sizes, and of a similar size with respect to each other. The maximum and minimum size of the cluster for the ATLAS detector pseudorapidity is presented in Figure 3.19.

Beam Spot Origin

The tracking task focuses on finding the primary track originating from the IP; therefore, the selected doublets should originate from it as well. The tracks can be approximated by a straight line in the $z-r$ plane (illustrated in Figure 3.6). The added selection cut restricts the maximum z distance from the beam spot; the r distance criterion was already in place.

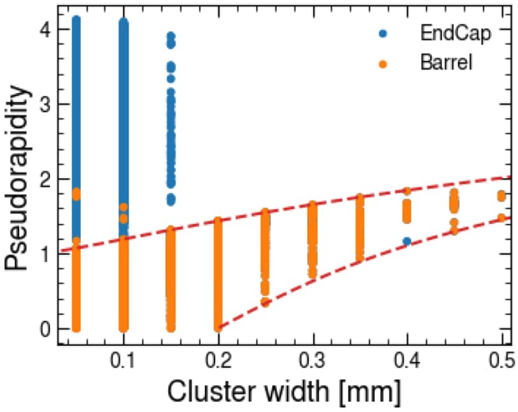


Fig. 3.19 Distribution of the pseudorapidity for different cluster sizes. The applied cut on minimum and maximum pseudorapidity is marked with a red dashed line.

Track Curvature

With the given restriction on the minimum transverse momentum p_T of the particle to reconstruct, a maximum curvature of the track can be calculated for minimum transverse momentum of target tracks $p_T = 900\,\mathrm{GeV}$ (Equation 2.6) and used as a selection criterion. The maximum expected value differs in the Barrel, where mostly high-p_T tracks are expected, compared to the EndCap region. Equation 3.6 is used to approximate the curvature k of a doublet, describing how fast the azimuthal angle ϕ of the track candidate SpacePoints would change with changing distance r.

$$k = \frac{\delta\phi}{\delta r} \tag{3.6}$$

Impact on Performance

By fine-tuning the doublet selection criteria, the track reconstruction efficiency improved significantly; the results are presented in Figure 3.20. The accuracy was the most affected by the beam spot restriction, successfully rejecting fake seeds in the Barrel region of the detector and improving the efficiency in that region from 0 to 0.8. The fake seeds, with a high d_0 score, due to low r distance but high z distance from the beam spot, were eliminated, allowing the true seeds to be chosen as candidates in the last step of the doublet matching.

When comparing the number of seeds before and after applying improvements (Table 3.2), the number remained stable with minor fluctuations after different

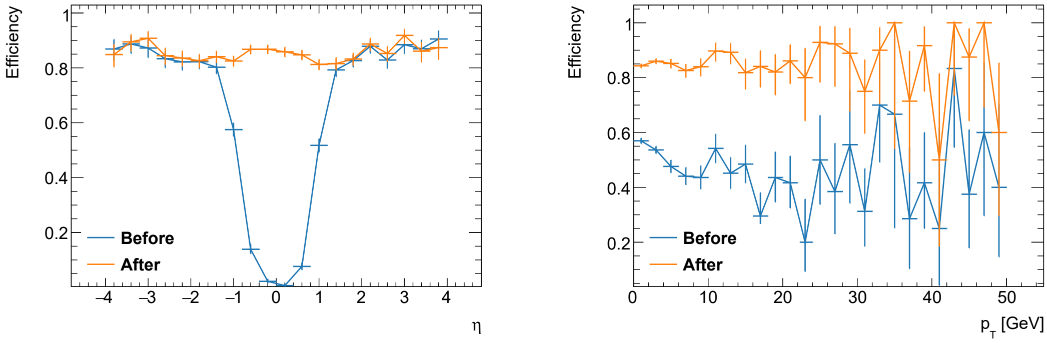


Fig. 3.20 Tracking efficiency after applying selection cuts related to the doublet selection.

enhancements. The track seeding time was reduced by 82% on average by decreasing the number of selected doublets and their early rejection (number of doublets was reduced by 97%, illustrated in Figure 3.21). The number of fake seeds is still high (90%), dominating the tracking computing performance. Therefore, the following steps focus on reducing the number of fake track seeds in order to improve the computing performance of track seeding.

Table 3.2 Summary of the impact of improvements done to the track seeding doublet finding on the number of track seeds and computing performance. "Baseline" refers to the track seeding with the input prefiltering optimisations included.

Optimization	Average number of seeds	Average number of true seeds	Track seeding time per event [ms]	Total tracking time per event [s]
Baseline	1.91×10^5	6.07×10^3	1000 ± 36	15.6 ± 0.3
Doublet Length	1.93×10^5	6.72×10^3	556 ± 18	14.9 ± 0.3
Cluster Width	1.86×10^5	6.66×10^3	503 ± 15	14.5 ± 0.2
Beam Spot z distance	2.10×10^5	7.13×10^3	280 ± 7	19.1 ± 0.3
Curvature	1.83×10^5	5.24×10^3	177 ± 5	17.1 ± 0.3

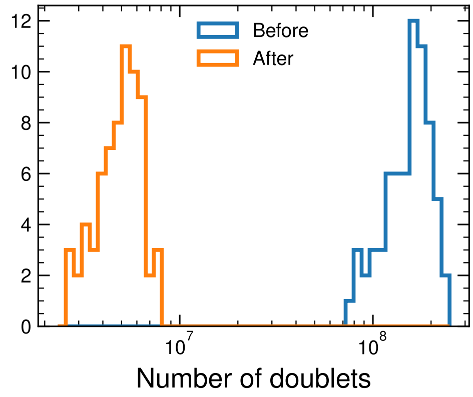


Fig. 3.21 Distribution of number of doublets before and after fine-tuning the input prefiltering and doublet selection.

3.4.3 Improvements to Triplet Selection

Breaking Angle

The first addition to the triplet selection involves an adjustment of the breaking angle cut, illustrated in Figure 3.22. If a doublet pair belongs to a true seed, the polar angle θ of the doublets must be within agreement, considering the uncertainties related to the interaction of the particle with detector material: the uncertainty on the cluster measurement and the multiple scattering effect, impacting the p_T.

The detector resolution determines the first value. For the ITk-pixel, the tracking resolution is much higher than for the Inner Detector because the pixel sizes are much smaller; therefore, the implemented calculation needs reevaluation. The comparison of errors for ID and ITk is included in Table 3.3.

Table 3.3 Error on cluster position for ID and ITk-Pixel detector.

	ID-Barrel	ID-EndCap	ITk-Barrel	ITk-EndCap
δr[mm]	0.01	0.13	0.01	0.02
δz[mm]	0.13	0.01	0.02	0.01

The second term, the multiple scattering effect, described by Equation 2.1, needs to be recalculated for a minimum transverse momentum of $p_T = 900\,\text{MeV}$. The error value depends on p_T and is inversely proportional to p_T; the lower p_T, the higher the uncertainty. Therefore, the highest possible error value for the target tracks is considered as a part of the breaking angle measurement.

The measurement of the breaking angle and its error are included in Figure 3.23 as well as their Gaussian distribution fit. The maximum ratio of the measurement to error is reduced from 3 to 2, covering 99.8% of possible values.

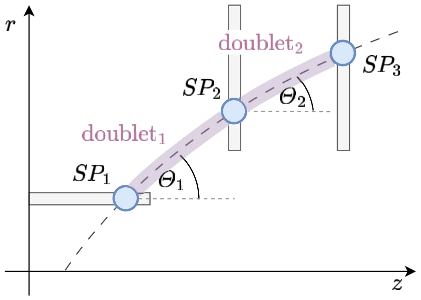


Fig. 3.22 Illustration of the doublet pair, a triplet candidate in the $z - r$ plane. Each of the doublets has its polar angle θ, related to the p_T of the track candidate containing this doublet as a track segment.

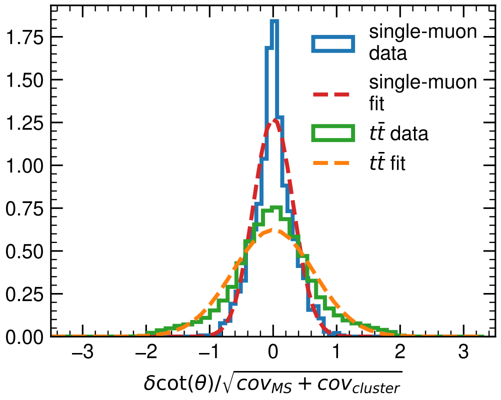


Fig. 3.23 Distribution of the breaking angle measurement and its error related to the cluster measurement and MCS effects.

Triplet Confirmation

After applying all the selection cuts related to the new detector geometry and parameters of target tracks, the number of fake and duplicate seeds still dominates the number of track seed candidates. However, the duplicate seeds can provide helpful information and be leveraged instead of being a redundancy. A new algorithm was added, where at the end of the doublet matching algorithm, all the seeds are searched in order to find duplicates, serving as a confirmation that the seed is correct. For a true track, multiple true seeds can be found (Figure 3.24). The quality score of a seed with duplicates (at least one) is increased to favour it for the final seed selection for the processing block (Subchapter 3.1.2).

To consider two seeds as duplicates, they must fulfil a list of requirements:

- They must share two SpacePoints (inner and middle),
- Their outer SpacePoints will not belong to the same detector layer,
- Their curvatures' direction must agree,

- The estimated difference in the estimated transverse momentum must be within the standard deviation (calculated based on the Single Muon sample).

All the duplicates are accepted - they are rejected in the next step of the track reconstruction at a low computing cost.

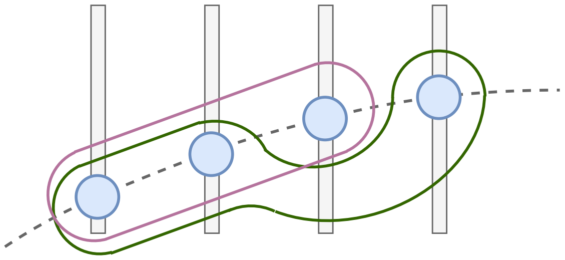


Fig. 3.24 Illustration of two seeds confirming each other. The blue circles represent hits, grey rectangles - detector layers and the green and purple loops - seed candidates. The triplets share two SpacePoints, and if their parameters are within agreement, they may belong to the same track. Predominantly, only true tracks have seed duplicates.

Impact on Performance

After including the adjusted selections in the triplet creation step, the efficiency improvement is observable primarily in the $|\eta| < 2.5$ region (Figure 3.25). The Barrel performance was impacted by the breaking angle cut, and the inclined Barrel region by triplet confirmation. The inclined Barrel ($1 < |\eta| < 2.5$) consists of the highest number of layers, therefore the highest number of SpacePoints, and is prone to contain fake seeds, eliminated by the triplet confirmation cut.

The number of seeds decreased by 74% while preserving the number of true seeds, resulting in a significant improvement in computing performance of the track finding algorithm, presented in Table 3.4. The computational cost of track reconstruction was reduced by 80% after the integration of the triplet confirmation kernel, which led to an 80% reduction in the number of fake seeds (Table 3.5). To further improve it by rejecting the fake track seeds, a global seed confirmation should be included, as described in the next subchapter.

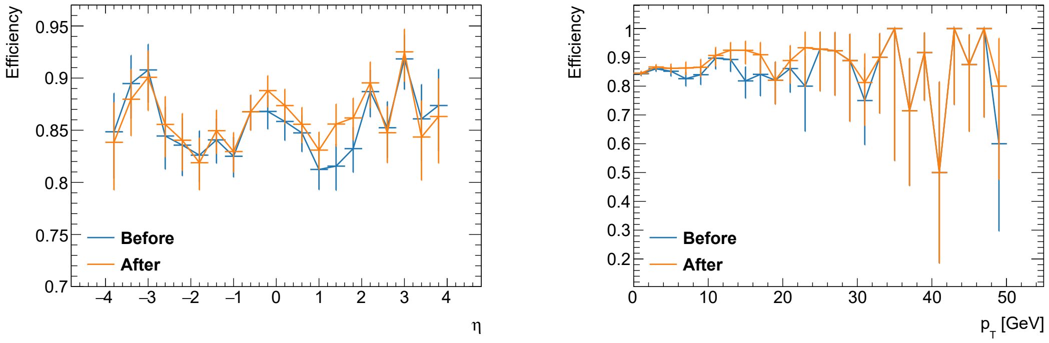


Fig. 3.25 Tracking efficiency after applying selection cuts related to the triplet selection.

Table 3.4 Summary of the impact of improvements done to the track seeding triplet finding on the number of track seeds and computing performance. "Baseline" refers to the track seeding with the doublet finding optimisations included.

Optimization	Average number of seeds	Average number of true seeds	Track seeding time per event [ms]	Total tracking time per event [s]
Baseline	1.83×10^5	5.24×10^3	177 ± 5	17.1 ± 0.3
Breaking angle	1.78×10^5	4.99×10^3	121 ± 4	12.9 ± 0.3
Triplet confirmation	4.56×10^4	4.84×10^3	143 ± 4	3.12 ± 0.11

Table 3.5 Average number of track seed candidates before and after adding triplet confirmation to the triplet selection.

	All	True	Fake	Duplicate
Before	1.78×10^5	4.99×10^3	1.60×10^5	1.31×10^4
After	4.56×10^4	4.84×10^3	2.62×10^4	1.45×10^4

3.4.4 Improvements to Track Seeding Algorithm

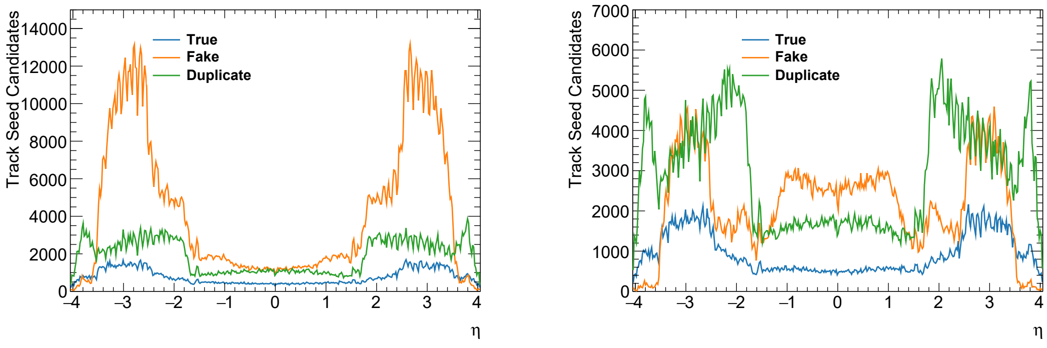
Track Seed Confirmation

In the triplet selection algorithm, each block processes a different middle SpacePoints; therefore, the considered seed duplicates always share the middle point. However, it is possible that in the set of all final track seed candidates, the overlapping seeds do not share a middle SpacePoint, for example, inner and middle points from one triplet and inner and outer seeds in the second one (Figure 3.24). Moreover, for true track candidates, it is expected that multiple seed duplicates exist. This knowledge can be leveraged as a global seed filtering algorithm to reduce the number of fake seeds by tracking duplicate seeds.

The criteria for track duplicates are:

- The seeds are entirely in the EndCap,
- They must share two SpacePoints,
- Direction of their curvatures must agree,
- The difference in the estimated transverse momentum must be within the standard deviation (calculated based on the Single Muon sample).

If the seed has at least two other confirming seeds, it is saved for further reconstruction; others are rejected. The confirmation is not applied in the Barrel region, because it consists of only five layers; therefore, the number of fake seeds and possible duplicates is low. A comparison of the number of seeds as a distribution of pseudorapidity η is presented in Table 3.26.



(a)

(b)

Fig. 3.26 Number of true, fake and duplicate seeds presented as a function of pseudorapidity η, before (3.26a) and after (3.26b), including the seed confirmation.

Impact on Performance

The summary of the change in the amount of track seed candidates is presented in Table 3.6. The number of the fake track seeds was reduced by 60%, constituting 36% of the seed candidates (62% before the cut). The number of duplicate seeds increased; however, they are rejected at a low cost, before the track extension. The number of true seeds decreased as well, leading to lower efficiency of track reconstruction in the EndCap region (Figure 3.27). Considering the significant improvement in the computing performance, the slight performance loss for tracks with low p_T is acceptable in this case.

Table 3.6 Average number of track seed candidates before and after adding track confirmation to the triplet selection.

Optimization	All	True	Fake	Duplicate
Before	4.56×10^4	4.84×10^3	2.62×10^4	1.45×10^4
After	2.39×10^4	3.73×10^3	8.64×10^3	1.15×10^4

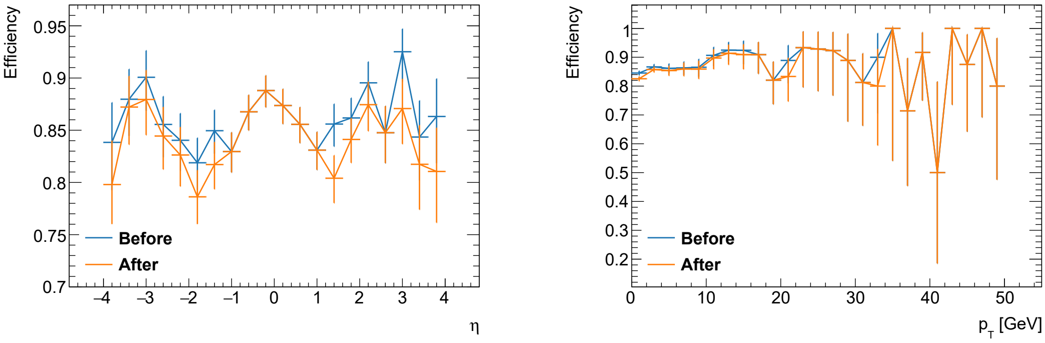


Fig. 3.27 Tracking efficiency before and after including triplet confirmation kernel.

The computing cost, included in Table 3.7, was reduced by 48% for tracking, making the track seeding algorithm competitive with the offline track reconstruction algorithms, therefore fulfilling the specified requirements. The new kernel caused an increase in the track seeding algorithm time; however, the cost of it is not as high as the improvement in the full track reconstruction algorithm.

Table 3.7 Performance of the track finding algorithm before and after including track seed confirmation.

	Track seeding time per event [ms]	Total tracking time per event [s]
Before	143 ± 4	3.12 ± 0.11
After	184 ± 6	1.62 ± 0.49

3.5 GPU Utilization

This subchapter focuses on the improvements in GPU utilisation of the Track Seeding algorithm. This code was implemented using native CUDA; therefore, the evaluation will focus on identifying and incorporating optimisations specific to the CUDA architecture. Performance on different hardware will be considered, as well as GPU occupancy, throughput, and memory access.

3.5.1 Computing Performance

First, the algorithm bottlenecks in the context of GPU acceleration will be analysed and optimised. The computing performance will be evaluated with the available NVIDIA profilers.

Floating Point Precision

One of the first improvements included in the considered algorithm is the reduction of numerical precision. Single-precision (32-bit) floating-point representation can be implemented in place of double-precision (64-bit) with minimal loss of accuracy. According to the NVIDIA Guide, the ratio of the performance of double to single precision is 64:1 [3], suggesting a substantial advantage in using single precision [4]. The improvement in the performance is presented in Table 3.8, where the processing time was slightly reduced. The change in the precision did not affect the accuracy of the algorithm. Not all of the floating point operations were using the double precision before the update (percentage double precision operations: doublet counting 75%, doublet making 71%, doublet matching 8%, seed confirmation 1%), predominantly it was the mathematical C++ functions, automatically casting the value to double precision.

Kernel Performance Analysis

The seed finding consists of four kernels after the addition of the triplet confirmation. In order to understand the performance of the overall algorithm,

[3] Value dependent on the hardware, available in the graphic card specifications, NVIDIA Tesla T4 32:1, NVIDIA RTX A5000 64:1, NVIDIA RTX 5000 Ada 64:1, NVIDIA A100 PCIe 2:1. In all cases, the double precision performance is worse than single precision performance.

[4] Nsight Compute Profiling Guide, profiling output for measurement on NVIDIA RTX A5000

Table 3.8 Performance of the track seeding algorithm with mixed and single precision of floating point representation.

	Track Seeding Time per Event [ms]
Single and Double precision	184 ± 6
Single precision	174 ± 5

the detailed performance of the kernels was analysed as well. The results are presented in Table 3.9.

Table 3.9 Measured performance of the individual kernels on NVIDIA A5000 24 GB.

	Duration [ms]	Average performance [TFLOP/s]	Memory bandwidth [GB/s]
Theoretical Performance	-	27.77	768
Doublet Counting	16.0 ± 1.7	1.98 ± 0.15	0.21
Doublet Making	13.1 ± 1.6	1.94 ± 0.16	1.84
Doublet Matching	35.6 ± 13.4	$(2.27 \pm 0.03) \times 10^{-2}$	1.09
Triplet Confirmation	38.5 ± 15.8	$(7.13 \pm 0.13) \times 10^{-4}$	0.15

The doublet counting and making kernels are reaching good computational performance. Due to the nature of this algorithm, focused on rejecting impossible combinations of two processed points, theoretical peak performance can never be achieved. Some of the threads terminate early to achieve the goal of the algorithm, causing thread stalls. It is impossible to predict which pairs of points will pass all the selections and saturate the performance and which will not; therefore, the stalls are unavoidable.

In contrast, the computing performance of the doublet matching kernel is suboptimal. A maximum of 1500 pairs is processed for each middle SpacePoint. If a separate thread processes each pair, it is not possible to process them all simultaneously; the maximum number of threads per block is 1024 for all modern GPUs. Moreover, this kernel consumes a significant amount of shared memory - 46.87 kB. Each streaming multi-processor has 102.4 kB of shared memory [5]

[5] For NVIDIA RTX A5000, value retrieved with CUDA built-in tools.

The shared memory size for all considered GPUs is available in Table 3.14. On the other hand, increasing the block size and allowing multiple threads to process point pairs concurrently leads to frequent thread stalls due to early termination due to pair rejection. In order to improve the performance, the kernel functionality could be split into two parts: one to reject the incorrect triplets and another to sort and select the best candidates. However, a kernel launch and memory transfer overhead may deteriorate the performance after the split. The second solution would focus on reusing the shared memory.

The triplet confirmation kernel is responsible for considering all the final track seed candidates and identifying those that belong to the same track. The amount of computations in this kernel is minimal - the seed properties, such as estimated transverse momentum, are reused from the preceding kernel. The thread with index 0 saves the confirmed seeds into a final list. During that time, the other threads are idle; moreover, the index 0 may not be coalesced to the memory, where the output is saved.

In summary, the performance of each type of kernel is limited by a different property: for doublet counting, computing is the limiting factor to improve the performance, the size of shared memory is a bottleneck for doublet matching, and for the triplet confirmation kernel, it is the memory access.

The measured triplet seeding algorithm time is longer than the sum of the execution times of the kernels. The complete step includes pre- and post-processing, for example, data preparation and transfer.

Search for Parallelization Configuration

An important factor to consider for CUDA kernels is the optimal block and grid configuration. The parameters for the doublet finding kernels are based on the processed event: number of layers in the detector, number of sectors to process, and the expected number of middle SpacePoints and doublet candidates. In contrast, the triplet finding and triplet confirmation kernels are based on preselected values.

In the CUDA programming model, the maximum number of threads per block is 1024, and optimal performance is achieved when this number is a multiple of 32, aligning with the warp size. For the considered GPU - NVIDIA RTX

A5000 - the maximum number of threads per SM is 1536 [6]. With 64 SM, the $N_{threadsperblock} \times N_{blocks}$ must be lower than 98 304 to process all the blocks concurrently. If the number is exceeded, the blocks will be queued until resources are free.

First, the doublet matching kernel will be considered. The more blocks, the more middle SpacePoints can be processed concurrently; the more threads, the more doublet pairs can be considered simultaneously. The results from processing a middle SpacePoint are independent of each other. While choosing the configuration, the bottlenecks related to the shared memory size limitation and the thread stalls need to be considered, as described in the previous subchapter. The results of processing time performance for different configurations are presented in Table 3.10.

Table 3.10 Performance of the doublet matching kernel with different thread-/block configurations on NVIDIA RTX A5000. The red colour marks the initial performance of the kernel, the blue one - the best one achieved.

	Execution time per event [ms]	Threads Per Block			
		128	256	512	1024
Blocks	512	-	42.1 ± 1.6	-	-
	1024	42.0 ± 15.4	40.2 ± 14.6	39.0 ± 12.4	**70.8 ± 26.6**
	2048	40.6 ± 15.0	38.6 ± 12.3	38.2 ± 14.1	-
	4096	39.5 ± 14.8	38.1 ± 14.1	-	-
	8192	-	37.5 ± 13.9	37.1 ± 13.7	-
	16 384	-	37.7 ± 13.9	36.6 ± 13.6	-
	32 768	-	-	**36.5 ± 13.5**	-

The best performance for the doublet matching kernel was achieved for 512 threads with 32 thousand blocks. During the experiments with an even higher number of blocks, the performance stayed stable. With this configuration, each block will process on average two middle SpacePoints. With the new configuration with a higher number of blocks, the total number of warps is much higher, giving the hardware more flexibility to schedule execution of warps and hide latencies. The chosen number of threads balances between the limitations from scheduling

[6] CUDA Device properties from NVIDIA CUDA Library

too few warps (a small number of threads) and the thread stalls (a high number of threads).

The second considered kernel is the triplet confirmation kernel. Similarly to the doublet matching kernel, each block processes a middle SpacePoint and threads the doublet pairs. The main bottleneck in this kernel is the memory access, described in the previous subchapter. The results of the configuration search are presented in Table 3.11.

Table 3.11 Performance of the triplet confirmation kernel with different thread-/block configurations on NVIDIA RTX A5000. The red colour marks the initial performance of the kernel, the blue one - the best one achieved.

	Execution time per event [ms]	Threads Per Block				
		64	128	256	512	1024
Blocks	512	-	-	55.3 ± 22.9	-	-
	1024	-	50.1 ± 21.0	47.4 ± 18.1	56.7 ± 24.6	**63.6 ± 28.2**
	2048	62.6 ± 25.8	41.9 ± 16.9	45.9 ± 18.3	55.9 ± 24.1	-
	4096	-	**39.6 ± 15.9**	46.2 ± 18.4	-	-
	8192	-	39.7 ± 15.1	47.0 ± 19.0	57.3 ± 25.4	-
	16 384	-	40.6 ± 16.3	48.2 ± 20.0	58.26 ± 26.4	-

The best performance results were achieved for 128 threads per block and 4096 blocks. With a smaller number of threads, the impact of thread stalls caused by the thread with index 0 saving the result is reduced, as fewer threads are blocked during the data saving. The low number of blocks can cause inefficiencies due to warp scheduling, and excessive overhead from kernel launch and scheduling.

The final configuration of 512 threads per block with 32 thousand blocks for the doublet matching and 128 threads per block and 4096 blocks for triplet confirmation will be used in the following studies.

3.5.2 Performance on Different Hardware

Different graphics cards released over the years have different properties, beneficial for various parallel tasks, discussed in Subchapter 2.4.1. In order to make the best choice for the ATLAS EF computing farm, the graphics card with the best algorithm performance should be chosen. This subchapter compares the performance of the track seeding algorithm on different available hardware.

Track Seeding Execution Time

The average processing time per event for the track seeding algorithm is presented in Table 3.12. The best grid and block size configuration was used, as discussed in the previous subchapter. When changing the configuration for other cards, the differences in the performance remained within uncertainty.

Table 3.12 Performance of the track finding algorithm with default threading configuration.

Execution Time per event [ms]	Tesla T4	RTX A5000	RTX 5000 Ada	A100
Track Seeding	433 ± 15	138 ± 3	94.2 ± 1.8	149 ± 3
Doublet Counting	33.1 ± 4.2	16.1 ± 1.7	10.7 ± 1.1	10.9 ± 1.9
Doublet Making	27.3 ± 3.9	13.1 ± 1.6	8.72 ± 1.05	10.9 ± 2.1
Doublet Matching	201 ± 76	35.6 ± 13.4	23.9 ± 8.9	33.2 ± 11.9
Triplet Confirmation	145 ± 63	38.9 ± 15.8	27.1 ± 11.3	34.9 ± 14.7

Among all the tested GPUs, the NVIDIA RTX 5000 Ada demonstrated the best performance. Its advantage is primarily due to its large number of CUDA cores, which enables a high degree of parallelism when executing the algorithm. A clear correlation was observed between the number of CUDA cores and the computing efficiency of the seeding process. Even though other GPUs can achieve higher performance when considering characteristics such as memory access, parallelism has the most significant impact on the final result. The NVIDIA RTX 5000 Ada card is already a technology with one of the highest numbers of CUDA cores on the market (12.8×10^3). The GPU cards that are expected to improve the performance even more are RTX PRO 6000 Blackwell with up to 24.1×10^3 CUDA cores or H200 with up to 16.9×10^3 CUDA cores.

Among all the kernels, the performance of the doublet matching changes the most on different hardware. The differences in performance due to kernel-specific bottlenecks will be discussed in detail in the following subchapter.

Computing Resource Utilization

Table 3.13 presents a summary of measurements of the number of FLOPs per second, compared to the theoretical peak performance that the considered card can achieve. Based on the previous studies (Table 3.9), only the doublet counting and making kernels are limited by the computational capabilities. The kernels do not fully saturate the available computational resources; however, it is not possible due to thread stalls that are unavoidable in this algorithm. The higher the number of CUDA cores in the hardware, the better the performance of the analysed kernels.

Table 3.13 Measured computing performance of the doublet counting and doublet making kernel on different hardware.

FP32 Performance [TFLOP/s]	Tesla T4	RTX A5000	RTX 5000 Ada	A100
Theoretical performance	8.141	27.77	65.28	19.49
Doublet Counting [TFLOP/s]	0.62 ± 0.05	1.97 ± 0.15	2.95 ± 0.23	1.87 ± 0.11
Doublet Making [TFLOP/s]	0.62 ± 0.05	1.94 ± 0.16	2.92 ± 0.25	1.57 ± 0.09

The next discussed bottleneck is the doublet matching kernel and the limitations of the shared memory size. The kernels in this implementation use only static shared memory - its size per SM is predefined, included in Table 3.14, summarising conducted experiments. Based on the available shared memory, only a limited number of blocks can be scheduled at the same time, which can lead to resource underutilization. The shared memory consumption for this kernel is very high - 46.87 kB per block. This way, for most of the available cards, only two blocks can be active at the same time. With each block having 512 threads grouped in 8 warps, the computing capabilities of the SM are underutilised. In order to improve this inefficiency, some of the cards offer dynamic shared memory that could be used to schedule more blocks simultaneously. Another solution, requiring code refactoring as well, is to reuse the shared memory or split the functionality of the kernel into separate selection and sorting, each using effectively less shared memory.

The performance of the last kernel in the track seeding, the triplet confirmation

Table 3.14 Measured GPU occupancy by a number of active concurrent blocks of the GPU for the doublet matching kernel on different hardware.

	Tesla T4	RTX A5000	RTX 5000 Ada	A100
Static shared memory per SM [kB]	65.5	102.4	102.4	167.9
Maximum warps per SM	32	48	48	64
Number of active blocks per SM	1	2	2	2
Average number of warps per SM	15.9	31.7	31.7	31.7
Average occupancy [%]	49.9	66.24	66.24	49.6

kernel, is limited primarily due to execution stalls, as indicated by the NVIDIA profiler. These stalls occur when threads are forced to wait for thread 0 to complete data write operations, which happens twice: at the beginning and the end of the kernel. The bottleneck of slow uncoalesced memory access suggests that the high memory bandwidth of the hardware card would improve overall performance.

Table 3.15 Measured computing performance for triplet confirmation kernel on different hardware.

	Tesla T4	RTX A5000	RTX 5000 Ada	A100
Number of CUDA cores per SM	64	128	128	64
Peak performance [TFLOP/s]	8.14	27.8	65.3	19.5
Memory bandwidth [GB/s]	320	768	576	1555
Duration [ms]	145 ± 63	38.8 ± 15.8	27.1 ± 11.3	34.9 ± 14.7
Compute throughput [MFLOP/s]	196 ± 45	717 ± 133	1031 ± 197	802 ± 147
Memory throughput [MB/s]	96 ± 29	154 ± 38	169 ± 62	150 ± 53

The measured performance of the triplet confirmation kernel is presented in Table 3.15. Although the NVIDIA RTX 5000 Ada exhibits lower memory bandwidth compared to the A5000 and A100 GPUs, this kernel achieves the best performance with this GPU, in terms of the average processing time of one event. In this case, a higher number of CUDA cores per streaming multi-processor and an increased peak floating-point operation throughput help to compensate for the bandwidth deficit. The system's ability to hide memory latency through parallelism is reducing the relative impact of memory access as a performance bottleneck.

Energy Consumption

The last measurement in this subchapter is the energy consumed during track seeding, summarised in Table 3.16.

Table 3.16 Measured computing performance for triplet confirmation kernel on different hardware.

	Tesla T4	RTX A5000	RTX 5000 Ada	A100
TDP [W]	70	230	250	250
Average Power Consumption [W]	65	213	224	193
Energy per event [J]	28.1	29.4	21.1	28.8

The NVIDIA A100 card does not reach the power limit, possibly due to its low computing capabilities. The least energy is consumed by NVIDIA RTX 5000 Ada, proven to be the best fit for this algorithm in the previous studies in this subchapter. This algorithm fully exploits the computing power of the card, operating at maximum power utilisation; however, due to the card's high computational efficiency, it achieves the lowest overall energy consumption.

3.6 Discussion

By applying a series of selection cuts to the track seeding algorithm, the track reconstruction efficiency was recovered from 65% to 85% on average; the detailed efficiency as a function of pseudorapidity and transverse momentum is included in Figure 3.28. Most notably, the efficiency in the Barrel region of the ITk-Pixel detector improved from close to 0 to almost 90%.

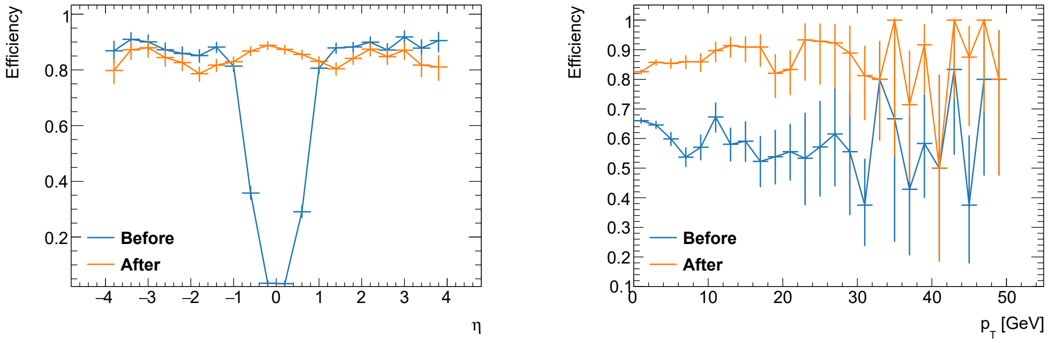


Fig. 3.28 Tracking efficiency after applying selection cuts described in this chapter.

The addition of the triplet and seed confirmation algorithm significantly improved the computing performance of the algorithm. The total number of processed seeds (including both true and fake) was reduced by 96%, while the number of fake seeds alone decreased by 97%. The proportion of their contribution to the total number declined from 91% to 36%. As demonstrated in Table 3.17, this resulted in a substantial improvement in computing performance. The event processing time matched one of the offline track reconstruction, fulfilling the predefined performance objective.

Table 3.17 Summary of the impact of the track seeding optimisations on the number of track seed candidates and the track seeding time. The optimisations with the biggest impact on the performance were chosen.

	Average number of seeds	Average number of true seeds	Seeding time per event [ms]	Tracking time per event [s]
Initial	3.35×10^5	9.46×10^3	1219 ± 43	26.7 ± 0.5
Middle SP restriction	1.91×10^5	6.07×10^3	1000 ± 36	15.6 ± 0.3
Beam spot cut	2.10×10^5	7.13×10^3	280 ± 7	19.1 ± 0.3
Triplet Confirmation	4.56×10^4	4.84×10^3	143 ± 4	3.12 ± 0.11
Final	2.39×10^4	3.73×10^3	184 ± 6	1.62 ± 0.49

The final track reconstruction efficiency (Figure 3.29) is in 98% agreement with the offline track reconstruction for tracks with $p_T < 10\,\mathrm{GeV}$, fulfilling the predefined performance objective. The reconstruction efficiency of tracks with $p_T > 10\,\mathrm{GeV}$ is suboptimal; however, the uncertainty of the result is high due to insufficient data volume.

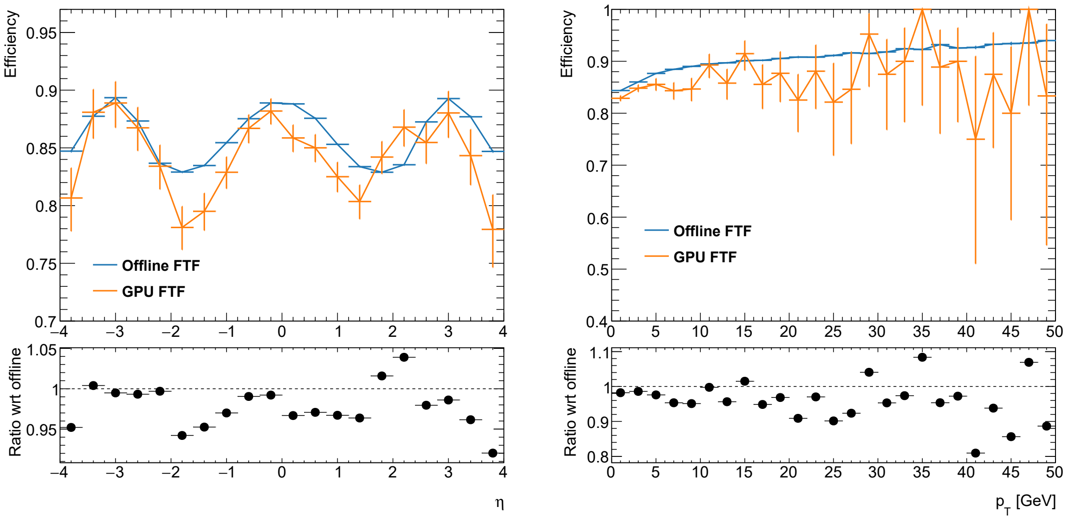


Fig. 3.29 Final tracking efficiency of the FTF algorithm with GPU-accelerated track seeding compared to the efficiency of the offline algorithm.

To further improve the algorithm's performance, the GPU utilisation was analysed in detail with available profiling tools. The reduction to single precision floating point and adjustment in the blocks and grid configuration improved the performance of the algorithm even further to 138 ms per event.

In order to choose the best hardware card for the described algorithm, four GPUs were compared; among them, the NVIDIA RTX 5000 Ada performed the best. It was observed that the computing efficiency increases with the number of available CUDA cores. With the recommended accelerator, the track seeding time per event reaches 94 ms per event, resulting in 93% improvement with respect to the initial performance. Moreover, this graphics card offers the lowest energy consumption per event processing out of the considered solutions.

The computing performance can be further improved by optimising memory offloading and management. The static structures of arrays (SoA) allocations can be replaced by a dynamic one, integration of one of the frameworks developed in the ATLAS Collaboration: vecmem [57] or Marionette [58]. This way, instead of preallocating the data structures for every event, the sizes can be configured dynamically, and the memory footprint of an event will be decreased. The dynamic shared memory could be used, improving the throughput of the memory-consuming kernels. Moreover, system failures due to exceeding the hardcoded maximum sizes of the data containers will be mitigated.

The track seed redundancy is a problem present in all traditional track reconstruction algorithms. The selection cuts described in this chapter were also applied in the ACTS framework [24], and can be further used by other projects in order to fine-tune the selection. The computationally expensive GBTS algorithm can significantly improve the redundancy.

Chapter 4

Memory Footprint Optimisation of Interaction Graph Neural Network

As an alternative to the traditional track reconstruction algorithm performing the track seeding, extension and fitting, discussed in the prior chapter, a machine learning solution was proposed by the GNN4ITk group. The particle tracks are reconstructed from the ATLAS detector hits by using an IGNN model, classifying the track segment candidates into true or fake based on their geometric properties and their graph neighbourhood. This chapter presents the details of such a model and applied optimisations improving the memory footprint of IGNN inference.

4.1 Interaction Graph Neural Network for Track Reconstruction

This subchapter describes the track reconstruction approach using the IGNN. Its steps are illustrated in Figure 4.1.

First, from the read-out detector hits, a graph is constructed using Module Map or Metric Learning algorithms described in [60]. The graph nodes represent hits, and the edges represent track segments. The first approach to graph construction uses the map between detector modules to reject node connections that are not possible. The map was created based on 90 000 $t\bar{t}$ simulated events, saving all the pairs of modules creating the target particle trajectories. The

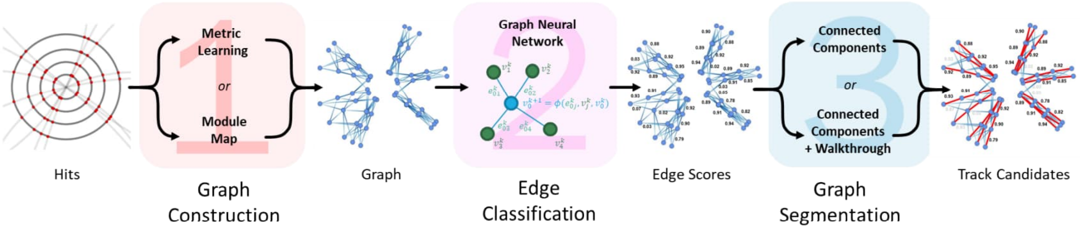


Fig. 4.1 Schematic overview of the GNN4ITk pipeline [59].

second method applies a neural network that embeds the SpacePoints into a latent space and connects them based on their proximity. The network is trained in a way that SpacePoints from the same target particle are close to each other in the embedded space, with the embedding radius being a hyperparameter of the training. This approach uses a fast Fixed Radius Nearest Neighbour algorithm [61] to create the connection graph, optimised for large-scale data processing under resource-constrained conditions.

After the graph construction, the next step uses the IGNN to classify the graph edges by evaluating the probability that the edge is a true track segment. This subchapter focuses on the detailed description of this step.

Finally, the Graph Segmentation algorithm walks through the created paths in the graph in order to extract the final tracks. The Connected Components algorithm [62] is illustrated in Figure 4.2. Graph edges with sufficiently high edge classification scores are grouped into connected components, where each node can be linked to others by different paths. If each node in a connected component has only one incoming and outgoing edge, the path is directly classified as a track candidate. Otherwise, the paths are disentangled by a Wrangler algorithm, creating a path from a node by choosing the outgoing edge with the highest score. If more than one edges have a high classification score $s > 0.8$, multiple paths are created, and eventually, the longest one is saved as a track candidate.

4.1.1 Input Data

In the input graphs to the IGNN, each node represents a point, where a passing particle interacted with the detector and has four features related to its position:

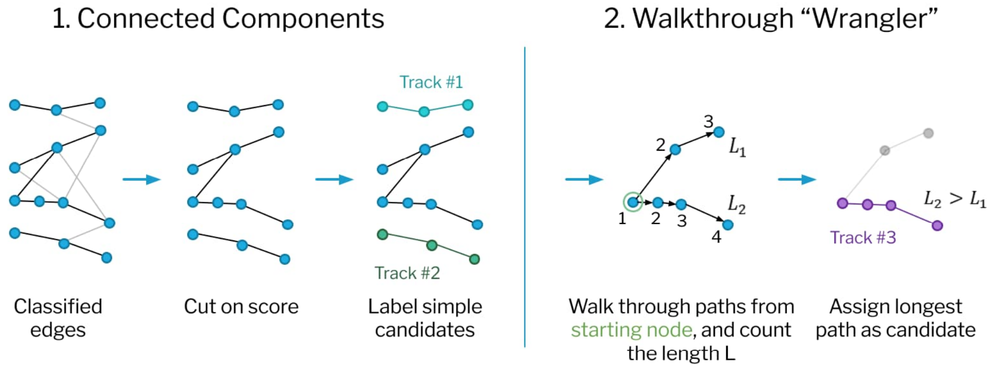


Fig. 4.2 Schematic overview of the connected components and walkthrough algorithms. Figure based on [63].

r, z, ϕ and η described in detail in Subchapter 2.2.2. A node from the ITk-Strip detector can additionally contain features of two clusters composing the hit, with the same features as the reconstructed position: r, z, ϕ and η. The cluster information is necessary to account for the point position uncertainty, described in more detail in Subchapter 3.1.1.

Each edge connecting two hits represents a track segment candidate, with six geometry-related features: δr, δz, $\delta\phi$, $\delta\eta$, ϕ_{slope} slope, and r-ϕ_{slope}. For n_1 and n_2 being the nodes connected by the considered edge, the features can be described by Equation 4.1.

$$
\begin{gathered}
\delta r = r_2 - r_1 \quad \delta z = z_2 - z_1 \\
\delta\phi = \phi_2 - \phi_1 \quad \delta\eta = \eta_1 - \eta_2 \\
\phi_{\text{slope}} = \frac{\delta\phi}{\delta r} \\
\bar{r} = \frac{r_1 + r_2}{2} \quad r\text{-}\phi_{\text{slope}} = \phi_{\text{slope}} \cdot \bar{r}
\end{gathered}
\tag{4.1}
$$

The chosen edge and node features are crucial for reconstructing the helix-shaped particle track and distinguishing it from the other tracks. The GNN4ITK group selected them throughout years of research to encode the most critical features of the true tracks for accurate reconstruction. During the graph construction stage, the graph is preprocessed to eliminate impossible track segments based on the requirements of the target tracks.

The size of the final graph, apart from the edge and node features, depends on the conditions of particle collision and the detector granularity. The example sizes for datasets considered in the chapter are presented in Table 4.1.

Table 4.1 Sizes of graphs from chosen data samples. The chosen dataset ATLAS consists of 10 000 events and TrackML of 1500 events.

Dataset Name	Number of nodes		Number of edges	
	avg	max	avg	max
ATLAS	3.36×10^5	5.83×10^5	9.15×10^5	3.17×10^6
TrackML	1.29×10^4	1.90×10^4	4.81×10^4	8.53×10^4

4.1.2 Model Description

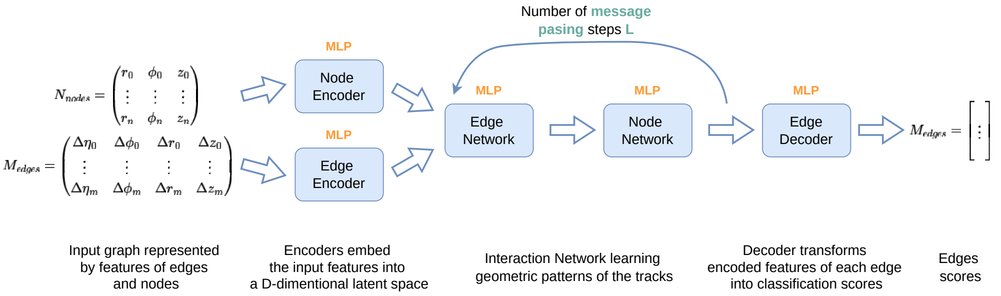

Number of message pasing steps L
MLP
Node Encoder
MLP
Edge Encoder
MLP
Edge Network
MLP
Node Network
MLP
Edge Decoder
$N_{nodes} = \begin{pmatrix} r_0 & \phi_0 & z_0 \\ \vdots & \vdots & \vdots \\ r_n & \phi_n & z_n \end{pmatrix}$
$M_{edges} = \begin{pmatrix} \Delta\eta_0 & \Delta\phi_0 & \Delta r_0 & \Delta z_0 \\ \vdots & \vdots & \vdots & \vdots \\ \Delta\eta_m & \Delta\phi_m & \Delta r_m & \Delta z_m \end{pmatrix}$
$M_{edges} = \begin{bmatrix} \vdots \end{bmatrix}$
Input graph represented by features of edges and nodes
Encoders embed the input features into a D-dimentional latent space
Interaction Network learning geometric patterns of the tracks
Decoder transforms encoded features of each edge into classification scores
Edges scores

Fig. 4.3 Schematic overview of the IGNN forward step. The input features of nodes and edges are encoded into a latent space. Encoded features are the input to the edge and node networks, learning the geometric patterns of the tracks by a message passing mechanism. As the last step, features of the edges are decoded into edge scores. All the neural networks are MLPs with 2-3 hidden layers, depending on the network. The diagram is based on Ref. [40].

This edge classification model consists of the following steps, illustrated in Figure 4.3. First, the edges and nodes are encoded into a latent space of size D with two dedicated MLPs, each with two hidden layers. Next, the edge and node features are aggregated through L message passing steps. During the aggregation, the features of the neighbouring edges or nodes are encoded into the latent space with a separate MLP for each iteration, and propagated through the neighbourhood. After the last step, the edge features are decoded and transformed into a score.

Considering all the processing steps and the number of message passing iterations $L = 8$, the model consists of 20 MLPs. Eight message passing iterations is a default setting for the IGNN; this way, the neighbourhood of distance eight can be considered for each hit, covering even the longest forward region tracks. The detailed functions of the MLP types are presented in Table 4.2 and the summary of their sizes in Table 4.3. For graphs, where the number of edges is higher than the number of nodes (which is true for the graphs representing the collisions), the edge encoder is the network with the highest number of parameters, as well as one with the largest input tensor.

4.1.3 Model Evaluation

To evaluate the classification, the GNN Edge-wise Efficiency and Purity scores [40] are used as the accuracy metric, presented in Equation 4.2. High efficiency of IGNN indicates that the majority of the edges were correctly classified as true segments, and high purity, that the number of incorrectly classified edges is low. By considering both metrics, the model will be able to correctly identify particles and maintain the number of track segment candidates as low as needed, thereby reducing the associated computational costs of the following track reconstruction steps.

$$\begin{gathered}\text{efficiency} = \frac{\text{number of correctly classified true track segments}}{\text{number of true track segments}} \\ \\ \text{purity} = \frac{\text{number of correctly classified true track segments}}{\text{number of track segments classified as true}}\end{gathered} \tag{4.2}$$

The loss function of the IGNN model is prepared to separately consider the fraction of the true and fake classified track segments, and balance their contribution to the loss based on model hyperparameters. In the process of calculating the loss function, the background edges, linking two successive SpacePoints of a non-target particle, need to be distinguished from simply incorrect, nonexistent edges. The background edges should be masked to prevent the model from classifying edges from non-target particles with similar topology to target particles as false, which could mislead the training process and impact the performance of the model.

Table 4.2 Names and function of the MLPs building the IGNN.

Name	Function
Node Encoder	Encode node features into latent space
Edge Encoder	Encode edge features into latent space
Edge Network	Compute new edge features based on the features from the source and destination nodes
Node Network	Compute new node features based on the aggregated features from the incoming and outgoing edges
Edge Decoder	Decode the edge features into latent space features
Edge Output Transform	Decode the latent space features into a score describing if the edge can be a track segment

Table 4.3 Sizes of the model parameters, inputs and outputs for MLPs building the IGNN. Node and edge networks are created L times, separate for each message passing step; the numbers in the table reflect just one step. Parameters: N_N - number of input graph nodes, N_E - number of input graph edges, F_N - number of node features, F_E - number of edge features, D - size of the latent space.

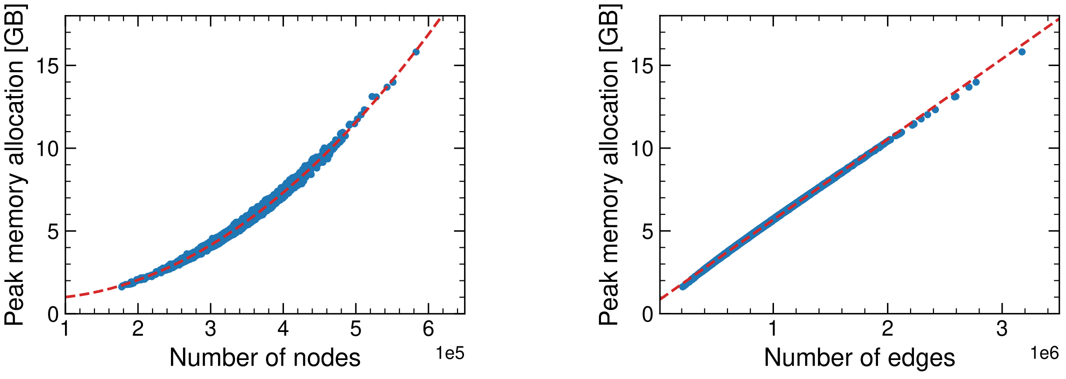
Name	Input Size	Output Size	Sizes of layers	Number of parameters (weights)
Node Encoder	$N_N \times F_N$	$N_N \times D$	$F_N \times D, D \times D, D \times D$	$F_N \cdot D + 2D^2$
Edge Encoder	$N_E \times F_E$	$N_E \times D$	$F_E \times D, D \times D, D \times D$	$F_E \cdot D + 2D^2$
Edge Network	$N_E \times 3D$	$N_E \times D$	$3D \times D, D \times D, D \times D$	$5D^2$
Node Network	$N_N \times 2D$	$N_N \times D$	$2D \times D, D \times D, D \times D$	$4D^2$
Edge Decoder	$N_E \times D$	$N_E \times D$	$D \times D, D \times D, D \times D$	$3D^2$
Edge Output Transform	$N_E \times D$	$N_E \times 1$	$D \times D, D \times 1$	$D^2 + D$

4.2 Memory Footprint of Interaction Graph Neural Network

Despite achieving competitive accuracy of the track reconstruction, the IGNN faces several inefficiencies limiting its computational performance. The first area of improvement for the GNN4ITk track reconstruction pipeline discussed in this work is its memory footprint.

The proposed IGNN model with $HL = 128$, achieving the highest accuracy, requires up to 80 GiB/event during training and up to 16 GiB/event during inference[1]. By reducing the instantaneous memory consumption, it would be possible to use cost-efficient and market-available GPUs with smaller memory for both training and inference, or process multiple graphs simultaneously. This work focuses on the application of the track reconstruction in the online trigger; therefore, this chapter concentrates on improving the memory consumption only during the inference.

The peak memory consumption per input graph for the chosen IGNN is presented in Figure 4.4. Even though the average memory consumption is 5.26 GB, the algorithm should be able to process the largest events, representing the busiest collisions, on the chosen accelerator. Within the current configuration, the peak memory consumption for the biggest event with 3.17×10^6 edges reaches 15.8 GB; however, this value is highly dependent on IGNN configuration as well as the input graphs and the graph construction algorithm.



(a)

(b)

Fig. 4.4 Peak memory consumption of the IGNN inference as a function of number of nodes (4.4a) and edges (4.4b) in the input graphs. The red dashed line represents data fit: quadratic for 4.4a and linear for 4.4b.

In order to understand the memory footprint, one of the available PyTorch performance analysis tools was used to create a memory snapshot. An example of a snapshot is presented in Figure 4.5, and a summary of the most expensive allocations is included in Table 4.4.

[1] For the largest event in the considered ATLAS dataset

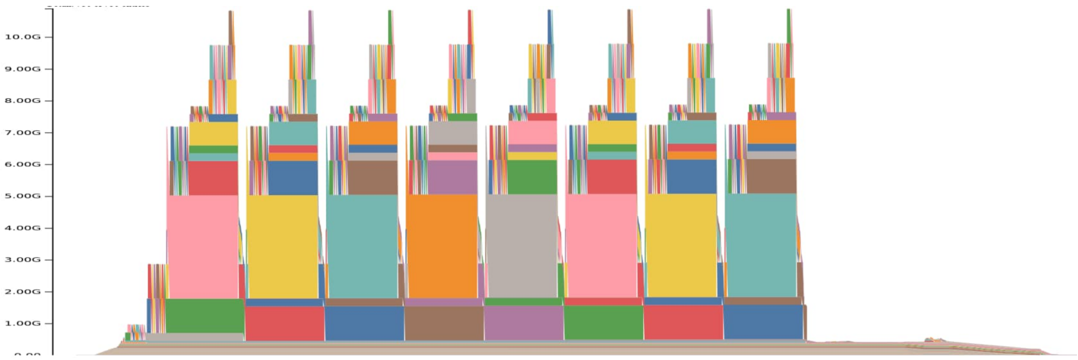


Fig. 4.5 Snapshot of memory allocations during a forward step of the IGNN inferring a simulated $t\bar{t}$ event with pile-up 200 with ITk geometry. Different blocks represent different memory allocations. Eight peaks are visible, corresponding to message passing steps.

Table 4.4 Selected most memory-consuming allocations in the IGNN inference. Measured on NVIDIA RTX A5000 with 32-bit float precision.

Function name	Allocation size	Estimated size for the biggest graph [GB]
Edge network input tensor	$N_E \times 3D$	4.87
Edge network output tensor	$N_E \times D$	1.63
Edge decoder output tensor	$N_E \times D$	1.63
Node network input tensor	$N_N \times 3D$	0.89
Node network output tensor	$N_N \times D$	0.29

Memory footprint is dominated by one allocation, namely the input tensor to the MLP classifying the encoded features of edges and their source and destination nodes (edge network). The size of that tensor is $3D \times N_E$ (Table 4.3), where $3D$ equals the features of the edges, and related source and destination nodes, each encoded to a latent space of size $D = 128$. With the average number of edges per graph $N_{\text{edges}} = 0.91\,\text{M}$ (maximum observed $N_{\text{edges}} = 3.17\,\text{M}$) and using single-precision floats, the average size of the structure is equal to 1.40 GB (4.87 GB).

This chapter focuses on improving the peak memory consumption of the inference by applying the substepping mechanism, sequentially filling and processing memory-consuming temporary tensors, such as the input to the edge network MLP.

4.3 Related Work

Optimisation of the inference memory consumption is a crucial task for enabling efficient deployment of neural networks, particularly in resource-constrained environments. There is a range of techniques available to reduce the memory footprint.

Quantisation [64], [65] is a well-known compression model reducing the precision of weights and activations of the model to a lower-bit floating point representation. It is known for effectively reducing memory footprint and accelerating computation, making it a widely adopted optimisation technique in FPGA development. However, the process of quantisation can lead to a degradation in model accuracy and a need for additional fine-tuning to recover some of the lost performance.

The second widely-known method for memory footprint optimisation is *pruning* [66], [67], reducing the size and computational complexity of neural networks by eliminating redundant or less important parameters. However, the memory footprint of the presented model is driven by the size of the input structures rather than the number of model parameters themselves.

The neural networks often use a mechanism called *batching*, where the input is split into smaller sequences and processed independently. In the context of the track reconstruction and its message passing mechanism, partitioning the input requires additional preprocessing. One batch should contain the neighbourhood of size eight (configurable parameter) of the considered edges to classify them correctly. The regionalisation for the IGNN track reconstruction is currently studied as a compression technique for FPGA acceleration [68]. This solution can also be deployed in the future for the GPU-accelerated version of the algorithm; however, more studies are required on the region overlap to remove the duplicates, which can degrade computing performance.

4.4 Reducing Instantaneous Memory Consumption by Substepping

During the MLP evaluation on a GPU, the input tensor is split among the threads and blocks by the underlying PyTorch scheduling. However, the evaluation input is created in the memory before the parallel scheduling; therefore, it can exceed the available memory capacity.

To fit the memory-consuming structures into any size-restricted hardware, the algorithm must be able to split the inputs and process them sequentially. It is not always possible, however, in the case of the IGNN, such a mechanism can be implemented. The edge network input, identified as the most memory-consuming allocation, is only a temporary tensor, and processing it sequentially will not have an impact on the accuracy. The splitting of the tensor is described by Equation 4.3. For example, a tensor of size $N = 3\,174\,201$ can be decomposed into $q = 3$ substeps of size $S = 100\,000$ and a last substep with a remainder $r = 174\,201$. The mechanism of explicit splitting of the workload into steps of predefined size is called *substepping*.

$$N = q \cdot S + r \tag{4.3}$$

where:

N = size of the tensor
q = number saturated substeps
S = substep size
r = remainder

The pseudocode for the substepping mechanism is presented in Listing 4.1. The maximum size of a processed segment is defined as an inference hyperparameter. The last segment, with the size of the remainder, is often underfilled, containing fewer elements than the maximum allowed.

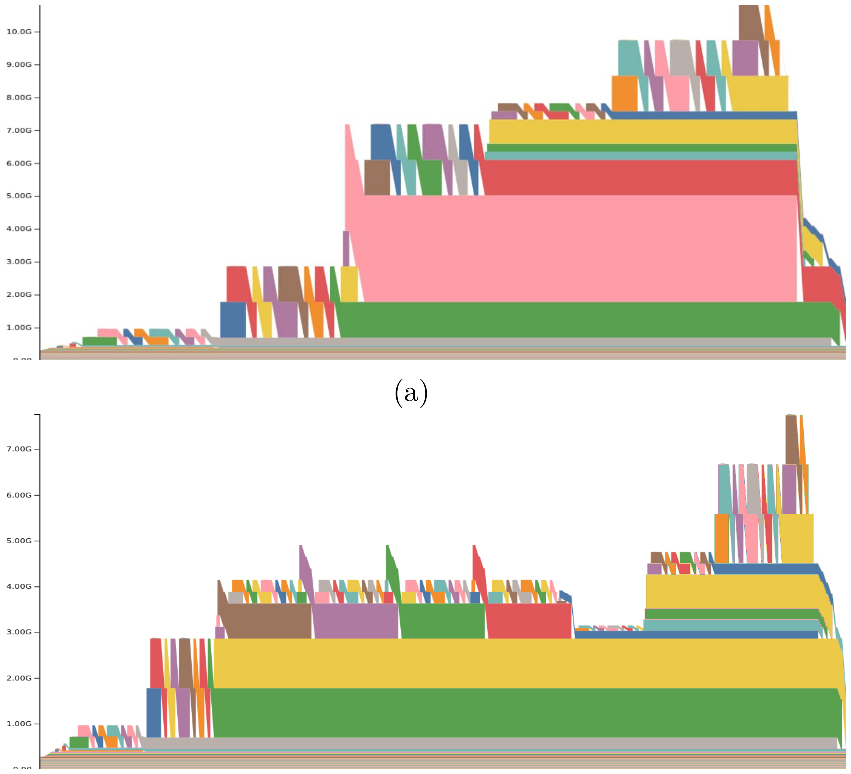
Listing 4.1 Pseudocode of substepping mechanism applied in IGNN to evaluate the edge network

```
n_substeps = ceil(n_edges/max_edges_per_step)
result[n_edges][latent_space_dim] = 0
for i from 0 to n_substeps:
  # Calculate the start and end of the substep
  start_index = i*max_edges_per_step
  end_index = (i+1)*max_edges_per_step - 1
  if end_index >= n_edges:
    end_index = n_edges

  # Prepare input to MLP for the currently processed edges, consisting of edge features and source and destination node features for each edge
  mlp_input = cat([edge_features[start_index:end_index], node_features[src[start_index:end_index]], node_features[dst[start_index:end_index]]], dim=-1)

  # Evaluate MLP on the partial input and save the result
  result[start_index:end_index] = mlp(mlp_input)
```

An example memory snapshot with the substepping mechanism applied is shown in Figure 4.6. The biggest allocation is split into five sequential substeps, four with the maximum step size, and the last one containing the remainder.

4.5 Impact of Substepping on GNN Inference

In order to analyse the impact of substepping on the performance of the algorithm, different computing metrics will be collected regarding the memory consumption, inference time, and FLOPs. All the tests are done with acorn [69] implementation of the IGNN with 1000 simulated ATLAS $t\bar{t}$ events with $<\mu>= 200$ transformed into graphs with the module map algorithm. The hardware used was NVIDIA RTX A5000 24GB, unless specified otherwise.



(a)

(b)

Fig. 4.6 Snapshot of memory allocations during one message passing step of the IGNN without (4.6a) and with (4.6b) applied substepping mechanism of 500000. The dominating allocation (4.6a pink) was replaced by the edge output preallocation (4.6b yellow) and five substeps (4.6b brown, purple, green, red and blue). The last allocation (blue), with the size of the remainder, is only deallocated at the end of the message passing function.

Inference Memory Footprint

To evaluate the effects of the substepping on the inference memory footprint, first, the peak memory consumption for different substeps is examined. To measure the memory consumption during the inference, PyTorch CUDA memory-related tools were used to read the peak memory consumption recorded during processing. The peak memory consumption as a function of substep size is presented in Figure 4.7.

Adding the substepping mechanism does not increase the memory consumption compared to the baseline model, even with the preallocation. The trend of the measurements is determined by the number of edges in the processed graphs and the substep size. With the applied substepping, the MLP input is overwritten for each substep, and only the last allocation with remainder remains

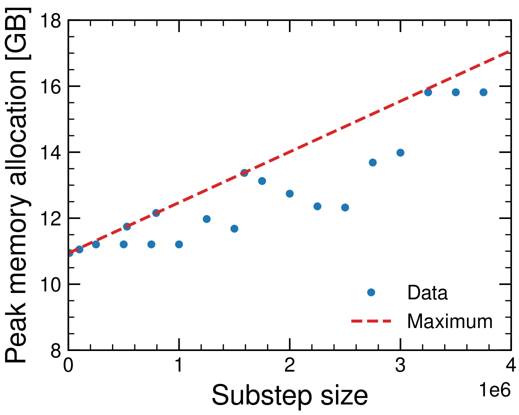


Fig. 4.7 Peak memory consumption for 100 graphs with the highest number of graph edges for different substep sizes. The trend of measurements is determined by the size of the last substep - the edge network input is only deallocated after finishing processing one message passing iteration. The data measurements are highly dependent on the number of edges in processed graphs - the red line marks the maximum memory consumption assuming the last substep is saturated.

until the processing unit leaves the function and the memory is deallocated. The sizes of the last allocation for an example graph are presented in Table 4.5. Whenever a substep size exceeds the next divisor of the number of edges, the remainder is low, and the last allocation will be significantly smaller at a cost of executing more substep iterations. The lower the substep size is, the smaller the remainders will be.

Table 4.5 Example number of edges in the last substep (remainder) for a selection of substep sizes for an example graph.

Substep size	Number of substeps	Remainder
3 500 000	1	3 174 201
1 590 000	2	1 584 201
1 580 000	3	14 201
500 000	7	174 201
100 000	32	74 201
50 000	64	24 201

The model should not be fine-tuned to fit the data sample; therefore, the maximum possible allocation should be considered rather than the measurements themselves, marked with a red line in Figure 4.7.

The experiments were repeated on other graphics cards as well; no difference in the memory-related performance was observed.

Inference Time

The measurement of edge network input preparation and evaluation time with included substepping is presented in Figure 4.8. The performance without substepping applied is measured at 128 ms ± 2. With a substep size of 500 000, which achieved the best performance during tests, the processing time is measured at 130 ms ± 1; therefore, the cost of substepping is within the standard deviation of the measurement.

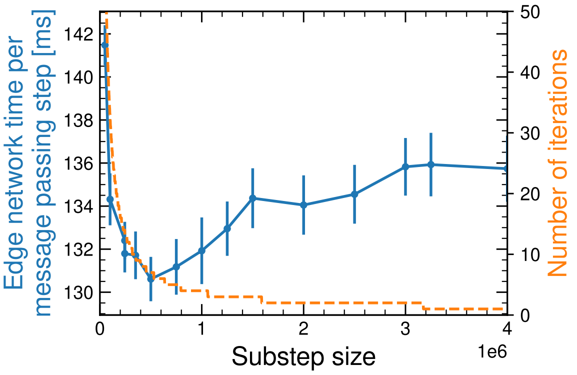


Fig. 4.8 Elapsed time of edge network input preparation and evaluation of IGNN over repeated experiments on a graph with the highest number of edges in the available data sample.

The cost of substepping is driven by two operations: concatenation of the tensors with features of edges, incoming, and outgoing nodes, and the model evaluation. With the number of steps increasing exponentially with decreasing substep size, the cost of the repeated operations affects the performance drastically for steps smaller than 500 000 (up to 6 iterations per input graph). For each processing step, time spent in a function can be described by Equation 4.4.

$$t_f(N_{edges}) = \left\lfloor \frac{N_{edges}}{N_{substep}} \right\rfloor \cdot t_f(N_{substep}) + t_f(N_{edges} \bmod N_{substep}) \tag{4.4}$$

where:

t_f = elapsed execution time

N_{edges} = number of edges in the processed graph

$N_{substep}$ = substep size

The inference performance of the IGNN on different graphics cards will be considered in the Chapter 5.

Computing Resource Utilization

Figure 4.9 presents the number of FLOPs as a function of a substep size. The increase for steps smaller than 100 000 is driven by an increase in the number of operations of the matrix multiplication (SGEMM) (executed for each substep). The performance of the hardware, presented in Figure 4.10, decreases for substeps smaller than 100 000 due to the exponential increase of the number of FLOPs. The decline of the performance could be related to the time overhead from the other repeated functions, increased kernel launch overhead - multiplication for each substep is launched separately, or reduced parallel efficiency - the multiplication of a smaller matrix does not fully saturate the available hardware.

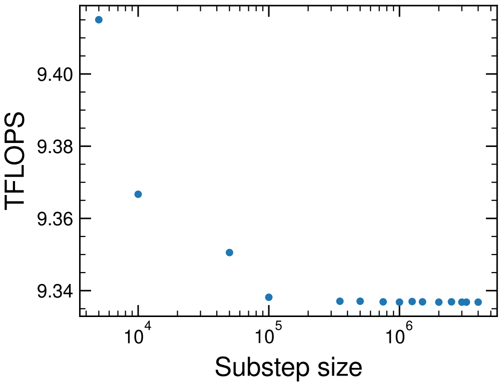


Fig. 4.9 Number of FLOPs as a function of substep size. The measurement was done for one event, with the highest number of edges in the available data sample.

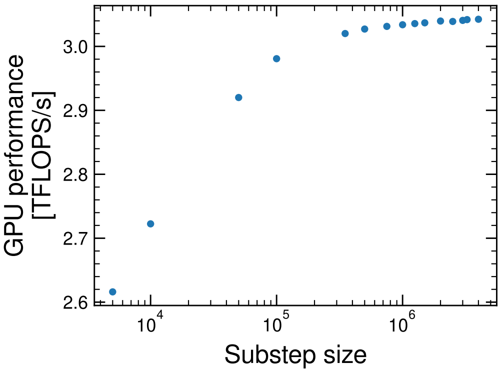


Fig. 4.10 Number of FLOPs per second as a function of substep size. The measurement was done for one event, with the highest number of edges in the available data sample.

4.6 Discussion

The first presented optimisation for the IGNN track reconstruction, substepping, significantly reduces the instantaneous memory consumption during IGNN inference. With this mechanism applied, graphs of any input size can be processed, a critical requirement for the ATLAS experiment's data-taking operations, where every event must be processed. The track reconstruction efficiency is not affected; only the temporary structure is split for independent input matrix rows. Application of substepping does not require retraining of the network and can be adjusted based on the used hardware.

By analysing the performance of the model with different substep sizes, presented in Figure 4.11, the optimal substep size for NVIDIA RTX A5000 with 24 GB of memory is 500 000. It guarantees optimal inference time with no performance degradation observed with respect to the baseline, while achieving a substantial 30% reduction in peak memory consumption.

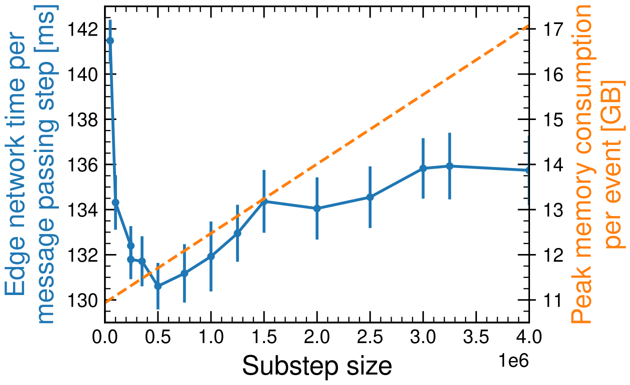


Fig. 4.11 Measured time of edge network input preparation and evaluation of IGNN on the graph with the highest number of edges in the available data sample.

The substepping mechanism is highly dependent on the neural network implementation; therefore, it cannot be easily transferred to other models. However, if, after the memory consumption analysis, a temporary memory-consuming structure is identified, a similar mechanism can be applied.

Chapter 5

Inference Optimisation of Interaction Graph Neural Network

The chapter focuses on the inference optimisation of the IGNN for the particle track reconstruction, described in detail in Subchapter 4.1. There are no specific execution time requirements in the EF Tracking requirements document [26], described in Subchapter 2.2.4; however, the online trigger system must be able to perform track reconstruction at a rate of 1 MHz on a computing farm, with potential accelerator cards, created with a limited budget. At the time of writing, the computing farm design was not finalised yet.

Apart from the track reconstruction step, other algorithms must be executed as well within this time frame, including the clusterisation, selection, or muon reconstruction; therefore, the IGNN inference time should be as fast as possible while preserving the accuracy of the model. Moreover, while studying the inference time, one must not only consider the average time per input graph but also the time of processing the biggest graph possible, to consider the cost of processing the busiest collisions and understand the performance scaling trends. Of course, the simulated data is limited, but the biggest graph was selected from the available sample, and it will be used for the benchmarking.

The measured inference time of the IGNN forward step is presented in Figure 5.1. The inference time scales linearly with the number of graph edges, driving the cost of executed operations. To understand its performance in detail and identify the bottlenecks, a detailed profiling was performed, with its results presented in Table 5.1.

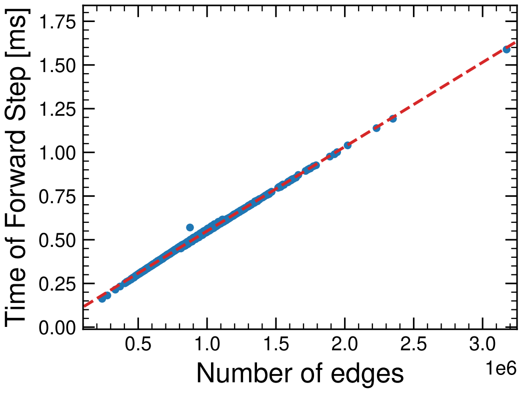


Fig. 5.1 Measured IGNN inference time on NVIDIA RTX 5000 Ada for 1000 input graphs.

Table 5.1 Summary of fraction of inference time of the four most expensive PyTorch functions, executed during IGNN inference. Measurement was done with the cProfile profiler on NVIDIA RTX A5000.

Function	Fraction of processing time [%]
Fully connected layer evaluation `Linear`	32.1
Layer normalization `LayerNorm`	7.94
Rectified Linear Unit activation `ReLU`	6.40
Tensor concatenation `cat`	1.73
Other functions	51.8

The most time-consuming function of the IGNN inference is the linear layer evaluation. The matrix multiplication of the large input tensors and the weight matrices drives the performance cost. The most expensive multiplication - the Edge Network evaluation - performs multiplication of $N_{edges} \times 3D \otimes 3D \times D$ (Table 4.3), for the model configuration used in this work and the biggest event is it equal to $3.17 \times 10^6 \times 384 \otimes 384 \times 128$.

Reducing the dimensionality of the matrices involved in the linear transformations is a promising approach for reducing the computational cost of the IGNN inference.

Pruning

Pruning [66], [67] is a model compression technique aimed at reducing the size and computational complexity of neural networks by eliminating redundant or the least important parameters. It identifies and removes the parameters, most commonly weights, that contribute the least to the accuracy of the model. Pruning is also proven to prevent overfitting of the model [70], [71]. However, due to the unstructured nature of pruning, the parameter removal introduced by pruning does not necessarily lead to improved computational performance on massively parallel processors. In order to take advantage of introduced sparsity, sparse operators could be used [72]–[74]; however, they tend to perform better with regular sparsity patterns.

In contrast to unstructured methods that remove individual elements, *structured pruning* removes subtensors of defined shape that embrace multiple individual elements. This way, it not only reduces model complexity, but it is also in line with the computing performance optimisations for massively parallel processors. By introducing sparsity in the structure, sparse operators can be used, or the pruned tensors can be resized. The studies in this chapter focus on the second approach - by reducing the size of the weight matrices, the functions should take less time.

5.1 Related Work

The model compression techniques used for reducing the memory footprint, described in Chapter 4, can also be used for model compression targeting the inference time. Quantisation can successfully reduce the model size by reducing the precision of weights and activations of the model to a lower-bit floating point representation. They are proven to not only improve the memory footprint of the model but accelerate its inference as well [75]. Unstructured pruning, covered in the previous subchapter, successfully identifies the least important parameters; however, without additional operators leveraging the sparsity, no performance benefits can be observed.

Another technique to reduce the model size, the *knowledge distillation method* [76]–[78] was proposed, where a small model, called a student, is trained by a complex model, the teacher. The student learns from the teacher's output probabilities or intermediate representations, which encode richer information than if it were trained from scratch. This allows for better generalisation for the student model. Knowledge distillation is particularly used in resource-constrained environments, as it produces a compact model with the power of the original one. However, it still requires sufficient capacity to capture the complexity of the teacher model.

5.2 Search for Optimal Pruning Configuration

The search for the optimal pruning configuration is a time-consuming task, requiring fine-tuning of multiple independent parameters. To optimise this process, the search will be performed with an IGNN model with the same architecture but with a much simpler dataset TrackML, described in Subchapter 2.1.4 The size of a model was adjusted to 32 HL as a compromise between efficiency and timing; results from the search for the right size are presented in Figure 5.2. The training of TrackML IGNN is 11 times faster than ATLAS IGNN. It is assumed that the same conclusions will translate to the network trained on ATLAS data.

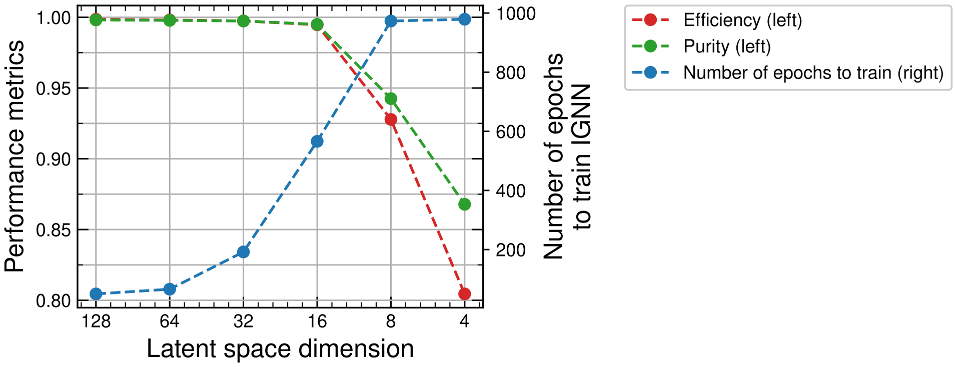
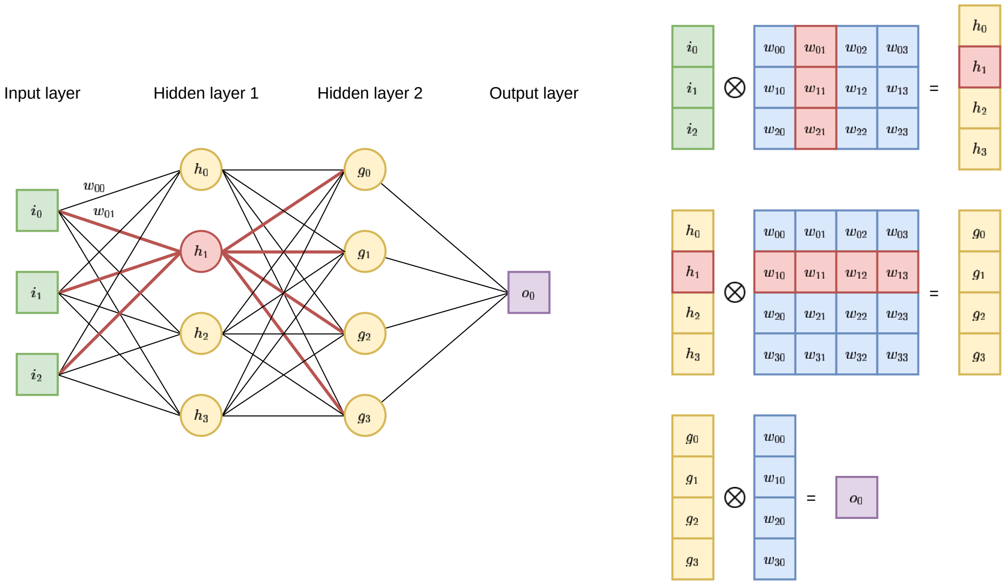


Fig. 5.2 Accuracy metrics and the training time for the IGNN model with different latent space dimensions. To compromise between efficiency and the time needed to train, HL=32 was chosen for further studies. Its accuracy loss is minimal compared to the base model of HL=128, but it is not as overparameterised. It requires around 200 epochs to reach the minimum.

During the structured pruning, we aim to reduce the size of the model by removing the unimportant neurons in the hidden layers of the network. In order to achieve that, the pruning algorithm masks the rows and columns of the weight matrices to remove the corresponding neurons in the hidden layers. The process is illustrated in Figure 5.3.



$$\begin{bmatrix} i_0 \\ i_1 \\ i_2 \end{bmatrix} \otimes \begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} \otimes \begin{bmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}$$

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} \otimes \begin{bmatrix} w_{00} \\ w_{10} \\ w_{20} \\ w_{30} \end{bmatrix} = \begin{bmatrix} o_0 \end{bmatrix}$$

Fig. 5.3 Illustration of structured pruning of an example MLP. The figure on the left side presents the connections between input, output, and hidden layers in a fully connected network. The right side illustrates the matrix multiplications performed during the evaluation of this network. One neuron h_1, marked in red, is pruned in this example, and related connections to the previous and following layers need to be removed. The removal will result in a smaller size of the weight matrices used in the model.

While searching for a setting that reaches the highest pruning percentage without a significant accuracy loss, different criteria are considered. The pre-trained model will be pruned and fine-tuned for 10 epochs, which is sufficient to recover as much prediction quality as possible. The number of epochs was chosen based on conducted experiments with IGNN.

In the remainder of this work, prediction quality is assessed based on the metrics described in Subchapter 4.1.3. The minimum GNN Edge-wise efficiency for the model is marked on the plots with a red dashed line, and it is defined as 0.99 for IGNN. The pruning percentage on all the presented figures refers to the percentage of parameters that can be pruned - that is, only from parameters from hidden layers.

The following subsections will present a selection of various experiments to highlight the findings with regard to different pruning configurations.

Importance Scores

To decide which subtensors of the pruned weight matrices should be kept and which should be removed, first, an importance score needs to be assigned to each of the considered subtensors. Three different scoring methods are tested in this work; the first, and the most popular, is **weight magnitude pruning (WMP)**. A high absolute value of a parameter implies a high importance in the accuracy score; therefore, it should not be pruned. However, during training, when pruning is applied, some of the weights have not reached their minimum, and their value can change. To mitigate this issue, in **gradient magnitude pruning (GMP)**, the current gradient on a weight is factored into an overall score of $|w_i \cdot g_i|$. This method is not perfect either; it informs only if the weight is going to change in the current epoch, but not about the direction of the change. To consider that factor, the third method, **optimal brain damage (OBD)** [66] is used. It estimates the saliency of a parameter by approximating the second-order derivative of the loss function. This way, parameters that are contributing least to the loss, and thus accuracy, can be removed.

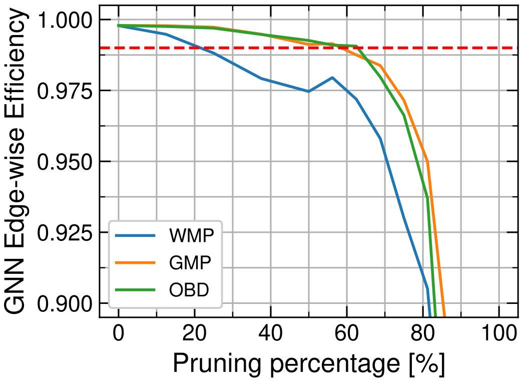


Fig. 5.4 Comparison of GNN Edge-wise Efficiency with chosen importance scores: WMP, GMP, and OBD. The plots show the accuracy averaged over five experiments. Pruning is applied iteratively with 10 epochs of fine-tuning in between iterations, with L2 norm for importance score aggregation and TopK pruning criterion.

The effects of structured pruning for these three scores are presented in Figure 5.4. The accuracy is not highly impacted by changing the method; the WMP score performed the worst, not reflecting the importance of the neurons well enough. The other two methods maintain similar efficiency with the same pruning amount.

To decide the best pruning configuration, the cost of the methods during training should be considered as well. In the case of the importance scores, GMP and OBD require additional computations. Compared to the WMP, the GMP accounts for only 0.2% increase of the average training epoch time, which falls within the standard deviation of the measured results, while for the OBD method it is 3% [1].

In the following studies in this chapter, the GMP importance score will be used as the best importance score method, reporting the highest efficiency at a low cost.

Importance Score Aggregation Methods

The structured pruning considered in this work focuses on masking rows and columns of weight matrices; therefore, it requires aggregation of the importance scores for each subtensor. To perform it, two metrics are considered: **L1 norm** and **L2 norm**.

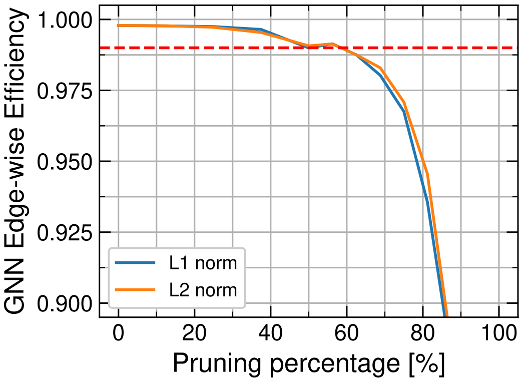


Fig. 5.5 Comparison of different importance score aggregation methods. The plots show the accuracy of the model averaged over five experiments. Pruning is applied iteratively with 10 epochs of fine-tuning in between iterations, with GMP importance score and TopK pruning criterion.

Based on the results of the experiments, illustrated in Figure 5.5, the L2 norm achieves the best results. It emphasises the importance of outliers in the weight distributions; thus, assuming a magnitude correlates with importance, it preserves important weights better.

Considering all the factors, the L2 norm is a better choice to be used for applying structured pruning to GNNs, as also reported in Reference [79].

[1] Measured for IGNN, with 1200 training input graphs NVIDIA RTX A5000 24 GB

Pruning Criteria

To select which of the subtensors should be removed while pruning, a criterion should be applied to the aggregated importance scores. This paper considers two of them: **TopK** and **threshold**. The first method removes the k least important parameters, based on the importance score. For $\mathbf{x}$ being the aggregated importance scores of the given weight matrix, and $M^{m\times n}$ being the mask for the structures to prune, the TopK pruning procedure can be described by Equation 5.1.

$$\begin{gathered}\operatorname{topk}(\mathbf{x},k) = \underset{S\subset\{1,\ldots,m\},|S|=k}{\arg\max} \sum_{i\in S}|x_i| \\ m_{i,j}^{(k)} = \begin{cases} 1, & \text{if } i \in \operatorname{topk}(\mathbf{x},k) \\ 0, & \text{otherwise} \end{cases} \quad \text{for all } j \in \{1,\ldots,n\}\end{gathered} \tag{5.1}$$

The second considered criterion, the threshold method, removes all subtensors with an aggregated importance score below a defined threshold. For $\mathbf{x}$ being the aggregated importance scores of the given weight matrix, $M^{m\times n}$ being the mask for the structures to prune, and t being the chosen threshold, the threshold pruning procedure can be described by Equation 5.2.

$$m_{i,j}^{(t)} = \begin{cases} 1, & \text{if } |x_i| \geq t \\ 0, & \text{otherwise} \end{cases} \quad \text{for all } j \in \{1,\ldots,n\} \tag{5.2}$$

In the experiments, multiple layers are pruned at the same time. In order to simplify the process and use one threshold value for all the layers, the importance score was normalised before applying the threshold. For $\mathbf{x}$ being the aggregated importance scores of the given weight matrix, the normalisation is described by Equation 5.3. Since the pruning is applied iteratively, some of the values in the importance score vector are already equal to zero because they were pruned in past iterations. For the current pruning, the threshold value should be applied to unpruned structures; therefore, the current minimum is recognised.

$$x_i = \frac{x_i - \min(\mathbf{x}_{x>0})}{\max(\mathbf{x}) - \min(\mathbf{x}_{x>0})} \tag{5.3}$$

Based on conducted experiments, the results of which are presented in Figure 5.6, the TopK criterion performs better than the threshold-based selection.

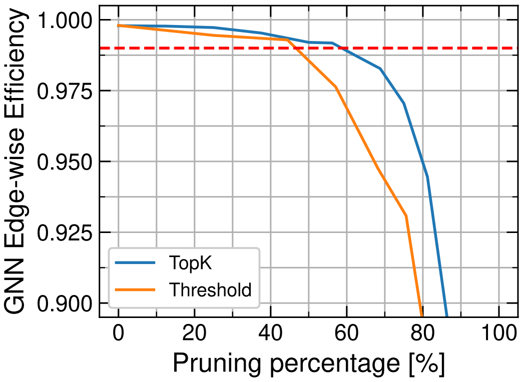


Fig. 5.6 Comparison of GNN Edge-wise Efficiency after applying different pruning criteria: TopK and threshold. Pruning is applied iteratively with 10 epochs of fine-tuning in between iterations, with GMP importance score and L2 norm for importance score aggregation.

Considering that the pruning is applied iteratively, one needs to take into account that the TopK pruning is fine-tuned for more epochs in total. It is hard to control the cumulative pruning percentages for threshold-based pruning because it is highly dependent on the values of the importance scores. There is no control over how many of the weights in the network are below the defined threshold; therefore, one iteration can prune a very small or a very high percentage of the weights. Depending on how many weights are pruned in each of the iterations, threshold pruning can terminate earlier compared to TopK, where the pruning amount for each iteration is defined.

Pruning Procedure

Finally, it is essential to consider when and how often the pruning is applied throughout the training process.

The experiments were conducted in two ways: **one-shot pruning** with the desired amount and fine-tuned or **iterative**, with 10 epoch fine-tuning in between epochs, as illustrated in Figure 5.7. The pruning amount in each iteration is chosen based on the experiments; the initial steps are bigger, but as the accuracy limit is closer, the steps become smaller. Moreover, the memory alignment should also be taken into account; therefore, the sizes of steps have a value of 2^N.

The GPU kernels, such as SGEMM, are scheduled by the underlying PyTorch libraries with consideration of the warp organisation, where each warp consists of 32 threads. A range of kernel tile shapes [2] is available to maximise warp utilisation. When matrix dimensions align with the available tile shapes, all threads within each warp remain active and fully utilise available computing resources. In case of "edge tiles" containing residual matrix elements that do not fit the tile shape, their performance is suboptimal due to low thread occupancy.

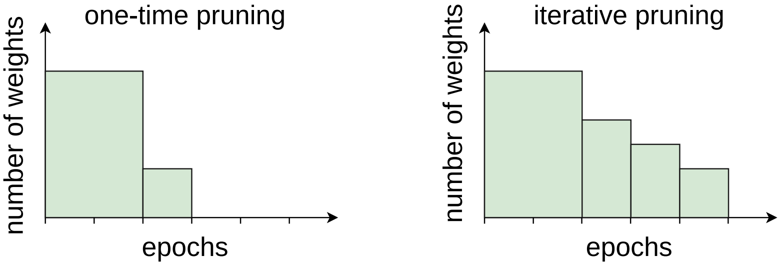


Fig. 5.7 Illustration of one-shot and iterative pruning. To achieve the same sparsity, one-shot pruning is applied once, but the iterative approach applies pruning in small steps, separated by fine-tuning.

In the case of the IGNN, where the layer normalisation is used, the alignment to the multiplication of four is even more important - without it, inference time can increase by even 57% with respect to the baseline model. This performance decline is attributed to the PyTorch CUDA implementation of covariance computation within layer normalisation. PyTorch employs two different methods for variance calculation — vectorised layer normalisation and the Welford algorithm — depending on whether the input tensor is memory-aligned. The non-aligned variant is much more computationally expensive.

To consider the size of the pruning step, minimum accuracy is usually reached around 60% of pruning; therefore, close to this value, pruning steps should become more granular. The final procedure, for p_{def} being the pruning amount in the current iteration epoch and p_{perc} being the current pruning percentage of the layer, can be described by Equation 5.4.

$$p_{def} = \begin{cases} 4, & \text{if } p_{perc} < 50 \\ 2, & otherwise \end{cases} \tag{5.4}$$

The results from pruning each of the models are presented in Figure 5.8. In both cases, iterative pruning maintains accuracy better at higher pruning

[2] Tile refers to part of the matrix processed independently

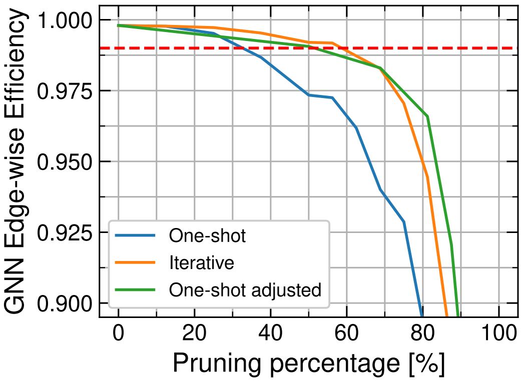


Fig. 5.8 Comparison of model accuracy after applying pruning with different frequencies and number of fine-tuning epochs. "Adjusted" one-shot pruning is fine-tuned for the same number of epochs as it would happen in iterative pruning to reach the same pruning amount. The plots show the accuracy averaged over five experiments. Pruning is applied with the TopK pruning criterion, GMP importance score and L2 norm for importance score aggregation.

percentages, compared to one-shot pruning. It can be explained by the fact that when pruning a small amount followed by fine-tuning, the model can recover lost efficiency. When pruning a significant number of parameters at the same time, some of the relations may be lost. However, iterative pruning takes significantly more time, depending on the number of iterations.

When the cost of iterative pruning is considered, it requires significantly more training epochs, as fine-tuning is performed multiple times compared to one-shot pruning. This makes direct accuracy comparisons with one-shot pruning less fair due to the additional optimisation effort. When the number of fine-tuning epochs is the same for both one-shot and iterative pruning, one-shot pruning tends to preserve accuracy with a higher pruning percentage. It could be related to the fact that in iterative pruning, each new pruning step can overwrite or diminish the improvements gained during previous fine-tuning phases, leading to a gradual loss of recovered information.

On the other hand, iterative pruning offers the advantage of being stopped when accuracy falls below an acceptable threshold, making it more suitable for experimental settings. However, for the final production model, one-shot pruning should be considered with an adjusted number of training epochs.

Structured Pruning Procedure Evaluation

A high efficiency is maintained for the smallest models when using GMP importance score, L2 norm, TopK criterion, and pruning applied iteratively. With this configuration, for IGNN we can prune up to 55% while maintaining over 99% of the GNN edge-wise efficiency.

A comparison of the best structured pruning configuration and unstructured pruning is illustrated in Figure 5.9. Despite the best efforts to optimise it, the model accuracy degrades sooner than the unstructured variant. By pruning subtensors but not individual elements, loss of important information is inevitable. On the other hand, the structured pruning allows resizing the model and gaining performance improvements, which is usually not possible for unstructured pruning, as discussed in the introduction of this subchapter.

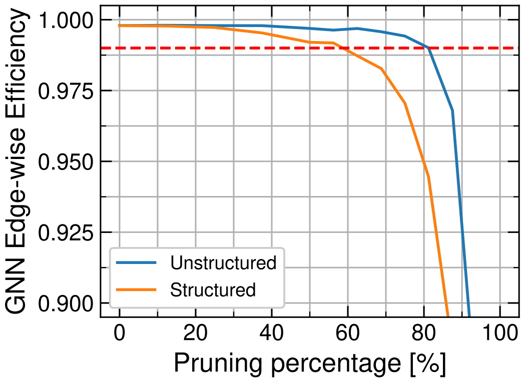


Fig. 5.9 Comparison of the best structured and unstructured pruning methods. The plots show the accuracy averaged over five experiments. Pruning is applied iteratively with 10 epochs of fine-tuning in between iterations, TopK pruning criterion, GMP importance score and L2 norm for importance score aggregation.

Application of Structured Pruning to ATLAS Model

The chosen pruning procedure with the TrackML IGNN can now be applied to ATLAS IGNN; the measured accuracy while pruning is presented in Figure 5.10. The pruning amount per iteration was adjusted to fit the network dimension.

In order to reduce the pruning time, the fine-tuning in between pruning iterations was only done on a restricted dataset of 780/100/100 training/validation/testing graphs. Comparison of fine-tuning with a full and restricted dataset is presented in Figure 5.11. The accuracy of the model is expected to

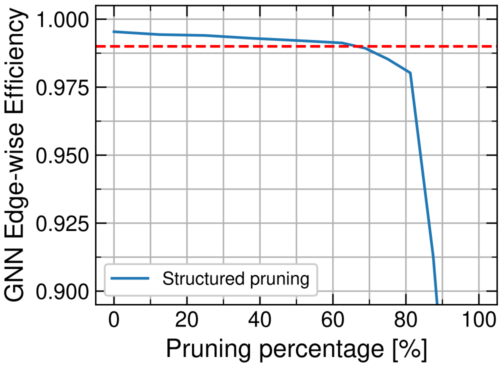


Fig. 5.10 Accuracy of the ATLAS IGNN after applying structured pruning with the best chosen configuration. The fine-tuning was applied to a limited dataset.

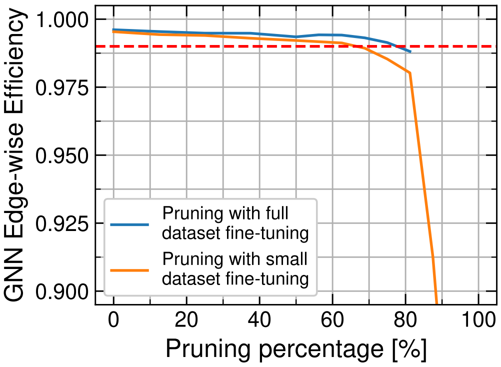


Fig. 5.11 Comparison of model accuracy with fine-tuning on full and small datasets.

be preserved for higher compression; however, the same patterns are expected. With the small dataset, the complete pruning time was reduced approximately 10 times.

Based on the conducted experiments, to maintain the GNN edge-wise efficiency over 99%, the ATLAS IGNN model can be pruned up to 65%. However, the final model can be further fine-tuned to regain lost accuracy. This work does not include this type of study; however, it will be conducted in the future.

5.3 Model Sensitivity

One of the disadvantages of the pruning with the TopK strategy presented in the previous subchapter is that all layers are pruned the same amount per pruning step. However, some layers may require larger or smaller dimensions than others to perform optimally. As a result, they may benefit from being pruned more conservatively, depending on their importance. The process to find the optimal set of hidden layer sizes is similar to model architecture search.

In this work, to find the optimal number of parameters of each layer, also known as *sensitivity*, each of the layers is pruned individually with progressively smaller amounts. Then, the epoch when the accuracy reaches a value below a defined threshold is tracked. The score is an exponential function of that epoch, as a linear scoring function did not provide sufficient performance. Based on those scores, *a sensitivity dictionary* is created, storing sensitivity scores for each layer.

When applying the pruning, the sensitivity score affects the pruning amount in the current pruning step p_{def}, thus k for TopK or t for threshold-based pruning, respectively. For a given layer l, a sensitivity score $s_l \in (0,1]$, the pruning amount p_l is determined by Equation 5.5. The first two cases in the Equation apply to TopK pruning, where the number of layers defines the pruning amount to prune, and the third case applies to TopK pruning, where the pruning amount is defined by the fraction of total parameters to prune and threshold pruning.

$$p_l = \begin{cases} \lfloor p_{def} \cdot s_l \rfloor, & \text{if } |p_{def} \cdot s_l| >= 1 \quad \text{and} \quad p_{def} \in \mathbb{Z} \\ 1, & \text{if } |p_{def} \cdot s_l| < 1 \quad \text{and} \quad p_{def} \in \mathbb{Z} \\ p_{def} \cdot s_l, & \text{if } p_{def} < 1 \end{cases} \tag{5.5}$$

There are other methods for assessing the best model size [80]–[82]. The chosen approach is simple, yet the performance improvements can be observed during the tests with TrackML IGNN. In future work, other methods can be explored to improve the sensitivity search.

Additionally, when resizing the model pruned with sensitivity scores, the memory alignment must be taken into account as well. The pruning amount per iteration, described in 5.2, is adjusted by the sensitivity scores, breaking the aligned sizes. To mitigate this issue, an alignment option was added to the resizing procedure, avoiding the more expensive execution path for layer normalisation.

The new resizing procedure ensures that the number of remaining rows in the weight matrix is rounded up to the nearest multiple of four. The impact on the performance is presented in Table 5.2. Alignment significantly improved the performance on the pruned IGNN inference (reduced by 50%) with a small cost of the model size increase (1%).

Table 5.2 Model size and performance pruned with and without alignment

Pruning option	Model size	Inference time
Pruning without alignment	3.61E+04	31.2 ± 0.1
Pruning with alignment	3.65E+04	15.9 ± 0.3

The results of applying the sensitivity scores to the TrackML IGNN are presented in Figure 5.12. The model accuracy is improved. By recognising how much of which layer of the model can be pruned, the accuracy is maintained above the desired minimal threshold for longer than when treating all layers in the same way.

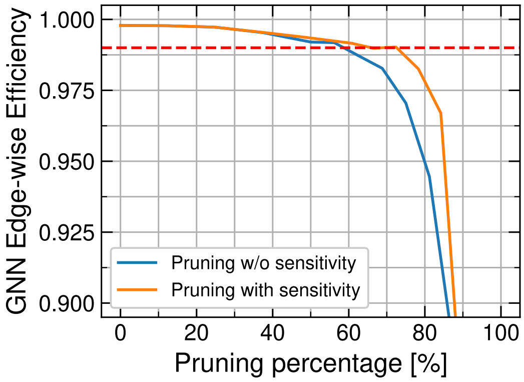


Fig. 5.12 Comparison of structured pruning with and without sensitivity scores. The plots show the accuracy averaged over five experiments. Pruning is applied iteratively with 10 epochs of fine-tuning in between iterations, TopK pruning criterion, GMP importance score, and L2 norm for importance score aggregation.

Applying Sensitivity to ATLAS IGNN Pruning

Before applying the sensitivity-guided structured pruning to ATLAS IGNN, a test was conducted to determine if the TrackML IGNN sensitivity scores are in line with the ATLAS model. Experimental results show that, in the case of TrackML IGNN, reducing the dimensionality of encoder and decoder networks leads to the most significant decline in model accuracy. In contrast, for the ATLAS IGNN, although pruning the encoder networks does affect the performance of the model, a greater impact is observed when reducing the size of the edge network components. In graphs representing high-complexity collision events from the ATLAS dataset, the features of neighbouring nodes and edges are critical to determine whether an edge belongs to a true track. A substantial decrease in performance was observed when restricting the size of edge networks from the first four message-passing iterations, suggesting that the edge neighbourhood of distance four is generally sufficient for reliable edge classification. Similar conclusions were drawn for independent tests, with the IGNN model trained with only four message-passing steps.

The results of pruning with sensitivity scores of ATLAS IGNN are presented in Figure 5.13. In comparison to TrackML IGNN, the benefit of sensitivity-aware pruning is much less significant. By introducing irregular pruning amount over the layers, an additional 5% of parameters were removed.

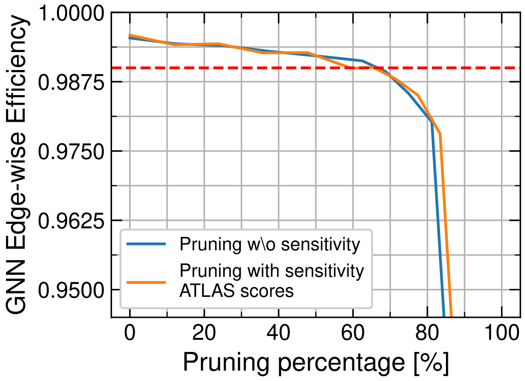


Fig. 5.13 Comparison of structured pruning with and without sensitivity scores. Pruning is applied iteratively with 10 epochs of fine-tuning in between iterations, TopK pruning criterion, GMP importance score, and L2 norm for importance score aggregation. A small dataset was used for the fine-tuning.

The lack of a significant improvement can be explained by multiple restrictions applied to the experiments. The retraining on the ATLAS IGNN was done on a reduced dataset, proven to cause faster accuracy loss (Figure 5.11). Moreover, the impact of the sensitivity scores' aggressiveness should be studied further. Additionally, other architecture search techniques could be used to describe the per-layer sensitivity, which could be more efficient than the chosen simple technique.

5.4 Application of Structured Pruning to Graph Neural Networks

To study the portability of optimisation techniques applied to IGNNs, two other example GNNs were chosen: GCN and GAT, described in Subchapters 2.3.1 and 2.3.1. Different pruning techniques will be applied to both of them to choose the best pruning strategy for the GNNs.

5.4.1 Models

Graph Convolutional Network

An example GCN chosen for the experiments on portability of structured pruning is solving the node classification task on the CORA dataset [83]. It consists of 2708 academic publications, represented as the nodes, each classified in one of the seven classes representing different machine learning topics, and the 5429 citation links between them, represented by edges.

The model prepared to predict classes of the nodes uses three GCN Convolutional layers, implementing the graph convolutional operator [31] from the PyTorch Geometric library [84]. In each step, first, the message passing takes place, where all the features are aggregated, including the features of the node itself. The features are normalised, depending on the size of the neighbourhood (degree), and aggregated by summing the messages. Last, a linear transformation is applied to aggregated features by multiplying them by a learnable weight matrix.

The accuracy of the network prediction is evaluated by the simple ratio of the number of correctly predicted labels to the total number of tests.

Graph Attention Network

A GAT example considered in this work performs node classification on a Protein-Protein Interactions (PPI) dataset [85]. From the input graphs representing different tissues in the human body, where nodes are proteins with 50 different features and edges represent interactions between them, the goal is to label each of the nodes with multiple labels out of 121, corresponding to biological functions.

The prepared model consists of 3 hidden GAT layers, each starting with an MLP encoding the node features into a latent space. For each of the nodes and their neighbourhoods, an attention score is calculated, reflecting how important the neighbouring nodes are for each node. The features are aggregated for each of the heads, and at the end, the outputs from multiple attention heads are concatenated (or averaged for the last layer).

To assess the accuracy of the network, the micro-F1 score [86] is used, which is common for multi-label classification tasks. It aggregates the total true positives (TP), false positives (FP), and false negatives (FN) across all classes and nodes, providing a balanced measure of overall classification quality, described by Equation 5.6.

$$\text{Micro F1} = \frac{2 \cdot \text{TP}}{2 \cdot \text{TP} + \text{FP} + \text{FN}} \tag{5.6}$$

It is particularly used for imbalanced datasets, as it gives equal importance to each prediction rather than each class.

5.4.2 Experimental Setup

The differences in model sizes and inputs are summarised in Table 5.3. The toy models contain only two hidden layers, which are the target of structured pruning, compared to 39 in IGNN, with much more complex dependencies between them.

Table 5.3 Overview of models and datasets used in experiments.

Name	Task	Dataset	Total number of parameters	Number of parameters in hidden layers	Average graph size
IGNN	Edge Classification	TrackML [21]	1.25×10^5	1.05×10^5	1.2×10^4 nodes 5.0×10^4 edges
IGNN	Edge Classification	ATLAS	1.45×10^6	1.38×10^6	3.4×10^5 nodes 9.1×10^5 edges
GCN	Node Classification	CORA [83]	4.72×10^4	4.69×10^4	2.7×10^3 nodes 5.4×10^3 edges
GAT	Node Classification	PPI [85]	3.7×10^6	7.8×10^4	2.4×10^3 nodes 7.0×10^4 edges

5.4.3 Graph Convolutional Network Experiments

This subchapter presents a study on the pruning of the GCN. Each of the convolutional layers has a linear layer, applying a learnable transformation to the aggregated data. The weights of those linear layers will be pruned. The size of the input and output to the GCN will be preserved, analogous to IGNN.

Different pruning configurations were studied for the presented GCN; only the best results will be shown and discussed in the subchapter. The results averaged over repeated experiments are presented in Figure 5.14. The highest achieved accuracy of the GCN pretrained model is 0.815, and the desired minimum accuracy after pruning is 0.75 for this study.

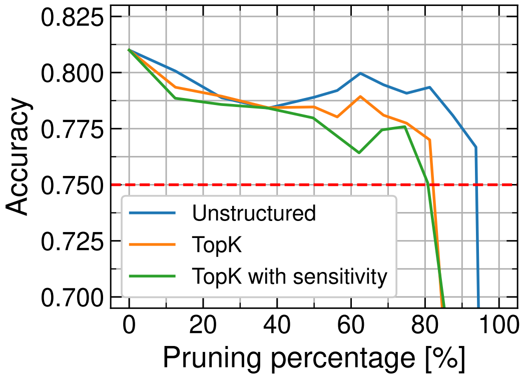


Fig. 5.14 Comparison of model efficiency of different pruning methods applied to GCN: unstructured pruning, structured TopK pruning, and structured TopK pruning with sensitivity. Pruning is applied iteratively with 10 epochs of fine-tuning in between iterations, with GMP importance score and L2 norm for importance score aggregation.

The best results for structured pruning were achieved with the TopK iterative pruning with L2 norm and GMP importance score, similarly to IGNN. The unstructured pruning maintains the desired efficiency when pruned 15% more than the structured approach. Applying the sensitivity to the pruning procedure did not improve the accuracy. The difference in the impact of pruning each of the layers on the overall accuracy is not significant - the second layer can be pruned slightly less.

What is also important to notice here is that the first layer has many more parameters than the second one; each of the 1433 features is a separate input. Pruning the output features of the first layer (weight matrix size 1433×32), each with 1433 parameters, will have a bigger impact on total pruning percentage than pruning the output parameters of the second layer (weight matrix size 32×32), where each weight matrix row contains 32 parameters.

5.4.4 Graph Attention Network

In the Graph Attention Network, each attention layer begins with a transformation of the input using a linear layer. Next, attention coefficients are calculated to determine the influence of each neighbour, and features are updated by aggregating neighbouring information weighted accordingly. The weights of the linear layer transforming the input are pruned in this work. The highest accuracy of the GAT pretrained model is 0.966, but the model is reported to achieve a score of 0.973. The desired minimum accuracy is 0.9.

The results of repeated experiments with different pruning methods of GAT are presented in Figure 5.15. The differences between different configurations were not as significant as those observed for the IGNN.

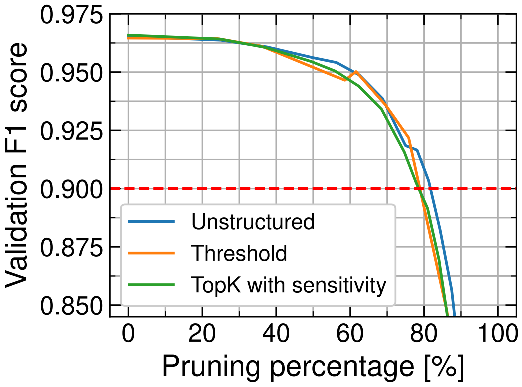


Fig. 5.15 Comparison of model efficiency of different pruning methods applied to GAT: unstructured pruning, structured threshold pruning, and structured TopK pruning with sensitivity. Pruning is applied iteratively with 10 epochs of fine-tuning in between iterations, with GMP importance score and L2 norm for importance score aggregation.

The final pruning configuration can achieve results almost as good as the unstructured pruning approach. The threshold pruning criterion was chosen as the best approach to structured pruning (without sensitivity) because it performed better than TopK (Figure 5.16). To understand this difference in comparison to IGNN, the pruning percentage of each of the layers needs to be taken into account separately. During the threshold pruning, the first layer is pruned much less than the second layer. It may indicate that the first layer has a significantly greater impact on the overall accuracy of the model. Since the second hidden layer has more parameters than the first one, it has a bigger impact on the pruning percentage metric as shown in Figure 5.17.

However, after applying the sensitivity scores, accounting for the impact of each layer on the total efficiency of the model, the TopK pruning can achieve results as good as the threshold pruning approach (Figure 5.15).

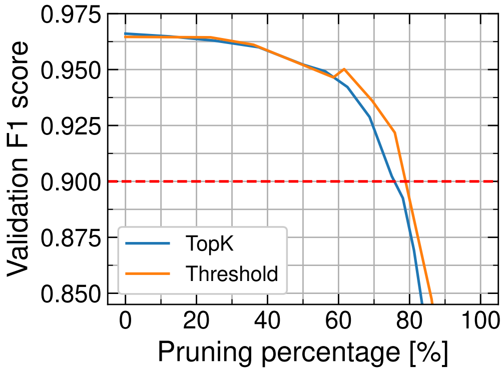


Fig. 5.16 Comparison of efficiency after applying different pruning criteria: TopK and threshold. Pruning is applied iteratively with 10 epochs of fine-tuning in between iterations, with GMP importance score and L2 norm for importance score aggregation.

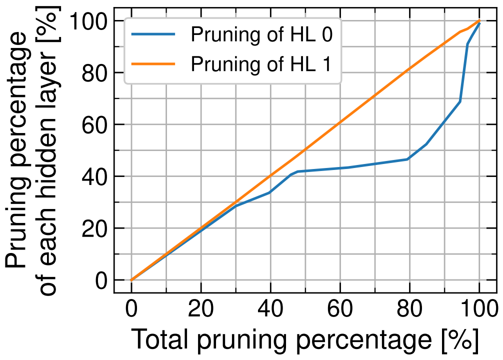


Fig. 5.17 Pruning percentage of each of the hidden layers for the measured total pruning percentage for the threshold pruning.

5.5 Impact of Pruning on IGNN Inference

This subchapter evaluates the computing performance of IGNN model inference for TrackML and ATLAS datasets, with emphasis on the impact of applied structured pruning. For each example, three models will be considered: baseline (not resized version), pruned and resized without sensitivity scores (regular shape), and pruned and resized with sensitivity scores (irregular shape) at the point of the minimum allowed accuracy. The models with unstructured pruning, which are competitive in terms of performance, cannot be easily resized with the proposed method; therefore, their performance will be the same as the baseline version for GPU inference.

After applying the pruning, the sizes of the models were reduced by 50%/65% (TrackML IGNN/ATLAS IGNN), and their sizes are presented in Table 5.4. By applying the sensitivity while pruning, the model size was reduced by another 40%/15% (TrackML IGNN/ATLAS IGNN). The impact of the model size reduction and the irregularity patterns in the removed parameters will be studied in this subchapter.

Table 5.4 Overview of the number of parameters in models before and after pruning.

Dataset	Applied Pruning	Number of Parameters
TrackML	baseline	1.25×10^5
	pruned	6.13×10^4
	pruned with sensitivity	3.65×10^4
ATLAS	baseline	1.45×10^6
	pruned	5.28×10^5
	pruned with sensitivity	4.49×10^5

Inference Time

The measured inference time of baseline, pruned and pruned with sensitivity models is presented in Table 5.5. For each of the considered models and the datasets, pruning improved the inference time by 18%. By including the sensitivity guidelines, a slight improvement can be observed; however, its impact is not significant.

Table 5.5 Measured inference time per input graph for an average and the biggest graph in the input datasets. The measurements were done on an NVIDIA RTX A5000 24GB and are averaged over repeated experiments.

Model	Input graph size	Baseline model [ms]	Model pruned w/o sensitivity [ms]	Model pruned with sensitivity [ms]
TrackML	avg	19.7 ± 0.3	16.7 ± 0.6	15.9 ± 0.3
	max	33.4 ± 0.1	28.2 ± 0.1	27.1 ± 0.1
ATLAS	avg	721 ± 7	588 ± 3	566 ± 3
	max	2373 ± 16	1932 ± 13	1876 ± 16

The improvement in the inference time of IGNN is correlated with improved computing performance of the linear layer evaluation, which performs the matrix multiplication. Reduction in size of matrices leads to multiplication of matrices with a smaller size, executed faster. The detailed performance of this operation is considered in the following subchapter.

Computing Resource Utilization

The next considered metric is the number of FLOP per inference step. This measurement helps to estimate the computational demand placed on the processor and directly indicates how intensively a model utilises the resources. The results of the conducted measurements are included in Table 5.6.

Table 5.6 Measured number of executed FLOP per input graph of the inference step for an average and the biggest graph in the input datasets. The measurements were done on an NVIDIA RTX A5000 24GB and are averaged over repeated experiments.

Model	Input graph size	Baseline model	Model pruned w/o sensitivity	Model pruned with sensitivity
TrackML [GFLOP]	avg	15.7	12.6	11.9
	max	27.3	21.7	20.6
ATLAS [TFLOP]	avg	2.88	1.21	1.2
	max	6.35	2.64	2.61

For the TrackML IGNN, the reduction in FLOPs for the pruned model is approximately only 20%, and 25% for the model pruned with sensitivity. These gains are notably smaller than those observed for the ATLAS IGNN, where FLOP performance improved by 60%. The number of FMA operations, particularly those associated with SGEMM, demonstrated a smaller change than in the ATLAS IGNN.

Due to the small dimensions of the TrackML IGNN ($D = 32$), the dimensions of the pruned hidden layer are rounded to fit the kernel tile size by the underlying PyTorch library. Due to the small hidden-layer dimension of the TrackML IGNN ($D = 32$), the dimensions of the pruned hidden layers are rounded by the underlying PyTorch libraries to match the available kernel tile sizes. In the experiments presented in Figure 5.18, the used kernel tile sizes during IGNN inference are 128×64 and 128×32. All SGEMM operations in the TrackML IGNN employed the 128×32 tile size, whereas in the ATLAS IGNN, the initial model utilised the 128×64 tile. The 128×64 tile was observed to be selected only when the hidden-layer dimension exceeded 64, which does not occur in the

TrackML IGNN. With pruning applied to ATLAS IGNN, decreasing the size of the hidden layer to below 64, the more efficient kernel with 128 × 32 tile is used, causing a substantial performance gain.

The performance scaling of kernels for each tile size does not directly correspond to the associated matrix size; the performance gain is smaller.

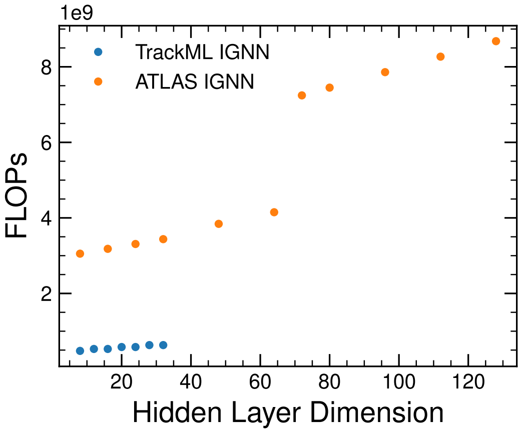


Fig. 5.18 Number of FLOP during evaluation of an MLP with an architecture of IGNN edge network reflecting sizes of ATLAS IGNN and TrackML IGNN. The input to MLP reflects the IGNN input of a tensor with 5×10^5 edges. For MLPs with dimensions larger than 64, the SGEMM kernel with a tile size of 128×64 is used, increasing FLOPs.

Memory Footprint

To measure how the memory footprint of the model inference is impacted by the structured pruning, peak memory consumption during graph inference was analysed. Similarly to the previous studies, the memory consumption was measured on average and for the largest input graphs. The results are summarised in Table 5.7.

The peak memory consumption improved only by 5%/10% (TrackML IGNN/ATLAS IGNN) after applying pruning. There is no observable difference for models pruned with and without sensitivity - the difference in model size is too low to make an impact after different layers of tooling.

The memory footprint of the IGNN is driven by the size of the input graphs, rather than the model itself. The model with 1.45×10^6 parameters translates to 5.8 MB [3] while an average input graph with 3.4×10^5 nodes with 12 features

[3] With single-precision floating-point representation

Table 5.7 Measured peak memory consumption per input graph of the inference step for an average and the biggest graph in the input datasets. The measurements were done on an NVIDIA RTX A5000 24GB and are averaged over repeated experiments.

Model	Input graph size	Baseline model [GB]	Model pruned w/o sensitivity [GB]	Model pruned with sensitivity [GB]
TrackML	avg	0.117	0.110	0.110
	max	0.190	0.179	0.179
ATLAS	avg	5.17	4.69	4.69
	max	15.8	14.2	14.2

each and 9.14×10^5 edges with 6 features each to 38 MB. Moreover, there are multiple temporary structures, serving as inputs to each of the MLPs creating the IGNN; their sizes are not reduced by pruning.

5.6 GPU Utilisation of IGNN

After a detailed analysis of the computing performance of the IGNN before and after pruning, similar studies were conducted on a selection of hardware. Based on the inference time measurements, summarised in Table 5.8, the shortest inference time was achieved for NVIDIA A100.

Table 5.8 Measured inference time of the IGNN on an average-sized graph on selected hardware. The presented results are averaged over repeated experiments.

Inference Time [ms]	Tesla T4	RTX A5000	RTX 5000 Ada	A100
Baseline model	2769 ± 114	721 ± 7	493 ± 1	458 ± 1
Pruned model	2122 ± 82	588 ± 3	382 ± 1	378 ± 1
Model pruned with sensitivity	2058 ± 93	566 ± 3	375 ± 1	363 ± 1

To understand those results, selected performance metrics will be compared across the cards to determine which offers the best performance and to identify the characteristics that contribute to its advantage. The tests were conducted on an input graph with an average size from the ATLAS dataset.

Memory Throughput

The results of measured memory throughput during IGNN inference are presented in Table 5.9. For all the cards, the memory bandwidth is not saturated, achieving up to 50% of available bandwidth. The higher the theoretical performance is reported, the higher the observed throughput is. NVIDIA A100 achieved the highest memory throughput, which substantially affects the inference time of the network - the IGNN inference operates on large input graphs, and operations on them can cause substantial latencies.

Table 5.9 Measured memory throughput of IGNN inference on different hardware for an average graph.

Memory Throughput [GB/s]	Tesla T4	RTX A5000	RTX 5000 Ada	A100
Theoretical Performance	320	768	576	1555
Baseline model	165	390	325	670
Pruned model	160	349	291	628
Model pruned with sensitivity	161	352	290	628

Based on the detailed performance study, the memory throughput is limited for the most time-consuming operations by the computational throughput - the SGEMM operations saturate the SMs to 95% while the memory throughput stays at a level of 50%.

Computing Resource Utilization

The next considered metric is the number of operations that a card can perform per second. The measurements are included in Table 5.10. Even though NVIDIA A100 does not have the theoretical highest performance, it achieves the best performance out of the measured cards.

For each of the considered cards, the compute resource utilisation decreased by half after pruning - only one input graph is processed simultaneously, with a reduced number of weights, and fewer FLOPs are performed.

The results presented in Table 5.10 can be misleading due to the way this metric is calculated. The number of recorded FLOPs is divided by the elapsed

Table 5.10 Measured computing performance of IGNN inference on different hardware.

FP32 Performance [TFLOP/s]	Tesla T4	RTX A5000	RTX 5000 Ada	A100
Theoretical Performance	8.141	27.77	65.28	19.49
Baseline Model	1.12	3.07	4.56	5.12
Pruned Model	0.60	1.51	2.34	2.58
Model Pruned with Sensitivity	0.61	1.56	2.36	2.65

GPU time, which includes synchronisation stalls, kernel launch overhead or memory access latency. The number of operations is the same across cards; however, the NVIDIA A100 recorded shorter GPU time due to the substantially higher memory bandwidth. Even though NVIDIA RTX 5000 Ada has a considerably higher number of CUDA cores, due to lower memory bandwidth, they may be underutilised, unlike NVIDIA A100.

Power Consumption

The last measurement considered in this subchapter is the energy consumed during IGNN inference. Table 5.11 summarises the recorded measurements. No significant differences in the instantaneous power consumption were observed for full and reduced models; therefore, only the performance of the baseline model was presented.

Table 5.11 Measured average power and energy of an inference step consumption of IGNN inference on different hardware.

	Tesla T4	RTX A5000	RTX 5000 Ada	A100
TDP [W]	70	230	250	250
Instantaneous Power Consumption [W]	66	225	242	246
Baseline [J]	182	162	119	112
Pruned model [J]	140	129	93.6	86.2
Model pruned with sensitivity [J]	135	124	91.9	82.4

For all the cards, the power limit is reached. After the pruning application, the energy is reduced by 20% for all presented cards.

Despite the smallest power consumption, when considering the performance of the complete inference step, NVIDIA Tesla T4 consumes the most energy out of the considered cards. NVIDIA A100 reports the smallest energy consumption out of reported GPUs; the maximum power consumption is similar to the other cards, yet, as reported in this chapter, its inference time is faster due to high memory throughput benefits. Low energy consumption directly translates to long-term savings, even though the initial cost of this GPU is higher than other considered models.

5.7 Discussion

Structured pruning, removing the hidden layer neurons, proves to be a promising model compression solution for IGNNs as well as other complex GNNs. Based on the conducted experiments, compression of 60% on average can be achieved without compromising the accuracy of the model. Combined with resizing, structured pruning gives promising results in 20% faster inference and 20% of energy savings.

The structured pruning for the GNNs can be configured in many different ways regarding various aspects, with the solutions proposed by the community over the years. This work proposes the best set of pruning parameters based on extensive studies on the importance scores and their aggregation methods, pruning criteria and pruning frequency. The configuration of GMP importance score, aggregated with L2 norm, TopK pruning criterion, and iterative pruning, yields the best results, maintaining model accuracy in the model with the highest compression.

Complicated, multi-layer models can be further pruned while preserving the accuracy by adjusting the pruning amount based on the sensitivity score. The chosen shape of structures to prune, combined with resizing of the model, significantly reduces the model size and the inference computing resource consumption. Additionally, applying sensitivity while pruning reduces model size by an additional 20%, but for even stricter accuracy requirements, the gain in model compression can be even higher. After application of sensitivity-guided structured pruning, no significant performance gains were observed. The proposed structured pruning can be further studied and improved by considering a different sensitivity score, for example, inspired by the architecture search tasks [80]–[82] or analysing the impact of reemerging weight subtensors [87], [88].

The proposed pruning strategy, as well as the sensitivity-guided pruning, were evaluated on benchmark GAT and GCN models with promising results. However, the unique architecture of IGNN, comprising of multiple but rather small MLPs exploits the sensitivity scores more effectively than the benchmark models, which contain fewer layers.

Based on the performance studies of the IGNN on selected hardware, NVIDIA A100 shows the best inference performance, offering substantial time and energy savings—the performance benefits from extremely efficient memory type HBM2e, delivering outstanding memory bandwidth. Even though the cost of the card is the highest among those examined, its long-term energy efficiency is a cost-effective investment over time. To reach even higher memory throughput, graphics cards from Hopper and Blackwell generations can be considered, with the memory bandwidth reaching over 4 TB/s.

Chapter 6

Conclusions

The upcoming upgrade of the LHC will enable particle collisions with up to 200 interactions of single bunch crossing, facilitating more extensive studies in particle physics. The LHC experiments, such as ATLAS, must adapt to the increased rate and complexity of these collisions. To fully leverage the physics potential of the enhanced accelerator, the ATLAS detector will undergo an upgrade of the tracking detector, incorporating a substantially larger number of modules with improved precision and expanded coverage around the IP.

Track reconstruction is a challenging task, involving resource-consuming, advanced algorithms capable of distinguishing particles traversing the tracking detectors. Its accuracy is crucial for successful studies of the interactions of particles. Moreover, in the event selection system environment, additional challenges arise due to the limited computing resources; all particle collisions must be processed to avoid data loss. Experiments such as ATLAS must consider alternatives to traditional CPU-bound track reconstruction algorithms to improve the efficiency of the event selection system. Recent advances in GPU acceleration, coupled with the growth of machine learning capabilities, have emerged as a promising solution to address these computational challenges.

This work presents two track reconstruction solutions involving graphics card acceleration, one enhancing the traditional track reconstruction algorithm, and one proposing a novel approach with the IGNN. Both of the solutions are effectively optimised to reduce computational resource usage while maintaining satisfactory accuracy.

Fast Track Reconstruction Algorithm

The first discussed solution, fast track reconstruction, has been successfully used in the ATLAS trigger in Run 2 and Run 3. This algorithm can be partially accelerated with a GPU, as one of the proposed EF track reconstruction algorithms. Specifically, the track seeding component, identified as a strong candidate for parallel processing, is offloaded to the graphics card for optimisation.

The optimisation of the algorithm required adjustments to both track seed selection and GPU resource configuration. By fine-tuning the selection cuts to fit the expected data from the new ATLAS tracking detector, the average accuracy was improved from 65% to 85%. By rejecting the incorrect seed candidates, the efficiency in the Barrel region of the detector improved from 0% to almost 90%. The number of fake track seeds was reduced by 97%, the full track reconstruction time improved from 26.7 s to 1.62 s on average, matching the performance of the offline track reconstruction algorithm.

The performance improved due to the novel triplet confirmation kernel proposed in this work. The supposedly redundant duplicate seeds, belonging to the same true track, were used to confirm whether the seed can belong to a true track. The novel adjustments to selection cuts were applied in track seeding algorithms of other ATLAS track reconstruction algorithms. The triplet confirmation kernel can be adapted to other frameworks as well, helping with the present problem of a high number of fake track seeds, significantly deteriorating the computing performance of track finding.

Furthermore, the GPU usage of the algorithm was analysed in detail, identifying bottlenecks of the kernels. The most crucial parts were addressed, including floating-point precision and scheduling configuration. However, further optimisations are possible, including modifications to the code architecture to address issues arising from limited shared memory.

The conducted studies show that shared memory as well as the number of CUDA cores limit the performance of this algorithm. The graphics cards with the highest number of CUDA cores (NVIDIA RTX 5000 Ada) reported the best performance of the algorithm, as well as the lowest energy consumption.

The current fast track reconstruction track seeding algorithm can benefit from refactoring of the last two kernels to fully leverage the available parallelism. Moreover, memory management can be optimised by replacing the static allocations by frameworks allowing modern memory management. The fast tracking algorithm can be further improved by implementing the Graph Based Track Seeding algorithm for track seeding, proven to substantially reduce the computing cost of track building by providing preselected, high-quality track seeds.

Interaction Graph Neural Network

The second considered track reconstruction algorithm, involving IGNN, has proven to achieve high track reconstruction accuracy; however, it is resource-consuming. The performance bottlenecks arise from a high memory footprint and suboptimal inference time. Both of those challenges are addressed in this work.

The architecture of the IGNN consists of multiple MLP networks, processing the propagated features of graph nodes and edges. The temporary structures, which serve as inputs for those networks, are identified as the most memory-consuming parts of the model, yet they are only temporary. A mechanism called substepping was proposed, processing those temporary structures iteratively, split into predefined sizes, reducing the instantaneous memory footprint to a chosen size. This way, the model can be deployed on memory-limited hardware, or multiple graphs can be processed in parallel.

The impact of the substepping on the computing performance was analysed - the time penalty for iterative processing is noticeable only for steps smaller than 500 000. The number of iterations increases exponentially with decreasing step size, outweighing the performance gains from executing functions on all sizes of tensors. Additionally, the additional substepping operations cause overhead. The optimal performance was achieved for the substep size of 500 000, matching the inference time of the non-substepping model while reducing the memory footprint by 30%.

To address the suboptimal inference time, a structured pruning model compression technique was applied. It has proven to be compatible with modern hardware, reducing the model complexity by introducing structured sparsity in the underlying tensors. This work explores different structured pruning configurations applied to IGNN as well as GNN benchmark models to reveal which pruning achieves the best results.

A search for the GNN pruning configuration that the importance scores including both values of weights and their gradients sufficiently reflect the importance of the network neurons, preventing premature accuracy loss. The tests with scores with Optimal Brain Damage achieved similar performance; however, its computational cost is much higher compared to the gradient based method. The L2 norm has been demonstrated to be the optimal score aggregation method, a result widely reported in the machine learning community. Based on conducted experiments, the TopK pruning criterion proved to perform better than threshold-based selection, and the iterative pruning stands as an optimal choice for the pruning search. After establishing the maximum model compression for desired accuracy, the model should be one-shot pruned, followed by fine-tuning epochs.

The found pruning configuration was applied to the ATLAS IGNN, allowing for 62.5% compression while preserving the accuracy of 99%. This accuracy can be improved even more by additional fine-tuning.

Structured pruning can be further improved for models with a large number of layers by guiding the pruning amount by the sensitivity scores. Each of the layers was analysed to determine how its compression affects accuracy. The pruning sensitivity guidelines allow for higher model compression, reducing the model size by 20% compared to structured pruning without sensitivity guidelines. Additionally, minor performance improvements were observed for the model pruned with the sensitivity scores.

In the future, the sensitivity scores can be further fine-tuned by using neural architecture search algorithms or studying the impact of the aggressiveness of the scores in more detail. Moreover, the pruning strategy can be improved by including pruning of input and output layers of IGNN. This way, the temporary structures, serving as MLP inputs, can have reduced sizes and offer substantially lower memory footprint.

The conducted measurements of the IGNN performance show that its main limitation corresponds to the memory throughput possible on the GPU. NVIDIA A100 demonstrated the shortest inference time and the smallest energy consumption due to its exceptional memory bandwidth.

To further improve the performance, multiple compression techniques, including model compilation or mixed precision, can be applied to IGNN, which are not discussed in this work.

Comparison of the Algorithms

The presented algorithms achieve satisfactory accuracy, making them suitable candidates for the algorithm choice for the future ATLAS trigger. Their computing performance was thoroughly optimised, making them competitive with other proposed solutions.

When comparing the complexity of the algorithms, the fast track reconstruction seeding requires expertise and a deep understanding of the particle track reconstruction, its geometry and the impact of the data taking conditions (rule-based algorithm). In contrast, the IGNN seems to be a much simpler algorithm, requiring the user to have a sufficient understanding of track reconstruction to select the appropriate input features. Rather than explicitly implementing selection criteria, the network autonomously learns them from the data, thereby reducing the user's workload (learning-based approach). However, because these selections are deeply embedded within the model, diagnosing potential performance issues and verifying correctness is more challenging than in the case of the FTF.

The direct comparison of the performance of those algorithms is not obvious. While the FTF is already integrated in the ATLAS trigger, the studies of the IGNN were performed in a standalone environment of the acorn [69] framework. To assess its efficiency as the track reconstruction algorithm, the IGNN and its pre- and post-processing steps need to be integrated into the ATLAS trigger framework and evaluated with dedicated tools.

Table 6.1 summarises improvements to the GPU algorithms proposed in this work. Both of the algorithms require additional pre- and post-processing on the CPU, including the computationally expensive CKF algorithm. The performance of the FTF Track Seeding algorithm on GPU achieves much better performance, consuming less time, memory and energy. The IGNN memory consumption can be further reduced by applying more restrictive substepping.

The algorithms exhibit different bottlenecks, requiring different optimisation approaches. The limiting factors in FTF are the parallel processing power related to the high number of processed data, related to the number of CUDA cores and the size of the shared memory. The latter can be addressed by algorithm redesign. In contrast, the performance of IGNN is limited by the memory bandwidth correlated to the substantial size of the model input graphs and

Table 6.1 Comparison of FTF GPU Track Seeding and IGNN algorithms. The included numbers present performance metrics recorded on the best identified GPU for each algorithm, for an average input size.

Metric	FTF Track Seeding	IGNN
GPU	NVIDIA RTX 5000 Ada	NVIDIA A100
Execution time per event [ms]	94	363
Execution time improvement	75%	20%
Number of FLOPs [GFLOP]	58	2610
Number of FLOPs improvement	65%	60%
Memory Bandwidth [GB/s]	0.51	628
Memory Footprint [GB]	2.5	5.26
Energy per event [J]	21.1	82.4

necessary SGEMM operations. The algorithms need hardware cards focusing on the number of CUDA cores or the memory bandwidth; neither of the analysed cards satisfied both requirements. However, the newest NVIDIA architectures, Hopper and Blackwell, include cards with promising properties.

To further evaluate the proposed solutions, the performance of the algorithms should be evaluated when processing multiple inputs simultaneously. Multithreading and multiprocessing are essential features of the ATLAS trigger. Moreover, this way, the computing resources can be fully saturated.

Comparison of Hardware

In the presented studies, four graphics cards were compared, with different architectures, and focusing on various aspects of performance.

The exceptionally high parallel processing power of NVIDIA RTX 5000 Ada driven by the number of available CUDA cores facilitating massive parallelism of floating-point operations, has proven to be the best accelerator for the parallel track seeding. During the conducted experiments, it was able to reach the highest memory and compute throughput. Even though the available memory bandwidth is higher for NVIDIA A100, it could not be fully leveraged due to the lower computing capabilities.

The high memory bandwidth of the NVIDIA A100 is particularly beneficial for algorithms involving GNNs, resulting in a notable improvement in inference. Out of all the considered accelerators, it was able to reach the highest compute throughput by avoiding memory access stalls, even if its theoretical performance was not the best out of the selected cards. Due to the discussed advantages, it achieved the lowest energy consumption of event processing.

Acronyms

ACTS A Common Tracking Software.

CKF Combinatorial Kalman Filter (Chapter 3.1.1).

CPU Central Processing Unit.

EF Event Filter (Chapter 2.1.3).

FLOP Floating Point OPeration (Chapter 2.4.2).

FMA Fused Multiply-Add (Chapter 2.4.2).

FTF Fast Track Finder algorithm (Chapter 3.1).

GAT Graph Attention Network (Chapter 2.3.1).

GBTS Graph-Based Track Seeding (Chapter 3.3).

GCN Graph Convolutional Network (Chapter 2.3.1).

GMP Gradient Magnitude Pruning (Chapter 5.2).

GNN Graph Neural Network (Chapter 2.3.1).

GPU Graphics Processing Unit (Chapter 2.4).

HL-LHC High Luminosity Large Hadron Collider (Chapter 2.1.1).

HLT High Level Trigger (Chapter 2.1.3).

ID Inner Detector (Chapter 2.1.2).

IGNN Interaction Graph Neural Network (Chapter 2.3.1).

IP Interaction Point (Chapter 2.1.1).

ITk Inner Tracker (Chapter 2.1.2).

LHC Large Hadron Collider (Chapter 2.1.1).

MLP Multi-Layer Perceptron (Chapter 2.3).

NN Neural Network (Chapter 2.3).

OBD Optimal Brain Damage (Chapter 5.2).

SGEMM Single precision GEneral Matrix Multiply.

SM Streaming Multi-processors (Chapter 2.4).

TDP Thermal Design Power (Chapter 2.4.1).

WMP Weight Magnitude Pruning (Chapter 5.2).

References

[1] ATLAS Collaboration, “The ATLAS experiment at the CERN Large Hadron Collider,” *Journal of Instrumentation*, vol. 3, S08003, 2008, Also published by CERN Geneva in 2010. DOI: 10.1088/1748-0221/3/08/S08003 (cit. on pp. 2, 7).

[2] ATLAS Collaboration, “The ATLAS experiment at the CERN Large Hadron Collider: a description of the detector configuration for Run 3,” *Journal of Instrumentation*, vol. 19, P05063, 2024. DOI: 10.1088/1748-0221/19/05/P05063 (cit. on pp. 2, 7, 9).

[3] L. Evans and P. Bryant, “LHC Machine,” *Journal of Instrumentation*, vol. 3, S08001, 2008. DOI: 10.1088/1748-0221/3/08/S08001 (cit. on pp. 2, 7).

[4] “High-luminosity large hadron collider (hl-lhc): Technical design report,” CERN, Tech. Rep., 2020. DOI: 10.23731/CYRM-2020-0010 (cit. on pp. 2, 8).

[5] ATLAS Collaboration, *Studies for the development of the Inner Detector trigger algorithms at ATLAS*, ATL-DAQ-PUB-2013-002, 2013. [Online]. Available: https://cds.cern.ch/record/1602918 (cit. on pp. 2, 37).

[6] CMS Collaboration, “The CMS experiment at the CERN LHC,” *Journal of Instrumentation*, vol. 3, S08004, 2008. DOI: 10.1088/1748-0221/3/08/S08004 (cit. on p. 7).

[7] ATLAS Collaboration, “Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC,” *Phys. Lett. B*, vol. 716, p. 1, 2012. DOI: 10.1016/j.physletb.2012.08.020 (cit. on p. 7).

[8] ATLAS Collaboration, “ATLAS Inner Tracker Pixel Detector: Technical Design Report,” ATLAS-TDR-030; CERN-LHCC-2017-021, 2017. [Online]. Available: https://cds.cern.ch/record/2285585 (cit. on pp. 10, 12, 39).

[9] ATLAS Collaboration, “ATLAS Inner Tracker Strip Detector: Technical Design Report,” ATLAS-TDR-025; CERN-LHCC-2017-005, 2017. [Online]. Available: https://cds.cern.ch/record/2257755 (cit. on p. 10).

[10] ATLAS Collaboration, “ATLAS Inner Tracker (ITk) Schematics,” General Photo, 2024. [Online]. Available: https://cds.cern.ch/record/2908925 (cit. on p. 10).

[11] K. Hanagaki, J. Tanaka, M. Tomoto, and Y. Yamazaki, “Particle identification,” in *Experimental Techniques in Modern High-Energy Physics: A Beginner‘s Guide*. Tokyo: Springer Japan, 2022, pp. 69–114. DOI: 10.1007/978-4-431-56931-2_6 (cit. on p. 10).

[12] ATLAS Collaboration, “Performance of the ATLAS track reconstruction algorithms in dense environments in LHC Run 2,” *Eur. Phys. J. C*, vol. 77, p. 673, 2017. DOI: 10.1140/epjc/s10052-017-5225-7. arXiv: 1704.07983 `[hep-ex]` (cit. on p. 11).

[13] ATLAS Collaboration, “The Expected Performance of the ATLAS Inner Detector,” CERN, Geneva, Tech. Rep., 2008. [Online]. Available: https://cds.cern.ch/record/1118445 (cit. on p. 12).

[14] ATLAS Collaboration, *Expected tracking and related performance with the updated ATLAS Inner Tracker layout at the High-Luminosity LHC*, ATL-PHYS-PUB-2021-024, 2021. [Online]. Available: https://cds.cern.ch/record/2776651 (cit. on pp. 12, 15, 54).

[15] ATLAS Collaboration, “The ATLAS trigger system for LHC Run 3 and trigger performance in 2022,” *Journal of Instrumentation*, vol. 19, P06029, 2024. DOI: 10.1088/1748-0221/19/06/P06029. arXiv: 2401.06630 `[hep-ex]` (cit. on pp. 13, 14).

[16] M. Michelotto, M. Alef, A. Iribarren, *et al.*, “A comparison of HEP code with SPEC benchmarks on multi-core worker nodes,” *J. Phys.: Conf. Ser.*, vol. 219, p. 052 009, 2010. DOI: 10.1088/1742-6596/219/5/052009 (cit. on p. 13).

[17] ATLAS Collaboration, *Athena*, version 21.0.127, May 2021. DOI: 10.5281/zenodo.4772550 (cit. on p. 14).

[18] A. Mello, S. Armstrong, and S. Brandt, “Region-of-Interest Selection for ATLAS High Level Trigger and Offline Software Environments,” Mar. 2003 (cit. on p. 14).

[19] ATLAS Collaboration, *Computing and Software - Public Results*, https://atlas.cern/discover/computing-and-software/public-results, Accessed: 2025-06-17, 2025 (cit. on p. 15).

[20] ATLAS Collaboration, *Event Displays from Upgrade Physics Simulated Data*. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/UpgradeEventDisplays (cit. on p. 16).

[21] M. Kiehn, S. Amrouche, P. Calafiura, *et al.*, *TrackML Particle Tracking Challenge*, https://kaggle.com/competitions/trackml-particle-identification, Kaggle, 2018 (cit. on pp. 16, 112).

[22] G. R. Lynch and O. I. Dahl, “Approximations to multiple coulomb scattering,” *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, vol. 58, no. 1, pp. 6–10, 1991, ISSN: 0168-583X. DOI: https://doi.org/10.1016/0168-583X(91)95671-Y (cit. on p. 17).

[23] ATLAS Collaboration, "Software and computing for Run 3 of the ATLAS experiment at the LHC," *The European Physical Journal C*, vol. 85, no. 3, Mar. 2025, ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-024-13701-w (cit. on p. 19).

[24] A. Salzburger, P. Gessinger, F. Klimpel, *et al.*, *acts-project/acts: v41.1.0*, version v41.1.0, May 2025. DOI: 10.5281/zenodo.15533345 (cit. on pp. 20, 77).

[25] ATLAS Collaboration, "Software Performance of the ATLAS Track Reconstruction for LHC Run 3," *Computing and Software for Big Science*, vol. 8, no. 1, Apr. 2024, ISSN: 2510-2044. DOI: 10.1007/s41781-023-00111-y (cit. on p. 22).

[26] V. Boisvert, S. Majewski, J. Adelman, F. Pastore, and B. J. Rosser, "ATLAS TDAQ Phase-II Upgrade: EF Tracking User Requirements Document: EF Tracking circulation," CERN, Geneva, Tech. Rep., 2023, on behalf of the EF Tracking community. [Online]. Available: https://cds.cern.ch/record/2870344 (cit. on pp. 23, 95).

[27] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Review*, vol. 65, pp. 386 –408, Nov. 1958. DOI: 10.1037/h0042519 (cit. on pp. 24, 26).

[28] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *CoRR*, vol. abs/1810.00826, 2019. [Online]. Available: https://arxiv.org/abs/1810.00826 (cit. on p. 29).

[29] C. K. Joshi, T. Laurent, and X. Bresson, "Graph Neural Networks for the Travelling Salesman Problem," INFORMS Annual Meeting 2019, 2019 (cit. on p. 29).

[30] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *CoRR*, vol. abs/1710.10903, 2018. [Online]. Available: https://arxiv.org/abs/1710.10903 (cit. on p. 29).

[31] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2017. [Online]. Available: https://arxiv.org/abs/1609.02907 (cit. on pp. 30, 111).

[32] T. N. Kipf and M. Welling, "Variational Graph Auto-Encoders," *CoRR*, vol. abs/1611.07308, 2016. [Online]. Available: https://arxiv.org/abs/1611.07308 (cit. on p. 30).

[33] K. Jha, S. Saha, and H. Singh, "Prediction of protein–protein interaction using graph neural networks," *Scientific Reports*, vol. 12, p. 8360, May 2022. DOI: 10.1038/s41598-022-12201-9 (cit. on p. 30).

[34] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *27th International Joint Conference on Artificial Intelligence*, ser. IJCAI, 2018. [Online]. Available: https://www.ijcai.org/proceedings/2018/0505.pdf (cit. on p. 30).

[35] Y. Wang, Z. Han, Y. Du, J. Li, and X. He, "BS-GAT: A network intrusion detection system based on graph neural network for edge computing," *Cybersecurity*, vol. 8, Apr. 2025. DOI: 10.1186/s42400-024-00296-8 (cit. on p. 30).

[36] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph Convolutional Neural Networks for Web-Scale Recommender Systems," in *24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18, ACM, Jul. 2018, 974–983. DOI: 10.1145/3219819.3219890 (cit. on p. 30).

[37] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for Quantum chemistry," in *34th International Conference on Machine Learning - Volume 70*, ser. ICML'17, Sydney, NSW, Australia: JMLR.org, 2017, 1263–1272 (cit. on p. 30).

[38] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," *CoRR*, vol. abs/1612.00222, 2016. [Online]. Available: http://arxiv.org/abs/1612.00222 (cit. on p. 30).

[39] K. Huang, C. Xiao, L. M. Glass, and J. Sun, "Skipgnn: Predicting molecular interactions with skip-graph networks," *Scientific Reports*, vol. 10, no. 1, p. 21 092, 2020. DOI: 10.1038/s41598-020-77766-9 (cit. on p. 30).

[40] X. Ju, D. Murnane, P. Calafiura, *et al.*, "Performance of a Geometric Deep Learning Pipeline for HL-LHC Particle Tracking," *The European Physical Journal C*, vol. 81, no. 10, p. 876, 2021. DOI: 10.1140/epjc/s10052-021-09675-8 (cit. on pp. 30, 82, 83).

[41] Y. Li, Y. Ji, S. Li, *et al.*, "Relevance-Aware Anomalous Users Detection in Social Network via Graph Neural Network," *CoRR*, vol. abs/2104.06095, 2021. [Online]. Available: https://arxiv.org/abs/2104.06095 (cit. on p. 30).

[42] NVIDIA Corporation, *CUDA C Programming Guide*, Version 11.0, NVIDIA Developer, Sep. 2020 (cit. on pp. 31, 32).

[43] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable Parallel Programming with CUDA," *Queue*, vol. 6, pp. 40–53, Mar. 2008. DOI: 10.1145/1401132.1401152 (cit. on p. 31).

[44] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: an imperative style, high-performance deep learning library," in *33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019 (cit. on p. 31).

[45] NVIDIA Corporation, *CUDA C++ Programming Guide*, Version 11.5, NVIDIA, 2021. [Online]. Available: https://docs.nvidia.com/cuda/archive/11.5.0/pdf/CUDA_C_Programming_Guide.pdf (cit. on p. 32).

[46] *Graphics Double Data Rate (GDDR6) SGRAM Standard*, JEDEC Standard JESD250D, Published: May 2023, May 2023 (cit. on p. 33).

[47] *High Bandwidth Memory DRAM (HBM1, HBM2)*, JEDEC Standard JESD235C, Revision C, Published: December 2018, Dec. 2018 (cit. on p. 33).

[48] A. Delgado and D. Emeliyanov, “ATLAS trigger algorithms for general purpose graphics processor units,” Oct. 2016, pp. 1–6. DOI: 10.1109/NSSMIC.2016.8069670 (cit. on pp. 37, 50).

[49] ATLAS Collaboration, “The ATLAS inner detector trigger performance in pp collisions at 13.TeV during LHC Run 2,” *The European Physical Journal C*, vol. 82, no. 3, Mar. 2022, ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-021-09920-0 (cit. on pp. 38, 42).

[50] J. D. Burleson, S. Caillou, P. Calafiura, *et al.*, “Physics Performance of the ATLAS GNN4ITk Track Reconstruction Chain,” CERN, Geneva, Tech. Rep., 2023. [Online]. Available: https://cds.cern.ch/record/2882507 (cit. on p. 39).

[51] P. Billoir, “Progressive track recognition with a Kalman-like fitting procedure,” *Computer Physics Communications*, vol. 57, no. 1, pp. 390–394, 1989, ISSN: 0010-4655. DOI: https://doi.org/10.1016/0010-4655(89)90249-X (cit. on p. 40).

[52] R. Frühwirth, “Application of Kalman filtering to track and vertex fitting,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 262, no. 2, pp. 444–450, 1987, ISSN: 0168-9002. DOI: https://doi.org/10.1016/0168-9002(87)90887-4 (cit. on p. 40).

[53] T. G. Cornelissen, M Elsing, I Gavrilenko, *et al.*, “The global χ^2 track fitter in atlas,” *Journal of Physics: Conference Series*, vol. 119, no. 3, p. 032 013, Jul. 2008. DOI: 10.1088/1742-6596/119/3/032013 (cit. on p. 40).

[54] S. Amrouche, L. Basara, P. Calafiura, *et al.*, “The Tracking Machine Learning challenge : Throughput phase. The Tracking Machine Learning Challenge: Throughput Phase,” *Comput. Softw. Big Sci.*, vol. 7, no. 1, p. 1, 2023. DOI: 10.1007/s41781-023-00094-w. arXiv: 2105.01160 (cit. on p. 51).

[55] S. Caron, N. Dobreva, A. Ferrer Sánchez, *et al.*, “Trackformers: in search of transformer-based particle tracking for the high-luminosity LHC era,” *The European Physical Journal C*, vol. 85, no. 4, Apr. 2025, ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-025-14156-3 (cit. on p. 51).

[56] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” in *31st International Conference on Neural Information Processing Systems*, ser. NIPS’17, Long Beach, California, USA: Curran Associates Inc., 2017, 6000–6010, ISBN: 9781510860964 (cit. on p. 51).

[57] S. N. Swatman, A Krasznahorkay, and P Gessinger, “Managing heterogeneous device memory using C++17 memory resources,” *Journal of Physics: Conference Series*, vol. 2438, no. 1, p. 012 050, Feb. 2023. DOI: 10.1088/1742-6596/2438/1/012050 (cit. on p. 76).

[58] N. Dos Santos Fernandes, “GPU Acceleration and EDM Developments for the ATLAS 3D Calorimeter Clustering in the Software Trigger,” CERN, Geneva, Tech. Rep., 2025. [Online]. Available: https://cds.cern.ch/record/2924377 (cit. on p. 76).

[59] ATLAS Collaboration, *Track finding performance plots for a Graph Neural Network pipeline on ATLAS ITk Simulated Data*. [Online]. Available: https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/IDTR-2022-01/ (cit. on p. 80).

[60] S. Caillou, P. Calafiura, S. A. Farrell, *et al.*, “ATLAS ITk Track Reconstruction with a GNN-based pipeline,” 2022. DOI: 10.5281/zenodo.8119762 (cit. on p. 79).

[61] R. C. Hoetzlein, “Fast fixed-radius nearest neighbors: Interactive million-particle fluids,” in *GPU Technology Conference*, vol. 18, 2014, p. 2 (cit. on p. 80).

[62] D. J. Pearce, “An improved algorithm for finding the strongly connected components of a directed graph,” 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:7781255 (cit. on p. 80).

[63] D. T. Murnane, “Graph Neural Networks for High Luminosity Track Reconstruction,” EP-IT Data Science Seminar, 2022. [Online]. Available: https://indico.cern.ch/event/1104699/ (cit. on p. 81).

[64] R. Gray and D. Neuhoff, “Quantization,” *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998. DOI: 10.1109/18.720541 (cit. on p. 87).

[65] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, “Degree-Quant: Quantization-Aware Training for Graph Neural Networks,” in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=NSBrFgJAHg (cit. on p. 87).

[66] Y. LeCun, J. Denker, and S. Solla, “Optimal Brain Damage,” in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., ser. NIPS, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf (cit. on pp. 87, 97, 100).

[67] C. Liu, X. Ma, Y. Zhan, *et al.*, “Comprehensive Graph Gradual Pruning for Sparse Training in Graph Neural Networks,” *CoRR*, vol. abs/2207.08629, 2022. [Online]. Available: https://arxiv.org/abs/2207.08629 (cit. on pp. 87, 97).

[68] S. Dittmeier, “Online track reconstruction with graph neural networks on FPGAs for the ATLAS experiment,” CERN, Geneva, Tech. Rep., 2025. [Online]. Available: https://cds.cern.ch/record/2923275 (cit. on p. 87).

[69] M. J. Atkinson, J. Burzynski, P. Butti, *et al.*, *ACORN - a charged object reconstruction network*, https://gitlab.cern.ch/gnn4itkteam/acorn (cit. on pp. 89, 129).

[70] X. Ying, “An Overview of Overfitting and its Solutions,” *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 022 022, Feb. 2019. DOI: 10.1088/1742-6596/1168/2/022022 (cit. on p. 97).

[71] B. R. Bartoldson, A. S. Morcos, A. Barbu, and G. Erlebacher, “The Generalization-Stability Tradeoff In Neural Network Pruning,” *CoRR*, vol. abs/1906.03728, 2020. [Online]. Available: https://arxiv.org/abs/1906.03728 (cit. on p. 97).

[72] D. Gurevin, M. Shan, S. Huang, M. A. Hasan, C. Ding, and O. Khan, “PruneGNN: Algorithm-Architecture Pruning Framework for Graph Neural Network Acceleration,” in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 108–123. DOI: 10.1109/HPCA57654.2024.00019 (cit. on p. 97).

[73] H. Mao, S. Han, J. Pool, *et al.*, “Exploring the Regularity of Sparse Structure in Convolutional Neural Networks,” *CoRR*, vol. abs/1705.08922, 2017. [Online]. Available: https://arxiv.org/abs/1705.08922 (cit. on p. 97).

[74] T. Gale, M. Zaharia, C. Young, and E. Elsen, “Sparse GPU kernels for deep learning,” in *International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’20, Atlanta, Georgia: IEEE Press, 2020, ISBN: 9781728199986 (cit. on p. 97).

[75] S. Roy, “Understanding the Impact of Post-Training Quantization on Large-scale Language Models,” Sep. 2023. DOI: 10.48550/arXiv.2309.05210 (cit. on p. 97).

[76] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” *CoRR*, vol. abs/1503.02531, 2015. [Online]. Available: https://arxiv.org/abs/1503.02531 (cit. on p. 97).

[77] H. He, J. Wang, Z. Zhang, and F. Wu, “Compressing Deep Graph Neural Networks via Adversarial Knowledge Distillation,” *CoRR*, vol. abs/2205.11678, 2022. [Online]. Available: https://arxiv.org/abs/2205.11678 (cit. on p. 97).

[78] Y. Yang, J. Qiu, M. Song, D. Tao, and X. Wang, “Distilling Knowledge from Graph Convolutional Networks,” *CoRR*, vol. abs/2003.10477, 2021. [Online]. Available: https://arxiv.org/abs/2003.10477 (cit. on p. 97).

[79] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, “Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks,” *CoRR*, vol. abs/1808.06866, 2018. arXiv: 1808.06866. [Online]. Available: http://arxiv.org/abs/1808.06866 (cit. on p. 101).

[80] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” in *AAAI Conference on Artificial Intelligence*, ser. AAAI, 2019. DOI: 10.1609/aaai.v33i01.33014780 (cit. on pp. 108, 123).

[81] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," *CoRR*, vol. abs/1611.01578, 2016. [Online]. Available: https://arxiv.org/abs/1611.01578 (cit. on pp. 108, 123).

[82] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *International Conference on Learning Representations*, ser. ICLR, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX (cit. on pp. 108, 123).

[83] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the Construction of Internet Portals with Machine Learning," *Inf. Retr.*, vol. 3, no. 2, 127–163, Jul. 2000, ISSN: 1386-4564. DOI: 10.1023/A:1009953814988 (cit. on pp. 111, 112).

[84] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019 (cit. on p. 111).

[85] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," *CoRR*, vol. abs/1706.02216, 2018. [Online]. Available: https://arxiv.org/abs/1706.02216 (cit. on pp. 111, 112).

[86] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008, ISBN: 978-0-521-86571-5. [Online]. Available: http://nlp.stanford.edu/IR-book/information-retrieval-book.html (cit. on p. 111).

[87] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," *CoRR*, vol. abs/1803.03635, 2019. [Online]. Available: https://arxiv.org/abs/1803.03635 (cit. on p. 123).

[88] G. Schindler, W. Roth, F. Pernkopf, and H. Fröning, "Parameterized Structured Pruning for Deep Neural Networks," in *Machine Learning, Optimization, and Data Science - 6th International Conference, LOD 2020, Siena, Italy, July 19-23, 2020, Revised Selected Papers, Part II*, G. Nicosia, V. Ojha, E. L. Malfa, *et al.*, Eds., ser. Lecture Notes in Computer Science, vol. 12566, Springer, 2020, pp. 16–27. DOI: 10.1007/978-3-030-64580-9_3 (cit. on p. 123).