Imperial College London
Department of Physics

New directions for particle tracking at the High-Luminosity LHC

Liv Helen Våge

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy, April 2024

© The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Abstract

The High Luminosity upgrade of the Large Hadron Collider (LHC) will increase the instantaneous luminosity from 2 $\times 10^{34} \mathrm{cm}^{-2}\,\mathrm{s}^{-1}$ to $5 - 7.5 \times 10^{34}$ $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$. This will increase the average number of simultaneous proton-proton collisions from the current value of 60 to 140 and eventually 200. This poses a large challenge for the experiments, particularly for the trigger systems which process data quickly to determine which events to store. The CMS High Level Trigger (HLT) will receive data at a rate 5-7.5 times higher than at nominal operation. It will need to accommodate an acceptance rate up to 7.5 kHz, which will require around 20 times the computing power that is required now.

This thesis presents detailed timing studies of the HLT algorithms, identifying charged particle tracking as the main cause of the increased computing requirement. The computational performance of the current Kalman filter based tracking algorithm is evaluated along with a parallelised algorithm called mkFit. The potential for these algorithms to be accelerated with co-processors is discussed. Two machine learning algorithms are then explored. Graph neural nets (GNNs) have shown promising performance for tracking. This thesis applies them to CMS data and removes commonly used simplifications. Building graphs is shown to be the most time consuming element of the GNN pipeline, making low latency applications challenging. Two novel reinforcement learning methods for tracking are introduced, exploring both a continuous and a discrete environment. They show a hit classification score up to 90%, depending implementation options. The classification score is not yet sufficient for future LHC requirements, but with a small and simple neural net, it shows potential for speeding up tracking.

Statement of originality

This certifies that the work presented within contains my own work unless stated otherwise. Any results, conclusions or figures that are not my own are referenced. In Chapter 3, all the results presented are from my own experiments using algorithms that were developed by the groups referenced. In Chapter 4, the GNN algorithm was developed by the Exa.TrkX collaboration. Any performance related studies and extensions to the algorithm is my own work which builds on their codebase. Several of the plots in this section is based on their code.

Acknowledgements

Firstly, I want to thank Prof. Alex Tapper for being an absolutely excellent supervisor. Few would show the patience you have shown and let me continue working on something that seemed like it would not work for a very long time. The creative reign I was given is a rare gift that made my PhD so much more enjoyable. Your responsiveness and continued support was truly appreciated.

A large thanks goes to my other colleagues at Imperial. To Marco Barboune for teaching me about FGPAs and giving me invaluable input throughout my PhD. To Lucas Santiago for flying out to Geneva, and jumping into my work to help me out. To Andy Rose and Mikael Mieskolainen for patiently listening to my meandering RL attempts for years. I am especially grateful to Benjamin Radburn-Smith and to Rob Bainbridge for setting aside a large amount of time to help me shake data out of CMSSW. A large thanks also to Tobias Becker for our many conversations.

A very special thanks goes to the Pearson family for becoming an unexpected second family to me. Without Christine's endless hospitality, patience and understanding, this thesis would never have been half of what it is now. I really can't express my gratitude enough. A big thanks to Jay for listening to my rants about my work and always being a fantastic support. I want to thank Jess for her emotional support, even if she sometimes had to be bribed. Big thanks go to my mum, dad and sister for always reminding me that I have a home to come back to. Though you might not understand my work, I know you understand me.

A huge thanks goes to my friends, especially to Sammy, Kathryn, Cathrine, Mina, and Ragnhild. You have shown me so much support, and you are truly the best friends anyone could ever wish for. I am so lucky to have you in my life. A long overdue thank you goes to Alexandra - I don't think I would ever have become so interested in physics if I didn't see it through your eyes first.

The largest thanks is reserved for Daniel who had to be hospitalised, deported and become a barista for this thesis to come to life. This is as much your work as it is mine. There are no words to speak to the support you have given me for the past few years. I won't be able to make it up to you, but I will enthusiastically delve into trying.

This thesis was made possible with funding provided from the STFC.

Dedication

Til Daniel, verdens største og beste blåbær

"How did I escape? With difficulty. How did I plan this moment? With pleasure."

— Alexandre Dumas, The Count of Monte Cristo

Contents

Abstract **i**

Statement of originality **iii**

Acknowledgements **iii**

1 Introduction **1**

1.1 The Large Hadron Collider 2

1.1.1 Operating principles of the LHC 3

1.1.2 The four main LHC detectors 7

1.2 High luminosity LHC 8

1.2.1 Hardware upgrades for the LHC 9

1.2.2 Physics motivation for the HL-LHC 10

1.3 Machine learning 18

1.3.1 Loss functions 19

1.3.2 Gradient descent 19

1.3.3 Artificial neural networks 21

1.3.4 Practical ML development strategies 22

1.4 Computing architecture and co-processors 24

1.4.1 CPU 25

1.4.2 GPU 27

1.4.3 FPGA 27

2 The CMS detector **30**

2.1 Tracker system 31

2.1.1 Pixel detector 33

2.1.2 Silicon strip detector 33

2.2 Electromagnetic calorimeter 34

2.3 Hadronic calorimeter 35

2.4 Muon detector 36

2.5 Level-1 Trigger 36

2.6 High Level Trigger 37

2.7 Track reconstruction 38

2.7.1 Pixel and silicon hit reconstruction 39

2.7.2 Seed generation 39

2.7.3 Track building 41

2.7.4 Track fitting and selection 45

2.7.5 Tracking performance 46

2.8 The Phase-2 CMS detector 48

2.8.1 Phase-2 calorimeters 48

2.8.2 Phase-2 muon detector 50

2.8.3 Minimum ionising particle timing detector 50
2.8.4 Phase-2 Tracker 51
2.8.5 Phase-2 L1 Trigger 52
2.8.6 Phase-2 HLT 53

3 Computational performance of the Phase 2 High Level Trigger 55

3.1 Measuring computational performance 55
3.2 The Phase 2 and Run 3 HLT menus 57
3.2.1 Phase 2 pixel tracking 61
3.2.2 Phase 2 tracking 64
3.3 The mkFit algorithm 70
3.3.1 Computational performance of the mkFit algorithm 70
3.4 HLT track reconstruction algorithms under development 73
3.5 Summary 73

4 Graph neural nets for tracking 75

4.1 Graph neural nets 76
4.2 Interaction network for particle tracking 78
4.3 GNNs for the TrackML competition 79
4.3.1 Applying a GNN to TrackML data 81
4.4 GNNs using CMS Monte Carlo events 83
4.5 Removing simplifications 85
4.5.1 Allowing more than one hit per layer per track 86
4.6 How much information is needed for tracking 92

4.7 CMS Run 3 data . . . 93

4.8 Related work . . . 95

4.9 Summary . . . 97

5 Reinforcement learning for tracking **98**

5.1 Policy gradient methods . . . 101

5.1.1 Vanilla policy gradient . . . 102

5.1.2 Trust region policy optimization . . . 103

5.1.3 Proximal policy optimisation . . . 104

5.2 Q-learning methods . . . 104

5.3 Actor-critic methods . . . 106

5.3.1 Deep deterministic policy gradients . . . 106

5.3.2 Twin delayed deep deterministic policy gradient . . . 106

5.3.3 Soft actor critic . . . 107

5.4 Summary of common RL algorithms . . . 107

5.5 Tracking in a continuous action space . . . 108

5.5.1 Learning to navigate to a point . . . 108

5.6 Discrete action space . . . 114

5.6.1 Learning a state quality for each possible next hit . . . 116

5.6.2 Learning the quality of two possible actions . . . 117

5.7 Learning tracks in the barrel . . . 118

5.8 Learning tracks from the entire detector . . . 120

5.8.1 Including all the hits in the hit selection pool . . . 122

5.9 Choosing between n hits . . . 123

5.10 Towards a complete RL tracking implementation . . . 125

5.10.1 Implementing a simple track propagation algorithm . . . 125

5.10.2 Track propagation with an RL filter . . . 127

5.10.3 Seeding algorithm . . . 129

5.11 Summary . . . 132

6 Conclusion **134**

Bibliography **135**

List of Tables

1.1 Beam parameters for the LHC design and the HL-LHC [16]. 7

List of Figures

1.1 Size and latency of LHC data 2

1.2 The CERN accelerator complex 4

1.3 Plan for the upgrades of the LHC to high luminosity 9

1.4 The status of the current knowledge about interactions of the Higgs boson . . . 12

1.5 Discovery reach of supersymmetric light particles at the HL-LHC 15

1.6 Track left by a long lived particle 16

1.7 Gradient descent and a neural net 20

1.8 The development of microprocessors over 42 years 25

1.9 Architecture of CPUs, GPUs and FPGAs 26

2.1 The CMS detector 31

2.2 The CMS coordinate system 32

2.3 The Phase-1 tracker 32

2.4 A Higgs candidate event 35

2.5 A particle in the pixel detector 39

2.6 Generating tracking seeds 41

2.7 Track building 43

2.8 The Kalman filter . . . 45

2.9 Tracking efficiency and fake rate for the Phase-2 CTF algorithm . . . 47

2.10 Tracking efficiency and fake rate as a function of η . . . 47

2.11 The Phase-2 CMS detector . . . 49

2.12 The link between the CMS detector upgrades and physics goals . . . 49

2.13 The Phase-2 tracker . . . 52

2.14 Track stub finding and the p_T module . . . 52

2.15 The Phase-2 level one trigger . . . 54

3.1 Relative timings of the Run 3 and Phase 2 HLT menus . . . 59

3.2 Relative timing of modules in the Phase 2 HLT . . . 60

3.3 Relative timing of the pixel hit triplet and quadruplet functions . . . 62

3.4 Ratios of high pileup tracking elements compared to no pileup . . . 63

3.5 Callgraph of advancing a track one layer . . . 65

3.6 The proportional run time of updating a track with a hit using the Kalman filter 66

3.7 A simplified loop-flow diagram for track building within the CMSSW. . . . 67

3.8 Microarchitecture of CkfTracking algorithm . . . 68

3.9 Timing performance of the mkFit track building algorithm. . . . 71

3.10 Callgraph of the mkFit algorithm . . . 72

4.1 The graph data structure . . . 77

4.2 GNN message passing . . . 78

4.3 The interaction network used for particle tracking . . . 79

4.4 The TrackML detector . . . 80

4.5 Graph build performance for TrackML and CMS data 82

4.6 Implication of applying simplifications while building graphs 83

4.7 Time taken to build one graph 84

4.8 Tracking efficiency of a GNN using CMS MC data from the pixel detector . . . 86

4.9 Edge classification of graphs at different p_T cuts. 87

4.10 Complexity of allowing more than one hit per layer per track in GNNs 88

4.11 The number of nodes when building a graph for the outer tracker 89

4.12 Building a graph for the outer tracker using stubs 89

4.13 The effects of using a module mapping when building a graph neural net 91

4.14 Minigraphs 93

4.15 Tracking using a GNN for CMS Run 3 data 94

4.16 Track reconstruction using a GNN for Run 3 data 95

5.1 A taxonomy of common RL techniques 101

5.2 A standard vanilla policy gradient algorithm using a Gaussian distribution . . . 103

5.3 The distance between hits and their closest neighbours 109

5.4 A VPG algorithm trying to navigate to a goal 109

5.5 Learning how to reconstruct a single track 111

5.6 Learning to reconstruct barrel tracks 112

5.7 Changing the RL prediction to the closest hit 113

5.8 Two approaches to learning the quality of possible next track hits. 116

5.9 State-quality method 118

5.10 Learning barrel tracks with the direction quality method 119

5.11 Tracking using the direction-quality method 120

5.12 Hit prediction accuracy as a function of detector position 121

5.13 Curriculum learning in the barrel of the detector 123

5.14 Learning to choose between a selection of n hits 124

5.15 Performance of a simple tracking algorithm 126

5.16 Track propagation with an RL filter 127

5.17 Tracking efficiency of an RL assisted track propagation algorithm 130

5.18 Reconstructing tracks in an event from Run 3 data. 131

Chapter 1

Introduction

The Large Hadron Collider (LHC) [1] is one of the largest scientific endeavours in the world, and aims to investigate the fundamental forces and particles of the universe. Since the discovery of the electron in 1897, a zoo of particles have been unveiled, revealing the building blocks that make up our world [2]. As we probe further, it takes increasingly advanced equipment to make new discoveries. The LHC is perhaps the pinnacle of particle physics experiments, providing highly engineered detectors that serve a large array of physics needs.

The LHC produces data at a rate on the order of tens of petabytes per second [3]. Not all this data can be stored, and the main experiments at CERN have triggers in place to reduce the data size. Several of the experiments have implemented this in two steps; a Level-1 trigger (L1T) and a High Level Trigger (HLT). The data size and latency of these are shown in Fig. 1.1. Latency refers to the time taken to compute or transfer data. As shown in the figure, the triggers at the LHC represent one of the more sizable data processing challenges in the world. The LHC is set to undergo a set of upgrades that will see the data size increase even further, requiring faster data processing rates. This thesis will look at methods for tackling this problem, in particular for the HLT at the Compact Muon Solenoid (CMS) experiment [4]. Section 1.1 introduces the LHC experiment and explains the motivation behind the upgrade. Chapter 2 outlines the CMS experiment and how this experiment will change with the planned upgrade. Chapter 3 then presents measurements of the computational performance of the CMS HLT for

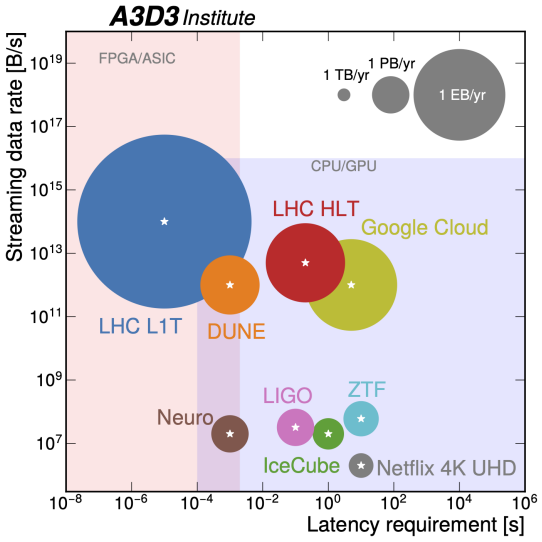


Figure 1.1: Data size and latency requirements for some of the large science experiments compared to Netflix and Google cloud. Figure from [5].

the upgrade. It shows that reconstructing charged particle tracks is especially challenging and discusses potential solutions. The final two chapters discuss the potential of applying graph neural nets and reinforcement learning to the track reconstruction problem. The main focus of the thesis is evaluating the possibility of using machine learning for the HLT, while also considering the use of co-processors - dedicated chips like GPUs or FPGAs, discussed further in Section 1.4.2.

1.1 The Large Hadron Collider

One of the world's largest scientific instruments is the Large Hardron Collider (LHC) [1], situated on the border between France and Switzerland. Its 27 km tunnel was repurposed from the Large Electron-Positron (LEP) [6] collider, which operated between 1989 to 2000. LEP helped probe the unification of the electromagnetic and weak forces, but did not find the predicted Higgs boson [6]. It was thought that a large energy upgrade could solve this while continuing to probe the Standard Model even further. This resulted in the construction of the LHC, which saw particle collision energy go from the order of hundreds of GeV to the TeV scale. The main results from LHC so far has been the discovery of the Higgs boson in 2012 [7],

[8] and improved precision on standard model measurements. There has been no conclusive evidence of new physics, and many big questions remain unanswered. Some of these are:

- Dark matter - We observe that around 95% of the matter in the universe is not visible, but we do not know what this matter is [9], [10].
- Gravity - All forces we know of apart from gravity have been associated with a force carrying boson. Gravity is also many orders of magnitude weaker than the other forces, and we don't have a reason for this.
- Flavour puzzle - Why are there three generation of quarks, and why do their masses differ?
- Neutrinos have mass, but only neutrinos with spin opposite to their momentum (left-handed) are observed. This suggests that they do not interact with the Higgs field, which inverts handedness. It might imply that neutrinos are their own anti-particles, and their interaction is how their mass is generated. The neutrino mass generation mechanism, and why the mass is so small compared to other particles, remains unknown [11].
- Matter-antimatter asymmetry - Most of the matter in the universe is matter rather than anti-matter. There is no experimentally proven reason for this.

Work at the LHC continues to search for answers to these several other questions by direct searches for new phenomena and by performing precision measurements of the Standard Model.

1.1.1 Operating principles of the LHC

The LHC accelerates two beams of protons in opposite directions at near light speed. At four different points in the tunnel, the two beams can be brought to collision, creating a plethora of other particles that can be measured with its four detectors. The LHC is a synchrotron, which means the particles are kept in a fixed loop by magnets and are accelerated by an electric field. The magnets are superconducting electromagnets and need to be kept at an extremely low temperature [12].

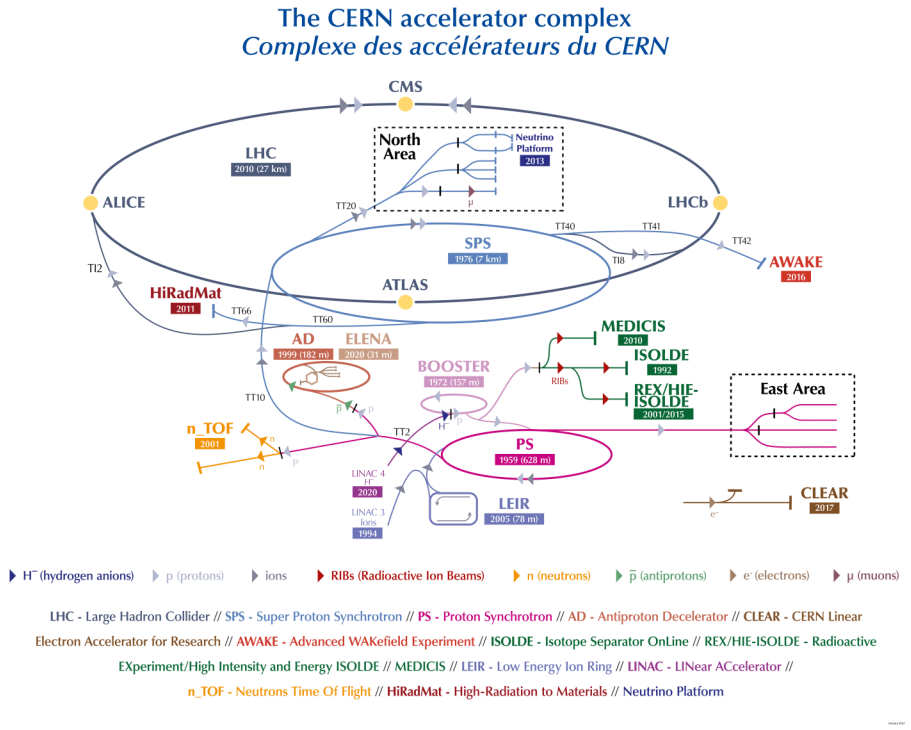


Figure 1.2: The CERN complex. Figure from [14]

Proton source and injector chain

The protons supplied to LHC and several nearby experiments start from hydrogen gas, H-. The newly built linear accelerator LINAC4 [13] combines the gas with an ion source and a cathode to produce H-. Using H- instead of protons, as was done before LINAC4, gives a smaller beam loss, a more focused beam, and makes injection easier. The ions are then accelerated to 160 MeV. During injection of H- into the next step - the proton synchrotron booster (PSB) [1] - the ions are stripped of their two electrons with a thin carbon foil. The PSB is the first and smallest synchrotron in the injection chain, accelerating the protons to 2 GeV. The subsequent proton synchrotron further pushes the beam to 25 GeV, and delivers the beam to the superproton synchrotron in 25 ns spaced bunches. The SPS finally increases the beam energy to 450 GeV. Part of the beam is deflected, so the two beam lines of the main ring are filled. A schematic of this setup is shown in Fig. 1.2.

Beam acceleration

Once the beam is injected into the main LHC ring, the particles circle the tunnel until they are accelerated to 6-7 TeV. This happens in 16 radio frequency (RF) cavities, 8 for each beam line. An oscillating electric field is delivered to the cavities by electron beam tubes called klystrons. These generate a fixed frequency of 400 MHz, which the cavities are designed to resonate with, amplifying the voltage. The particles are accelerated when they cross the cavities, requiring them to be in phase with the driving frequency. They therefore travel in packs known as bunches. A slightly asynchronous particle in the bunch would experience a slightly higher or lower potential, so that it would oscillate around the synchronous particles. The RF cavities therefore help the beam stay bunched.

Over 1200 dipole magnets enclose the beam pipes to bend the beams around the circular path. Double layers of cabling containing Niobium-Titanium strands enclose each beam pipe. These are superconducting, which is necessary to carry a very high current without overheating. The magnetic field of 8.3 T induced is in opposite directions over the two beam pipes, so that they both experience a force directed towards the centre of the ring. If the coils were not superconducting one would need the tunnel to be 120 km long to achieve the same result [15]. To maintain the superconducting phase, superfluid helium is circulated in a dedicated ring that keeps the magnets at a frosty 1.9 K. The magnets and the helium distribution lines are thermally insulated by a vacuum. The beam pipes also have to be a vacuum to avoid protons colliding with other materials. The resulting vacuum is emptier than the interstellar void and colder than the temperature of space. As the beams circle the tunnel, they are focused by over 800 quadrupole magnets. These magnets are arranged in a pattern called FODO; sequences of four quadrupoles that focuses vertically, deflects, focuses horizontally and deflects. There are also other multipole magnets that handle other corrections, such as electromagnetic interactions between bunches.

Proton-proton collision

When the beams have reached around 6-7 TeV each, they can be brought to collision. There are quadrupoles at each of the four collision points in the ring that squeeze the beam to the size of a human hair [1]. At these interaction points, the two beam lines are joined and see collisions every 25 ns with a centre of mass energy of 14 TeV. The number of collisions that can be produced in the detector per cm^2 per second is given by the machine luminosity L. This is often referred to as instantaneous luminosity to distinguish it from integrated luminosity - the luminosity integrated over time - which reflects the amount of data taken. Instantaneous luminosity depends on the number of bunches in the beam (N_b), the number of protons in each bunch (n), and the revolution frequency (f_v). The beams will collide at an angle to make sure the interaction only happens at certain point, and to reduce electromagnetic interactions between the beams. This is reflected by a reduction factor, F. The collision rate also depends on how well packed together the beams are, which is measured within the amplitude modulation (β) and the transverse emittance (ϵ). These reflect the parallelism and width of the beam, respectively. The formula for luminosity is shown in Equation 1.1, where γ is the relativistic gamma factor. [1].

$$L = \frac{N_b^2 n f_r \gamma F}{4\pi \epsilon_n \beta^*} \tag{1.1}$$

Not all head-on collisions will interact in an interesting way, that is, creating secondary particles. The number of desired events (N) is therefore given by the product of the luminosity and the probability that the desired event takes place (cross section σ) [1]. This is shown in Equation 1.2.

$$N = L\sigma \tag{1.2}$$

The beam parameters for the nominal design of the LHC and the HL-LHC, discussed in Section 2.12, are shown in Table 1.1. After an interaction point, the two beams are separated by dipole

Parameter	Nominal LHC	HL-LHC
Proton energy [TeV]	7	7
Peak luminosity* [$\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$]	1.0×10^{34}	8.11×10^{34}
N_b	2808	2760
n	1.15×10^{11}	2.2×10^{11}
f_r [kHz]	11.2	11.2
F	0.84	0.34*
ϵ_n [µm]	3.75	2.50
Minimum $\beta^*[m]$	0.55	0.15

Table 1.1: Beam parameters for the LHC design and the HL-LHC [16].

* without crab cavities [17]

magnets. After around 10-20 hours of continuous collisions, the beam is made much less intense by a magnet, and is made to collide with a block of concrete and graphite composite to fully stop the experiment.

1.1.2 The four main LHC detectors

LHC hosts four detectors in the main ring. ATLAS (A Toroidal LHC Apparatus) [18] and CMS (Compact Muon Solenoid) [4] are the general purpose detectors, representing two different solutions to the same challenge. Proton-proton interactions cover a large range of energies and angles, requiring large detector coverage around the interaction region and excellent momentum resolution for most particles. One can achieve good momentum resolution by a strong magnetic field or a by having a large distance over which a particle is bent [19]. CMS chose the former and ATLAS the latter. CMS is therefore compact with a strong magnet, and ATLAS is big with a weaker magnet. The result is a similar value for the product of the magnetic field, B, and length L for both experiments. There are also other design differences, like the material used in the calorimeters, tracking system and muon detectors. One of the main purposes of having two general purpose detectors is that they can verify each others' results, as they did with the Higgs boson discovery. The CMS detector will be discussed in detail in Chapter 2.

The smallest experiment by collaboration size is LHCb (LHC-beauty) [20]. It specialises in physics related to the bottom quark, but is also capable of other measurements such as exotic

decays. B-mesons and their decay products tend to stay close to the beam line axis, so while other detectors encapsulate the interaction point, LHCb instead stretches 20 metres along the beam pipe. Mesons with b-quarks leave a fairly distinct trace in the detector since the b-quark has a relatively high mass and low transition rate to its decay products, creating displaced vertices. This makes it ideal for studying charge conjugation parity symmetry (CP) violation. CP symmetry states that the laws of physics should be the same if a particle's charge is swapped and the spatial coordinates are inverted. LHCb has announced discovery of several decays violating CP symmetry [21]. CP violation is particularly interesting because it is observed in weak interactions but not in strong interactions, though there is not a known reason for it to be conserved within the strong interaction [22]. CP violation is also strongly linked to matter-antimatter imbalance, as CP violation is likely to have been present in the early universe [23].

The ALICE (A Large Ion Collider Experiment) [24] detector was designed with Quantum Chromodynamics (QCD) and nuclear physics in mind. For about a month each year, LHC provides heavy ion collisions like lead-lead. The collisions reach a temperature 100 000 times hotter than the centre of the sun [25]. Under these conditions, quarks and gluons are freed from the confinement of neutrons and protons. This creates a quark-gluon plasma (QGP) [26], similar to what the early universe consisted of before matter as we know it was formed. ALICE also takes data for proton-proton collisions with the specific intention to look closely at the strong interaction. The experiment has shed light on properties of the QGP, hadron formation and more [27].

1.2 High luminosity LHC

To continue operation and extend its physics potential, the LHC is set to undergo extensive upgrades [16]. This resulting phase is called the High Luminosity LHC (HL-LHC). Between 2020 and 2030, a series of upgrades will take the HL-LHC to five times the instantaneous luminosity, and a data volume ten times larger than the nominal LHC. Research efforts have

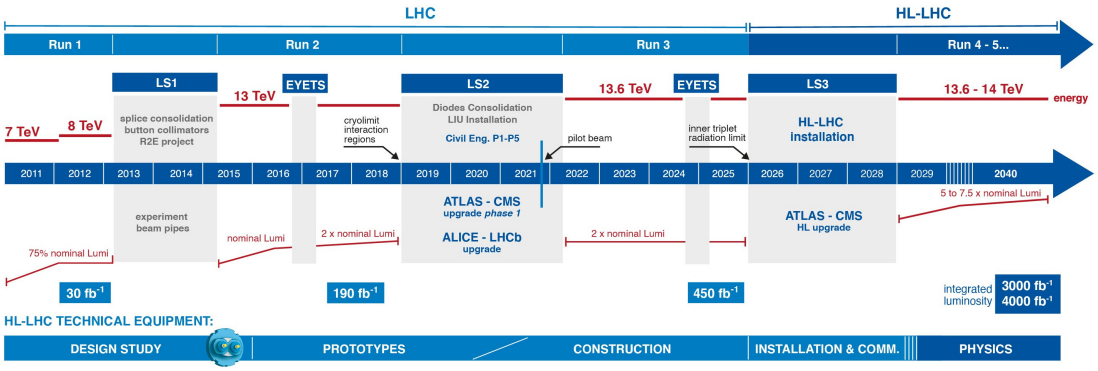


Figure 1.3: Plan for the upgrades of the LHC to high luminosity. Figure adapted from [16]

.

been ongoing for more than ten years to develop and test the technology that will make this possible. The main deliverable is a machine with a lifetime a decade beyond Run 3 and a luminosity of $5 - 7.5 \times 10^{34}$ cm^{-2} s^{-1}. Some of the main advancements include new and more powerful magnets, crab cavities and improved interaction regions [28]. The main changes will happen at the injection sites of ATLAS and CMS. The timeline of the project is shown in Figure 1.3.

1.2.1 Hardware upgrades for the LHC

To increase luminosity, some of the parameters in Equation 1.1 will need to change. A typical approach is to reduce the amplitude function β^*. This can be done by better focusing the beam with stronger and larger aperture quadrupoles. This is a fairly local change, so only the interaction regions in the LHC would need to change. At HL-LHC, novel Nb_3Sn quadrupoles capable of 12 T fields will be installed. Since the beam becomes more narrow in the direction transverse to the collisions, the crossing angle needs to increase to maintain the luminosity. This will be handled by newly developed crab cavities. These cavities tilt the beams into the interaction point, maximising the bunch crossing area, and then tilting the them out again so they don't interact with other nearby bunches from the opposite beam. Four cryomodules, each containing two crab cavities, will be installed at the sites of both ATLAS and CMS. Given the envisaged value of β^*, around 70% of the peak luminosity will be recovered from these crab cavities [17].

Again referring to Equation 1.1, another way of increasing the luminosity is to increase the beam brightness - the ratio between the number of protons in each bunch and the transverse emittance. This is handled by the upgrades to the injection chain, described in Section 1.1.1. Throughout the Long Shutdown 2 (see Fig. 1.3), LINAC4 was deployed and the other accelerators in the injection chain were updated to be ready to deliver 2.4 times the nominal beam brightness [13], [16]. A higher particle density in each bunch brings brings a higher radiation level and increased risk of damage in the case of e.g. beam loss. The collimator system will therefore be upgraded, including changing the material from carbon materials to tungsten based materials, which can better catch stray high energy particles [16]. Because of potential interactions with the collimators, stronger dipoles will be inserted as well. There will also be superconducting power lines and updates to the cryogenic systems. In total, about 1.2 kilometers of equipment at LHC will be upgraded. A final complication of increased beam brightness is increased pileup. As two bunches collide, there will be one or a few hard scatters, which will form the primary vertices of the collision. Many collisions other than the hard scatter will also occur. The average number of simultaneously colliding particles, known as pileup, will go from around 50 to 140 and eventually 200. This poses great problems for event reconstruction in the trackers and calorimeters. The added complexity from pileup is the focus of this thesis.

1.2.2 Physics motivation for the HL-LHC

The luminosity produced over time is directly related to the discovery potential of the LHC. Increased integrated luminosity will enable scientists to probe rare events and lower uncertainties for important measurements. The main goals of the HL-LHC will be to explore the Standard Model and QCD further, and to look for new physics, particularly within the Higgs sector. Each of the new physics measurements made available by the HL-LHC are directly related to how the detector will be upgraded. A summary of this is shown in Fig. 2.12. A description of the detector upgrades can be found in Section 2.8.

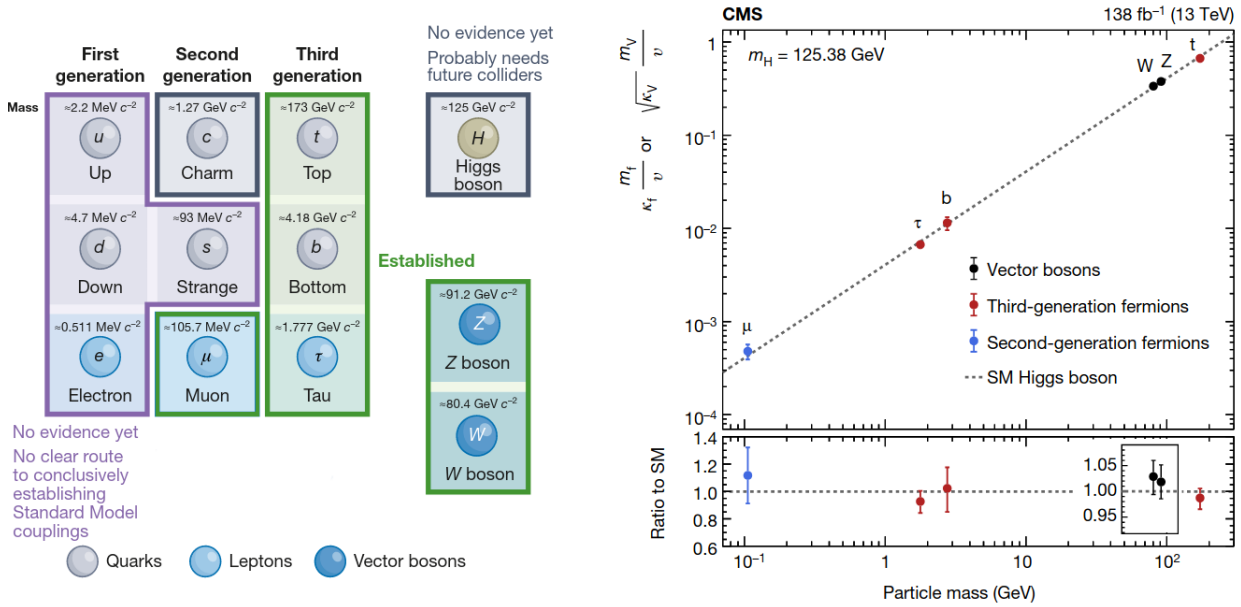
Precision Higgs measurements

The Higgs boson was one of the main motivations for building the LHC [29], [30]. It was theoretically predicted in the 1960s, but had not been discovered until the LHC announced discovery of a new boson in 2012. It is thought that as the very early universe was cooling down, the Higgs field underwent a phase transition, spontaneously breaking the symmetry of the electromagnetic and weak forces [29]. These forces have since then been distinct with separate strengths. The Higgs field settled into a non-zero vacuum expectation value, creating a field that interacts with particles to give them mass. During the rapid expansion of the early universe, potential fluctuations in the Higgs field could have caused variations in the distribution of matter and energy [31]. Baryon number conservation may be violated when electroweak symmetry is unbroken, and more baryons than anti-baryons can be produced. The nature of how Higgs broke the electroweak symmetry is therefore linked to the distribution of matter and antimatter of the universe [32]. All measurements of the Higgs boson so far agree with Standard Model predictions. It has been confirmed as a boson with a mass of 125 GeV, and it has been observed in all the third generation decay channels [32]. More recently, it was also confirmed in the second generation decaying to muons [33]. The strength of the Higgs coupling is proportional to the mass of fermions and to the square of the masses for bosons, so observing decays to lighter particles requires high precision methods. Not only will precision Higgs measurements help probe known physics, it is also strongly linked to new physics, as we will see in Section 1.2.2. Higgs boson properties remains the primary target of the HL-LHC programme [29]. The current state of the Higgs boson is summarised in Fig. 1.4.

Considering Fig. 1.4, there are two SM Higgs couplings that are projected to be within reach of the HL-LHC that were not available before; a direct decay to charm quarks and the self-coupling of the Higgs boson. The latter is particularly important, as the Higgs potential depends on it, but has not yet been directly measured. The Higgs potential can be written as

$$V(H) = \frac{1}{2}m_H^2 H^2 + \lambda v H^3 + \lambda H^4 \tag{1.3}$$

where m_H is the Higgs mass, λ is the self-coupling term, and v is the Higgs field vacuum



(a) The state of Higgs coupling measurements

(b) Higgs coupling measurements

Figure 1.4: The status of the current knowledge about interactions of the Higgs boson. Figure (a) adapted from [32] and Figure (b) from [34].

expectation value. The standard model predicts that $m_H = \sqrt{2\lambda}v$ where both m_H and v have been measured to a below percent precision. We consequently know that the standard model predicts that $\lambda \approx 0.13$ with a sub-percent precision [35]. The self-coupling is therefore an excellent probe of the shape of the Higgs potential and by extension new physics. The high energy behaviour of the self-coupling also affects the stability of the Higgs potential. The low mass of the Higgs means that there could be a lower energy vacuum state it can decay into. The decay is predicted to take on the scale of 10^{600} years, so it is not an immediate problem. However, it could have been shorter in the early universe, having an impact on cosmology [36]. If a measured self-coupling value does not agree with the SM, it is a clear signal of new physics and a revision of the electroweak symmetry breaking is needed. The lowest level self-interaction is a coupling between three Higgs bosons, which is easiest to measure with the production of two Higgs bosons. The production rate of this event is 1000 smaller than that of a single Higgs boson, so this measurement will be available only with the HL-LHC[35]. The final state that provides the best sensitivity is one where one Higgs boson decays into a b-quark pair, and the other to two photons. This emphasises the need of a HL-LHC detector with excellent b-tagging, photon identification efficiency and pileup mitigation. The second best channel has

a bb$\tau\tau$ signature, requiring excellent trigger capabilities. It is projected that the HL-LHC detectors will be able to provide a measurement of the self-coupling to a significance of 4σ [37].

There are several other precise Higgs measurements that will be of interest. Higgs decaying to two Zs and then to 4 leptons is a signature that can be reconstructed with a very high accuracy. By measuring the angular distributions of the decay products, the CP properties of the Higgs can be precisely measured. A direct measurement of the Higgs decay width has previously been limited by experimental resolution, but the $H \rightarrow ZZ \rightarrow 4\ell$ measurement should be accurate enough to provide a good measurement [28]. Excellent electron and muon reconstruction at a low p_T and high rapidity is required. Studies also show that it is necessary to measure the SM Higgs couplings to within at least a few percent to discriminate between a range of new physics scenarios [38]. It is estimated that the HL-LHC will be able to deliver this. The large data sample of the HL-LHC will also enable studies of complex final states. The link between this physics and the detector upgrades is shown in Fig. 2.12.

Beyond Standard Model Higgs physics

The relatively small measured mass of the Higgs boson could point towards new physics. Assuming that the Standard Model is correct up to the Planck scale, extremely short lived particles that pop in and out of existence would take the mass of the Higgs boson to around 10^{19} GeV [39]. In order for the Higgs to have its measured mass, there could be new, extremely finely tuned contributions that cancel out these large numbers. The same situation has historically arisen for charged fermions, but was solved by the discovery of the positron. Similarly for the pion, the problem was resolved by realising that it was a composite particle [39]. It is natural to think that the problem for the Higgs could be solved by new particles or the Higgs being a composite particle. All other previous particles that were considered fundamental scalar particles have turned out to be composite. There is a plethora of theories within these two options [39], but no evidence for either has been identified yet. This is a large driver for expecting new physics at the TeV scale. Evidence of a composite nature of the Higgs could be accessible at the HL-LHC. New particles could be found within SUSY, or through other searches [37]. Several

exotic Higgs models predict dark matter candidates, which is discussed in the next section. The HL-LHC will also be able to exclude new heavy Higgs bosons of masses up to 2.5 TeV using the $H/A \rightarrow \tau\tau$ channel, where A represents a new Higgs boson [37].

Beyond Standard Model physics

The HL-LHC will be sensitive to many beyond Standard Model (BSM) phenomena [29], [30]. Supersymmetry (SUSY) is one of the models that could help solve the Higgs mass problem described in the previous section. In this model, every fermion has a bosonic partner, and vice versa. The partners of the fermions have an s added to their name, and bosonic partners have an *-inos* suffix. For most of the SUSY scenarios, the mass reach that the HL-LHC is sensitive to will increase by 20-50% compared to Phase-1, which will cover a lot of the relevant SUSY phase space [40]. Sbottom and stop squarks could be produced at the HL-LHC at relatively large cross sections. The stop squark is one of the particles that could stabilise the Higgs boson mass, and this would imply a stop squark mass around 1-2 TeV. Many versions of SUSY also hypothesise new particles. The superpartners of the three neutral bosons could mix, creating four new particles known as neutralinos. Mixing of the W and a charged Higgs could also create charginos. The HL-LHC might be sensitive to these particles, depending on their mass. The HL-LHC is also projected to be able to discover or exclude stop squark masses up to 1.25 TeV, given small neutralino masses [37]. This is illustrated in Fig. 1.5. Gluinos have a relatively high cross section at HL-LHC, while neutralinos and charginos have slightly lower cross sections. Direct searches for these are both planned and ongoing [41].

Dark matter is another highly anticipated BSM puzzle piece. Several theoretical frameworks aim to explain it, including SUSY, axions, dark sector particles, or extra dimensions. SUSY could account for dark matter through the lightest neutralino, which is predicted to be stable and weakly interacting [37]. The axion, which was introduced as a solution to the strong CP problem, is also weakly interacting and long lived, providing another candidate [43]. A more particle agnostic solution would be the existence of new particles that interact very weakly with the standard model, typically called the dark sector [37]. Excited states of the standard

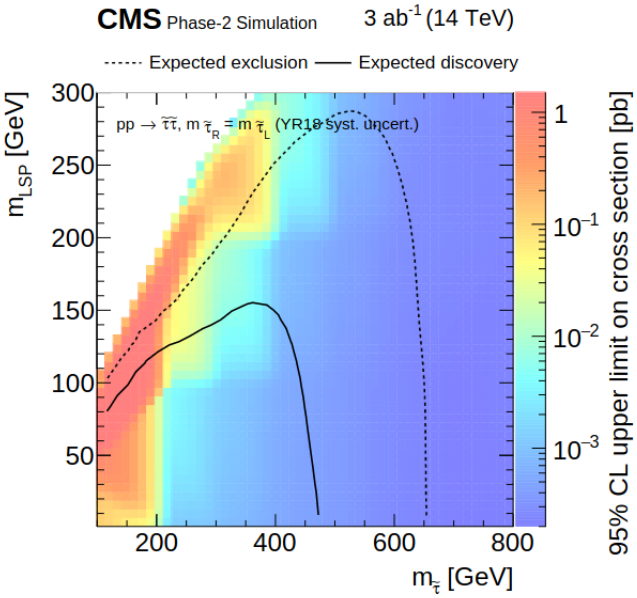


Figure 1.5: The 95% confidence level exclusion reach and 5σ contours of stop masses and the lightest supersymmetry particle (LSP) achievable at CMS using the complete data set collected at HL-LHC. Figure from [42].

model particles in extra dimensions might be yet another explanation [44]. Luckily, many of these theoretical models would leave similar signatures in the detector. A common feature of dark matter candidates is that they are weakly interacting. This can be seen by e.g. galaxy collisions, where normal matter decelerates and clusters, and dark matter doesn't [45]. We also haven't seen dark matter candidates at direct detection experiments yet, showing again that dark matter interacts very weakly [46]. We therefore don't expect dark matter to interact with our hadronic detectors, but to instead carry off missing transverse energy. Depending on the dark matter candidate, this energy could be very small, so systematic uncertainties must be quite small to be sensitive to it. The increased data set and detector resolution at the HL-LHC provides sensitivity to many of the theoretical models [37].

Both SUSY and dark matter models, as well as other BSM models, predict the existence of long lived particles (LLPs) [47]. These particles have lifetimes of a few nanoseconds or longer, and would generally live long enough to decay in the detector, or even pass through it. LLPs can be detected directly by recognising that LLP candidates would often be massive and ionise the subdetectors more than SM particles. They would travel at a low speed, and would therefore be seen as a late signal compared to the primary vertices. The more precise timing provided

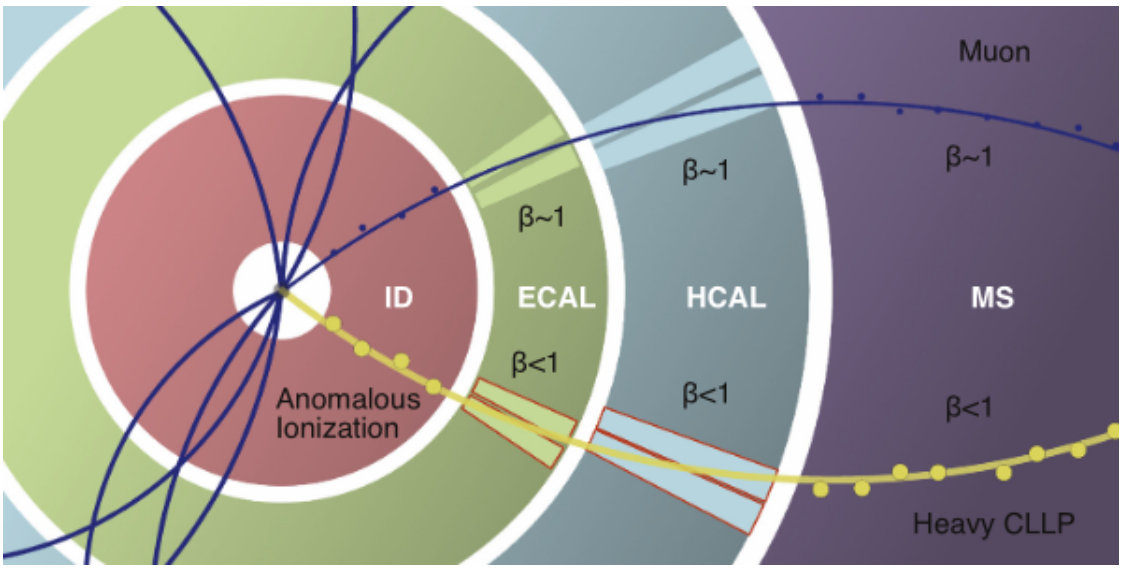


Figure 1.6: An illustration of what the track of a long lived particle (yellow) could look like. The long lived particle ionises the detectors more and could bend in an abnormal way. Figure from [47].

by the HL-LHC detectors, as described in Section 2.8.3, will be a great tool for identifying LLPs. LLPs could also be detected by their decay to SM particles. Because the LLPs take a longer path, the decay products would typically have displaced vertices. Uncommon tracks, like parabolic bending or disappearing tracks could also be signs of LLPs. The lightest neutralino is for instance predicted to be long-lived and slow in many SUSY models, presenting displaced and delayed di-jets with missing energy. An illustration of a LLP track is shown in Fig. 1.6.

The low uncertainty offered by the high integrated luminosity of HL-LHC might also shed light on the fermion and neutrino mass mystery described previously in 1.1. The generation of flavour structure could be at a high energy scale that is currently inaccessible to direct measurement, but could be seen by decay ratios that disagree with Standard Model predictions [37]. Theories of neutrino mass generation also predicts the existence of heavy neutrinos, which one could observe or at least constrain at the HL-LHC [37].

There are some results from both LHC and other experiments that present tension with the Standard Model. Experiments at Fermi National Accelerator Laboratories (FNAL) have for example measured the muon magnetic moment and the W mass to be more than expected, with 5.1 and 7 σ, respectively [48], [49]. Several measurements that have been in tension with the Standard Model have later disappeared through a higher amount of data, lower uncertainties, or

different analysis methods, like lepton universality violation at 3 σ significance at LHCb [50]. Contrarily, there could be tensions that are now hiding within the boundaries of statistical uncertainty that can be revealed at higher precision.

Vector boson scattering

Vector boson scattering is another interesting process that will be studied at HL-LHC [28]. Vector boson scattering occurs when two vector bosons, W and Z, scatter off each other by exchanging particles. It is a very rare process which the large data sample and extended acceptance of the HL-LHC detectors could make available. Some BSM Higgs models, especially those containing Higgs doublets, predict a modification of the Higgs-fermion couplings. The Higgs field normalises vector boson scattering, preventing the scattering rates from growing indefinitely with energy. If the observed scattering rate is different than that predicted by the SM, normalisation would have a contribution from an unknown mechanism. This could be a sign of an extended Higgs sector and new high mass vector bosons [37].

QCD

Pb-Pb collisions at HL-LHC will occur at ten times the nominal luminosity [51]. QCD is quite complex and difficult to probe mainly because of asymptotic freedom; prying quarks and gluons further away from each other strengthens their interaction. This motivates dedicated experiments since our understanding of QCD limits our understanding of the precise structure of hadrons, interactions like proton-proton collisions and the states of the early universe. HL-LHC will provide precise measurements of the wavelike behaviour and the parton behaviour of QGP. For the first time, there is a possibility that QCD can be described in a universal way, across smaller to larger systems [30].

1.3 Machine learning

Probing new physics often requires new tools. This thesis will explore Machine Learning (ML) in particular. ML has grown in popularity over the last decade, especially emphasised by the recent release of large language models like Chat-GPT [52]. The theoretical framework for ML started development already in the 1950s [53], but the implementations were limited due to the lack of computing power. As computing evolved according to Moore's law, described in Section 1.4, the potential of ML increased along with computing power. This increase, combined with unprecedented amounts of accessible data and algorithmic innovation, has led ML to an explosion of usage and popularity. Instead of being given explicit rules, ML learns from a sufficiently large collection of examples. There are many types of ML algorithms, but most of them have a configurable number of internal parameters whose values are tuned until the algorithm converges. The models are represented by mathematical functions where the parameters are iteratively changed by measuring the performance with a loss metric. ML can learn from labelled examples, known as supervised learning, or without, known as unsupervised. There are algorithms that sit between these boundaries, like reinforcement learning, discussed in detail in Chapter 5. For most models, there is no theoretical guarantee that ML will converge on an optimal answer, and ML can be time consuming to develop. There is also no one model that always gives the best solution, which is known as the the no free lunch theorem [54]. If one can easily program an explicitly rule-based algorithm, this is usually preferred to ML. ML presents an excellent alternative if this cannot be done, or it would be too slow to compute or to implement. ML is especially suitable for complex problems, or problems where the rules or desired behaviour change over time.

High energy physics uses ML extensively since the problems are often complex, the data size is large, and there are sophisticated simulation frameworks that can provide labelled data. An excellent collection of papers related to the uses of machine learning in HEP can be found in the "living review" [55]. Perhaps the most ubiquitous use of ML within HEP is the use of boosted decision trees to separate background and signal data. There is some resistance to the use of ML - both in HEP and in general - since the models can be quite opaque. It can be difficult

to retrace why a model has made its decisions, and there are concerns that this could impact the robustness of the physics performance and interpretability of the results. Machine learning therefore needs good levels of testing to address these issues.

1.3.1 Loss functions

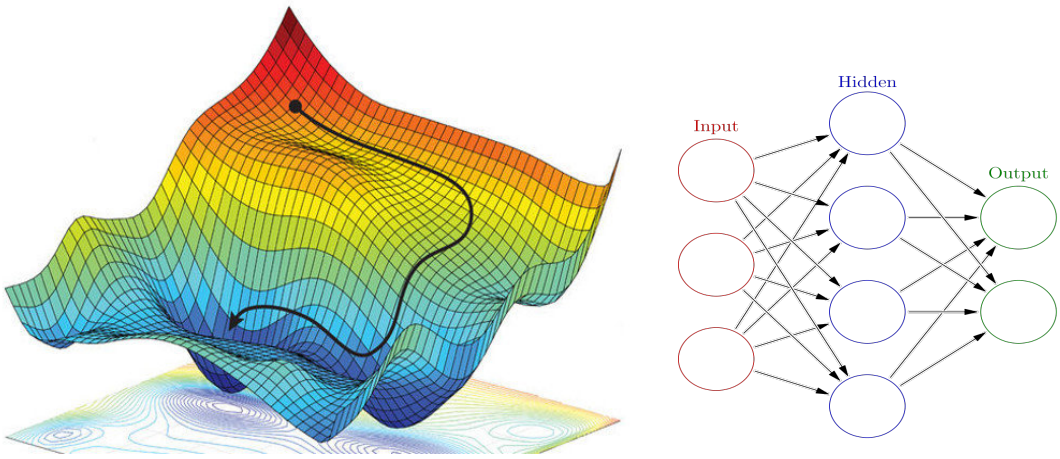
To understand ML, we first need to understand how we translate a problem into a goal that a machine can understand. The basic premise of ML is that we pass some features, x, to a model that performs transformations on x, and makes a prediction, y. The loss function is a measure of the difference between the prediction and the desired value, thus defining our goal. The loss function can be as simple as an average of the absolute difference between the model and the prediction. For classification problems, it is often a measure of entropy, that is the purity of each category. One can also use loss functions to penalize specific unwanted outcomes or data outliers. In general, a differentiable loss function is preferred so the gradient of the loss can be used in gradient descent. In models such as reinforcement learning, the loss function is replaced by a reward, which is discussed in more detail in Chapter 5. Most supervised neural nets, discussed in Section 1.3.3, use maximum likelihood estimators. This loss function, J, measures the probability of the model predicting y given input data x, averaged over input and output pairs x, y drawn from the data distribution. This can be written as

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \log p_{model}\left(\boldsymbol{y}|\boldsymbol{x}\right), \tag{1.4}$$

where $\mathbb{E}$ is the expectation value and θ are the parameters of the model [56].

1.3.2 Gradient descent

Most common ML models use gradient descent to determine how to change the parameters of the model to minimise the loss. Generally, a model initially has random or preset parameters. Data is passed through the model to make a prediction, and the loss is calculated using the loss function. The partial derivative of the loss function with respect to the parameters is



(a) Gradient descent (b) Neural net

Figure 1.7: Fig. (a) shows the task of minimising the loss function by iteratively taking steps towards a global minimum. Fig. (b) shows a fully connected feed-forward neural net with one hidden layer. Figure (a) from [57] and Figure (b) from [58].

calculated. This determines how the model should change the parameters to minimise the loss function. A learning rate can be set to determine how big the steps should be in the direction of the gradients. The learning rate can be adaptive, so e.g. bigger steps are taken when the gradient is steeper. This is then repeated until the model converges or until a certain number of iterations have passed. An illustration of this is shown in Fig. **??**. One can update the model parameters after a full pass through the data set, known as batch gradient descent. The difference between the predictions and the expected answer is calculated for the batch and the model parameters are updated. This leads to a stable gradient, but can result in slow training since a lot of the data samples usually represent similar information. A common strategy is to use mini-batches. This calculates the gradient and updates the model parameters every time it randomly samples data, known as a batch. This is generally faster than batch gradient descent, but makes the learning potentially sensitive to the size of the batches [56]. Using Equation (1.4), the gradient of the loss function is

$$\nabla_\theta J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \nabla_\theta \log p_{model}\left(\boldsymbol{y}|\boldsymbol{x}\right). \tag{1.5}$$

1.3.3 Artificial neural networks

Artificial neural networks (NN), henceforth called neural nets, are perhaps the most powerful ML method because of their number of tunable parameters and the way neurons interact. Most advanced models today, like large language models or image classifiers, use neural nets. Since the ML models discussed in Chapter 4 and Chapter 5 are based on neural nets, they will be discussed in some detail. Neural nets can be made from perceptrons, or neurons, that are connected to each other. Neural nets are loosely inspired by how the brain works; neurons work together to evaluate input, and there is an activation potential that evaluates whether the input is important enough for the neuron to fire. Each neuron performs an operation on the incoming data that can be written as

$$f\left(\sum_{i=1}^{n} x_i w_i + b\right) \tag{1.6}$$

where x is the input vector, w are the learnable weights, b is a bias vector and f is an activation function. The activation function is what allows for non-linearity in the model. There are many different activation functions, but a popular choice is the RELU function because of its generalisability and large and consistent gradients that are ideal for gradient descent [59]. It sets any negative input to zero, analogous to how neurons in the brain would ignore certain input. Neurons are connected to each other and arranged in layers, as shown in Fig. **??**. The input vector for the first layer of the NN are the features of the data. The function above is applied, and the neurons pass the output to the neurons in the next layer. Most general purpose neural nets are feed-forward, so neurons only pass information to neurons downstream in the network. Models that feed information back are called recurrent neural nets and are often used where context is needed, like time series or language processing. A neuron typically has connections to all the neurons in the subsequent layer. This is not always the case, such as in convolutional neural nets or graph neural nets. This will be discussed more in Chapter 4. The output of the neural net will depend on the type of problem, but can be a e.g. a predicted regression value or a vector of probabilities for categories in a classification. The final activation layer needs to be chosen carefully since this will determine the final output. The gradient of

the loss function is then calculated, as described in the previous section. Since the weights are dependent on each other, the chain rule is used when calculating the gradient in a process called back-propagation.

The number of neurons and layers, i.e. the architecture, will depend on the problem, and usually needs experimentation. The universal approximation theorem states that a neural net with only one hidden (i.e. not input or output) layer is sufficient to approximate any real, finite-dimensional continuous function if given enough neurons [60]. However, it is still possible that the neural net is not able to learn the task. The optimization process could be wrong for the problem, there could be overfitting, explained in Section 1.3.4, and there is no method for knowing exactly how many neurons are enough to represent the problem, though there are some approximations [61]. Though one hidden layer is enough in theory, it can be shown that one often requires fewer parameters and obtain a lower error by creating several layers instead of having a shallow network [56]. An intuitive way to understand this is that there is an underlying belief that the decision boundary consists of composite functions, where one takes the output of another as input. Since there is no formal methodology to design neural nets, ML usually needs experimentation to obtain good models.

1.3.4 Practical ML development strategies

Though neural nets can in theory approximate any function, finding a neural net that converges with an acceptable loss within a reasonable time can be challenging. One of the first steps is to perform feature engineering; creating input features that represent the problem well. The features can be cleaned, removed, transformed and presented in a way that makes the problem more conducive to learning. This can be time consuming, and is not always straight forward. This has partially led to an entire discipline, called representation learning [62]. Here, ML is used to learn how to represent features in a way that is useful for the problem. It has also been applied to many other tasks including outlier detection and data compression. The desired output of the model can similarly be changed to help the ML algorithm. This is usually done in conjunction with choosing a loss function. One could e.g. try to predict a likelihood that an

event will pass through the L1T, or one could classify it as a binary outcome.

Once the input, desired output and loss function is decided, the architecture and hyperparameters of the model must be chosen. Hyperparameter is a term used for parameters of the model that do not change during training. This includes the learning rate, batch size, neural net architecture, and loss function. The aim of tuning hyperparameters is to adjust the effective capacity of the model to represent the data. This is dependent on three factors; is the model the right size for representing the data, can the algorithm successfully minimize the loss function and is the degree of regularisation right for the problem. Regularisation is a term for strategies used to reduce overfitting and penalize complexity. The perhaps most important hyperparameter is the learning rate since this controls the effective capacity in a more complex way than other hyperparameters [56]. Hyperparameter tuning is often done with a grid search, where a grid of parameters are explored. To avoid bias when tuning hyperparameters, part of the data set is usually not used in the training process, and is called validation data. Another part of the data is held out for completely independent testing, which is known as the test set.

Even if the features, output and hyperparameters are well tuned, the no free lunch theorem implies that we have no guarantee that the chosen model is the best model for the data. Any model makes inherent assumptions about the data and has properties that make it favour one solution over another. Convolutional neural nets for instance assume translational invariance in the data and regularised loss functions will favour a simpler model over a more complex one [63]. This is known as inductive bias, and will be especially important in this thesis because the ML models will make implicit assumptions about the physics problem considered, and we will be comparing the success of different ML models. Especially in Chapter 5, where reinforcement learning is used, a large part of the work focuses on how to infuse the model with the right inductive bias.

We therefore need tools to evaluate the performance of a model. The loss function is the first metric to consider. If the loss does not decrease during training, there might be problems with the model, such as the gradient taking on very large or small numbers. The former would lead to a very unstable model and the latter would lead to learning in very small increments. If

the loss decreases well, one of the next things to look for is overfitting. This is most easily seen in differences in accuracy between the training and test set. Almost any ML model uses data that is split unevenly in two or more parts. Training data is used to teach the model, and a smaller set of test data is withheld and never used in training. Overfitting happens when the ML algorithm learns properties that are true for the training set, but not for the data in general. An extreme example would be an algorithm that has learnt decisions for individual data points, which result in a near perfect accuracy for the training set, but would likely be very poor for the test set. In contrast, underfitting can happen when the model does not effectively represent the data in the training set. This illustrates the idea of a trade-off between bias and variance. Bias is the error that is introduced by having a simple model approximate complex data. As we saw, some degree of bias is needed, but too much could underfit the data. Variance is the model's sensitivity to fluctuations in the training data. Again, some extent of variance is needed as it gives it flexibility to fit to complex data, but too much can lead to overfitting. A balance between the two must be struck. Both of these can be adjusted with hyperparameter tuning, and can be measured by the difference in performance between the training and test sets. There are also many other ways to measure performance apart from using the loss function. Measuring the physics accuracy is naturally very important for physics use cases. For applications that might be used with co-processors, the size of the neural nets is also important.

1.4 Computing architecture and co-processors

This thesis will evaluate the possibility of speeding up the run time of CMS HLT. Apart from addressing specific shortcomings, accelerating code is often done by parallelising CPU computations, and/or using co-processors. To choose the right alternative, it is important to be aware of some of the fundamentals of computer architecture. Today's High Performance Computing (HPC) landscape has in particular been shaped by Moore's law [64] and Dennard scaling [65]. In 1965 Gordon Moore presented an article about the development of the number of components on integrated circuits [64]. He showed that the number of transistors on an

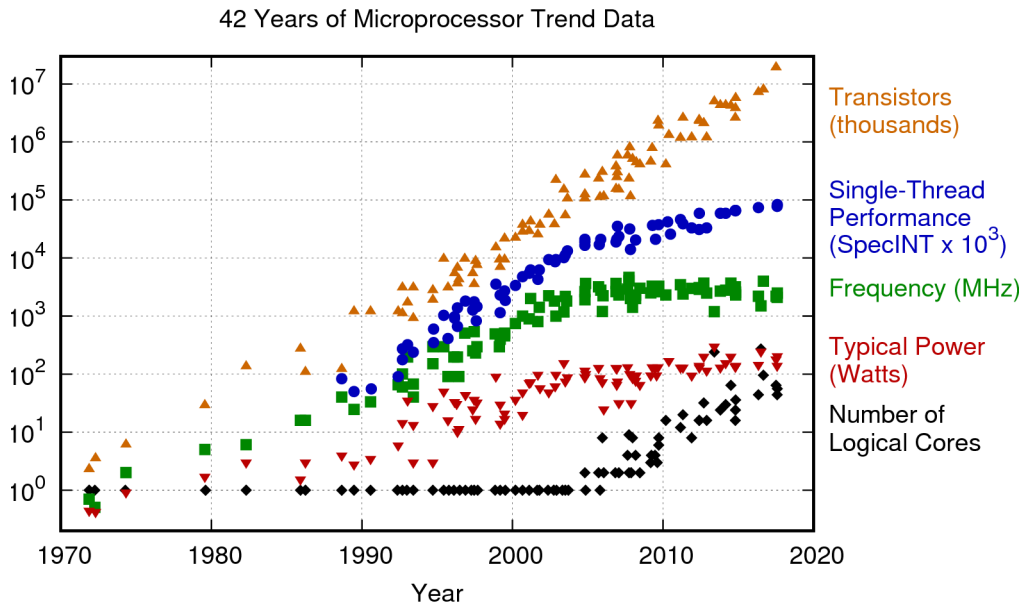


Figure 1.8: The development of microprocessors over 42 years. Figure from [68].

integrated circuit doubles about every two years, a trend which has held true since then. This is shown in Fig. 1.8. It is likely that the trend will continue to be true for at least a few more years, though it is showing signs of slowing down [66]. Fig. 1.8 also shows that the increase in frequency, power consumption and single-thread performance has tapered off since around 2005. This represents an end of Dennard scaling [65]. Dennard proposed that the power density of transistors stay constant, so as the transistors get smaller, so does the power consumption. The clock frequency could therefore be increased by adding more transistors without drawing more power. Around 2005, issues like leakage current, noise and heat dissipation meant that the scaling dramatically stopped. To continue putting more transistors on a chip without breaching an unacceptable power consumption, cost and overheating, mutli-core processor chips became popular. Several comparatively low clock cycle and low power chips work together to provide a scalable solution. Ways to parallelise computations also saw great performance increases, ushering an era of parallel computing [67].

1.4.1 CPU

Modern Central Processing Units (CPUs) are found in any standard computer. On a basic level, they operate with a *von Neumann architecture*, where an undivided memory stores data

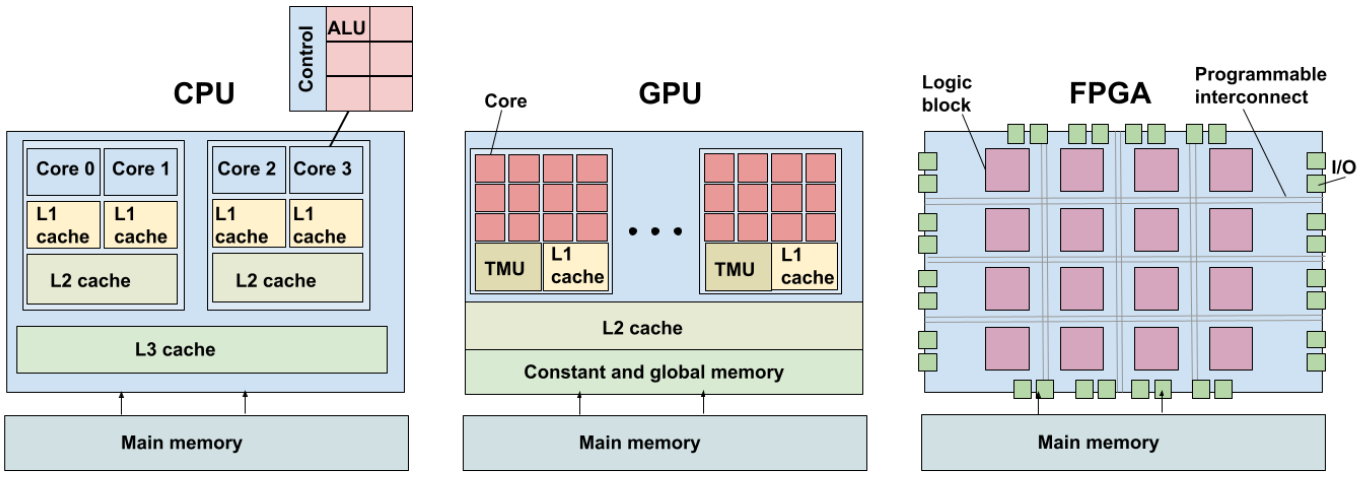


Figure 1.9: Architecture of CPUs, GPUs and FPGAs. CPUs have a complex cache and control system, allowing them to be general purpose. GPUs have smaller caches and blocks of cores, designed for multiprocessing. FPGAs have programmable interconnects, allowing hardware implementations.

and instructions, and a processing unit executes instructions and operates on data. As CPUs have developed, their performance and complexity has increased. There is instruction level parallelism, so many instructions can be executed per cycle. Many instructions will be "in flight" at once, meaning they are in various stages of completion. Instructions can be handled out of order, so in a different order than the program specifies as long as this does not change the result. Data can also be prefetched, i.e. speculatively fetched because the algorithm expects that it will be needed. CPUs will also guess whether $if-$statements will fail or pass, and will start speculatively executing code, known as branch prediction. The memory is typically handled in three stages, a main memory, caches and registers. Registers are what the processor operates on. The caches are quick access temporary storage of recently used data that is likely to be reused. There are typically several levels of caches of decreasing size and latency, and they are shared by multiple cores on the CPU. The architecture is shown in Fig. 1.9. Multicore CPUs distribute the workload across the cores. Most modern computer CPUs have several cores. To fully take advantage of the CPU architecture, data reuse and paralellism can be implemented by e.g. vectorising computations. We will see an example of this in Section 3.3.

1.4.2 GPU

Graphics Processing Units (GPUs) were originally developed to handle the small matrix multiplications needed when rendering graphics in low latency environments, like needed for e.g. computer games. Since machine learning also has many small matrix multiplications when calculating weights and losses, GPU use was extended to accelerating scientific computing along with machine learning training and inference. While CPUs have multiple instructions, GPUs have a single instruction, multiple data (SIMD) architecture. GPUs are designed to deliver a high-throughput - computing large amounts of data quickly, rather than focusing on delivering any single task quickly. This is achieved by separate threads working mostly independently. GPUs often have thread blocks, where threads in a block execute the same instruction. These blocks typically share a memory. If there are conditional statements, the thread that performs the true or the false statement will be executed while the other branch threads wait. This means that GPUs are typically not performant for code with a lot of branching. GPUs are typically suitable for applications with natural data parallelism and a large degree of parallelism. Code with many loops, large loops and nested loops are often good GPU candidates since the loops can often be parallelised. Limited data dependencies and regular memory access patterns are beneficial since there is latency incurred for transferring the data to the chip, and the cache is small. Code generally needs restructuring and optimisation to work well on GPUs. An important exception is machine learning, since high level packages tend to support GPU implementations without the need to change the code base. GPUs generally offer good acceleration for a relatively low development and maintenance cost.

1.4.3 FPGA

Designing circuits specifically for an application can lead to significant speedups since the hardware, memory use, instructions and parallelism is optimized. Optimized implementations also lead to a lower power consumption. The lowest latency alternative is Application Specific Integrated Circuits (ASICs). They typically take months to years to develop and can be quite expensive since they are highly optimized and uniquely designed. They often become more

beneficial when they are mass produced. Most of the raw data readout at CMS, as discussed in Chapter 2, is done using ASICs since they are robust, and deliver very low latency with low power consumption.

A cheaper and quicker to develop alternative to ASICs are Field Programmable Gate Arrays (FPGAs) which consist of reconfigurable logic blocks. Software applications are translated into a physical circuit by a hardware description language. This delivers the benefits of a hardware implementation while being reconfigurable. Hardware description languages like VHDL [69] and Verilog [70] require a lot more code per instruction compared to higher level languages. They can be difficult to learn, adding to the development time and maintenance threshold. For specific use cases, graphical languages, like Labview [71] can be a good option. High level synthesis (HLS) languages [72] have become increasingly popular. These use higher level languages like C++ or Python to translate the code into a hardware description language. These might lead to slightly less performant implementations, depending on the application. A significant contribution from CMS is hls4ml [73]. This takes models from the ML packages Keras and Pytorch and use HLS backends like Vivado and Intel's HLS to implement the ML on FPGAs. This dramatically reduces the threshold to put neural nets on FPGAs, which otherwise can take years to implement, especially for non-specialists. Several successful uses of hls4ml have been reported, including an application for particle tracking [74], [75].

FPGAs can be a good alternative when very low latency or deterministic latency is required. This is why they are the choice for the Level-1 trigger at CMS, as described in Section 2.5. FPGAs have a smaller amount of internal storage than GPUs. A good candidate is therefore often a relatively small amount of code that takes up a large portion of processing time. GPUs have an order of magnitude higher clock frequency than FPGAs. The clock frequency measures how quickly instructions can be processed. FPGAs can therefore be faster only for algorithms that are not able to exploit all the computing resources on CPUs and GPUs [76]. Workloads that are inefficient on CPUs generally have the following characteristics;

- High branch misprediction rate
- High cache miss rate

GPUs do not have branch prediction or prefetchers that fetches from the cache, so code with these characteristics become even less efficient. FPGAs implement branches in hardware and cache misses can be avoided with custom loading logic. These two points are therefore the main characteristics of a good FPGA candidate. FPGAs have a comparatively low on-chip memory and a limited amount of digital signal processors (DSP), so any potential algorithm must be carefully analysed to see that it would fit on an FPGA. It can take days or even weeks for an algorithm to be mapped to FPGA hardware, so the development time can be long. It is therefore crucial to analyse an algorithm carefully to see that it is suitable for an FPGA implementation before starting to develop it.

Chapter 2

The CMS detector

The CMS detector [4] is a 14 000 tonne general purpose detector sat 100 metres underground in the French village Cessy. It detects particles using a layered structure of a particle tracker, calorimeters and a muon detector. Electrically charged particles leave traces in the tracker, photons and electrons are detected in the electromagnetic calorimeter ECAL, hadrons in the hadronic calorimeter HCAL and muons in the muon detector. The HCAL is designed to be hermetic; fully stopping the relevant particles to accurately measure them. Some of the main design requirements of CMS were good resolution of charged particle momentum, electromagnetic energy, transverse and missing transverse energy, dijet mass, and good muon identification. The first requirement is addressed by a strong magnet that bends charged particles. At the heart of CMS sits a solenoid made from coiled Nb-Ti fibres, creating a magnetic field of up to 4 T. It was the largest magnet in the world when it was made [77], allowing the tracker and calorimeters to be placed inside, and creating a comparatively compact detector. The magnetic field is contained by a steel yoke, which at 12 500 tonnes is by far the heaviest component of CMS. The muon detectors are partially interweaved with this. The layout of the CMS detector is shown in Fig. 2.1.

By convention, CMS uses a Cartesian coordiante system with the origin at the nominal interaction point. The z-axis goes along the beamline, x points towards the centre of the LHC and y goes vertically upward. The x-y plane is the transverse plane in which the transverse

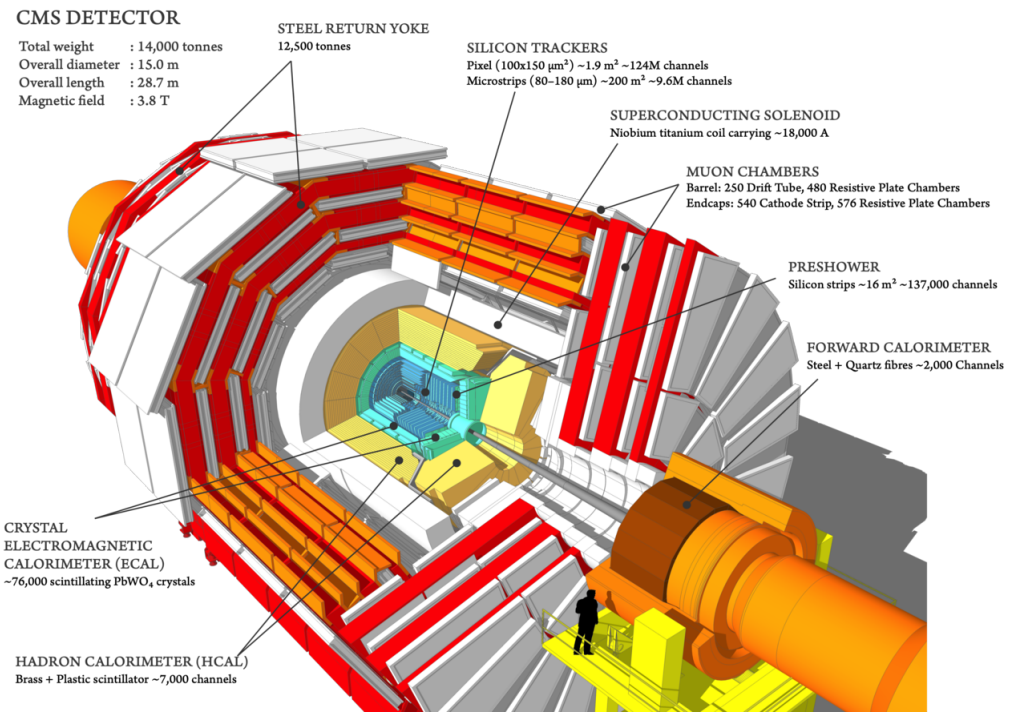


Figure 2.1: The CMS detector. Figure from [78].

momentum p_T and energy E_T are defined. By the assumption that particles initially only have momentum in the z-direction, one expects that the sum of the transverse momentum of the event should be zero. This makes missing transverse momentum and energy an important signal for new physics. A radial vector r can also be defined in this plane. The azimuthal angle ϕ is the angle between r and the x-axis, and the polar angle θ is the angle between r and the z-axis. The pseudorapidity η describes the angle between a particle and the beamline, and is given by $\eta \equiv -\ln \tan(\theta/2)$. Particles with a high pseudorapidity are close to the beamline and can be lost, escaping through spacing in the detector. A high pseudorapidity coverage is therefore a defining feature of a detector with a high transverse momentum and energy resolution. The coordinate system and angles are illustrated in Fig. 2.2.

2.1 Tracker system

Closest to the interaction point sits the tracker, a cylindrically symmetric detector consisting of around 1800 pixel and 15 000 strip modules [80], [81]. This yields total sensitive area that is approximately the same area as a tennis court. The tracker was designed to be radiation

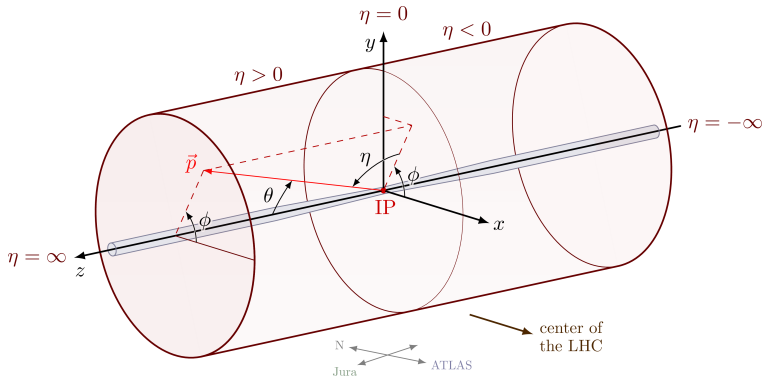


Figure 2.2: The CMS coordinate system. Figure from [79].

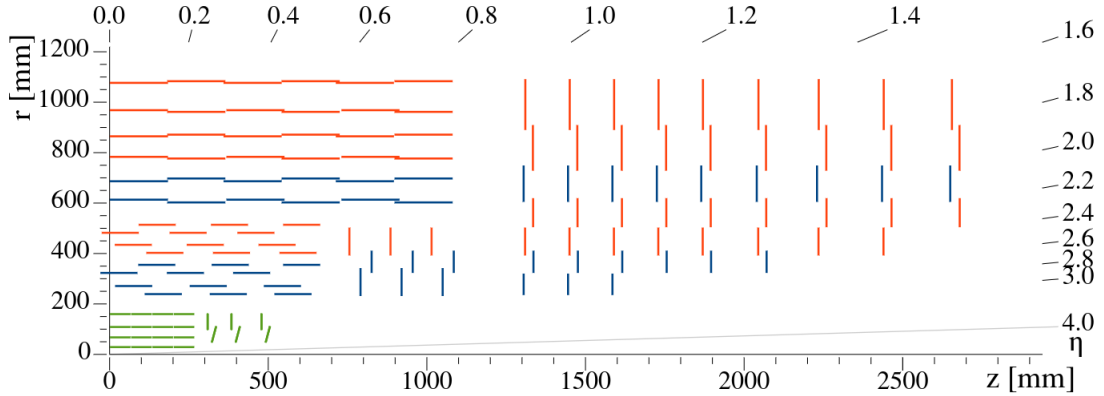


Figure 2.3: A quarter slice of the Phase-1 tracker. The pixel detector is shown in green. The strip detector has single and double-sided strip modules shown in red and blue, respectively. Figure from [81].

resistant, have an excellent position resolution with a fast response, while not introducing much material as this would result in too much multiple scattering, bremsstrahlung, photon conversion and other nuclear interactions. The solution was a tracker that is entirely made out of silicon. Silicon is relatively radiation resistant, only needs 3.6 eV to create an electron-hole pair, and has a low atomic number, causing minimal material interaction [82]. The magnetic field, $\boldsymbol{B}$, of CMS runs parallel to the beamline, so charged particles bend in the transverse plane with a radius, $\boldsymbol{R}$, proportional to their transverse momentum; non-relativistically $p_T \propto eBR$. A figure of the tracker is shown in Fig. 2.3.

2.1.1 Pixel detector

The pixel detector is the inner part of the tracker with 120 million pixels sized around 100×150 µm^2 distributed over four barrel layers and three endcap disks. This geometry is a result of an upgrade performed in 2016-2017 [83]. A smaller beam pipe was installed, which allowed adding a layer to the barrel and the endcaps. The other pixel layers were replaced to install thinner, smaller and more radiation resistant pixels. The readout bandwidth was increased by an order of magnitude by changing the output signal from analogue to digital. The inner pixel layer now sits only 3 cm from the interaction point, enabling the tracker to reconstruct the trajectories of very short-lived particles. The pixel detector is particularly important for primary vertex reconstruction, track seeding, and b and τ reconstruction since they often have displaced vertices and collimated decay products, respectively [84], [85]. The vertex resolution is around 10 µm and for high momentum tracks (100 GeV), the momentum resolution is 1-2% [4]. The pixels are made from n-on-n silicon diodes where silicon wafers have been enriched with oxygen, which has been proven to be very radiation resistant [82]. The pixels are bump bonded to readout electronics that amplify and buffer the signal. A bias voltage is applied to the pixels, creating a depletion region. When a particle traverses the pixels, it creates electron-hole pairs that are collected by electrodes. If a particle deposits charge in several pixels, known as charge sharing, one can reconstruct the position more accurately. The barrel is parallel to the magnetic field, so charge sharing happens naturally because of the angle of incidence of the particles. This is not the case for the forward disks, which are mounted in a turbine geometry at a slight angle to encourage charge sharing. The pixels provide 3-D position information with a resolution of O(10) µm [81].

2.1.2 Silicon strip detector

The pixel detector is surrounded by a silicon strip detector. The particle flux is lower further out from the interaction point, allowing the use of silicon strips instead of pixels. A lower granularity is cheaper, requires less cooling and leads to fewer material interactions. The detector consists of an inner and outer barrel, inner disks and endcaps. The barrel region has

ten layers with rectangular strips that run parallel to the beam axis. There are three inner disks and nine endcap disks with sensors of trapezoidal strips placed radially to the beamline. All the strips are p-in-n silicon, operating in a similar way to the pixels, with ionising particles creating electron-hole pairs. Several of the layers have modules where a strip detector module is mounted back-to-back at an angle with respect to a primary strip module. This allows a 3-D reconstruction of the hit position, whereas the other layers provide a 2-D hit position. The size of the silicon strips vary around 80-200 µm with a length of 10-20 cm [81]. Both the pixel and silicon strip tracker will see major changes in preparation for the HL-LHC. A more detailed description of this can be found in Section 2.8.4.

2.2 Electromagnetic calorimeter

The tracker is encompassed by the electromagnetic calorimeter (ECAL) [4]. The ECAL has a barrel and two endcaps made of 75 848 lead tungstate crystals that scintillate when electrons and photons pass through them [28]. This dense material stops particles quickly to produce fast and well-defined scintillation photons, resulting in excellent position resolution and a compact detector. The crystals are 22 (23) cm long with a 22 (30) $\times$ 22 (30) mm^2 face in the barrel (endcap), ensuring hermeticity and good granularity. The crystals have photodetectors at the back that convert and amplify light into an electrical signal. As photons and electrons interact with the crystals, they cascade into secondary particles, known as a shower. Neutral pions very dominantly decay into two photons, leaving a similar shower signature as photons. The two produced photons often seem like a single particle since they have a small angular separation. To help distinguish these scenarios, a preshower detector has been installed in front of the endcap crystals [86]. Lead absorbers initiate showers that are measured with silicon strip sensors. Neutral pions and photons leave different shower signatures here, making it possible to distinguish them. The preshower detector also helps distinguishing $e^{\pm}/\pi^{\pm}$, improves background rejection and helps estimate energy losses later in the ECAL. Electrons and photons are typically constructed in the $|\eta| < 2.5$ and jets in the $|\eta| < 3$ region.

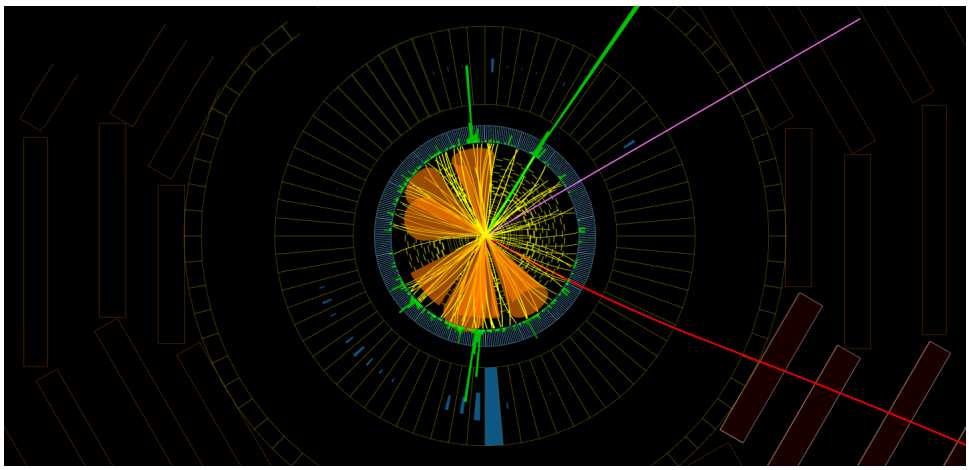


Figure 2.4: A Higgs candidate event. A Higgs boson and top quarks decay to seven jets (shown by the orange cones), an electron (green line), a muon (red line) and missing transverse energy (pink line). The energy deposited in the ECAL and HCAL is shown by the green and blue towers, respectively. Figure adapted from [88].

2.3 Hadronic calorimeter

Outside the ECAL sits the hadronic calorimeter HCAL [4]. It detects hadrons and is the only point in the detector that detects neutral hadrons. HCAL is divided into four sections; the barrel (HB), endcap (HE), outer (HO), and forward (HF) sections. These are sampling calorimeters that have alternating layers of a passive absorber and active detector layers. The absorbers are designed to strongly interact, causing processes like showers and bremsstrahlung that help stop particles. HB and HE use brass as an absorber and a plastic scintillator as the active material. Unlike the other layers, HO sits outside the CMS magnet, allowing it to have iron as the active material. It is designed to help measure particles that penetrate through the barrel. The forward calorimeter sits close to the beamline 6 metres downstream from the endcaps and receives the highest radiation of all the CMS subdetectors [87]. It uses steel absorbers and quartz fibers that generate Cherenkov radiation. It is designed to measure particles from very forward or highly energetic processes. In total, the hadronic calorimeter covers $|\eta| < 5$. Since missing energy is one of the key signatures of new physics, this large coverage and hermeticity is crucial. An illustration of an event passing through the calorimeters is shown in Fig. 2.4.

2.4 Muon detector

Unlike electrons, muons are usually not stopped in the ECAL owing to their heavier mass. They also have a long life time compared to e.g. τ particles. The signatures of new particles often include muons, and they have played an important role in the discoveries of the W, Z and Higgs boson as well as the top quark. Muons therefore need their own dedicated detection system [89]. The muon system has three types of detector where gas is ionised as a muon passes. The detector is arranged in a four-layered barrel and two four-layered endcaps. The muon barrel region is mostly contained in the steel yoke, and has a largely uniform magnetic field. This allows the use of drift tubes, containing a stretched wire in a gas volume. There are 250 drift tubes (DT) arranged both along and perpendicular to the beam line. In the endcap regions, the magnetic field is high and non-uniform, and muon rates and neutron-induced backgrounds are high. Neutrons can originate from showers or leaks in the forward shielding, and since they are long lived may interact with the detector material to create background signals. Cathode strip chambers (CSC) are used in the endcaps since they have a high background rejection rate and good timing and spatial resolution. The chambers have a crossed arrangement of anode wires and cathode strips resulting in a 2-D hit position. Both the drift tubes and cathode strips are complimented by resistive plate chambers (RPC). They have a lower position resolution than the other two detectors, but have a fast response time, aiding in triggering. In anticipation of the HL-LHC, they also add a level of redundancy to compensate for potential inefficiencies in a high luminosity environment. Information from the muon detector can be combined with information from the tracker, but all the muon sub-systems can trigger independently based on the p_T of the muons. CMS is also used to detect cosmic muons in times where there is no beam provided.

2.5 Level-1 Trigger

The first stage of the CMS trigger system is a hardware trigger that decides within 4 µs whether an event should be accepted or rejected [90]. Front end buffers store the full event until the

coarse grained detector readouts, known as trigger primitives, can be processed. Information from the calorimeters and muon detector are handled in separate steps before being combined into a global trigger. Calorimeter trigger primitives are formed by summing the deposited energy of local areas into trigger towers. In the barrel of the ECAL, energy is summed over a 5×5 crystal grid using front end electronics. This is sent via optical fibers to off-detector trigger cards. The same happens in the HCAL with towers of the same $\Delta\eta \times \Delta\phi$ size as the ECAL barrel. In the endcap and HF regions the layout of the crystals makes the geometry of the towers more complicated, and the size of the $\Delta\eta \times \Delta\phi$ window varies. The primitives are sent via optical links to off-detector electronics. The calorimeter towers are processed by letting high energy towers act as seeding points for clustering algorithms. Particle positions can be inferred from the energy-weighted cluster position. This yields e/γ and τ candidates as well as jets and energy sums. Trigger primitives from the muon detectors provide hit coordinates, timing and quality information. Muon tracks are reconstructed in three separate pseudorapidity regions by combining information from the three subdetectors. Some of the tracking is done with simple methods like straight line extrapolation and segment matching. In the transition region between the barrel and endcap, the geometry is complex, and a novel algorithm has been developed that performs tracking and p_T measurements in one step [91]. The muon trigger sorts and ranks the candidates, then sends the best ones to the global trigger. The global trigger has a menu that can select for particular physics interests, such as W decays, Higgs decays or supersymmetry signatures. The workload is divided across several FPGAs and merged before sending the quantities to the HLT.

2.6 High Level Trigger

When an acceptance signal is sent from the L1T, it triggers a read out of the full detector. The High Level Trigger (HLT) unpacks the raw detector data and combines it into reconstructed jets and identified particles. This algorithm is called particle flow [92], and a simplified version is implemented in the L1T. The HLT menu consists of paths that target reconstruction of final state objects, such as a single muon or double tau. Events have to pass at least one of the

paths in order to be saved to permanent storage. For the current Run 3 data taking, the HLT menu consists of over 800 paths [16]. The HLT menu is therefore modular and changes with physics interests. Each of the paths start with a seed from the L1T. Events are reconstructed in order of increasing complexity so that events can be rejected early in the pipeline. Candidates from the calorimeter and muon chambers are therefore reconstructed and filtered first. If the event passes, the relevant region of the tracker is reconstructed, and finally full detector reconstruction is considered. Within each of these steps, there are modules that perform partial reconstructions, known as Producers. These are also run in order of increasing complexity and followed by Filter modules that can reject the candidates. The result from a module can be cached if it is needed by another path to avoid duplicating computations. One of the core principles of the HLT is that the algorithm is a simplified version of the offline reconstruction software. This saves development time and allows for a consistent reconstruction system. The framework for offline reconstruction and the HLT is known as the CMS software (CMSSW) and is mostly implemented in C++ [93]. Due to the increased complexity of the algorithms compared to the L1T, the HLT is implemented on more than 30 000 CPU cores [94]. After HLT triggers on an event, the event is sent to local storage, which is later sent to a computing centre for offline processing and data storage. Chapter 3 describes the HLT and the paths in more detail. The HLT has to adhere to a maximum trigger latency of 175 ms [95].

2.7 Track reconstruction

Reconstructing the paths of charged particles from hits as they pass through the tracker is known as tracking. Since this is a core topic of the thesis, it will be considered in some detail. CMS uses a Combinatorial Track Finder (CTF) that iteratively creates track candidates [80]. As tracks are reconstructed, their hits are removed from the pool of available hits. High p_T tracks that are close to the interaction region are reconstructed first. Their hits are removed, which harder to reconstruct tracks would otherwise have to consider. The algorithm takes place in four steps; seed generation, track finding, track fitting and track selection. Before tracking starts, one first has to reconstruct the hit positions from the tracker.

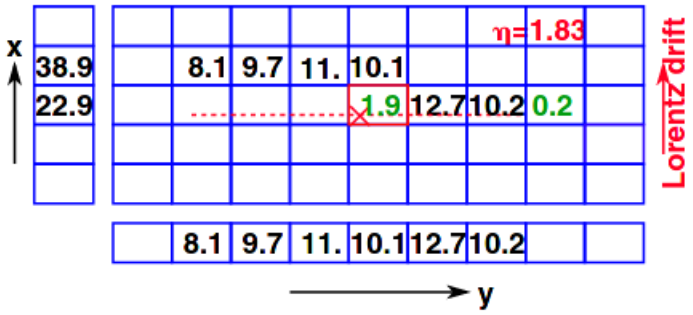


Figure 2.5: A particle traverses in the direction of the dotted red line and deposits charge in several pixels. The numbers represent the charge in 1000s of electron equivalent charges. The green numbers are below the readout threshold and the red cross indicates the true hit position. Figure from [96].

2.7.1 Pixel and silicon hit reconstruction

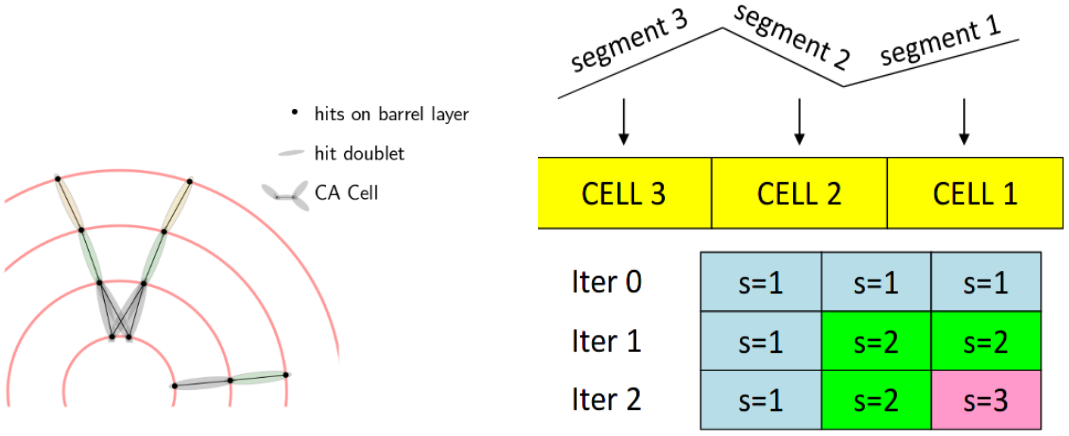
Because of their angle of incidence, particles usually deposit charge in several adjacent pixels or strips. Before tracking finding can begin, one must reconstruct these charge clusters into an estimate of where the particle intersected with the layer, known as a hit. Signal sizes over an adjustable threshold are read out from the pixel and silicon detectors. If only one pixel has been hit, the centre of the pixel is read as the hit position. If many pixels have been hit, the position must be estimated from adjacent pixels, as illustrated in Fig. 2.5. In the pixel detector, a fast algorithm that considers the edges of the cluster is used for seeding, and a slower, template based algorithm is used for the final track fit. The strip detector uses a charge-weighted averaging approach while adjusting for the thickness of the silicon sensors. The average efficiency for hit reconstruction is $> 99\%$ [80]. The resolution of the position is reduced by multiple scattering, and depends on the size of the cluster and the position and angle of the sensor, but is on the order of µm. The hit reconstruction also gives an estimate of the uncertainty of the hit positions.

2.7.2 Seed generation

The seeding process gives an initial estimate of the five parameters necessary to describe the helical path of a track. This is done by connecting hits in nearby layers primarily in the pixel detector. Although the outer tracker has a lower track density, the pixel detector measures a

3-D position, the multiple scattering effects are lower, and the high granularity results in a lower channel occupancy. Most of the tracks leave at least three hits within the pixel detector, and seeds can form from these [80]. For other tracks, the pixel hits must be combined with hits from the silicon strips, knowledge about the primary vertex or a combination of strip pairs and a beam spot constraint. Most algorithms start with high p_T tracks close to the beamline that are easy to reconstruct and continue with harder tracks. A global tracking region is defined based on what is compatible with the beam spot. Hit pairs and triplets are found by extrapolating the z and ϕ values of one or two hits to subsequent layers.

Thanks to the tracker upgrade described in Section 2.1, the pixel barrel has four layers, allowing hit quadruplets to be found. This is done by cellular automaton [97]. First, a graph of all possible two layer connections is created. Filters based on the curvature and the compatibility with the beam spot can be applied. The two layer doublets form the cells in the cellular automaton. All cells initially have a state of 1. If a cell has an inner neighbour with the same state, its state is incremented by one. This is repeated until there are no more neighbouring cells with equal states. This is illustrated in Fig. 2.6. A track candidate is made using a backward pass. Starting with the highest state cell, the neighbours with incrementally lower states are added to the segment. If there are several candidates, the one with the smallest angle is chosen. Since all the operations are local, the algorithm can be easily parallelised. This approach has also been explored for track building, but is generally not used in favour of the Kalman filter (see Section 2.7.3), since it typically requires an exhaustive search for track segments. The trajectory parameters can be estimated from the seeds using conformal mapping. A description of this can be found in [98]. Extending to a fourth layer to form quadruplets would be slow since it creates an extra step that can not be parallelised, and unless the algorithm is changed, doublets and triplets from the same subset of layers would be evaluated multiple times.



(a) Doublets from hits (b) Cellular automaton algorithm

Figure 2.6: A simple cellular automaton scenario. Cells are formed from hits in adjacent layers. All cells are initialised with a state of one. Each iteration sees the state increase by one if the neighbour that is closer to the beamline has the same state. Figure (a) is from [99] and (b) from [100].

2.7.3 Track building

A track is built for each of the seeds. Based on the track direction and position, the next potential layers the track can intersect with are identified. If there is more than one compatible layer, the track splits into different candidate tracks. The track helix and its errors are propagated to the given layer assuming a perfect helix. Detector modules that fall within a window of compatibility with the propagated track are found. Many of CMS's tracker modules slightly overlap, so a track can leave more than one hit in a layer. In CMSSW, modules are stored in groups where a track can not cross any more than one module per group. Modules at equal radii are for instance in the same group because it is impossible for a track to cross more than one of them at a time. The hits in the module groups are collected to form corresponding hit groups. It is possible to add a *ghost hit* to the groups, accounting for the possibility that a track crossed a module without leaving a hit due to detector inefficiencies. One hit per group is added to the track. It is possible to add just the best hit from the group, or new candidate tracks can spawn from the potential combinations of hits from each group. If there are e.g. two groups with two hits each, that means there could be one candidate with two hits, or four candidates each with two hits. To avoid a large number of candidates, only the best n - by default three to five - candidates are propagated from the same seed at any time. The quality

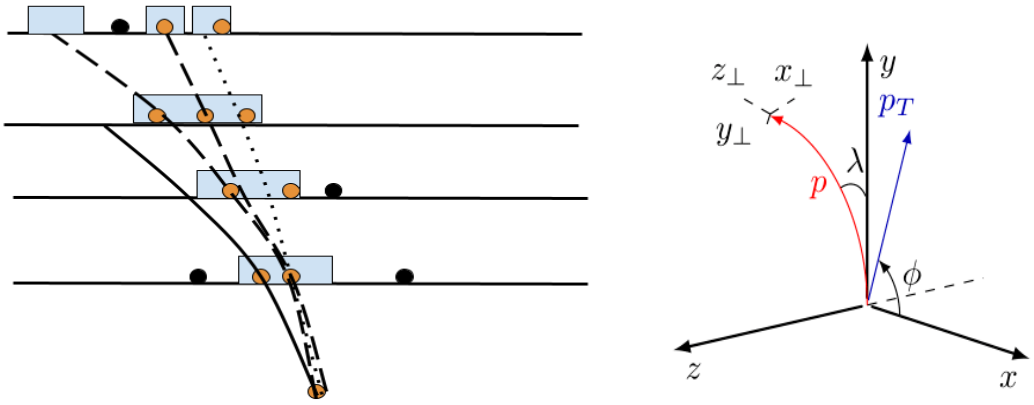
of the track is determined by the χ^2, the number of hits and the number of ghosts hits. When a track candidate has five hits, an inward search for hits is initiated. A track seed is made from the outer hits, and tracking is done just as described in this paragraph, going from the outside in. This can find additional hits in the seeding layers or other layers closer to the interaction region. Strip hits that are not matched with a pixel hit are not available in the seeding step, and these can now be added to the track.

The track building stops when the track reaches the end of the tracker, the p_T is too low, or it contains too many missing hits. In the HLT, a track is discarded if it contains too many missing hits, the p_T drops under a set threshold, or the χ^2 for the track is too high. An illustration of the tracking algorithm is shown in Fig. 2.7. The main challenge of track building is the combinatorics introduced when a track splits into different candidates. This is especially a challenge for the HL-LHC. The tracking algorithm used at the HLT and the effect of the combinatorics is explored in more detail in Chapter 3. The algorithm used for updating the track with each of the compatible hits is the Kalman filter method, described in the next section.

The combinatorial Kalman filter

Kalman filtering is a linear estimation technique from the 1960s, commonly used in target tracking [101]. It combines estimates of positions and uncertainties from observations and models, making it ideal for physics uses. It is the default method of track building both in the HLT and offline. It is sequential, so hits can be removed as tracks are reconstructed. It is robust to uncertainties and allows for small matrix calculations, making it a comparatively quick algorithm.

Five parameters are necessary to describe the helix path. For easy error propagation, CMS uses a coordinate system that moves a long the track in a curvilinear frame. The helix can be described by $x = (\frac{q}{p}, \lambda, \phi, x_\perp, y_\perp)$ where q/p is the charge of the particle divided by the momentum, λ is the dip angle, ϕ is the azimuthal angle and $x_\perp$ and $y_\perp$ are defined in a coordinate system local to the track. This is illustrated in Fig. 2.7.



(a) Assigning hits to track candidates

(b) Coordinate system used for particle tracking

Figure 2.7: Fig. (a) illustrates the track building algorithm. The blue squares are windows of compatibility and the circles represent hits. The track shown by the black line will stop propagation since it has too many missing hits. The long dashed lines show how several compatible hits lead to new track candidates. The short dashed line shows how the propagated helix can fit somewhat poorly with the compatible hits. Fig. (b) shows the coordinate system used for tracking. $z_\perp$ is defined along the direction of the track. $x_\perp$ is in the global $x-y$ plane, and $y_\perp$ is the coordinate that is orthogonal to these two. ϕ is the azimuthal angle, and λ is the dip angle. In Fig. 2.2, θ is also shown, and $\lambda = \frac{\pi}{2} - \theta$.

The Kalman filter algorithm propagates a state, looks for compatible measurements and combines these into an updated state. In the context of tracking, the first step is to propagate the helix to a given layer. More information on the helix propagation equations can be found in [102]. The predicted state on layer k is given by a projection matrix $\boldsymbol{F}$ applied to the state on the current layer $\boldsymbol{x}_{k-1}$;

$$\hat{\boldsymbol{x}}_k = \boldsymbol{F}_{k-1}\boldsymbol{x}_{k-1}, \tag{2.1}$$

and since $Cov(\boldsymbol{A}x) = \boldsymbol{A}Cov(x)\boldsymbol{A}^{\boldsymbol{T}}$ it follows that the covariance matrix $\widehat{\boldsymbol{C}}$ of the predicted state is

$$\widehat{\boldsymbol{C}}_k = \boldsymbol{F}_{k-1}\boldsymbol{C}_{k-1}\boldsymbol{F}_{k-1}^T + \boldsymbol{Q}_{k-1}, \tag{2.2}$$

where $\boldsymbol{Q}$ accounts for multiple scattering. We can map the projected state to where we expect

to find detector hits with the matrix $\boldsymbol{H}$, giving an expected measurement

$$\boldsymbol{z}_k = \boldsymbol{H}_k \hat{\boldsymbol{x}}_k + \boldsymbol{v}_k \tag{2.3}$$

where $\boldsymbol{v}_k$ is a measure of noise. In the current CMSSW implementation, the translation from the local coordinate system to a compatible window on the detector is handled outside of the Kalman module, so $\boldsymbol{H}$ is set to an identity matrix with negligible noise. By the properties of covariances mentioned, the covariance of $\boldsymbol{z}_k$ is $\boldsymbol{H}_k \boldsymbol{C}_k \boldsymbol{H}_k^T$. The expected measurement can be combined with the actual hit measurement by considering them Gaussian distributions. The hit measurement distribution has a mean given by the hit position $\boldsymbol{m}_k$ and a corresponding error $\boldsymbol{V}_k$. It can be shown that combining two Gaussian distributions with means $\boldsymbol{\mu}$ and covariance matrices $\boldsymbol{\Sigma}$ gives;

$$\boldsymbol{\mu} = \boldsymbol{\mu}_0 + \boldsymbol{K}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0), \quad \boldsymbol{\Sigma} = \boldsymbol{\Sigma}_0 - \boldsymbol{K}\boldsymbol{\Sigma}_0 \tag{2.4}$$

where $\boldsymbol{K}$ is the Kalman gain

$$\boldsymbol{K} = \boldsymbol{\Sigma}_0(\boldsymbol{\Sigma}_0 + \boldsymbol{\Sigma}_1)^{-1}. \tag{2.5}$$

We can apply this to the expected and actual hit measurements. To simplify the equations, we drop a $\boldsymbol{H}$ term from the Kalman gain and get

$$\boldsymbol{K}_k = \frac{\widehat{\boldsymbol{C}}_k \boldsymbol{H}_k^T}{\boldsymbol{H}_k \widehat{\boldsymbol{C}}_k^T + \boldsymbol{V}_k}. \tag{2.6}$$

The mean and uncertainty of the new, combined distribution becomes

$$\boldsymbol{\mu} = \boldsymbol{x}_k = \hat{\boldsymbol{x}}_k + \boldsymbol{K}(\boldsymbol{m}_k - \boldsymbol{H}_k \boldsymbol{x}_k), \quad \boldsymbol{\Sigma} = \boldsymbol{C}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k)\hat{\boldsymbol{C}}_k. \tag{2.7}$$

Since we have the error, the χ^2 can also be calculated. Details of these derivations can be found in [103], and the main idea of combining two distributions is illustrated in Fig. 2.8. Since there are five parameters, the matrices will take dimensions of 5×5, 5×2 etc. The Kalman filter is called many times in the HLT, including for track building, track fitting and primary vertex finding.

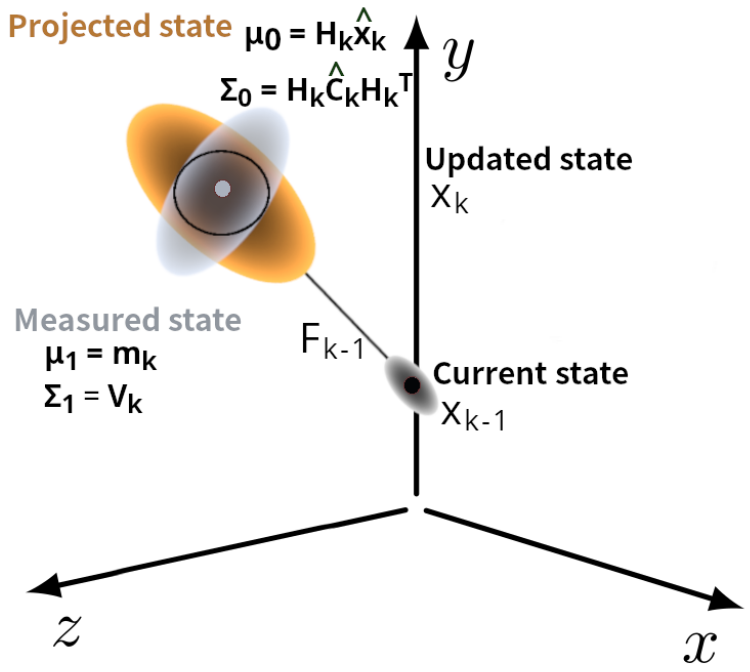


Figure 2.8: One iteration of the Kalman Filter. The current state is propagated to a projected state. A compatible hit is found within this, shown as a small blue-grey circle. The Gaussian distributions of the projected and measured state are combined into an updated state, shown by the black circle. This is repeated iteratively to build the track.

2.7.4 Track fitting and selection

After matching hits to tracks, the track parameters are recalculated using two iterations of Kalman filters, known as fitting as smoothing. The fitter starts from the initial seed and goes outwards, and the smoother goes from the outer hits inwards. The smoother is initialised with the track parameters found by the track fitter. The initial covariance matrices are scaled up by a large factor to reduce potential bias. The final track parameters are given by the weighted average of the parameters found by the two filters. The helix propagation used here is more accurate than the one used for track building. The Runge-Kutta method [104] - a common technique for first-order differential equations - takes into account that the magnetic field is not uniform. After the filtering and smoothing, the track is checked for outliers by considering the hits' χ^2 values. If an outlier is found, the filtering and smoothing is repeated.

Since a track can split into several candidates during track building, many of the reconstructed tracks are not tracks associated with a charged particle. To remove such fake tracks, a track

selection algorithm is applied. Selections can be made based on the maximum allowed χ^2, minimum number of hits, minimum number of layers intersected and compatibility with the beam spot. The z-distance from the closest point to the primary vertex is also considered.

2.7.5 Tracking performance

This thesis will focus on tracking methods alternative to the CTF that meet the requirements of the Phase-2 environment, as described in Section 2.8. The Phase-2 implementation of the CTF algorithm should therefore be used as a benchmark to gauge the necessary performance. The timing performance is discussed in detail in Chapter 3. The physics performance is often measured with either single particles like muons or electrons, or proton-proton events. The latter will be considered here since it is more relevant for high pileup scenarios. Testing performance requires access to labelled data, i.e. a ground truth, so simulated events are used. Events with top quark pair production ($t\bar{t}$) acting as a signal are commonly used as a standard candle since they are representative of a pp event, and represent a well understood and modelled process. After running the tracking algorithm, reconstructed tracks are matched with simulated tracks if at least 75% of the hits in the reconstructed track originate from a simulated particle. The performance is measured by an efficiency and fake rate. The tracking efficiency is the fraction of charged particles associated to at least one reconstructed track. The fake rate is the rate of tracks not associated with a simulated particle. The efficiency and fake rate for simulated $t\bar{t}$ events is shown in Fig. 2.9 and in Fig. 2.10.

Fig. 2.9 shows that for the standard CKF tracking algorithm, the tracking efficiency is around 90% for tracks with an energy between 1-100 GeV with a fake rate around 5-6%. It also shows several interesting physics effects. At high energies the efficiency drops. This is caused by highly collimated jets - i.e. a narrow cone of particles - that produce particles closer together than the positional uncertainty of the trajectories. The tracking efficiency is also low for low momentum particles because they spiral in the magnetic field, and are more affected by multiple scattering than higher energy particles. The fake rate similarly has peaks at low and high p_T. Another effect in the high pileup environment not directly shown in the plots is that hits in

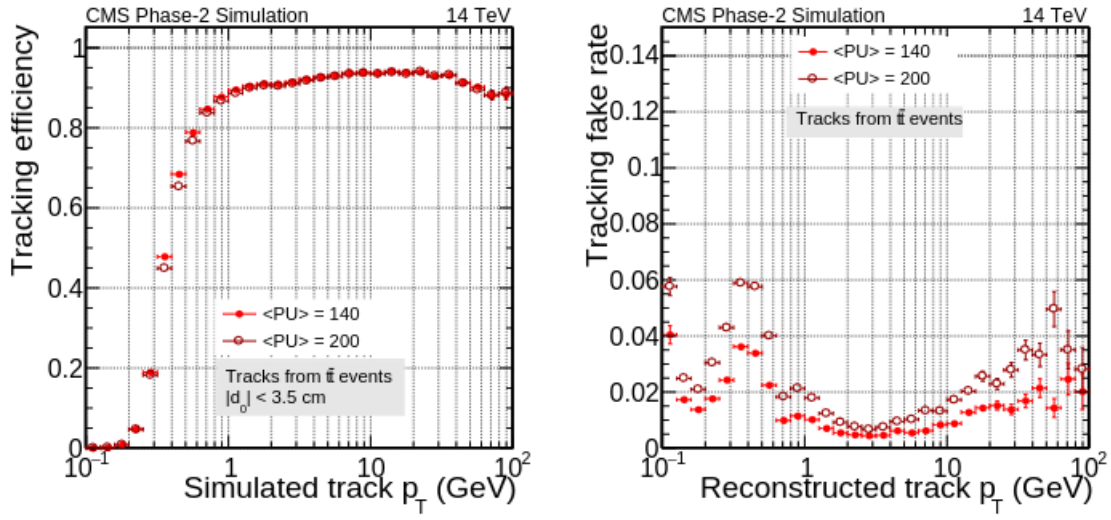


Figure 2.9: Tracking efficiency and fake rate of the Phase-2 tracking algorithm using simulated $t\bar{t}$ events. A Poisson distribution of the number of events with mean 140 or 200 is added to the event. Figure from [81].

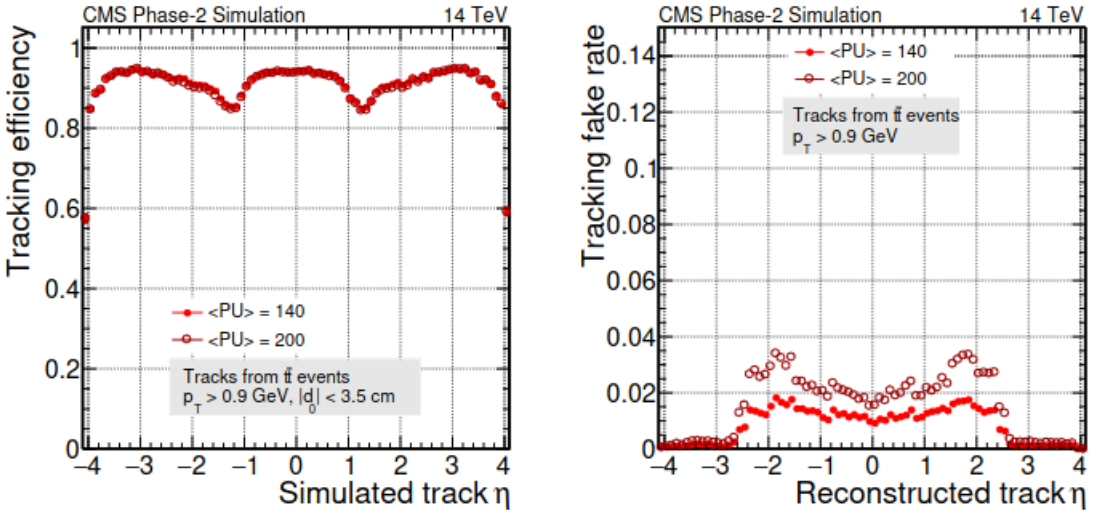


Figure 2.10: Tracking efficiency and fake rate as a function of pseudorapidity. Figure from [81].

the inner parts of the detector can be very close to each other, and could even appear as one hit. Fig. 2.10 shows the tracking efficiency and fake rate as a function of the position in the detector. Particles close to the beamline are difficult to reconstruct, as shown by the decrease in efficiency at a high absolute value of η. The are also two dips in the efficiency, or equivalently two increases in the fake rate. These represent the transition regions between the barrel and endcap, where there the geometry makes reconstruction challenging.

2.8 The Phase-2 CMS detector

The HL-LHC environment will require CMS to undergo upgrades to reach a better resolution, higher radiation tolerance, increased data bandwidth and improved trigger capabilities [28]. Some of the main changes will be an entirely new tracker and calorimeter endcaps, the ability to perform tracking in the L1T, extended detector coverage and a new timing detector sat between the tracker and the ECAL. The layout of the Phase-2 CMS detector is shown in Fig. 2.11. Each detector upgrade is motivated by physics targets, as illustrated in Fig. 2.12.

2.8.1 Phase-2 calorimeters

For the barrel calorimeters, the existing detector materials have been shown to be able to cope well with HL-LHC conditions [105]. To keep up with the required trigger rates, the backend electronics will be upgraded. In the ECAL barrel, the frontend electronics will be upgraded to read out from every crystal instead of 5×5 grids. This lets showers be matched to tracks at the L1T. The granularity will remain the same as in Phase-1 for HCAL. The ECAL and HCAL endcaps would receive too much radiation damage to operate at an acceptable level at HL-LHC, and will be replaced with a new High Granularity Calorimeter (HGCAL) [87]. Unusually for calorimeters, the HGCAL will provide 3-D positional measurements. It will consist of 50 longitudinal layers with a total of 6.5 million channels. The electromagnetic and large parts of the hadronic part of HGCAL will consist of silicon cells of size around 0.5-1 cm^2. The remaining hadronic part will have plastic scintillators of size 4-30 cm^2. The sensors are interleaved with

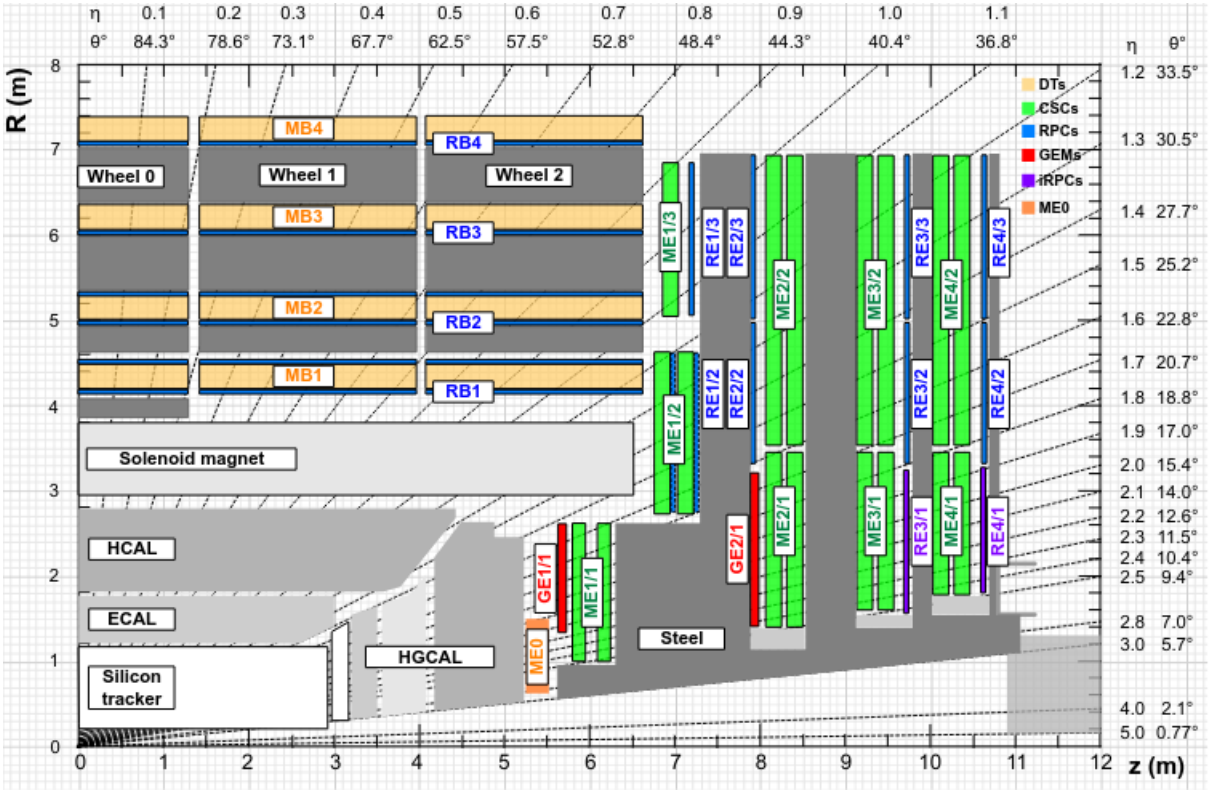


Figure 2.11: Layout of the Phase-2 CMS detector. The detectors shown in red, purple and dark orange are new components of the muon detector. The ME0 detector component is also a GEM detector. Figure from [89].

Performance/ Physics	Higgs VBF H→ττ	Higgs H→μμ	Higgs H→ZZ→4l	Higgs HH→bbγγ	Higgs HH→bbττ	SMP VBS	SUSY VH(bb) +MET	EXO A_{fb} (Z')	EXO Dark Matter	EXO HCP	BPH $B_{s,d}$→μμ
Tracker											
Performance		mass resolution	mass resolution	b-tagging	b-tagging						mass resolution
Extensions	forward jets / MET		acceptance		MET resolution	forward jets	MET resolution	acceptance	acceptance		
Trigger											
Bandwidth	acceptance				acceptance						
Track Trigger	background rejection				background rejection						background rejection
Calorimeter											
ECAL	forward jets / MET		acceptance	acceptance	MET resolution	forward jets	MET resolution	acceptance	acceptance		
HCAL	forward jets / MET				MET resolution	forward jets	MET resolution				
Muons											
Extension			acceptance					acceptance	acceptance		

Figure 2.12: The link between the CMS detector upgrades and physics goals. MET refers to missing E_T. Several of the physics processes are explained in Section 1.2.2. Figure from [28].

absorber materials. The high granularity is especially beneficial for separating jets from each other, and can help with pileup subtraction. To reduce irradiation effects and allow the use of silicon photomultipliers the HGCAL will be kept at -30 °C.

2.8.2 Phase-2 muon detector

Large parts of the muon detector will remain the same as in Phase-1 with an update of the electronics [89]. The RPC links will be upgraded, increasing the timing resolution by an order of magnitude. There will be new RPC detectors in the forward region, extending the acceptance of the RPCs from $|\eta|$ =1.9 to 2.4. When combined with the CSCs, this improves the time resolution by a factor of two and increases the position resolution, helping suppress low p_T tracks. New gas electron multipliers (GEMs) have already started to be installed. GEM detectors have a high efficiency while the thin profile allows them to be installed within the space constraints of the existing detector. They will be installed in the regions with the largest bending angels, helping momentum reconstruction. The GEM detectors extend the length of the measured particle paths, on average increasing the number of muon hits from six to eight in this region. This helps reconstruct more precise tracks and reject mis-measured tracks. An illustration of the Phase-2 CMS detector with the muon regions highlighted is shown in Fig. 2.11.

2.8.3 Minimum ionising particle timing detector

The new Minimum ionising particle Timing Detector (MTD) will provide time of flight information between the collision vertex and outside the outer tracker at 30-60 ps resolution [106]. This will help assign tracks to the correct vertices, exploiting the fact that interactions within one bunch crossing are generally not completely simultaneous. The MTD detector will enable the performance of the particle-flow algorithm to have a performance comparable with Phase-1. It will reduce the amount of reconstructed tracks associated with the incorrect particle by a factor of two. The removal of pileup tracks improves the identification efficiency of several final state observables. Time of flight information also helps distinguish low momentum charged

hadrons which is especially relevant for heavy ion collisions. It is also very useful for identifying long-lived particles, as described in Section 1.2.2. It is estimated that this detector will provide a reduction of 20-30% of the background across many signals [106].

2.8.4 Phase-2 Tracker

CMS will need a completely new tracker, as the current one would experience significant performance degrading in the high radiation environment of the HL-LHC [81]. The new tracker needs to maintain a channel occupancy near or below 1% and have a high radiation tolerance. It needs extended coverage in the forward region while reducing material in the tracking volume. The resulting detector is illustrated in Fig. 2.13. The pixel detector will have pixels that are 6 times smaller than the current ones. Smaller pixels are more resistant to radiation damage and provide better resolution. As shown is Section 2.7.5, the current tracker struggles with separating physically close particles from high-energy jets, and the higher resolution will mitigate this. The pixel tracker will extend the forward region by going from three to ten forward disks. This will almost double the active surface of the pixel detector compared to Phase-1. It has been shown that this will be a great benefit to the HL-LHC physics programme, for instance for vector-boson scattering events that are typically close to the beamline.

The outer tracker will have six barrel layers and five endcap disks on each side. This choice was made to reduce the amount of material in the tracker while still meeting physics requirements. The outer tracker will handle a data reduction of roughly an order of magnitude with novel p_T modules that will reject signals from particles below a set transverse momentum threshold at the L1T. It is not possible or necessary to read out all the tracker hits for Phase-2 since most of the hits will come from low-energy pileup tracks. Bandwidth and latency requirements are met by filtering these hits out before track reconstruction. The p_T modules measure transverse momentum with silicon modules spaced a few mm from each other. For each hit, a window is opened in the outer layer. If a hit is found within this window, a stub is created. The strong magnetic field of CMS allows an estimate of the transverse momentum by the angle of the stub. This is illustrated in Fig. 2.14. There will be two types of p_T modules, pixel-strip (PS) and

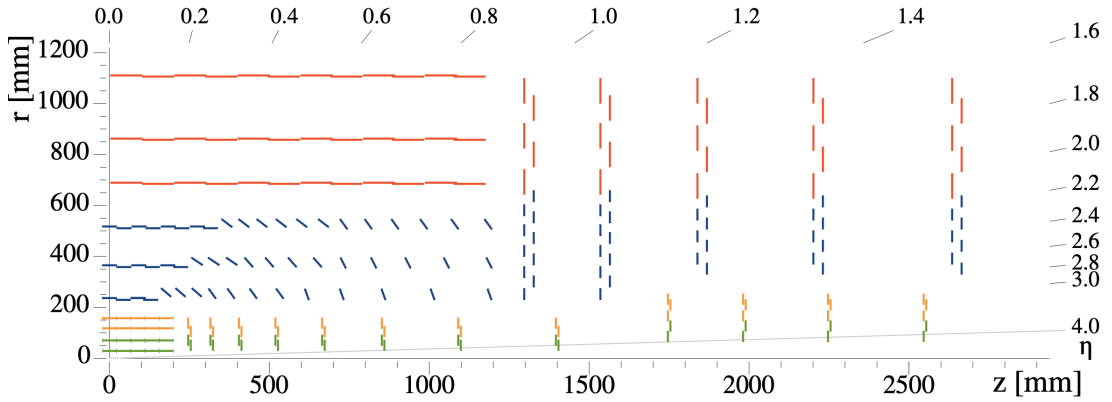


Figure 2.13: A slice of the Phase-2 tracker. The green lines represent pixel modules with two readout chips, and the orange lines have four readout chips. This forms the inner tracker. The outer tracker consists of pixel to strip modules, shown in blue, and strip to strip modules, shown in red. Figure from [107].

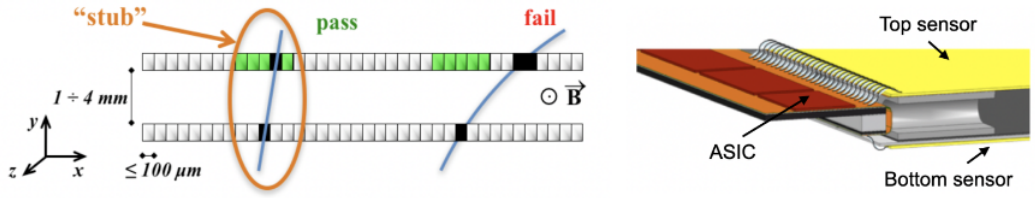


Figure 2.14: The track stub finding (left) and a p_T module (right). A low momentum track fails and does not produce a stub, while a high momentum track passes. The stubs are processed on ASICs. Figure from [107].

strip-strip (2S). The PS modules have a 1.5 mm × 100 µm pixel sensor mounted underneath a 2 cm × 100 µm silicon strip sensor. The pixel sensor provides a fine granularity z-position measurement, which is necessary in the inner part of the outer tracker. To encourage particles to cross both the bottom and top parts of the sensors, a tilted geometry is used. The 2S modules have two 50 cm long strip sensors with a size of 100 µm in the transverse plane mounted parallel to each other. This has a lower z-resolution, but provides sufficient momentum measurement. The processing will be done on front end ASICs and passed via optical fibres to the L1T.

2.8.5 Phase-2 L1 Trigger

The main changes for the L1T is input from the new HGCAL, and due the new p_T modules, tracking can be done at L1T for the first time. To allow for the increased algorithmic complexity,

the trigger latency will increase to 12.5 µs [90]. To reduce the increased data volume, a large part of the processing will be done by backend electronics. Tracking will be done at the backend and the track parameters are sent to the L1T as trigger primitives. A track trigger will reconstruct the primary vertices and trigger on tracker objects such as jets or missing transverse momentum. The tracks can also be propagated to other parts of the detector as needed. The barrel calorimeter and muon triggers will largely remain similar to their Phase-1 implementations. A notable exception is that the muon trigger will also match muon stubs to tracks from the track finder. The availability of tracks allows the reconstruction of physics objects and particle identification in a way previously only seen offline and in the HLT. The L1T will take advantage of the common reconstruction method particle flow, which is described in more detail in [92]. The possibility of including a pileup mitigation algorithm, described in [108], has also been explored. The particle flow reconstruction takes place in a correlation trigger and the output is then passed onto a global trigger which also receives information from the muon, calorimeter and track triggers. This part of the algorithm implements the final trigger menu. It also acts as an interface to the separate triggers that monitor the beam (PPS, BPTX and BRIL). An illustration of the new trigger layout is shown in Fig. 2.15.

2.8.6 Phase-2 HLT

The HLT will receive data at a rate 5-7.5 times higher than it did in Phase-1 [109]. It is estimated that if the current HLT code does not change, computing power requirements will be 24 times larger for a average pileup of 140 scenario. Though there is no set latency for the HLT yet, it needs to accommodate an acceptance rate up to 7.5 kHz. It would be both expensive and undesirable to address this merely by an increase in the number of CPUs. Instead, heterogeneous computing, parallel computing and new algorithms are being explored. GPUs are already in use, and using them more is strongly considered as they offer computational acceleration at a lower cost and developer threshold than e.g. FPGAs. The design of the trigger remains flexible and open to innovation. The underlying principles of an HLT with trigger paths of increasing complexity and an algorithm similar to the offline one are likely to

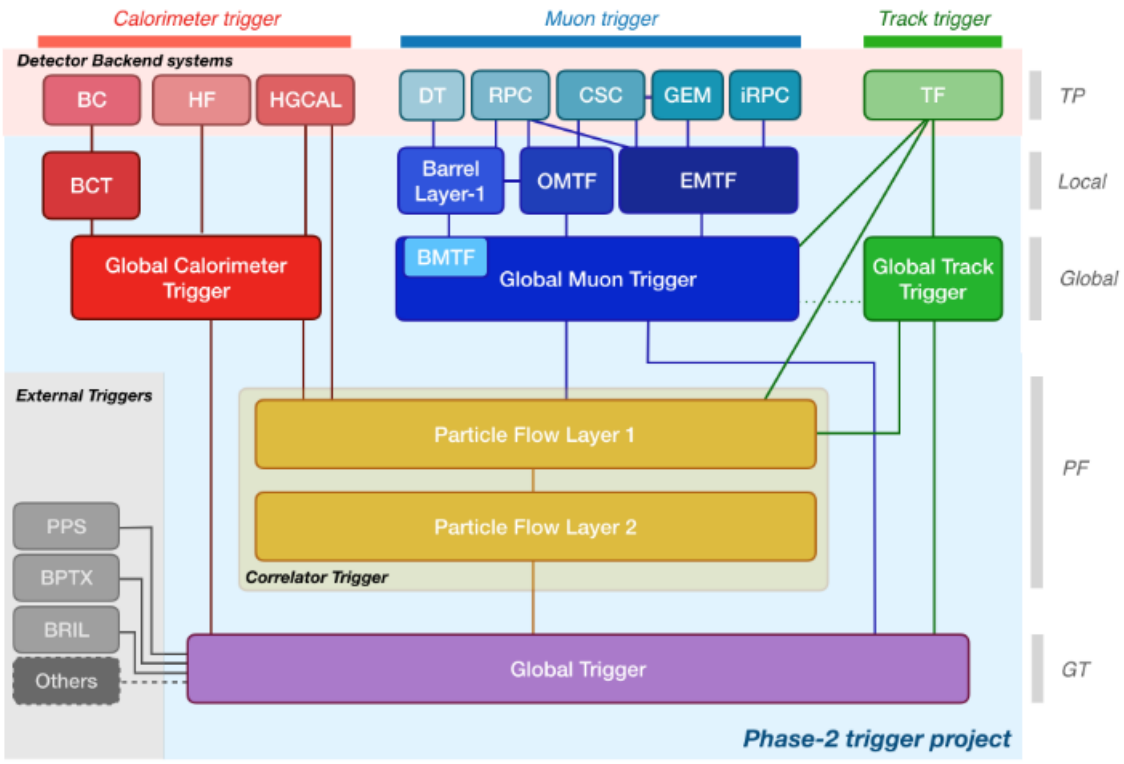


Figure 2.15: The Phase-2 level one trigger. The trigger has a muon and calorimeter trigger as in Phase-1, as well as new track trigger. Figure from [90].

remain. One of the biggest changes for the Phase-2 HLT is that it will need to accommodate information from the HGCAL and that the tracker is new and more complex. The tracking and HGCAL reconstruction are also the most time consuming paths of the HLT. Innovation for these algorithms is ongoing, and is discussed in further detail in Chapter 3.

Chapter 3

Computational performance of the Phase 2 High Level Trigger

As discussed in Section 2.8.6, the HLT must undergo changes in preparation for Phase 2. To understand what changes are necessary, it is important to study the computational performance of the trigger with the geometry and at the pileup levels expected for Phase 2. This chapter will discuss the parts of the trigger that scale badly with higher pileup and potential strategies to address it.

3.1 Measuring computational performance

Before discussing the computational performance of the HLT, it is worth outlining a strategy and some common tools. When creating a performance model, it is firstly important that the benchmark is representative, repeatable, and verifiable. The latter two are ensured by having a code repository with reproducible steps, available at [110]. The code is also representative since the selected HLT menu and input data follow the standard set by the CMS Trigger study group. When measuring computational performance it is also useful to be mindful of the characteristics that would make the code suitable for acceleration. The two main methods for measuring computational performance are to time the code and to use hardware counters.

Some of the useful metrics are;

- **Hotspots** - Timing functions or lines of code to identify what takes up a large portion of the run-time can often reveal prime candidates for acceleration. The potential acceleration achievable can be calculated using Amdahl's law, shown in Equation 3.2.
- **Memory structure** - The memory structure of an algorithm can reveal how resources are being used. Perhaps the algorithm isn't spending a lot of time on computations, but rather on e.g. moving data or waiting for instructions. Branching and cache mispredictions can also be revealing.
- **Loop structure** - The loop structure of a program is important for the performance. Large performance gains could be achieved by parallelising or restructuring the loops. A loop flow diagram is sometimes useful to study the number of loop entries and the information flow. If one is considering using a co-processor, it is important to understand loop dependencies and how the loops could be implemented and potentially unrolled; inner loops can e.g. be good candidates to accelerate since they are often called with a high frequency.
- **Function calls, function hierarchy and dependencies** - When considering potential co-processors, it is important to understand the data flow and the I/O that would be necessary.

These metrics can help determine whether code is suitable for co-processing. As explained in Section 1.4.2, code that is suitable for GPUs often has large loops or nested loops that can be parallelised. From Section 1.4.3, code suitable for FPGAs typically has one or several hotspots with relatively few lines of code. Branch misses and cache mispredictions are also indicators of good FPGA candidates. Accelerating on CPU is also possible with code optimisations and parallelisation. When a hotspot has been identified, the potential speedup can be calculated with Amdahl's law [111]. If the hotspot takes up a proportion p of the execution time, the initial execution time $T_{intitial}$ is the sum of the time spent inside and outside the hotspot. Given

that the hotspot is accelerated with a speedup s, one gets a new time estimate T_{new};

$$T_{intial} = (1-p)T_{intial} + pT_{initial}, \quad T_{new} = (1-p)T_{intial} + \frac{p}{s}T_{initial}. \tag{3.1}$$

Since the speedup, S, is the initial execution time divided by the new time,

$$S \leq \frac{1}{(1-p)+\frac{p}{s}}, \quad S_{max} = \frac{1}{1-p}. \tag{3.2}$$

This is Amdahl's law [111]. Because of the development time needed to port code to GPUs and especially to FPGAs, it is important to first make sure that the speedup is worth the development time. Given that the data must be copied into chip memory, the potential speedup must be large enough to comfortably hide the I/O time. Since FPGAs don't have caches, the latency can be accurately predicted.

A useful tool for measuring computational performance is Intel Vtune [112]. It is a software tool that can measure all of the elements listed above. All the results reported in this section were measured using Vtune apart from the full HLT menu studies which used the CMS Trigger Timing server. Vtune gives an estimate of whether enough data has been collected to estimate the run time of each function, and all the results reported here have a sample size that is sufficient to have reliable measurements for the most time consuming functions.

3.2 The Phase 2 and Run 3 HLT menus

To measure the Phase 2 HLT performance, two HLT menus were selected. The current Run 3 menu was chosen as a point of reference. Though the current menu is partially implemented on GPUs, the measurements here were done using CPUs. This gives a more fair comparison with the Phase 2 menu and with the more detailed timings presented later in this chapter. This menu has 839 paths. The Phase 2 HLT menu chosen is the standard menu for testing Phase 2 HLT performance in the CMSSW 13 release [113]. This is very close to the menu presented in the Phase 2 HLT technical design report [109]. That menu is a simplified version of the HLT

menu used at the end of Run 2 in 2018. It was designed to be representative of the Phase 1 physics, while being relatively simple to study. The majority of the paths used in the standard HLT menu are only responsible for around 1% of the acceptance rate. This simplified menu captures 50% of the HLT acceptance rate with only 15 paths instead of 600. To e.g. increase this to 70%, 100 new paths would have to be implemented [109]. The menu used for this timing study had 12 paths, and the menu can be found in [110].

The timing of the Phase 2 and Run 3 HLT menus were measured using Monte Carlo data samples of $t\bar{t}$ events. These events pass most of the trigger paths at the HLT, so it represents an overestimate of the timings compared to realistic data. This type of sample is standard to use for testing because it represents a worst case scenario, and it is easy to see which paths are not performant. To reflect Phase 2 conditions, the Phase 2 HLT used the Phase 2 detector geometry and an input sample with a constant pileup of 200. For the Run 3 menu, a sample with an average pileup of 61.5 was used since this reflects Run 3 conditions. A sample of 1000 events were used for the former, and 20 000 events for the latter. This takes about the same time to run, so the measurements have a comparable uncertainty. To have a reliable timing measure, the samples were run on the server that the CMS Trigger studies group uses for timing studies. This is a computing node with two AMD "Rome" EPYC 7502 processors, each with 32 physical cores at frequency of 2.5 GHz. The uncertainty of samples this size run on the same server are often measured be around ± 2 ms [109]. The results are shown in Figure 3.1.

Figure 3.1 shows how the relative timings for the HLT changes between Run 3 and Phase 2. The time taken per event increased by a factor of roughly twenty between the two samples. If one were to retain the exact same computing structure, one would naively need twenty times more CPUs to deliver the same acceptance rate. This is consistent with the estimate of the HLT computing needs given in Section 2.8.6. Figure 3.1 also shows that the HGCal and tracking are the paths that change the most with higher pileup. The tracking, which includes pixel tracking, takes up almost 50% of the HLT budget at high pileup. This takes up about 30% of the time with Run 3 data. By studying the Phase 2 menu more closely, one can see that the most time consuming paths of the HGCal paths are also related to tracking. This is shown in Figure 3.2b.

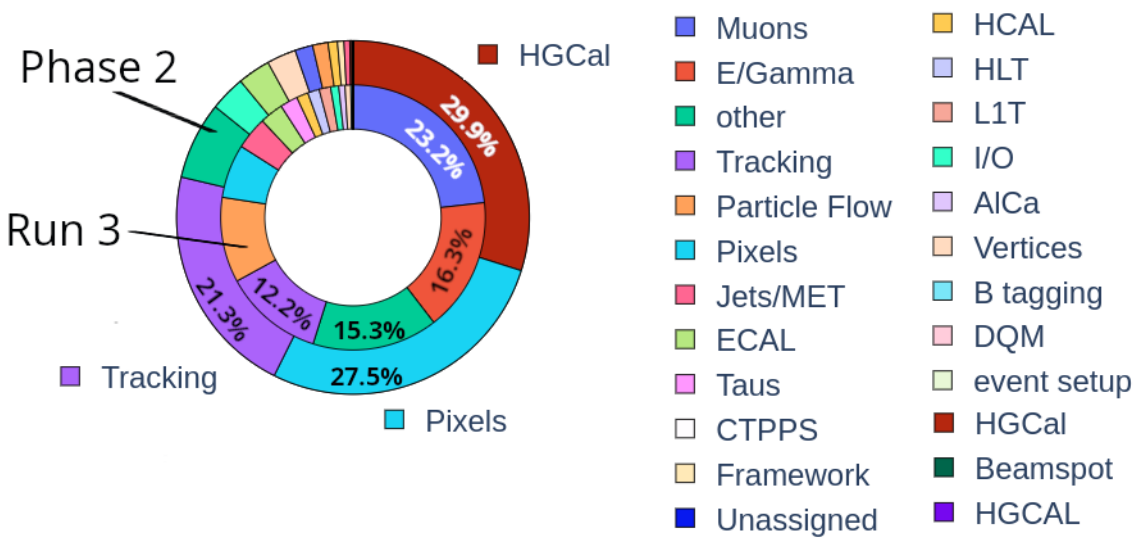
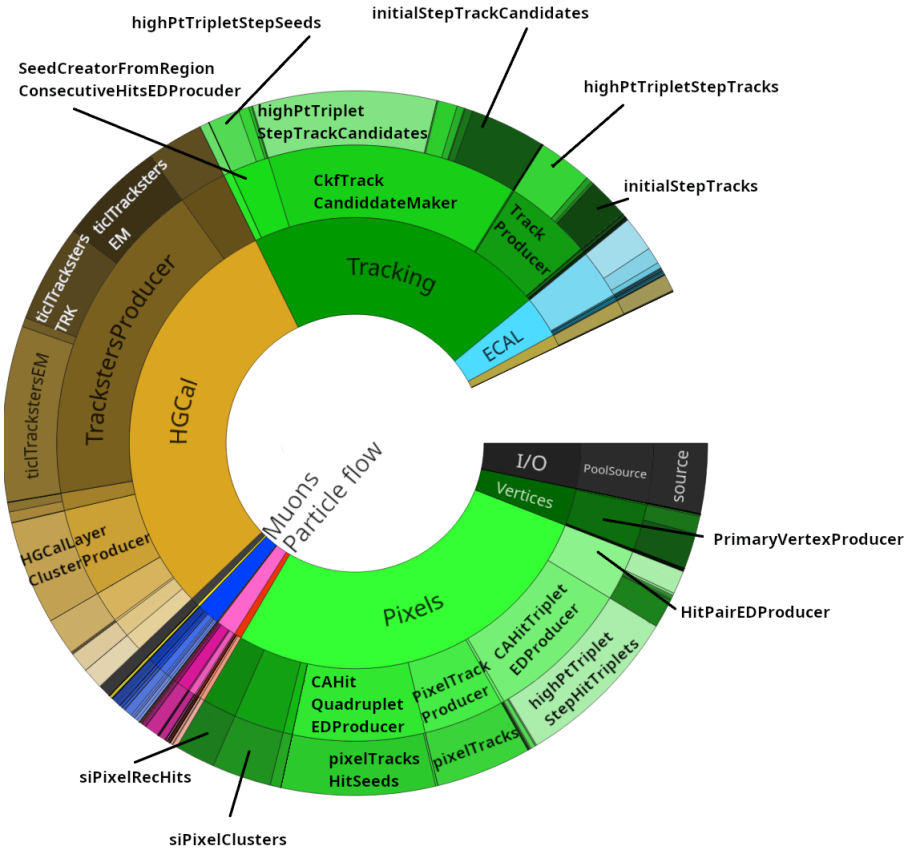


Figure 3.1: Relative timing of the modules in the Run 3 and the Phase 2 HLT. For Run 3, the modules from the muon chambers and the ECAL take up the most time. For Phase 2, the HGCal, tracking and pixel modules take up most of the time. The pixel paths are mostly pixel tracking, which means tracking takes up almost 50% of the HLT budget at PU 200.

Figure 3.2b shows a more detailed breakdown of the relative timing of the Phase 2 HLT menu. The most time consuming path is the one containing HGCal modules. Within the HGCal, hit clusters are created within each calorimeter layer by the `ClusterProducer`. The clusters from adjacent layers are then combined by the `TracksterProducer`. This producer takes up 17% of the HLT time budget and is responsible for reconstructing particle showers. It is part of The Iterative Clustering framework (TICL) [115], which is here implemented in four main steps. There are two electromagnetic steps, `ticlTrackstersEM`, which target reconstruction of e and γ particles. The `ticlTrackstersTRK` targets charged hadrons. The final iteration targets neutral hadrons and does not take up as much time as the other modules, and is hence not shown in the figure. All these steps use cellular automata to connect clusters from adjacent layers. This is the same method that is used for creating pixel quadruplets and triplets, and is described in more detail in Section 3.2.1. Since the most time consuming part of the HGCal reconstruction creates tracks, it is possible that a novel tracking algorithm could address both the HGCal, tracking and pixel tracking. By Amdahl's law, this would address around 78% of the HLT timing budget, unlocking a maximum possible speedup of a factor of around four. This is not enough for a speedup of a factor of twenty, but if this is to be achieved, 95% of the codebase needs to be sped up to its theoretical maximum. Consequently, there are plans to

Element	Time	Fraction
B tagging	6.8 ms	0.1 %
E/Gamma	85.8 ms	1.3 %
ECAL	197.9 ms	3.0 %
Framework	0.0 ms	0.0 %
HCAL	57.0 ms	0.9 %
HGCal	1774.8 ms	26.8 %
HLT	0.7 ms	0.0 %
I/O	214.7 ms	3.2 %
Jets/MET	34.8 ms	0.5 %
L1T	0.3 ms	0.0 %
Muons	107.8 ms	1.6 %
Particle Flow	92.5 ms	1.4 %
Pixels	1818.9 ms	27.5 %
Tracking	1365.6 ms	20.6 %
Unassigned	170.6 ms	2.6 %
Vertices	178.0 ms	2.7 %
event setup	39.3 ms	0.6 %
other	468.7 ms	7.1 %
total	*6614.1 ms*	*100.0 %*

(a)



(b)

Figure 3.2: Relative timing of modules in the Phase 2 HLT. The inner sections of the circle call the outer sections. The figure is adapted from plots made using the code in [114].

port large parts of the HLT to co-processors [109].

Tracking, along with HGCal tracking, is therefore the best algorithm to target in order to accelerate the HLT. Targeting any other path would lead to a maximum speedup of only a few percent. To understand why the tracking scales badly with high pileup, a detailed timing study was done. To access the hardware counters and detailed function calls, the timing server described here could not be used. Instead, a local installation of CMSSW on an external SSD was used. The CPU used is an Intel(R) Core(TM) i7-8665U running at 1.90GHz. Although the CMSSW installation was external, the USB 3.0 connection has a transfer rate of 400 MB/s, so it should have a minimal effect on the performance. Any effect should also be a systemic uncertainty, and this section only considers relative timings. Steps to replicate the measurements can be found in [110]. The data samples used were minimum bias samples. These have no non-collision background and represent a typical p-p collision. This choice was made to show a more realistic HLT menu than the $t\bar{t}$ events, and to reduce the run time since it was run locally.

3.2.1 Phase 2 pixel tracking

The pixel tracking starts by reconstructing pixel hits. This takes < 2% of the time for Run 3, and around 4% for Phase 2, and is shown by `siPixelRechits` and `siPixelClusters` in Figure3.2b. The next step is to create hit doublets with the `HitPairEDProducer`. This iterates through combinations of two seeding layers. For each hit in the outer layer, hits in the inner layer that are compatible with the beamspot and a minimum transverse momentum are combined with the outer ones to form hit doublets. The hits are sorted into a k-d tree [116], which is a data structure that partitions space and allows quick spatial queries. This takes up 1.9% of the run time for the Run 3 menu and 2.8% for Phase 2. These are relatively small changes, and can be attributed to there being more hits and doublets to reconstruct.

The more notable change is the creation of hit triplets and quadruplets. For Run 3, this takes up around 0.4% of the total run time. For Phase 2, it takes up 15% of the run time, and is therefore the main reason that the pixel tracking takes longer at higher pileup. Most of

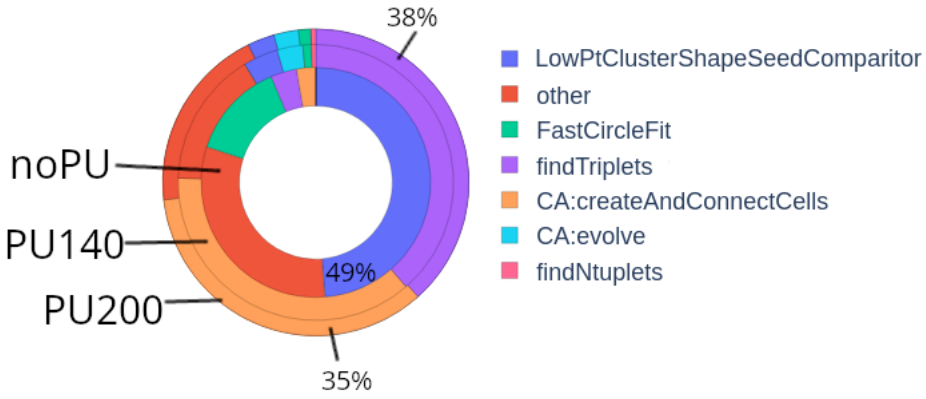


Figure 3.3: Relative timings of the functions responsible for creating pixel hit triplets and quadruplets at different pileup levels.

this is related to two functions, as shown in Figure 3.3. `findTriplets` creates triplets and `createAndConnectCells` reconstructs quadruplets. They both apply the cellular automaton method described in Section 2.7.2 to connect hit doublets. Cellular automaton methods generally scale badly because it is an exhaustive search. This is illustrated in Figure 3.4. This figure shows how the number of triplet and quadruplet candidates increase at higher pileup. It is an almost exponential increase because of the combinatorics introduced by connecting all possible hit doublets. It is not entirely exponential, tapered off by the geometric constraints that only allow certain combinations of hit doublets to be connected. The combinatoric hit search is also reflected in the rate of fake seed tracks that are reconstructed. At no pileup, the highest seed collection fake rate is 0.008, it is 0.47 at pileup 140, and 0.64 at pileup 200. The fake rate is reduced with a full track fit, but it takes time to reconstruct these fake seed tracks. Since the simplified HLT menu is used, only the first two steps of the pixel seeding is done. This targets prompt, high p_T triplets and quadruplets, which are responsible for almost 90% of the seed tracking efficiency. The `PixelTrackProducer` is the module that takes the track parameters from the quadruplets and triplets calculates the track uncertainties and updates the track state using the material in the final layer. This goes from taking 1.8% of the budget in Run 3 to 4.4% for Phase 2, again caused by the number of tracks.

The generation of pixel triplets and quadruplets together take up 73% of the pixel tracking budget at high pileup and is only a few hundred lines of code. This makes it a good candidate for acceleration. Since the HGCal also uses a cellular automaton method, an accelerated

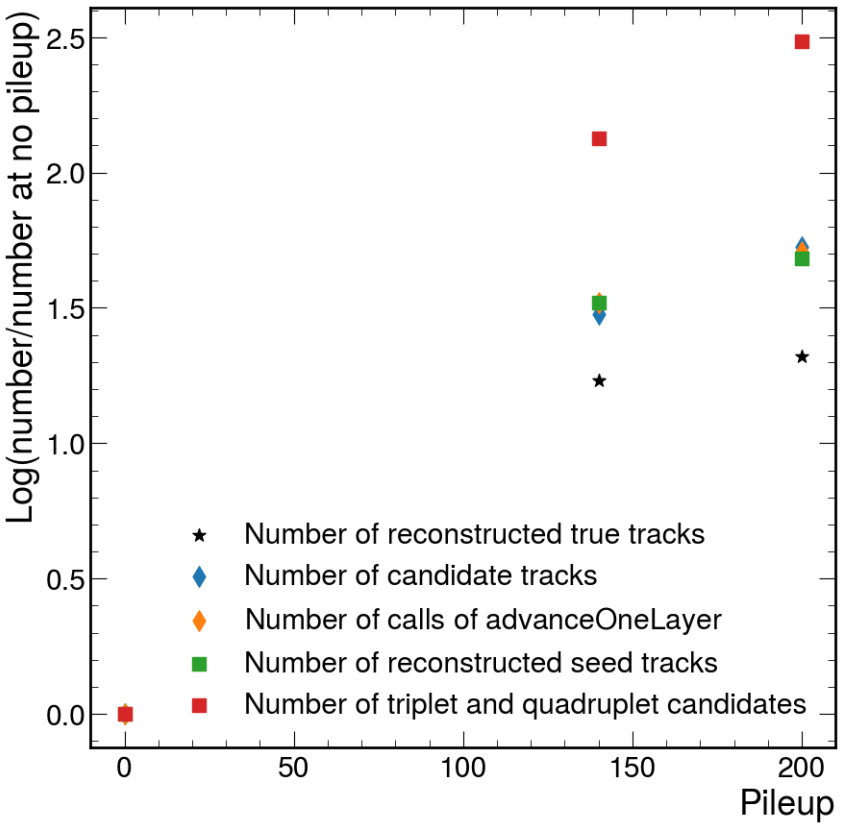


Figure 3.4: Ratios of high pileup tracking elements compared to a no pileup scenario.

implementation could also be used here.

Patatrack pixel tracking

The Patatrack collaboration [117] has accelerated the pixel seeding process by using GPUs. The raw pixel data is offloaded to GPUs, and multiple kernels are run to decode the data, cluster the pixel hits, form hit doublets and reconstruct triplets and quadruplets. This is done with a depth-first search from doublets. Each of these searches get their own thread, taking advantage of inherent parallelism of the problem. Vertices are also reconstructed, contributing to an increased physics efficiency and reduced fake rate compared to the CMSSW implementation. Patatrack has reported that the throughput of the triplet and quadruplet generation, as measured in event per second, has increased by a factor of around 1.8 to 2.8 [117]. They have also implemented the algorithm on CPUs and seen a small speedup here. The implementation is available within CMSSW, and a test was done with the Phase 2 menu using GPUs on the timing server described eariler. It shows a pixel reconstruction speedup of a factor of 2. Using Amdalh's law and

considering that pixel reconstruction takes up 27.5% of the HLT budget, we get a total HLT speedup of

$$S = \frac{1}{(1 - 0.275) + \frac{0.275}{2}} \approx 1.16. \tag{3.3}$$

Patatrack has achieved a very promising speedup and the algorithm has already been deployed and is being used in Run 3. Considering the total HLT speedup, it is clear that more needs to be done to address the overall Phase 2 HLT computing needs.

3.2.2 Phase 2 tracking

Around 63% of of the time spent in tracking takes place in the `CkfTrackCandidateMaker`. This module is also called by the muon reconstruction module. The `CkfTrackCandidateMaker` reconstructs tracks with the combinatorial Kalman filter (CKF), as described in Section 2.7. More than 80% of the `CkfTrackCandidateMaker` is spent in a function called `advanceOneLayer` which advances the tracks one layer. A description of the method can be found in Section 2.7.

Advancing a track one layer

The callgraph for advancing the track one layer is shown in Figure 3.5. `advanceOneLayer` first finds the next compatible layers with a quick track propagation. It then iterates over each of the layers using the `TrajectorySegmentBuilder`. First, the compatible modules are found with `groupedMeasurements`. As shown in Figure 3.5, this has different implementations depending on the region of the tracker because of the differing geometry. `TBLayer` takes up less time than the others because some of its tracking is already done in the seeding step. Almost all of the time for `groupedMeasurements` modules is spent propagating the path to the compatible detectors and calculating the material effects. This includes matrix operations, mostly of size $5 \times n$, because of the five trajectory parameters. Around 18% of the time spent in `advanceOneLayer` is spent on this accurate helix propagation, and around 10% on calculating the material effects. The hit measurements can then be fetched from the compatible modules. The group of compatible hits can be added to the track candidates. Depending on the number

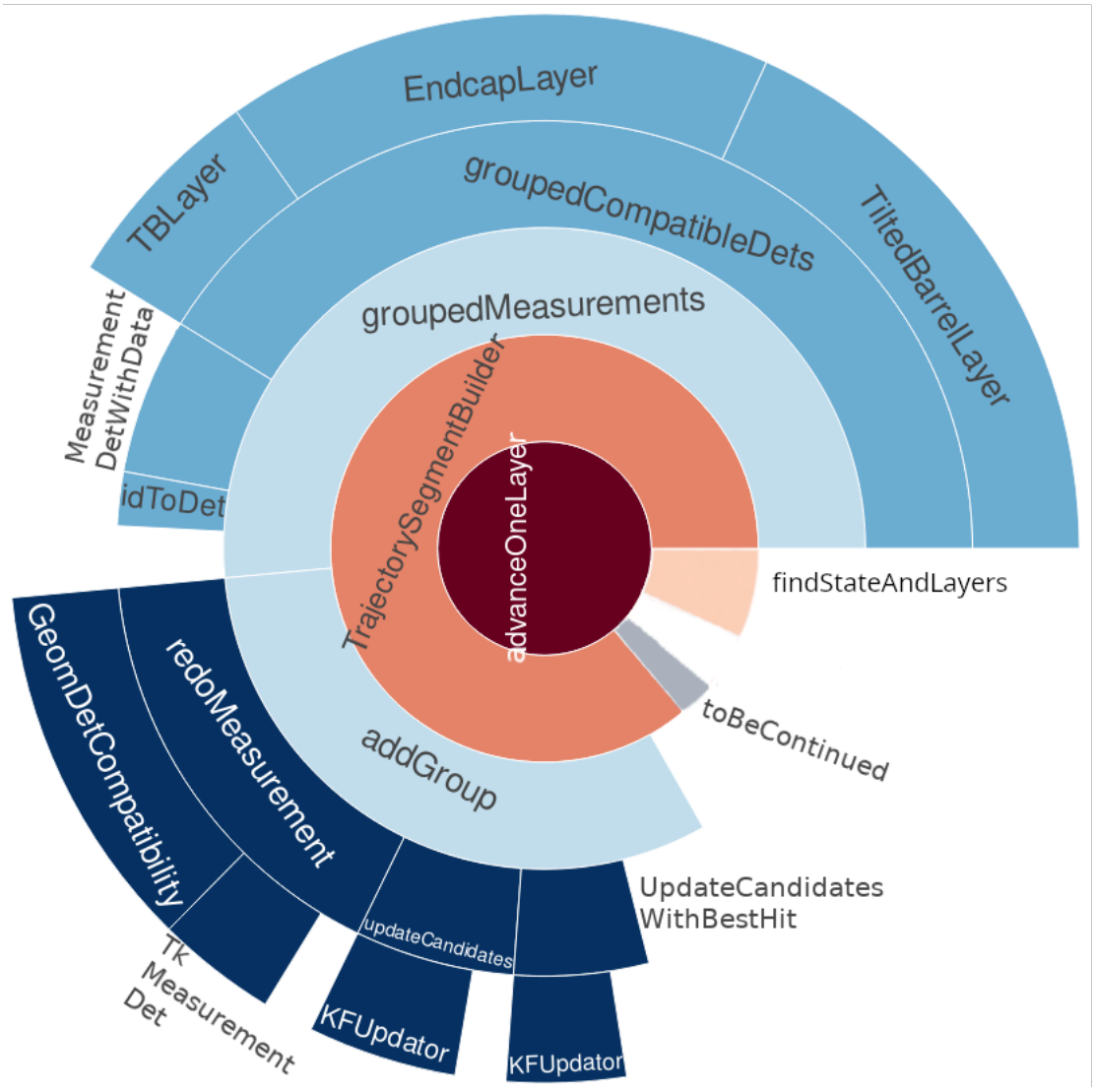


Figure 3.5: Callgraph of advancing a track one layer. The inner functions call the outer functions, and the functions are called in a counterclockwise manner starting from `findStateAndLayers`. The relative sizes of the functions reflect their relative timing.

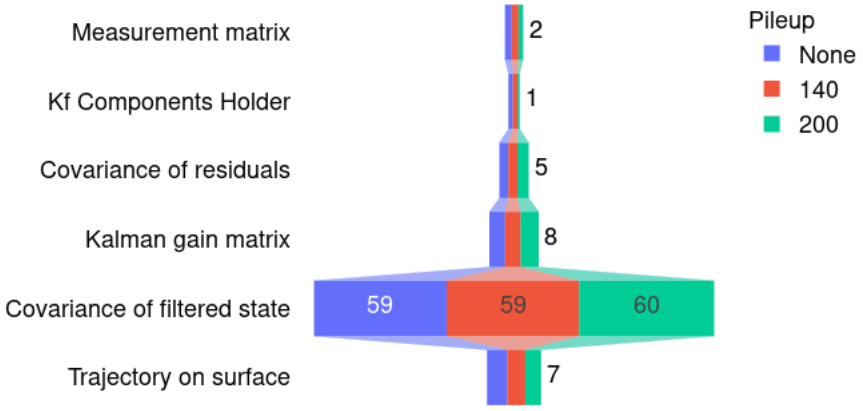


Figure 3.6: The proportional run time of updating a track with a hit using the Kalman filter.

of found hits, the best hit or several hits can be added. The `KFUpdator` is the Kalman filter implementation used to propagate the track with the new hit, and takes up around 7 % of the run time of advancing one layer. If the compatible measurements is empty, the detectors are checked again by the function `redoMeasurement`. There is a check on the number of measurements to see whether the track should be propagated further in `toBeContinued`.

Since the Kalman filter is at the heart of the tracking, a more detailed breakdown of its timing is shown in Figure 3.6. The necessary matrices are set up with the first two functions. The residual between the measurement and predicted state is calculated along with the covariance. The Kalman gain, as explained in Section 2.7.3, is then calculated. Computing the covariance of the newly updated state takes the longest, at around 60% of the run time. This contains two sets of triple matrix products of size 5×5. Again, it is the matrix multiplications that take up most of the run time. The Kalman filter has been ported to FPGA and tested for tracking purposes, but since the computation is relatively small, the speedup is negated by the I/O operations [118].

Interestingly, the proportions shown in Figure 3.5 stay the same at any of the tested pileups. One would perhaps expect that the number of compatible measurement groups or number of potential hit measurements would increase in a high pileup environment. Counting the iterations of each of the loops in `advanceOneLayer` showed that this is not the case. A simplified loop structure is shown in Figure 3.7. The number of detector groups and compatible hits

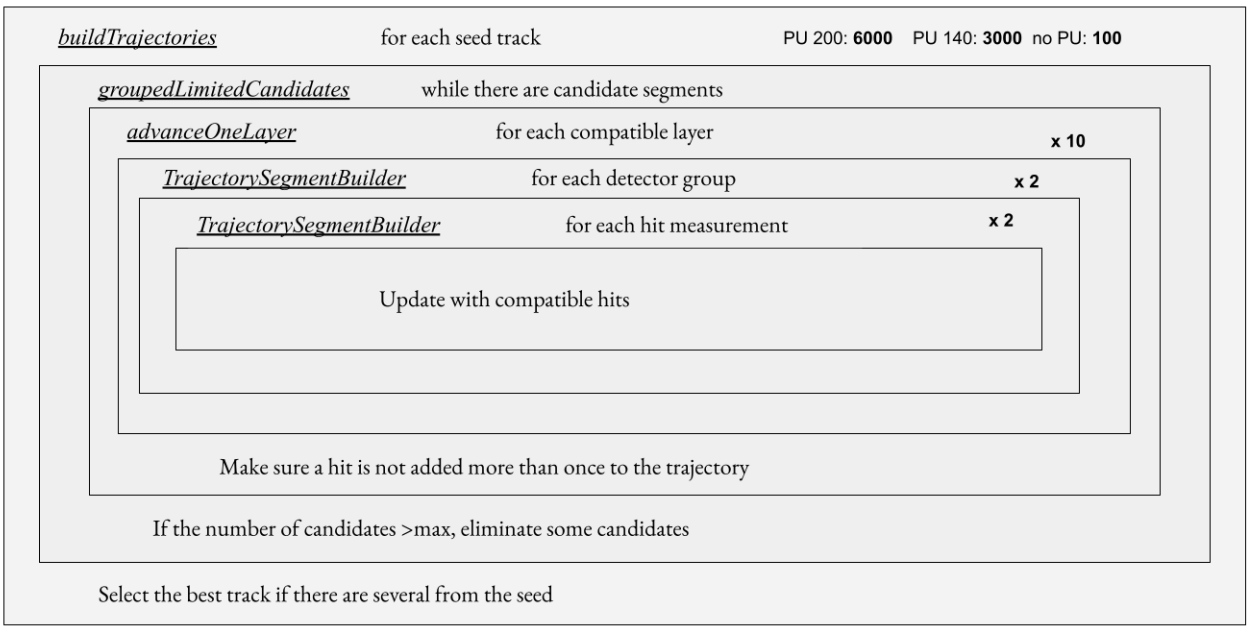
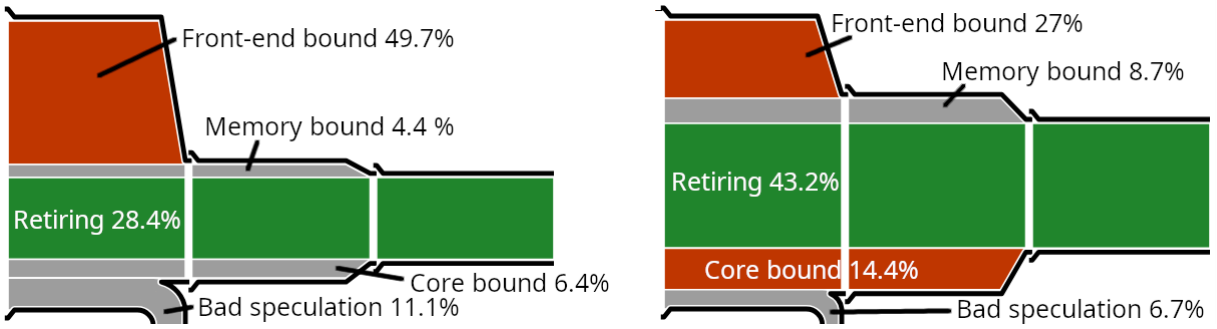


Figure 3.7: A simplified loop-flow diagram for track building within the CMSSW. The numbers in the top right corner show an estimate of the number of candidates at the different pileups as measured with a small sample. The numbers marked with an × show the average number of times the loops are called from each of the preceding loops.

within those groups stay roughly at two, no matter the pileup. This illustrates that it is useful to perform a very accurate helix propagation. Secondly, only three hit candidates were allowed to propagate at any time in this example. There are also various checks on the track and hit quality as tracks propagate [81]. It is clear that these successfully help restrict the number of candidates. The number of times one advances the track one layer therefore scales linearly with the number of track candidates, as shown in Figure 3.4. The number of candidates also seem to scale with the number of reconstructed seed tracks. Many of the seed tracks are not propagated further because of e.g. low momentum. The fraction of seeds that are continued seems to stay constant. As can be seen in Figure 3.4, the scale is in many ways set by the number of triplet and quadruplet candidates that are reconstructed, and fake tracks are eliminated at various stages. The combinatorial problem therefore really lies with the seed reconstruction.

A path to accelerate this code is not entirely straight forward. Matrix multiplications take up around 35% of the run time of the `CkfTrackCandidateMaker`. Though matrix multiplications might make the code a good candidate for both GPUs and FPGAs, there are a few complica-



(a) Pipeline efficiency of the `CkfTrackCandidateMaker`

(b) Pipeline efficiency of the mkFit tracking algorithm

Figure 3.8: Figure (a) shows the pipeline efficiency of the microarchitecture for the main CMSSW tracking algorithm when using one thread. Figure (b) shows the same for the mkFit algorithm using four threads. The mkFit algorithm is described in Section 3.3.

tions. The maximum HLT speedup by porting just this part of the code would be 1.5. The computations are also very quick, but repeated many times. This would mean the I/O would likely be a limiting factor, as seen with the Kalman filter implementation. An alternative would be to port a larger part of the code. A quick estimate of the number of arithmetic operations in the code base showed that it could potentially fit within a typical FPGA. To further explore the GPU or FPGA suitability, an analysis was done of the michroarchitecture usage of the algorithm.

Memory use of the CMSSW track building algorithm

The memory use of the track reconstruction was measured with ten events at pileup 200 and 0 run on a single CPU thread. Using multiple threads would not be possible because Vtune was paused and unpaused for the relevant part of the code. If Vtune was not paused, it would collect data for all of the HLT, resulting in an huge, unmanageable result. The HLT is designed to be multithreaded, so this measurement will show a worse resource exploitation than the real implementation. The CMSSW code is designed to paralellise over events rather than parallelisation within each event, so it is still a relevant measurement. This was confirmed with a measurement where four threads were used. Less time was spent loading instructions, but the measurement was largely the same as single threaded measurements. The result of measuring the track building is shown in Figure 3.8.

The microarchitecture graph in Figure 3.8 shows the CPU architecture use in terms of pipeline slots. These represent the hardware resources needed to process low-level hardware operations. The front-end fetches and decodes operations that will later be executed by the back-end. The memory bound segment shows where the pipeline could be stalled because of load or store instructions. Because of the complex geometry of the CMS detector, one might expect the tracking to be memory bound, but as described in Section 2.7, this has been optimised by sorting the module positions very effectively. The green slice shows the fraction of the pipeline that runs the "useful" work, i.e. running the instructions. If this was at 100%, the maximum number of microoperations per clock cycle would have been utilised. The core bound slice shows the issues that are not memory related, such as stalls due to software dependencies, like long-latency arithmetic operations. Bad speculation reflects slots that were issued but not used, or when the pipeline is blocked because it is recovering from an earlier incorrect speculation.

Figure 3.8(a) shows that most of the time spent in the `CkfTrackCandidateMaker` is spent in the front-end. This is abnormally high for an HPC application, where the front-end is usually expected to take up to around 10% for an optimized HPC algorithm [112]. Most of the front end latency here is caused by ICache misses (see Section 1.4.1 for an explanation of caches). These are cache misses from the level 1 instruction cache when the instructions fetched are not in the cache. This can typically happen with large code bases or fragmentation between hot and cold code. The fraction of bad speculation is also higher than expected for an HPC algorithm, which is usually up to 5% [112]. Here, all of the bad speculation is caused by branch mispredictions. Many of the functions in the `CkfTrackCandidateMaker` have a rate of branch misprediction well above 10%. An example is branching based on the geometrical location of the track. Other examples include functions that contain if-statements based on whether the hit is valid, or if a track is a subset of another track.

Based on Figure 3.8 there is some evidence that an FPGA could be a suitable accelerator. There are branch predictions and cache misses that affect the performance of the code. It also seems from the size of the arithmetic operations measured that a well thought through design could fit on FPGAs. There are small matrices that are called many times, which means a large part of the `CkfTrackCandidateMaker` would need to be ported in order to avoid the

I/O being a major bottleneck. Considering that the code base for tracking is more than 10 000 lines, it would however be an extensive undertaking. Again, using Amdahl's law, the `CkfTrackCandidateMaker` takes up 63% of the 21.3 % of the HLT budget that is spent on tracking at PU 200. This means a maximum overall HLT speedup of 1.15. Porting the algorithm to an FPGA would therefore be a large amount of effort for a relatively small gain. An alternative is to speedup the code base by vectorisation for CPU or GPU, which is what the mkFit algorithm, described in Section 3.3, implements.

3.3 The mkFit algorithm

The mkFit collaboration [119] started working on accelerating the track reconstruction in 2014, and was in 2022 included within CMSSW. They implement vectorisation of the track building by parallelising over events, detector regions and tracks. The detector geometry and material is also simplified compared to CMSSW to reduce memory operations. mkFit uses SIMD operations (same instruction, multiple data) for the track propagation and a class they developed called Matriplex that vectorices small matrices. The matrices are batched together and operate in lockstep. Targeting the small matrices is consistent with the finding in the previous section that a large amount of the computation is spent in matrix computations that are called many times. mkFit initially targeted the initialStep and HighPt iterations of the track building, though other iterations have been implemented later. mkFit has achieved a similar efficiency and fake rate as the current CMSSW implementation. They reported a speedup of a factor of three compared to the CMSSW track building. This is shown in Fig. 3.9.

3.3.1 Computational performance of the mkFit algorithm

The mkFit algorithm achieves good acceleration for CPUs. Since it is a smaller and simpler code base than the CMSSW tracking, it might have the potential to be a good starting point for a GPU or FPGA implementation. To evaluate this, a performance study was done. This was done in 2020, so any improvements made by the collaboration since then will not be reflected

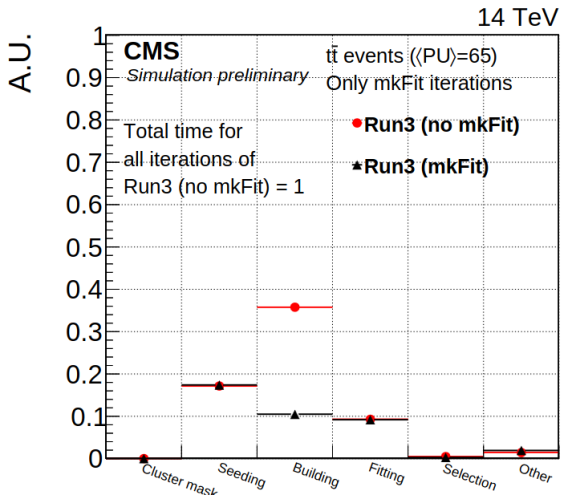


Figure 3.9: Timing performance of the mkFit track building algorithm. Figure from [119].

here. This implementation was not developed for the Phase 2 detector, so the Phase 1 geometry and a $t\bar{t}$ sample without pileup was used. It is therefore not a one-to-one comparison with the Phase 2 measurements used in the rest of the chapter, but is still a meaningful comparison. Fig. 3.10 shows the callgraph for mkFit.

The callgraph in Fig. 3.10 shows the structure of function calls measured with 100 events. The functions higher up on the graph call the ones that are directly connected to them. The colour indicates the proportion of the run time that was spent inside the function. The callgraph fits within a page, which is not the case for CMSSW, which has a more complicated structure. This illustrates that the mkFit algorithm could be a good starting point for heterogeneous computing. Unsurprisingly, we see again that it is the helix propagation and the material effects that are the hotspots of the algorithm. The large, turquoise rectangle in Fig. 3.10 shows the two main hotspots; `sincos4` (9.26%) and `sincosf` (24.07%). The percentages show the relative time spent inside these functions. These take long because of the number of times they are called. There might be some potential for FPGA acceleration here. If the trigonometric functions were stored inside a look-up table, they would not have to be recalculated for every use. The hotspots is clearly so small that a much larger part of the mkFit algorithm would need to be ported in order to offset the time spent in I/O. It is therefore necessary to consider the suitability of the rest of the codebase. To asses the suitability for a co-processor, as explained in Section 1.4, the microarcitecture was considered. This is shown next to the CMSSW measurement in Fig.

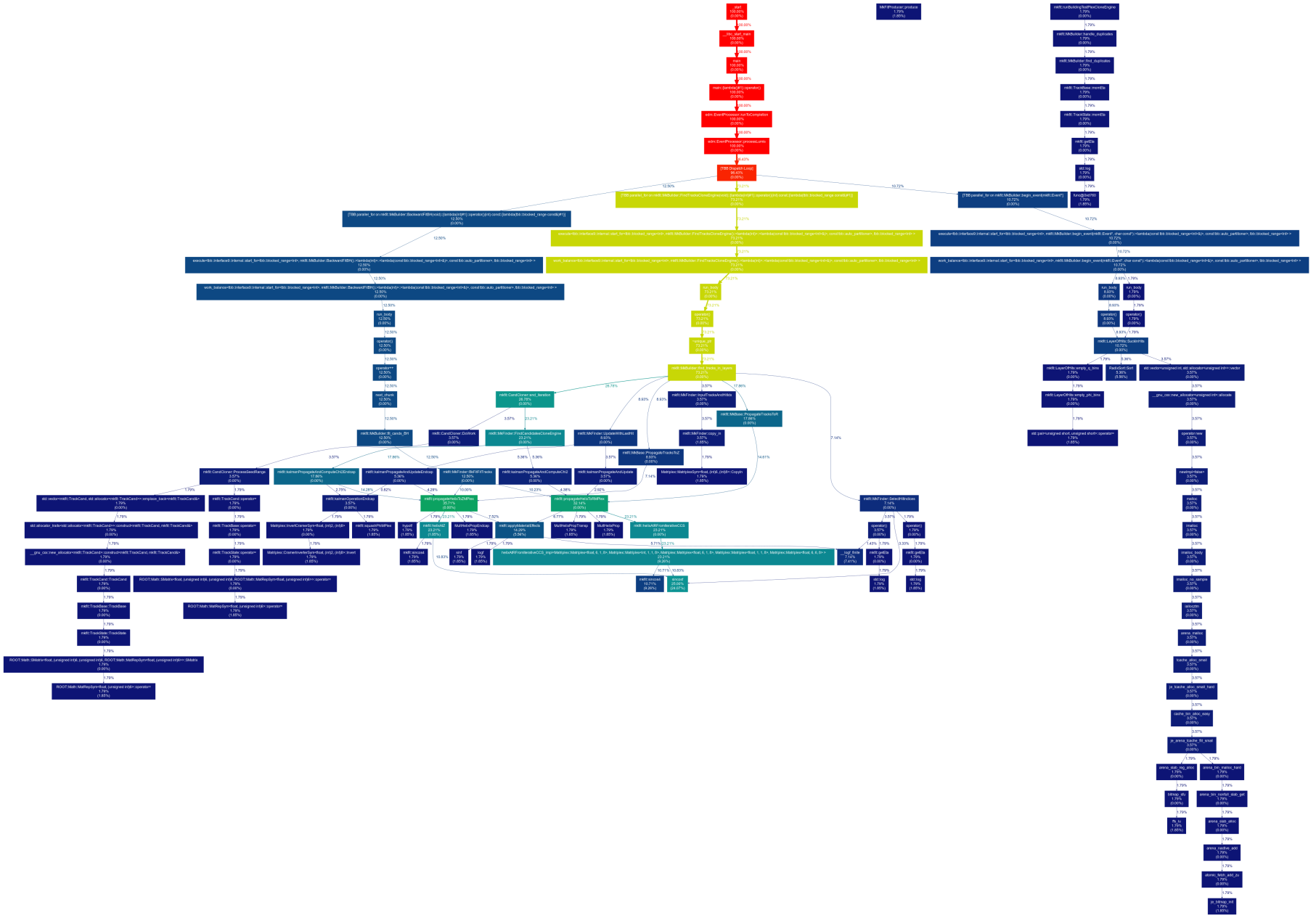


Figure 3.10: Callgraph of the mkFit algorithm. Each box represents a function and the colour indicates the time spent in that function. The upper percentage in each box shows the relative time spent inside the function and its daughters, and the number in brackets shows the relative time spent inside the function itself.

3.8. The figure shows that mkFit reduced the bad speculation and front-end issues that the CMSSW implementation had. There is still a relatively high percentage of bad speculation. This illustrates why the mkFit has not published a GPU implementation, though there has been work on this. It is hard to achieve a good GPU implementation when the code base has a large degree of branching. An FPGA implementation could address this, but it might take much longer to implement.

3.4 HLT track reconstruction algorithms under development

Apart from the mkFit and Patatrack algorithms, there are other track reconstruction algorithms currently in development at CMS. There is a line segment tracking algorithm [120] that aims to take advantage of both GPUs and the new p_T modules to accelerate tracking. Doublets are created from p_T modules in the outer tracker. Line segments are then created by linking doublets in neighbouring layers. The links can be extended inwards and then combined with tracks from the pixel seeds. A neural net that classifies real and fake track candidates has been developed to reduce the fake rate. They have achieved high efficiency and good timing results, but the physics performance is not quite as high as the CMSSW tracking yet. Work is ongoing to improve the timing and physics performance further and to integrate the package into CMSSW. Another machine learning alternative is graph neural nets, discussed in detail in Chapter 4.

3.5 Summary

The CMS HLT algorithm takes twenty times longer to process an event for Phase 2 than for Run 3. This is mainly caused by track reconstruction algorithms, which together take up 50-70% of the HLT algorithm, depending on whether one includes the HGCal tracking. The pixel tracking algorithm is subject to an almost exponential increase in the number of seeds explored at higher

pileup. This is a somewhat simple algorithm that has been accelerated on GPUs. The track reconstruction is more complex, with a high degree of branching and few clear timing hotspots. The mkFit collaboration has accelerated the track reconstruction on CPUs. Nothing has yet achieved the order of magnitude speedup one would ideally want. Though there are indications that either the CMSSW or mkFit would be suitable for e.g. an FPGA implementation, it would be a large undertaking. Because of the size and complexity of the existing code, alternative algorithms might be appealing. Machine learning could perhaps address both the pixel tracking and tracking, and potentially avoid some of the branching and cache mispredictions seen in the CMSSW. Machine learning is also more easily ported to co-processors because of native support of ML packages. The remainder of this thesis will explore the ML potential of the tracking.

Chapter 4

Graph neural nets for tracking

As seen in Chapter 3, track reconstruction takes up half of the HLT time budget at high pileup. Machine learning might be able to create a speedup that is difficult achieve by only accelerating the existing algorithm or using co-processors. Some of the main arguments for using ML for tracking are;

- ML can leverage learning from experience instead of exploring all possible hit options.
- The training time for an ML algorithm is in this case largely irrelevant; one could train a model for a long time and achieve a small model that performs quick inference.
- There could be special cases in the data that would normally lead to fake tracks or inefficiencies that the ML model could learn to handle.
- The level of branching and cache misses seen in the CMSSW tracking could be reduced, enhancing performance.
- The complex control flow of the tracking in CMSSW could potentially be handled by a simpler codebase, making maintenance easier.
- ML is generally much easier to accelerate than the current CMSSW tracking algorithm, allowing potential speedups.

A potential drawback is that, as mentioned in Section 1.3, ML can be opaque, and it can be difficult to see how the decisions are made. With a good level of testing, this can be addressed. This can be done by e.g. stress testing where the model fails, trying to visualise the decision making process or validating across many randomly sampled training and test sets. There are many other techniques, some of which can be found in [121]. Work on ML for tracking has been ongoing for several years, and convolutional and recurrent neural nets have been explored [122], [123]. These are theoretically well motivated since they can take into account the previous hits or nearby hits of a track. They have not shown as much promise as graph neural nets, which have been studied primarily by the Exa.TrkX group [124].

4.1 Graph neural nets

Graph neural nets (GNNs) [125] are extensions of neural nets that can learn from data structured as graphs. This can be used for predictions on a node level, on the connectivity of the nodes, or on a graph level. They are suitable for any data that has a graph structure, and have seen successful applications in tasks ranging from traffic flow to fluid dynamics [126], [127]. A graph typically contains nodes and a list of node connections. It might also contain information about the nodes, the node connections or the graph as a whole. This is illustrated in Fig. 4.1. An adjacency list contains indices of which nodes are connected to each other. The format of this is important. We could have used a matrix where e.g. the columns and rows represent individual nodes, and a 1 means that the two are connected. This would result in a large and sparse matrix that is not permutation invariant; depending on the ordering of the nodes, the connectivity could be described by different matrices that give different results in a neural net. It is crucial that the information is permutation invariant or equivariant. If the nodes were reordered, the output would be the same, or have a constant change. This is achieved with adjacency lists that describe the pairwise connectivity of the nodes, as seen in Fig. 4.1.

Usually, GNNs learn to encode and decode the graph data structure. The power of GNNs comes from learning about a node's neighbourhood through message passing. First, node features are

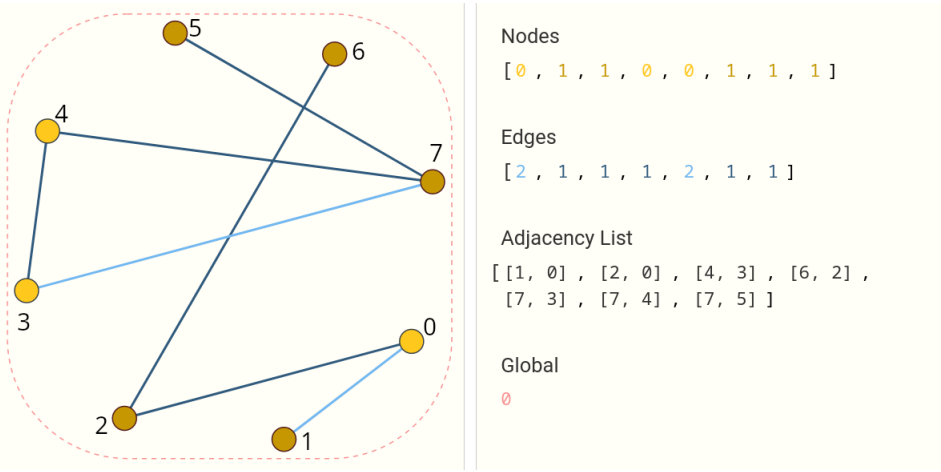


Figure 4.1: The graph data structure. The numbers next to the nodes represent the node index. The node and edge lists contain features for the nodes and edges in the order of the index. The adjacency list describes the connectivity of the nodes using the node indices. The node and edge features often have more than one dimension. Figure from [128].

embedded. Embedding refers to learning a new, often more compact, representation by passing information through a neural net. The embedding, h, for node u in layer k of a GNN is a combination of its representation in the previous neural net layer and an aggregation of the embeddings in its immediate neighbourhood:

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{self}^{(k-1)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{neigh}^{(k)}\sum_{v\in\mathcal{N}(u)}\mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right), \tag{4.1}$$

where W are the weight matrices and b is the bias term. In the first layer, the embedding for node u is simply its own feature matrix multiplied by an initialised weight. This is combined with an initialised weight multiplied by the features of the first-hop neighbours. In the second layer, all the embeddings for neighbours v also include information about their own neighbours, thereby giving node u information about the second-hop neighbours. One can therefore determine how many neighbours are considered by the number of message passing layers in the network. An illustration of this formula is shown in Fig. 4.2. There are many different message aggregation methods. A popular modification to Equation 4.1 is to normalize the aggregation. Some nodes might have many connections while others have few, and normalizing this can lead to increased performance [129].

Since many physics problems naturally have a graph structure, GNNs have been successfully

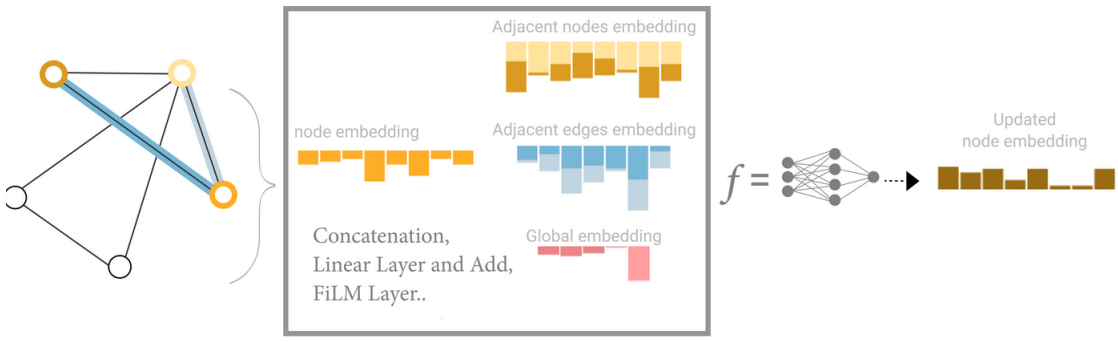


Figure 4.2: GNN message passing. Information about nodes and edges are aggregated. These elements can then be concatenated or combined with e.g. a linear layer. This is passed through a neural network to create an updated embedding. Figure adapted from [128].

applied to many tasks in HEP. For node level tasks, it has been used to correct cluster energies, identify pileup particles and for particle flow reconstruction. At a graph level, it has been used for jet tagging, to estimate shower energy and to discriminate signal and background in various analyses. At the edge level, it has been used for vertex reconstruction and to identity track segments. The latter will be the focus of this chapter. An overview of many of the applications of GNNs in HEP can be found in [130].

4.2 Interaction network for particle tracking

One of the main efforts to use GNNs for tracking has been led by the Exa.Trkx collaboration [124]. They create one graph from each collision event where the tracker hits are nodes. The hit position is described in cylindrical coordinates, (r, ϕ, z). Hits that might be connected by a track have edges that connect them. Edges are only built between hits in adjacent layers, with some simple geometrical constraints that are set to exclude false edges while keeping the efficiency around 99%. The features of the edges include the difference in position between the hits, and the length of the edge measured in $\eta - \phi$. The edges are labelled as true edges if there is a track that connects them and false otherwise. The task is to predict which edges are true and which are false. This represents the track building, and a separate track fitting method can later be applied.

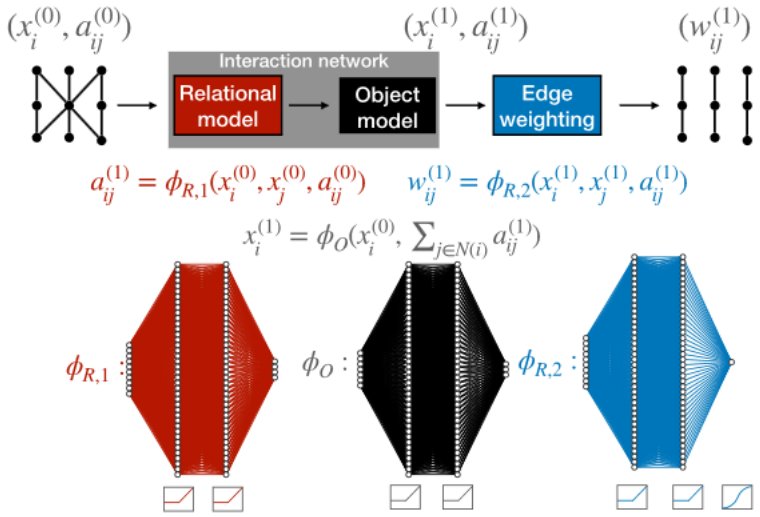


Figure 4.3: The interaction network used for particle tracking. The relational model encodes node and edge features. The object model calculates the "effect" of this by concatenating the relations with the node features. This information is combined to weigh the edges. Figure from [133].

The GNN used by Exa.Trkx is based on an architecture that was specifically developed for physics use cases. It is called an interaction network and was created by Gooogle DeepMind [131]. The idea is to encode one object, and then calculate the effect it has on another object. Here, the hits act as objects, and the edges as relations. First, node features and edge attributes are embedded by a neural net. This is the relational model in Fig. 4.3. The effect on the nodes is considered by concatenating this learnt embedding with the node features, and passing this through a neural net. This is the object model. Finally, the information from the relational network and the object model are concatenated and passed through a final neural net. This outputs a score for each edge, which reflects the learnt likelihood of the edges being true edges. Exa.Trkx first applied this model to data from the TrackML competition [132].

4.3 GNNs for the TrackML competition

In 2018, CERN hosted an open competition on the ML competition webpage Kaggle to inspire new algorithms for particle tracking [134]. Instead of tying the competition to the ATLAS or the CMS detector, a generic Phase 2 detector layout was used, as shown in Fig. 4.4. Similar to the ATLAS and CMS detectors, the central regions have a barrel structure, and the endcaps

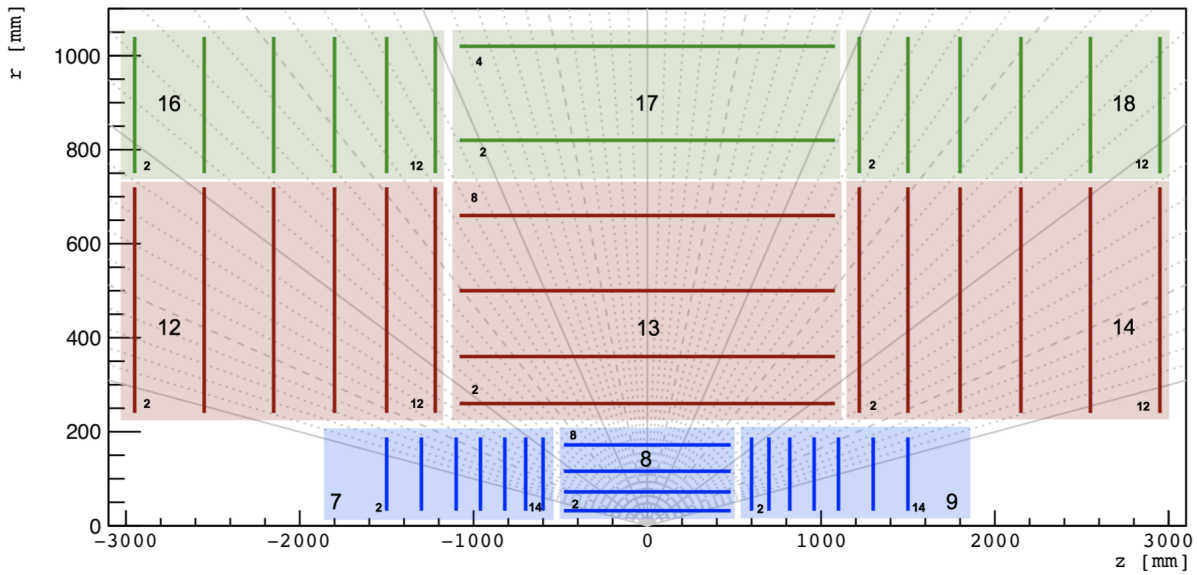


Figure 4.4: The TrackML detector. The pixel detector is shown in blue, short silicon strips in red, and long silicon strips in green. Figure from [134].

have a disk structure. The pixel detector has a spatial segmentation of 50 µm × 50 µm. The short strip region has a size 80 µm × 120 µm and the long strips are 0.12 mm × 10.8 mm. The modules overlap and cover up to $\eta = 3$. Simulated $t\bar{t}$ events with a pileup of 200 were produced using the Pythia 8 [135] generator and the detector response was simulated using the ACTS software [136]. This includes material interactions, like multiple scattering and energy loss. Inefficient sensors and noise hits are also included.

The resulting data contains 50 000 events with information about the hit positions, true track parameters, vertices and hits. The competition consisted of an accuracy and a throughput phase [137], [138]. The highest accuracy achieved was 0.92 with an algorithm quite close to what the experiments currently use, where seeds are generated and extended. The second most accurate solution used an adjacency matrix to predict hit pairs, quite similarly to graph neural nets. This method depended on a fairly extensive combinatorial search. The fastest high scoring solution in the throughput phase spent 0.56 seconds per event, and was an iterative, non-ML solution. One of the takeaways from the competition was that many of the highest scoring solutions in both phases were close to the traditional approaches, and perhaps more time was needed for the ML models to be competitive. The TrackML data has been continued to be used as a benchmark for the tracking problem, because it is experiment agnostic, easy to

use, and there are many benchmarked solution to compare against.

4.3.1 Applying a GNN to TrackML data

Results from applying an interaction network architecture to the pixel detector in the TrackML data can be found in [133]. Some of the results reported in this paper have been reproduced here in order to study the GNNs in more detail. The only difference is that we have used 14 neurons in each layer of the GNN, whereas their paper uses 40. We have made this choice because we only lose at most a percentage point of edge classification accuracy, while giving us the ability to train faster. All the code presented in this chapter is available at [139]. First, we consider the graph construction. It is worth emphasising that graph neural nets typically require two steps. A graph has to be built before a graph neural net can be applied. Some metrics of the graph building are shown in Fig. 4.5. The efficiency shows how many of the true edges were successfully built, and the purity shows the ratio of true to total edges. We can see that we are consistently creating edges where 70 to >90% of the edges are false. This results in millions of edges even though we are only considering the pixel detector. The edge classification accuracy when applying a GNN to these built graphs is, however, very good. As reported in [133], the edge classification accuracy is around 0.98 for $p_T > 0.6$ GeV. While this is excellent performance, it is important to keep in mind that there are many fake edges, and this does not necessarily translate to an equally high track efficiency. The track efficiency is consistently above 90%, with a fake rate around 4%. This is a very good performance.

A common simplification when studying tracking is to remove pileup. Since these studies are done with Phase 2 in mind, this is not done here. Some other simplifications have been applied that we should be aware of, including several truth level filters;

- Only particles above a certain p_T threshold were considered. Hits belonging to tracks below the threshold were removed
- Only one hit per particle per layer was considered
- Noise hits were removed

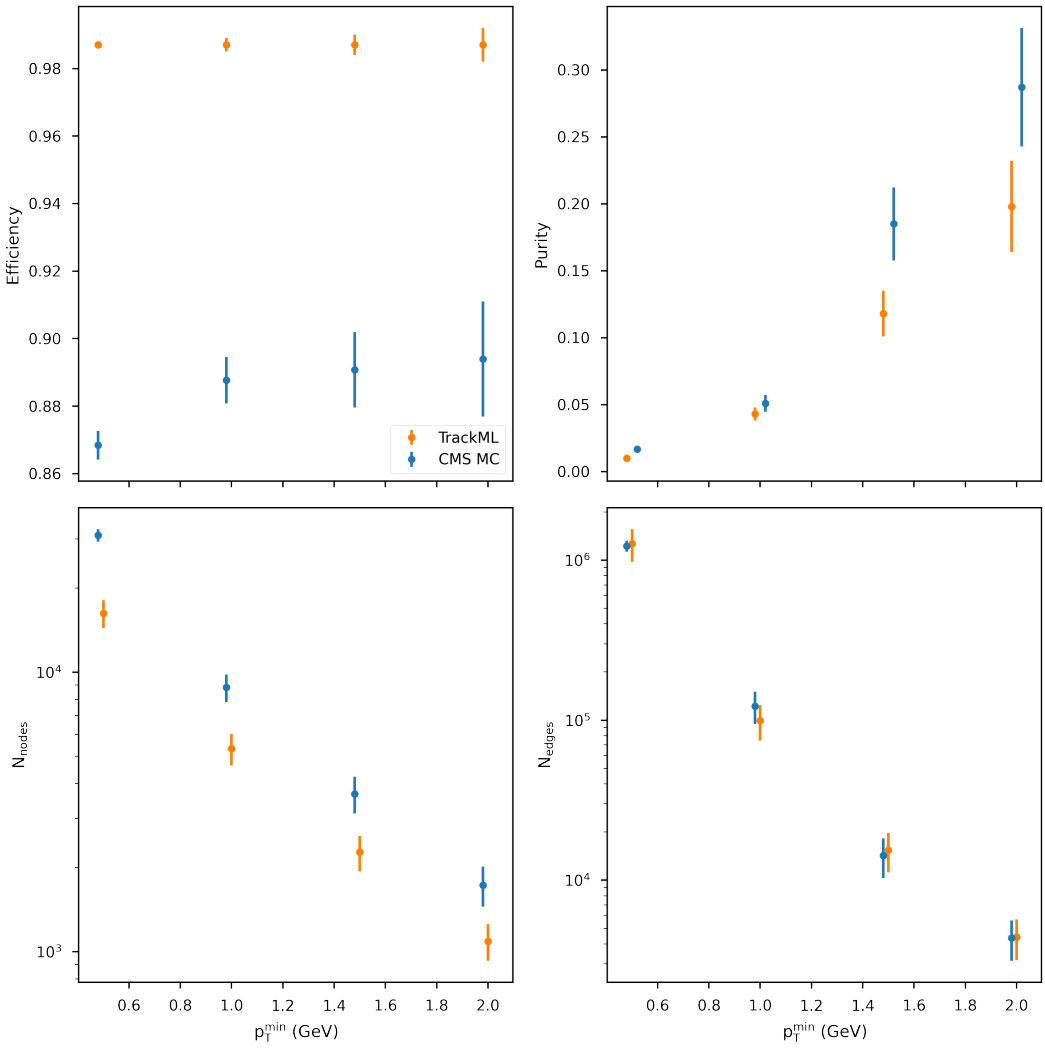


Figure 4.5: Graph build performance for TrackML and CMS data. The efficiency, purity, number of edges and nodes when comparing the pixel tracker of the TrackML and CMS detectors. Efficiency here is number of true constructed edges divided by the number of actual true edges. Purity is the number of true edges to the total number of edges. CMS MC refers to data from CMS Monte Carlo simulations, as described in Section 4.4.

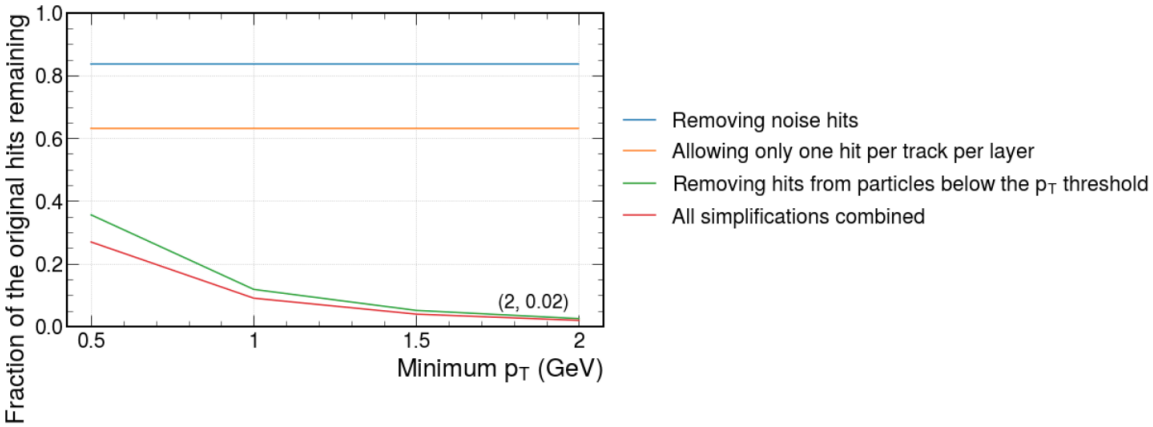


Figure 4.6: The fraction of the number of original hits in an event remaining after applying each of the simplifications shown to the right as measured at four p_T cuts.

- Only the pixel tracker was considered
- Only adjacent layers were connected, with some extra hard coded extensions in the transition between barrel and endcap

The implication of these limitations is shown in Fig. 4.6. Since the aim is to use GNNs for the entire tracker, the limitations were measured using the full tracker, not just the pixel detector. The figure illustrates that the p_T threshold used in [133] means that only 2%-27% of the total hits are considered. There is no way to make any of these simplifications without having truth-level data. Since graph building is a combinatorial problem, removing these simplifications might be difficult. To explore this, and to test the algorithm on a more realistic dataset, the GNN algorithm was applied to CMS Monte Carlo data.

4.4 GNNs using CMS Monte Carlo events

We will now extend the work of the Exa.Trkx collaboration using a sample of $t\bar{t}$ Monte Carlo events with a constant pileup of 200. As a starting point, the GNN was implemented in the same way as the TrackML dataset, with the same simplifications and only using the inner tracker. The only difference was that noise hits were not removed. Firstly, it is clear that building the graphs is a performance limitation. This is illustrated in Fig. 4.7. Even when using a p_T cut of 2 GeV and the pixel tracker only, building one event takes 0.2 seconds when

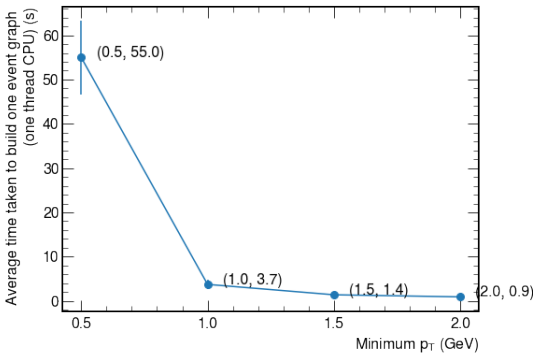


Figure 4.7: Time taken to build one graph.

using one thread on a standard CPU. Considering that the current HLT menu takes around 0.3 s/event, this is a very long time. The graph inference has been implemented on FPGAs by the Exa.Trkx collaboration [140]. The graph building is at least an order of magnitude slower than the inference, so this would also need acceleration if one were to consider co-processor implementations. It would be challenging to accelerate this on FPGAs because of the size of the graphs, so GPUs would likely be needed.

The physics performance of the algorithm is still high, as shown by the track reconstruction efficiencies in Fig. 4.8. An example of graphs and their corresponding inference pattern at two different p_T thresholds is shown in Fig. 4.9. The figure shows that there is a large fraction of fake edges. It also shows that because of the high granularity in the pixel detector, the hit density is lower in the endcaps. The number of misclassifications is therefore lower here than in the barrel. Because edge classification is imperfect, a clustering algorithm is used after the GNNs to cluster hits where edges are labelled as true. The DBSCAN algorithm [141] is used here, as it is a simple method that does not require specifying the number of clusters. In Fig. 4.8, we have applied the same convention as [133], where there are three different constraints on matching the hits. *LHC match* is a track where over 75% of the hits belong to the same track. *Double-majority* are tracks where over 50% of the track hits belong to the same track, and the track contains more than 50% of the true hits of the particle. *Perfect match* is where the track contains only hits from the same particle, and every hit generated by that particle.

We want to compare these results to both the TrackML results reported in [133] and to the CMS

efficiencies in Fig. 2.9 and Fig. 2.10. Starting with the former, we see a general decrease in the efficiency using CMS data compared to TrackML data. This is expected, since we now include noise hits, the detector geometry is more complex, and there are more hits in each event. The p_T distributions shows the same general trend of decreasing performance with increasing p_T. This is a common feature of edge classifying tracking schemes [133]. There is an unusually high performance for the LHC match at low p_T, but not for the double-majority or perfect match. It is possible that we are creating several, disjointed tracks here. This would create many tracks where most of the hits belong to the same particle, but the tracks do not contain most of the hits from that particle. This could be adjusted by adjusting the number of hits in each cluster. The shape of the pesudorapidity distributions is somewhat similar to the TrackML data. We see a similar decrease at a low absolute value of pseudorapidity as the high hit occupancy leads to a high rate of fake edges. This is illustrated in Fig. 4.9. We see a high performance at high η. The reason for this can be seen in Fig. 4.9. At a higher p_T cut, the occupancy in the endcaps is much lower, making it easier to reconstruct the edges. Comparing the results to the CMS efficiencies in Fig. 2.9, 2.10, we consider the LHC match, as this is the definition used by CMS. We see a similar shape of the plots, with slight decreases in efficiency around the transition region between barrel and endcaps. The overall LHC match efficiency is close to the one in the CMS results. This is encouraging, but we must remember that these are not directly comparable plots since we have made several simplifications.

4.5 Removing simplifications

We can now address some of the constraints used so far. Firstly, the allowed layer connections have been hard-coded. We can make this slightly more realistic by using data to find combinations of layers that are usually connected. A mapping was created by counting each layer connection combination over a few hundred events. Connections that were infrequent were assumed to be due to noise hits and/or missing hits, and were excluded from the allowed connections. The connections were inspected by eye so as to include connections that are rare, but physically valid. The mapping included about 90% of the hit connections.

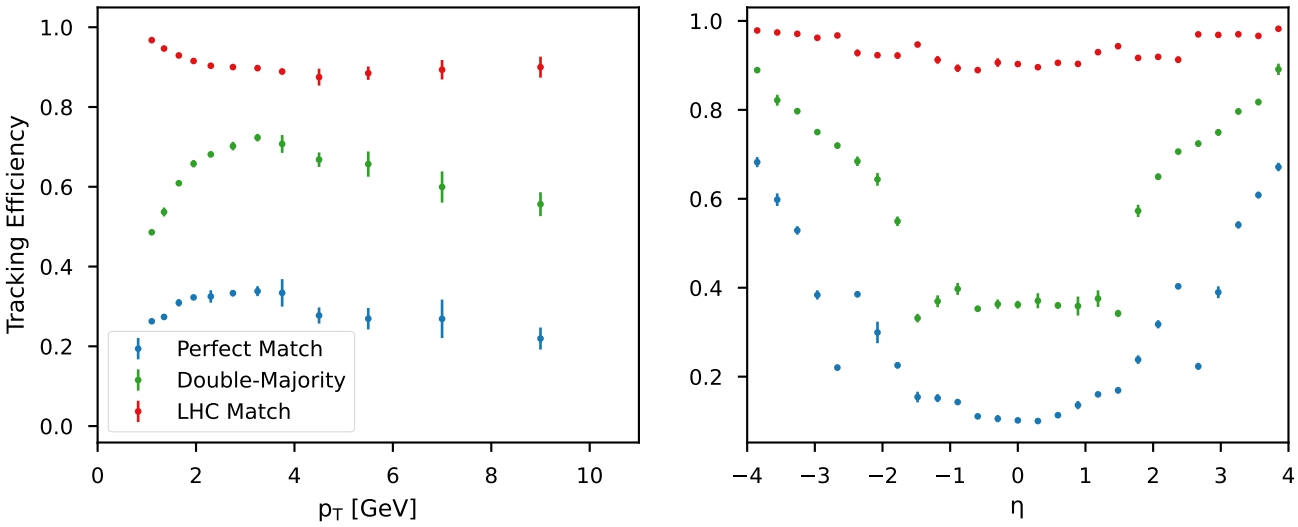
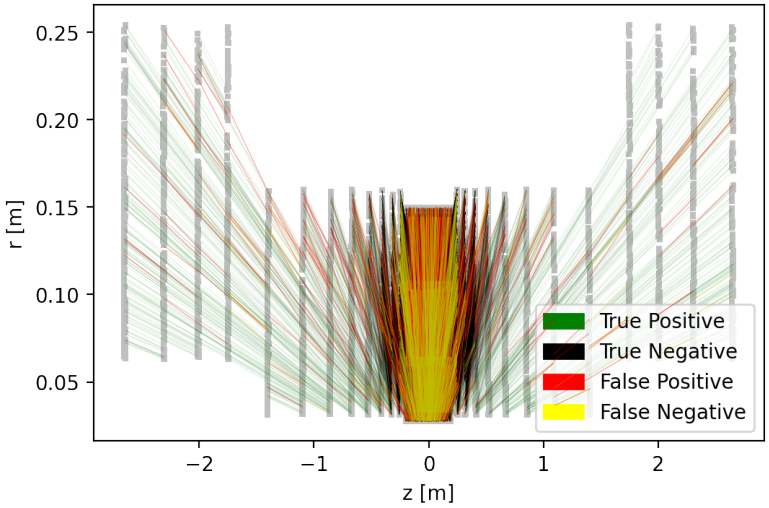


Figure 4.8: Tracking efficiency of a GNN using CMS MC data from the pixel detector. The η graph was made from tracks with a p_T above 2 GeV.

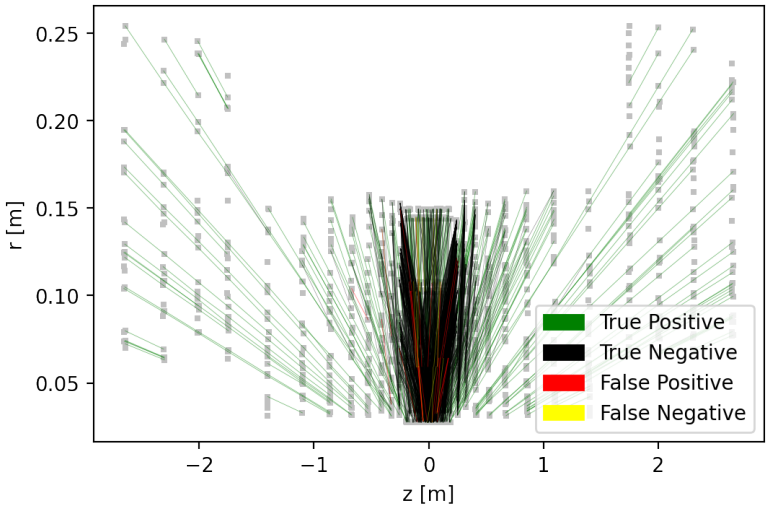
It would perhaps seem simple to extend our implementation from the pixel detector to the full detector, and to include all the hits in the detector. This is not feasible because of the number of edges. Let us illustrate this by considering a randomly chosen event. If we only consider hits that originated from tracks above a p_T of 2 GeV, the most populated layers that are connected contain about 800 and 500 hits. If we include all the hits, this number becomes 35 000 and 25 000. Even if me make good geometrical cuts on these edges, we would still first have to consider all the possible hit combinations. It is very simple to see that the combinatorics of this situation becomes prohibitive. Using the full detector also means creating very large graphs, which will often run out of memory on many CPU and GPUs. We will therefore build up to using the full detector and all the hits by considering some of the other simplifications first.

4.5.1 Allowing more than one hit per layer per track

So far, we have removed hits where there are more than one hit per layer per track. Reintroducing these hits creates a complication, illustrated in Fig. 4.10. In our simplified scenario, we would simply connect one hit from each layer, shown in the middle of the figure. This would create one edge. If we include all the hits, there are four to six possible connections, depending on whether we allow connections in the same layer. We can still label only one edge as a true



(a) Inference graph with $p_T > 1$ GeV



(b) Inference graph with $p_T > 2$ GeV

Figure 4.9: Edge classification of graphs at different p_T cuts.

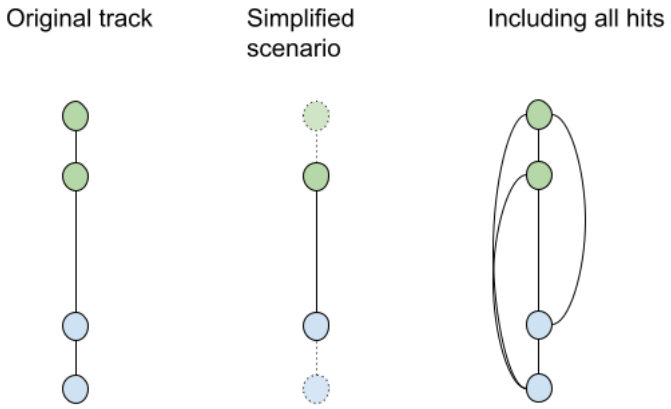


Figure 4.10: Allowing more than one hit per layer per track when building graphs can lead to building many more edges than necessary.

edge, but it would be hard to create geometrical constraints that would allow fewer of these edges without also eliminating true edges. As shown in Fig. 4.5 and Fig. 4.7, the number of edges is already a challenge, and we would like to avoid exacerbating the problem further. We can address this in several ways. One potential solution is to use take advantage of the new p_T modules in the outer tracker.

Using stubs from the outer tracker

Most tracks leave several hits in each layer because of the way the tracker is designed. In the outer tracker, this is even used as a feature in the new p_T modules, described in Section 2.8.4. If we use tracking stubs from the outer detector, we can avoid allowing more than one hit per layer per track while also avoiding using truth-level information. Since the tracking stubs also estimate the momentum of the tracks, we can even remove hits that come from a low momentum track without using truth-level information. While Level-1 tracking already uses the stubs, there are a few reasons why stubs are not typically used for the HLT. Firstly, some precision is lost by not using the full hit granularity. This could be amended by doing a track fit with all the hits after the GNN. Secondly, the momentum of the stubs is estimated by assuming that the tracks originated close to the beamline. The momentum of stubs from displaced tracks are often wrongly estimated. If it is underestimated, the track could be wrongly rejected. Since we are already using more restricting simplifications at this point, we shall ignore this for now.

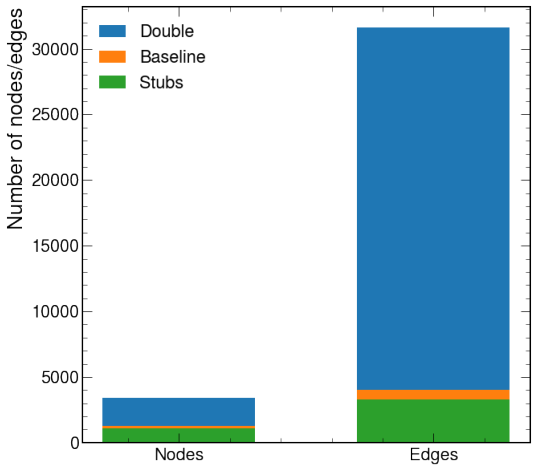


Figure 4.11: The number of nodes when building a graph for the outer tracker using a simplified approach, allowing double hits per track in a layer, and using stubs.

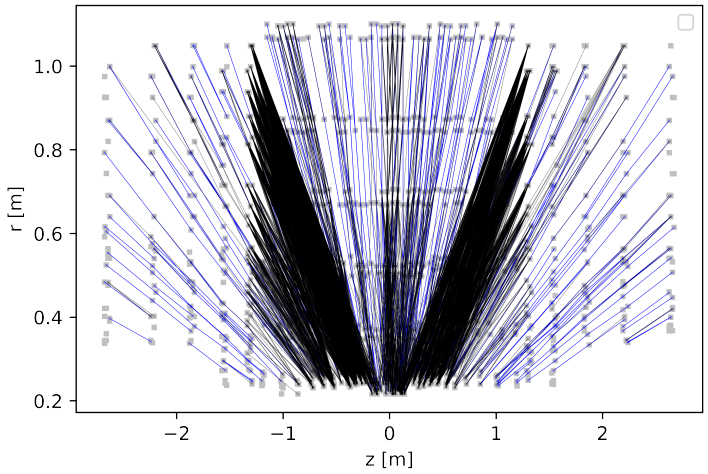


Figure 4.12: Building a graph in the outer tracker by removing hits from stubs with $p_T < 2$ GeV and only using stub hits. Black edges are false and blue edges are true edges

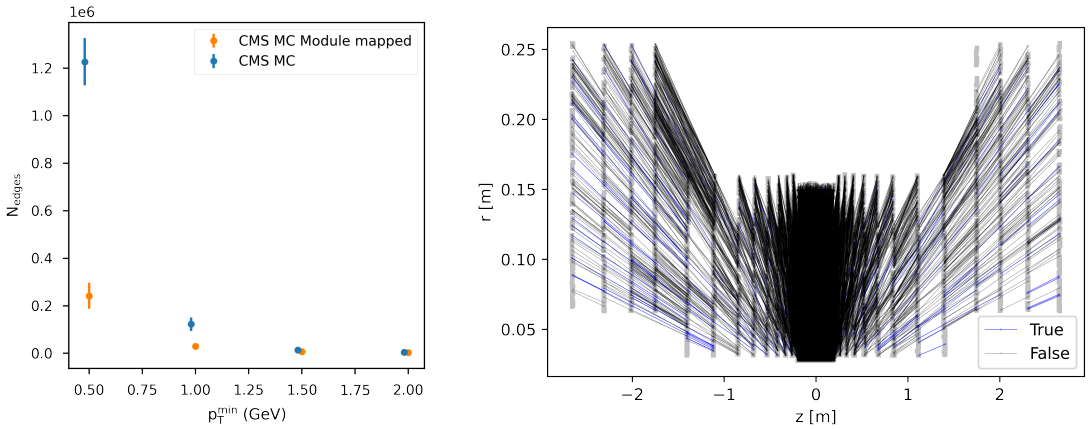
We will, however, keep in mind that a full implementation would need to address this. The results of building graphs with stubs from the outer tracker are shown in Fig.4.12.

Fig. 4.11 shows that allowing any number of hits per track per layer could be very problematic if we do not use the stubs. While the number of nodes roughly doubles, the number of edges is multiplied by roughly a factor of eight, due to the problem shown in Fig. 4.10 and because of the increased combinatorics. Using stubs is a good way to reduce the number of nodes and edges in a realistic way. Fig. 4.12 shows that the fake edges are most common in the transition from the barrel to the endcaps. This is likely because the allowed layer connections should ideally have been recalibrated for using the tracker stubs. Though certain connections might be common for

low p_T or displaced tracks, that might not be the case for the tracker stubs. We therefore have the potential of decreasing the number of edges even further. The edge classification score from using the stubs was similar to a scenario where low-p_T hits and double hits are removed using truth-level information, as shown in Fig. 4.14. Including more information from the stubs, like the angle or estimated momentum of the stub, might have helped the neural net learn faster. We chose not to use this, since we do not have this information in the inner tracker. We also do not have the option of using stubs in the inner part of the detector, so we might have to try another solution here.

Module mapping

When we set geometrical constraints on the built edges, we first have to build the edges, then remove the edges that do not meet our criteria. These constraints can be found in [139], and were made to decrease the number of fake edges while still retaining $> 99\%$ of the true edges. We also use the same geometric constraints throughout, even though it might be better if they were optimised for different parts of the detector. A way to address both of these points, and to leverage simulation data better, would be to consider which detector modules are typically connected by tracks. By considering a few hundred events, we can create a map of which modules are frequently connected to each other. When building the graphs, we can create edges only between these modules, potentially eliminating many fake edges. We could use the IDs of the modules in the detector for this mapping, but they vary in size. We will instead create a new mapping where every square centimetre within the detector gets its own module ID. We are essentially opening a window in the compatible layers, which is commonly used in CMSSW tracking. The main difference is that we can use past events to learn the size of the window instead of the window always having the same size. A simple module map for the CMS detector was created by considering 500 MC events. Only modules that were not in the same layer were allowed, and only particles with p_T above 2 GeV were considered. A threshold was set so that a connection between two modules must have happened at least two times in the 500 events, so as not to include connections from noise hits. The number of edges built in the pixel detector with and without the module mapping method is shown in Fig. 4.13. Since the



(a) The number of edges built for events in the pixel detector when using a module mapping and when not using it.

(b) The built edges in the pixel detector when including all hits and using a module mapping

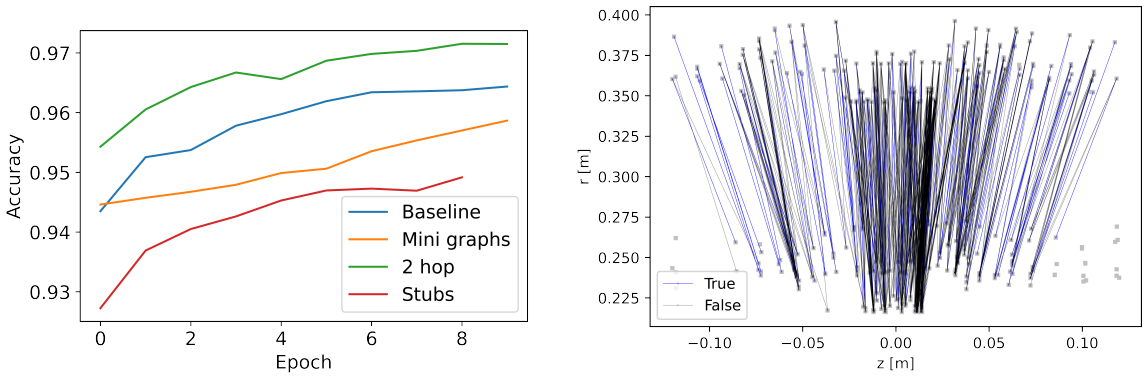
Figure 4.13: The effects of using a module mapping when building a graph neural net.

module mapping was created using tracks with a p_T over 2 GeV, we expect that the number of fake edges is very similar in the two scenarios for a p_T cut of 2 GeV. The difference gradually increases as we include more hits. At a p_T cut of 0.5 GeV, we have removed almost an order of magnitude of the fake edges.

The module mapping method is sufficient to be able to avoid using a p_T cut in the inner tracker. The graph of a such an event is shown in Fig. 4.13. The number of edges is still a problem because it takes at around 45 seconds to a minute to build a graph using one thread on a CPU. The ratio of fake to real edges is also around 98%. This creates a very large bias when training a classifier, since we would achieve a 98% accuracy by always predicting that the edges are false. We therefore would need a more sophisticated method to deal with the combinatorics in the inner tracker. We shall see in Section 4.8 that one group did this by considering module mappings for three layers at a time. Before we look at this work, we shall pause to consider some of the implications of our findings so far.

4.6 How much information is needed for tracking

The fact that we achieve a good edge classification performance for CMS MC data using the GNN from the Exa.Trkx collaboration has profound implications. This GNN is a one-hop GNN, so that a node only sees information from its directly connected neighbours. The edges are also directed, meaning that information flows from a source node to a target node. Each node will only see information about itself and directly connected hits in the previous layer. This means that, under the constraints we have used, we can get close to solving the tracking problem by only considering two layers at a time. We can confirm this explicitly by building graphs that only contain combinations of two layers. We shall call these minigraphs. The result is shown in Fig. 4.14. What the GNN architecture in this chapter learns is essentially the same as is shown to the right in this figure. The minigraph method performs about just as well as our baseline method. The small discrepancy can probably be explained by how the minigraphs are randomly sampled for each epoch. It is perhaps quite unexpected that we can accurately predict the next hit solely based on the previous hit position. If that were truly the case, we would not need to propagate several hit candidates in the CKF algorithm. Intuitively, two tracks could also leave a hit in the same position, but have a different next hit depending on the momentum and vertices of the track. Therefore, it is likely that we have simplified the problem to an extent where it does not actually solve the hard problem of distinguishing close hits based on the track parameters, but instead learns to eliminate hits that are further away. We can confirm this hypothesis by creating a GNN where we consider e.g. a two-hop neighbourhood. This means that each node contains information about the two previous layers. With three layers, there is enough information to implicitly contain information about the tracks. As shown in Fig. 4.14, the performance is only very slightly improved, indicating that our hypothesis is right; we have simplified the problem to the point of just showing that we are solving a different problem. The question then remains whether the GNN architecture is powerful enough to solve the more physical problem, where we include all the hits in the event. At this point, our work continued on to the work presented in the next chapter, but there was a group in the ATLAS collaboration that later took the GNN work further. We will return to this in Section 4.8.



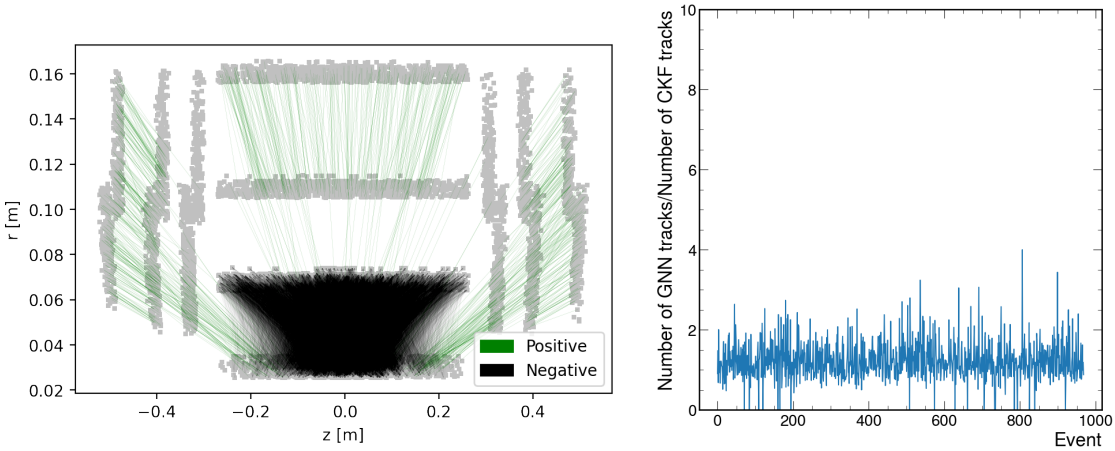
(a) Edge classification accuracy when building graphs in the outer detector. Minigraphs only see two layers at a time, and 2-hop uses the regular implementation, but allows a 2-hop neighbourhood. The stub method uses the stubs in the outer tracker.

(b) The minigraph method, where we only build edges between two layers for each individual graph.

Figure 4.14: Building minigraphs only has a slight decrease in performance compared to considering the full event at once, showing that the two methods are equivalent here. Using a 2-hop neighbourhood only slightly increases the performance.

4.7 CMS Run 3 data

In order to see how well the GNNs perform in a real scenario, we will apply our method to data collected in Run 3. The detector geometry in Phase 2 differs from that of Run 3. To fully implement the GNNs as we have in the rest of the chapter, we would need to retrain the GNN using MC data that has Run 3 geometry and create new module mappings. We would also need data from CMSSW that associates each of the hits to reconstructed CKF tracks. This could be quite time consuming. Instead, we will take a simpler approach. The Phase 2 pixel detector is very similar to that of Run 3, apart from the extended endcaps. The GNN trained on Phase 2 data used earlier in this chapter should therefore be quite applicable. If we only focus on the pixel detector, we can also build graphs without needing a module mapping. A sample was used that contained 1000 p-p events at an average pileup of 63, collected with standard p-p trigger configurations. Graphs were built from the hits, and the GNN trained on Phase 2 CMS MC data was used to classify the edges. An example event is shown in Fig. 4.15, which illustrates a sample of the built and classified edges in an event. All the allowed layer combinations have a similar amount of negative edges as the ones shown between the first two layers, though these are not plotted. It is very clear that the ratio of fake to real edges is a



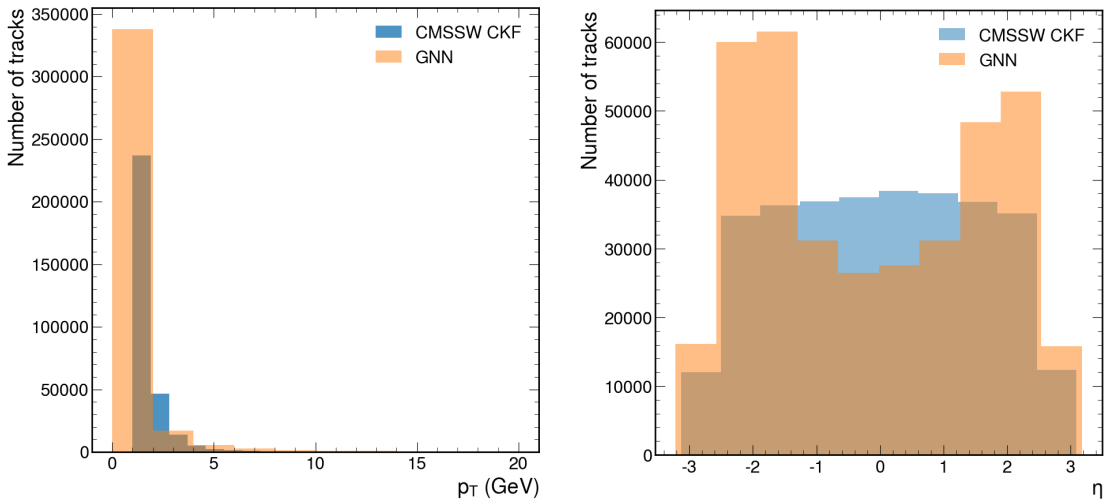
(a) A graph of the classified edges of an event from Run 3 data. Only some of the edges classified as negative edges are shown for illustrative purposes.

(b) Ratio of number of reconstructed tracks in events using the CMSSW CKF method compared to a GNN.

Figure 4.15: Tracking using a GNN for CMS Run 3 data. The GNN was trained on Phase 2 MC data.

large problem. Since we have no ground truth, we can not simplify the problem by artificially moving hits from the event, as we have previously. Each pixel graph took around 13 seconds to build on a standard CPU using one thread. Again, it is clear that the graph building is a limiting factor, both in run time and memory usage.

To evaluate the physics performance, tracks were made using DBScan. Track parameters were then estimated by fitting track parameters to the three innermost hits of the hit clusters. It would be more accurate to fit tracks to the entire clusters, but this approach allowed reuse of code for tracking used in Chapter 5. The difference should be negligible since we are only using the pixel detector. Since the GNN was trained to reconstruct tracks over 1 GeV, the GNN is compared to tracks from the CMSSW that are above 1 GeV. The physics performance shows somewhat positive results. The ratio of the number of tracks in each event of the GNN compared to the CMSSW CKF is shown in Fig. 4.15. The median value is around 1, showing that most often, the number of tracks reconstructed by the GNN is comparable to the number found in a CKF reconstruction. This is an indicator that the GNN can handle both the seeding and track building stage of the track reconstruction process without creating a very large amount of fake tracks. The number of tracks varies by about a factor of 2. A large part of this will be because we are comparing tracking done using only the pixel detector to tracking done with



(a) p_T distribution of tracks. (b) Distribution of tracks in η

Figure 4.16: Reconstructing tracks from a sample of 1000 events using the CMSSW CKF method and using a GNN.

the full detector. Distributions of the tracks in p_T and η are shown in Fig. 4.16. We can see that the GNN method still reconstructs tracks that have a low momentum, even though it was trained to reconstruct higher p_T tracks. This could be addressed by e.g. performing cuts on cluster p_T. There are also more tracks reconstructed with higher η for the GNN than the CKF method. Considering Fig. 4.15, it could seem like this is due to many false negative edges between the second and third barrel layers. Having a several-hop neural net could help address this. Another improvement would be to recreate these plots using the Monte Carlo data from Section 4.4. If those plots look dissimilar to the ones shown here, it is likely that the difference in geometry is the cause of the difference in reconstructed tracks between the GNN and the CKF. Overall, there seems to be a reasonable agreement between the GNN and the CKF methods, with deviations that can easily be addressed.

4.8 Related work

The GNN architecture used in this chapter has been applied to simulated data from the ATLAS detector [142]. This group used a pileup of 200 and were able to use the full detector and hit granularity. They had two different approaches. One is the module mapping, similar to the one

described in Section 4.5.1. Instead of just considering two-module connections, they consider sequences of three modules. This takes slightly longer to run, but can eliminate many more fake edges. The second method is metric learning. This method embeds the hits into a latent space, where hits belonging to the same track are close to each other. Both of these methods make it possible to use all the hits and consider the full detector at once. The methods have on the order of 300 000 nodes and a few million edges per event. The number of nodes is on the same order of what is seen in the CMS detector without removing any hits. Instead of only using a single-hop neural net, they have used a 8 hop-neural net, and they achieve a very good physics accuracy. Though they have not officially reported on how long it takes to build the graphs, the number of edges is around the same number of edges as we see in the pixel detector at a p_T cut of around 0.5 GeV in Fig. 4.13. It is therefore safe to assume that the graph building is still a limiting factor. An eight-hop neural net will also be somewhat slow when it runs the inference on the graphs. This work has not yet been applied to CMS, but it is very likely that it would show similar results. The CMS collaboration instead focused on integrating the GNN workflow into CMSSW. There was a hackathon aimed at this, that built many parts of the necessary pipeline [143].

Another interesting GNN approach for tracking is the HyperTrack method created by Mikael Mieskolainen [144]. This is a pipeline with both clustering, a GNN and a transformer. During training, a K-means algorithm is used to split the tracker space into V segments, or voxels. A connectivity matrix is created based on which voxels are typically connected to each other. During inference, hits are assigned to voxels based on a distance metric. If two voxels are connected, all the hits in each voxel will be connected to each other. A GNN is used to score each of the edges, and low scoring edges are removed. Nodes also have self connections, and a low scoring self connection means that the node is a noise hit. A weakly connected component search is then done on the edges and nodes. Weakly connected components simply make subgraphs based on which nodes are connected while ignoring the direction of the edges. This creates a set of smaller, disconnected graphs. Finally, a transformer is applied to cluster hits within the subgraphs into tracks. This was applied to the TrackML dataset with the pileup reduced to an average of 60. This achieved a double majority score of 0.83.

4.9 Summary

In this chapter we have seen that GNNs can perform edge classification between tracker hits very well if we apply a few simplifications, mainly removing hits that originate from a particle with a p_T below a certain threshold. If we use all the hits, the combinatorics explodes. There are a few ways to mitigate this. We can take advantage of the tracking stubs in the outer detector to only consider hits over a certain p_T threshold without using truth level information. We can also create a mapping of the modules in the detector are usually connected to each other, and only build edges between these when we build our tracks. The main problem that seems unsolved so far is that creating the graphs takes a prohibitively long time due to the combinatorics of the problem. The graph neural net is somewhat large and we still have not addressed missing hits in the detector. We shall therefore seek an entirely new approach, presented in Chapter 5.

Chapter 5

Reinforcement learning for tracking

Learning from feedback in an environment is perhaps our most intuitive way of learning, informing our behaviours since birth. Reinforcement learning (RL) is a computational implementation of this paradigm. An agent acts in an environment and receives rewards or punishments based on its behaviour. RL has been an area of research since the 1950s [145]. Since then, it has outcompeted humans in games like chess and Go [146], and has recently become the underpinning of hugely successful large language models [52]. Good candidates for reinforcement learning typically have three identifying features - the dynamics of the system are sequential and intractable, one has access to large amounts of data to train on, and the agent's actions affect the environment that the agent experiences.

Particle tracking fulfills these criteria. We can envision an RL agent reconstructing a track by gradually adding hits to a track. This is sequential, and the hits that the agent sees will depend on the previous steps. The problem is intractable, because there is no way to know a priori which hits are best without exploring the alternatives. There is also a large amount of Monte Carlo data available on which to train. RL has the potential to address the limitations of graph neural nets outlined in the previous chapter. While even the best tuned graph neural net will have to rebuild fake edges, RL could avoid much of the combinatorial problem by accurately predicting where the next hit is. While there will always be sources of randomness like multiple scattering, RL might remove a large part of the combinatorial problem. Whereas tracks have

to be built after the GNN has classified each edge, RL could reconstruct the track parameters as part of the state description. Missing hits could also be less of a problem since edges don't have to be connected as in GNNs. Because RL uses a reward, there is a lot of flexibility to tailor the algorithm to the specific needs of a problem. Tracking with RL also has the potential of massive parallelism since many agents could operate independently at the same time.

RL rephrases ML as choosing an action, a, given a state s. A chosen action will have an effect on the environment, changing the state. This is repeated for a trajectory, τ. The trajectory can have a fixed end-goal, like a winning state in chess, or a fixed number of steps. Each set of steps from a start to an end state is known as an episode. It is also possible to have an infinite horizon, like a walking robot, so that the episode never ends. The goal is to learn which actions to chose in order to maximise the reward for the episode. Rewards can be given for each step, every few steps, or at the end of an episode. It is important to not just consider the immediate reward, since we often have long-term goals. In chess, one might for instance sacrifice a piece, which would typically carry a low reward, but win the game, which should lead to a high reward. We typically require that the environment is Markovian; choosing the best subsequent action only depends on the current state. If this is not the case, choosing the best action depends on information that the agent does not have, making learning difficult. The actions can be chosen from a certain number of allowed actions - a discrete action space, or from a continuous distribution - a continuous action space. Which of these two spaces we choose will determine how we learn, and we shall see that some algorithms are designed for only one of the two.

If we have a continuous action space, it is useful to learn a policy π that dictates the probability that the agent will take an action given a certain state at time t;

$$\pi(a, s) = \Pr(A_t = a \mid S_t = s). \tag{5.1}$$

The policy is often a function, like a Gaussian distribution, whose parameters are learnt by using a neural net. This is discussed further in Section 5.1. The optimal policy, π^*, is the one that maximises the expected reward, R, of the agent's trajectory. The RL problem can then

be written as;

$$\pi^* = \arg\max_{\pi} J(\pi), \qquad J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \underset{\tau\sim\pi}{E}[R(\tau)], \tag{5.2}$$

where $\underset{\tau\sim\pi}{E}$ is the expectation value when the trajectory follows a policy π. RL algorithms that only try to learn the optimal policy are known as policy gradient methods. This will be discussed further in Section 5.5.

In discrete action spaces, it is often more useful to learn the quality, $Q(s, a)$, of the possible actions in a state. The quality is typically a measure of the reward and a the potential reward in the next step. Learning the quality is often done by training a neural net that takes states and actions as inputs and outputs a quality score. The chosen action is simply the action with the highest quality. There are also methods that combine policy and Q-learning; a policy will recommend an action, and the quality function will estimate its quality. These are called actor-critic methods. Generally speaking, policy methods are suitable for continuous action spaces, Q-learning for discrete spaces, and actor-critic methods for either. There are also adaptations to certain methods that make them suitable for either action space.

Another defining feature in RL is whether we have to learn a model of the environment. An RL model learning to play a PC game might for instance learn how to simulate game dynamics instead of having to directly interact with the game. Learning a model is often time-consuming, and is not applicable for tracking, since we already know a lot about the environment, like how a hit affects the track parameters. There are also some model-based methods where the model is not learnt, like AlphaZero [146], which uses a model to simulate game-play. The main advantage is that the agent can plan ahead by simulating what would happen in many different scenarios. Though this has similarities with tracking, we shall see that there are certain attributes of our problem that makes this less attractive. We shall therefore focus our efforts on model-free RL approaches, which is generally more common and well-explored. A taxonomy of some of the common RL methods is shown in Fig. 5.1. In accordance with the no free lunch theorem, many of these models will be explored, since there is no way to know which will be the best model without trying a few options.

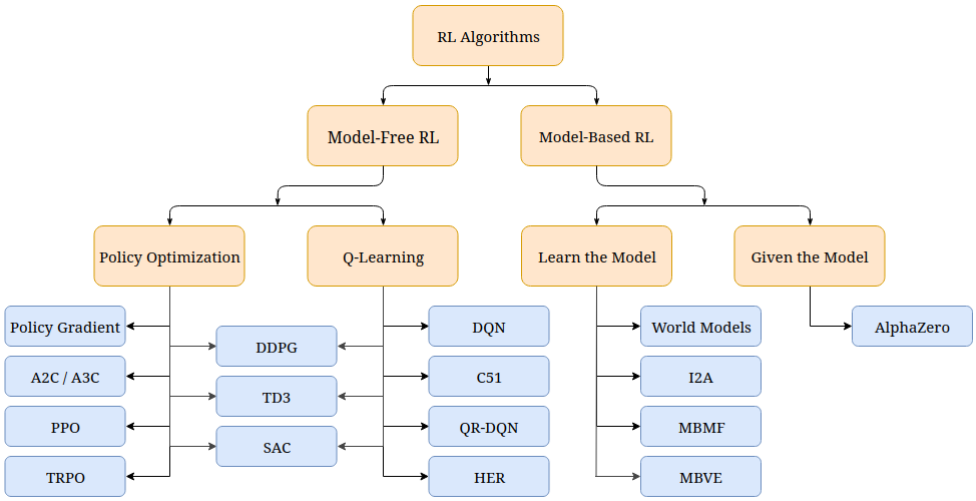


Figure 5.1: A taxonomy of common RL techniques. Most commonly, we do not learn a model of the environment. Often, we instead try to optimise the policy that dictates our actions and/or learn the quality of the possible actions. Figure from [147]

5.1 Policy gradient methods

Policy gradient methods focus on learning a policy without estimating the quality of the actions. They usually do so by learning parameters of a function, as illustrated in Fig. 5.2. The policy can in general be any differentiable function. It is preferable that it does not become deterministic, to avoid that the agent stops exploring the phase space. Gaussian policies are common for continuous action spaces, and the parameters are typically learnt using neural nets. Policy gradient methods are particularly useful when dealing with large action spaces, or continuous action spaces that have infinite possible actions. They have been shown to be very stable, and also give much stronger theoretical guarantees for convergence than quality-based methods. This is largely due to the policy gradient theorem, which gives a method for calculating the gradient of the policy without considering how the policy affects the states that the agent will see. The objective function of a policy is usually a measure of the cumulative reward. This aligns the learning directly with the goal, and addresses one of the limitations of learning quality functions; a small change in the quality function can lead to very different actions, which can prevent models from converging. This again means that there are stronger convergence guarantees for policy methods than for action-quality methods. Because policy improvement takes place in small steps, they can be sample inefficient and show slow convergence. Since the

policies tend to get less random as they learn, the can also get stuck in local minima.

5.1.1 Vanilla policy gradient

A vanilla policy gradient (VPG), also known as REINFORCE or Monte-Carlo policy gradient [148], is often a logical starting point in RL since it is a very simple algorithm. Starting from our general expression of the RL problem in equation 5.2, it can be shown that the gradient of the loss function J, is

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right], \tag{5.3}$$

where we are summing over the time steps, t, in the episode. The update of the policy is then given by the formula shown in Fig. 5.2. The loss function has some intuitive value, since the gradient is directly proportional to the expected return of the trajectory. This is estimated from the samples we collect, which makes VPG a Monte-Carlo method. Such methods often have high variance, which can produce slow learning. A way to stabilise this is to introduce a baseline. Instead of considering just the reward, we can consider how the reward compares to what we usually experience. A value function measures the reward we expect for the trajectory if we start in a certain state and follow the current policy;

$$V^\pi(s) = \underset{\tau\sim\pi}{E}\left[R(\tau)|s_0 = s\right]. \tag{5.4}$$

This value function can be be subtracted from the reward function to reduce the high variance. This is sometimes called the advantage function because it measures a relative advantage. If the difference between our reward and the value is high, we have an important step, so a larger step is taken in the gradient descent. Since it would be very time consuming to calculate the value function for each state, it is typically estimated with a neural net.

The VPG methods used in Section 5.5 are intended to be very simple, and do not use a baseline or a value function. Instead, they follow the simple method shown in Fig. 5.2. Trajectories are collected by running the policy in the environment, sampling actions from the Gaussian

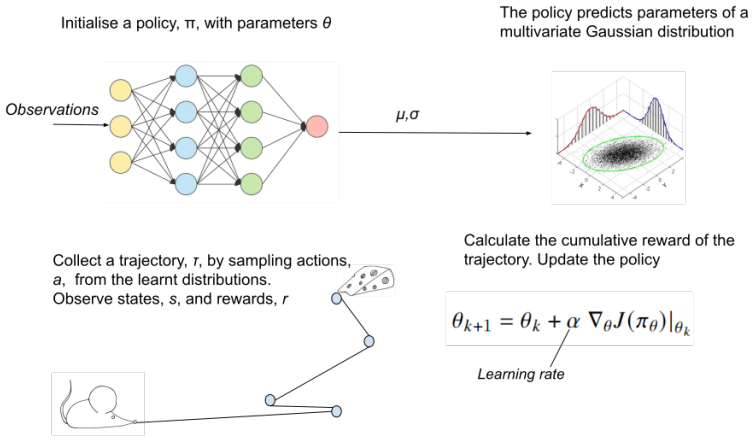


Figure 5.2: A standard vanilla policy gradient algorithm using a Gaussian distribution.

distribution. The policy is updated by considering the cumulative reward of running the policy. Since the actions are stochastically sampled, it ensures that exploration continues, while exploiting learnt information by decreasing the standard deviation over time.

5.1.2 Trust region policy optimization

A common problem in VPG is that small changes in parameter values can lead to significant behaviour changes, to the point that the policy collapses and starts performing poorly. This can be caused by having a lot of parameters to learn, complex environments, or the fact that we typically consider only one sample before updating the policy. To avoid a policy collapse, VPG needs a small learning rate. Trust region policy optimization (TRPO) aims address this limitation by letting the agent take the largest step possible within a trusted, safe region [149]. This is done by introducing a new loss function;

$$L_{surrogate}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)} \hat{A}_(s_t, a_t) \right], \tag{5.5}$$

where $\hat{A}$ is the advantage function. Maximising the surrogate loss function ensures that the expectation of the new policy is larger than the previous one. The maximimsation is usually

done under a constraint of the Kullback-Leibler (KL) divergence between the new policy and the old one. KL divergence is a measure of the similarity between two distributions, or more formally, the information lost when using one distribution to approximate the other. We therefore make sure that we have a better policy, that is not too far away from our previous policy, avoiding a policy collapse.

5.1.3 Proximal policy optimisation

Calculating the constrained optimisation in TRPO can be computationally expensive, complex, hard to parallelise and sensitive to the size of the trust region. The proximal policy method (PPO) aims to reduce these drawbacks implementing the TRPO method in a simplified way [150]. PPO uses a similar loss function to TRPO, but changes the KL constraint. There are two main strategies of doing this - one creates a penalty term and one uses a clip method. We will focus on the PPO-Clip method, which is more common, and is known for its simplicity. Instead of imposing a KL constraint, it clips the ratio between the new and old policies. Using the gradient of the KL constraint is complicated, and clipping the ratio of the policies instead lead to a simpler algorithm. It takes only a few lines of code to change a VPG to a clipped PPO. Clipping the ratios encourages a smooth policy, making it generally more stable and with a higher sample efficiency than TRPO. Since it is simpler than TRPO, it is more easily scaled, and can handle high-dimensional action spaces better. It has demonstrated robust performance across a wide range of tasks.

5.2 Q-learning methods

An early breakthrough in RL was the development of a method called Q-learning [151]. Q-learning usually does not learn a policy, but instead learns the quality of state-action pairs, and chooses the actions that have the highest quality. In a simple scenario, Q-learning could tabulate all the possible different state-action pairs and store the quality of each combination. In reality, the state-action space is usually too complex to be exhaustively explored. It is therefore

common to estimate the Q-function with a deep neural net. This is known as Deep Q-Learning, or DQN [152]. There are variations on this that e.g. emphasise the idea of exploring novelty (C51), do not predict just a single Q-value (QR-DQN) [153] or are specialised for the sparse rewards of high-dimensional action spaces (HER) [154].

The methods discussed so far have been on-policy methods, learning from enacting a current policy. Off-policy methods can instead learn by sampling from different policies. Though DQN does not usually learn a policy, it still uses a policy; choosing the action with the maximum value. Instead of always considering the most recent samples, experiences are stored in a replay buffer. The neural net learns by drawing random samples from this buffer. Since the parameters of the neural net used for these episodes were different, this is an off-policy method. Off-policy methods often show a good sample efficiency since they can reuse experience from a a variety of policies. They are often suitable for situations where exploration is challenging, since it separates exploration and exploitation by choosing random actions at a given frequency.

The policy gradient methods generally use Monte Carlo sampling, where the expected return is averaged over sampled trajectories. Q-learning instead uses the temporal difference (TD) method, which means it learns the quality by considering the difference between the current estimate and a bootstrapped estimate of the quality of the next state. This can be written as;

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s', a') - Q(s_t, a_t) \right]. \tag{5.6}$$

We measure the quality by the current reward and the highest reward attainable from the next action, a' in the subsequent state, s'. We discount the reward in the next state by a discount factor γ since it is generally less important than our current reward. We then subtract the current quality from this to get a measure of the difference of our new quality. We multiply this term by a learning rate and add it to our previous quality estimate. Because it bases the update on an existing estimate, it is known as a bootstrapping method. A challenge with this formula is that the target is constantly changing as the Q-value updates. This problem is usually solved by having a target network. This is a copy of the original network that updates

less frequently than the Q-network, which stabilises the training. Q-learning furthermore often overestimates the Q-values, since overestimating the value would create a compounding error by adding the term to the previous estimate. The target network mitigates this by letting us use a Q-estimate from the target network. This is called double DQN (DDQN) [155]. All the DQN methods implemented here use DDQN.

5.3 Actor-critic methods

Actor-critic methods combine the two methods of learning a quality and a policy. This can in theory combine the strengths of the two methods, creating both stable and efficient models.

5.3.1 Deep deterministic policy gradients

Deep deterministic policy gradient (DDPG) is a somewhat newer algorithm than DQN or VPG [156]. It is an actor-critic method that extends DQN to a continuous action space. As was seen in Equation 5.6, learning a state-action quality relies on a term that selects a maximum reward from the possible next actions. This is near impossible to use in a continuous space since there are infinite possible states. DDPG instead uses a policy to select the best action. Instead of trying all possible actions, it learns the qualities of the actions chosen by the policy μ, so that $Q(s,a) \approx Q(s,\mu(s))$. DDPG uses a deterministic policy, so noise is added to the actions to encourage exploration. DDPG can achieve great performance, but it can be unstable, and can overestimate Q-values.

5.3.2 Twin delayed deep deterministic policy gradient

Twin delayed DDPG (TD3) addresses the brittleness and hyperparameter sensitivity of DDPG by using three tricks [157]. Firstly, it learns two Q-functions, thereby the name *twin*. It uses the smaller of the two Q-values to avoid DDPG's issue of overestimating the Q-value. It uses delayed policy updates, so the Q-function updates more frequently than the policy. This again

works to protect the policy from breaking due to wrong Q-values. Finally, it adds noise to the target action so it is harder to exploit overestimation of Q-values. This has shown to provide a consistent improvement over DDPG. Similarly to DDPG, it is suitable for continuous action spaces.

5.3.3 Soft actor critic

The soft actor critic (SAC) method was published around the same time as TD3 [158]. It aims to directly address one of the issues particular to RL; the trade-off between exploration and exploitation. An agent must find the balance between exploiting what it has learnt and exploring new actions so that the learning does not stop. SAC does this by using a stochastic policy and maximizes a trade-off between expected return and entropy. This is done by modifying the policy, shown in Equation 5 to include an extra term that measures how random the policy is. Like TD3, SAC also learns two Q-functions and uses the lower of the two to choose the action.

5.4 Summary of common RL algorithms

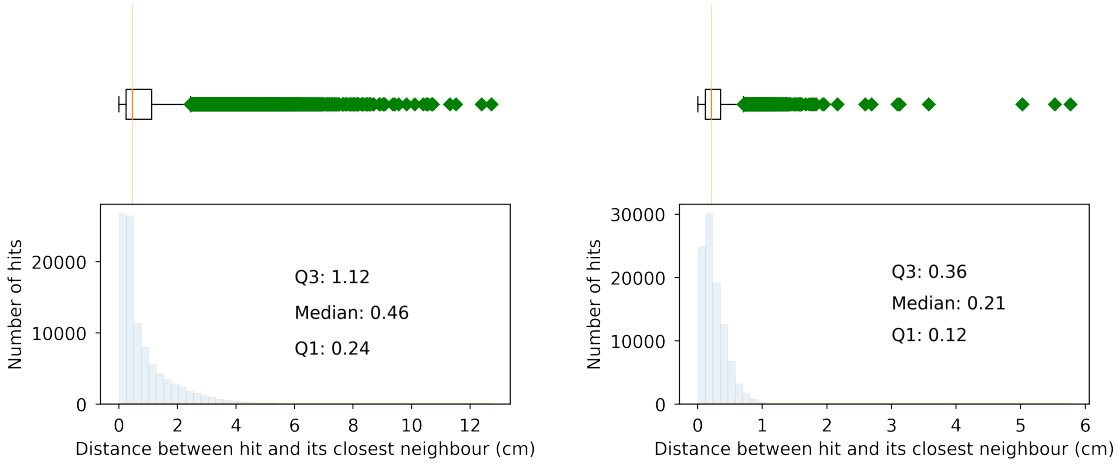
We have seen that there are three main approaches to RL; learning a policy, learning a quality function, and using both in an actor-critic method. Among these, DDPG, TD3 and SAC are typically appropriate only for a continuous action space and DQN only for discrete action spaces. The remaining methods are appropriate for either. There have been several studies that compare the methods on a variety of tasks [159], [160]. As is the case with ML in general, starting simple is usually a good idea. This is especially true in RL, where the environment can be hard to debug, and the learning can be very sensitive. Rather than always choosing what the literature suggests is the best performing or most modern methods, we shall instead aim to start as simple as possible and only add complexity as needed. This is also in keeping with our goal of finding a simple algorithm that can easily be accelerated using a co-processor.

5.5 Tracking in a continuous action space

We now need to recast the tracking problem in the framework of an agent acting in an environment. One of the most important decisions is whether to represent the environment as a continuous or discrete action space. This decision influences which models can be used and what the model learns. In theory, there is a finite number of states in the detector since the modules have a finite granularity. In practice, this granularity is so fine that there are hundred of millions of accessible states. An easy starting point is therefore to use a continuous action space where the agent can predict the position of the next hit. We start with a two-dimensional space because it is easier to learn than a three dimensional one, the tracks go in a straight line, and the GNNs in Chapter 4 showed that it is sufficient for tracking. The state is most easily described by a hit position. In order to be Markovian, the state also needs information about the previous hit or estimates of the track parameters. The reward can be given based on how close the agent is to the correct hit position. To get a sense of how precise such an algorithm needs to be, the distance between the hits in a CMS MC and a TrackML event were studied. This is shown in Fig. 5.3. The CMS hits are on average around half the distance apart compared to TrackML. If we wanted to predict the correct hit for CMS more than 75% of the time, we would have to predict to within 0.1 cm. Given that the r-z tracking space is around 600×120 cm, achieving a level of precision of this order could pose a challenge.

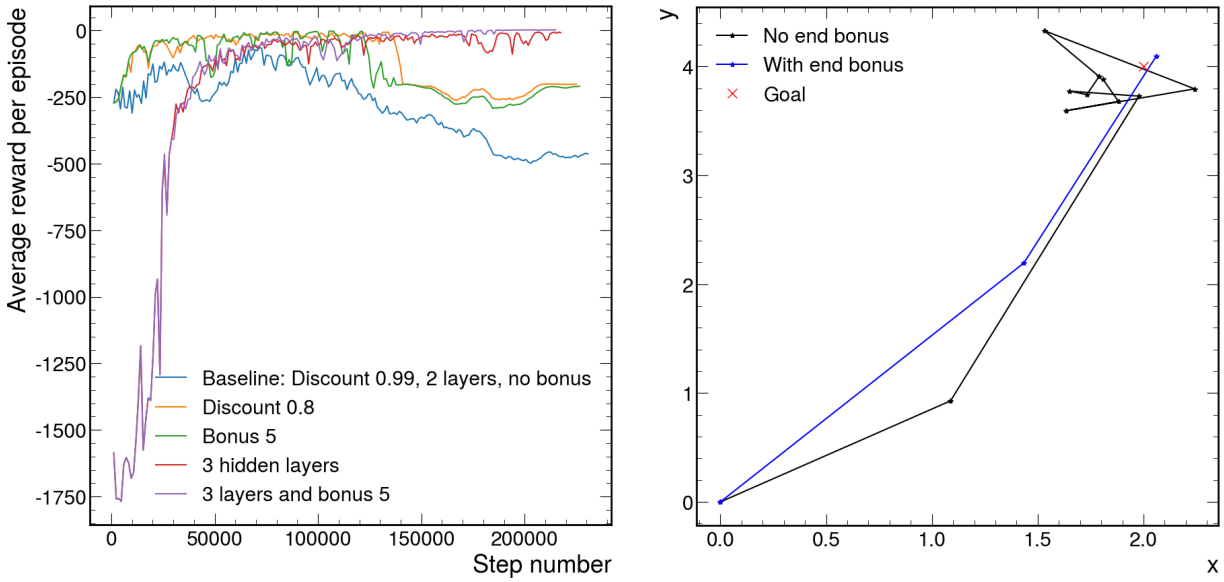
5.5.1 Learning to navigate to a point

To break down the complex task of learning particle paths, we can first learn how to navigate to a point. An environment of size 5 by 5 in arbitrary units was created. It had a starting position at $(0, 0)$ and a goal at $(2, 4)$. Each episode continued until the agent was within 0.1 from the goal. The observation was the two-dimensional position. A few experiments using a VPG with varying hyperparameters is shown in Fig. 5.4. The baseline policy here has two hidden layers of 32 neurons each. The reward discount was 0.99, and there was no additional bonus for reaching the set goal.



(a) TrackML (b) CMS

Figure 5.3: The distribution of the distances between hits and their closest neighbours in sample events from TrackML and from CMS data.



(a) Learning curves of a VPG algorithm trying to navigate to a goal

(b) The path of two different strategies trying to navigate to a goal

Figure 5.4: Learning how to navigate to a point. Rewards (left) and path taken (right). The higher the reward, the closer the agent is to the desired point. The lower the episode reward, the longer and/or further away the agent spends from the goal. An episode starts from a starting point and ends when the agent is within a certain distance from the end point. An extra bonus can be added to the reward when reaching the end goal.

Fig. 5.4 shows that it is very possible to learn to navigate to a point using a simple VPG algorithm. We can see that some of the parameters lead to the policy collapse that was discussed in Section 5.1.2. Adding an extra layer to the neural net seems to be a potential solution, likely because it can hold more information. Decreasing the reward discount also seems to help the agent learn faster. This is because it weights the future rewards more, so that it can better learn which is the correct direction. Finally, adding a bonus to the reward for reaching the end goal has an interesting effect. This seems to have little effect on the average reward, but it has an effect on the learnt strategy. This is shown in Fig. 5.4 (b). Without a bonus for reaching the end, the agent quickly gets close to the goal, but navigates around it for a long time. With the end bonus, it goes very directly to the goal. The incentive for finding a quick solution is simply stronger. This is important when reconstructing tracks, since one would want only true hit positions, not nearby positions that are not associated with the right hits. Similar experiments were also done when starting from a random point, or navigating to different starting points based on two different starting positions. They all showed that a simple VPG algorithm was sufficient.

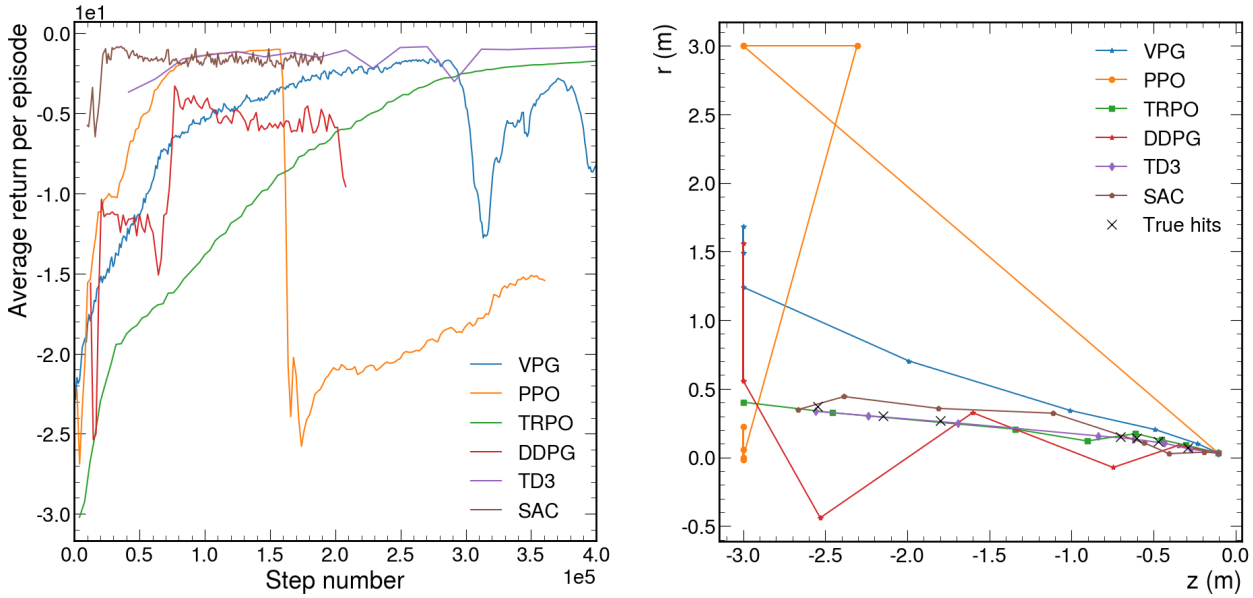
Learning the trajectory of one particle

Having learnt how to navigate to a point, we can continue to learn the trajectory of a single particle. First we need to adjust the boundaries of the environment to reflect the size of the detector. We will again be using data from TrackML, and we create the following environment:

$$E = \{(z, r) \mid z \in [-3, 3],\, r \in [0, 1.2]\} \quad A = (\Delta z, \Delta r) \in [-3, 3]. \tag{5.7}$$

The environment is described in terms of metres because it is more numerically stable in the environment due to how the network is initialised. The episode ends after five predictions have been made. The reward is the negative distance between the predicted point and the correct hit position. The starting point is the position of the first hit.

Fig. 5.5 shows that several algorithms can learn approximate hit positions of a single track. VPG is subject to a catastrophic policy update, as we have already seen can happen. Somewhat



(a) Learning the positions of hits in a track using different algorithms

(b) Predicted hit positions of a track using different algorithms

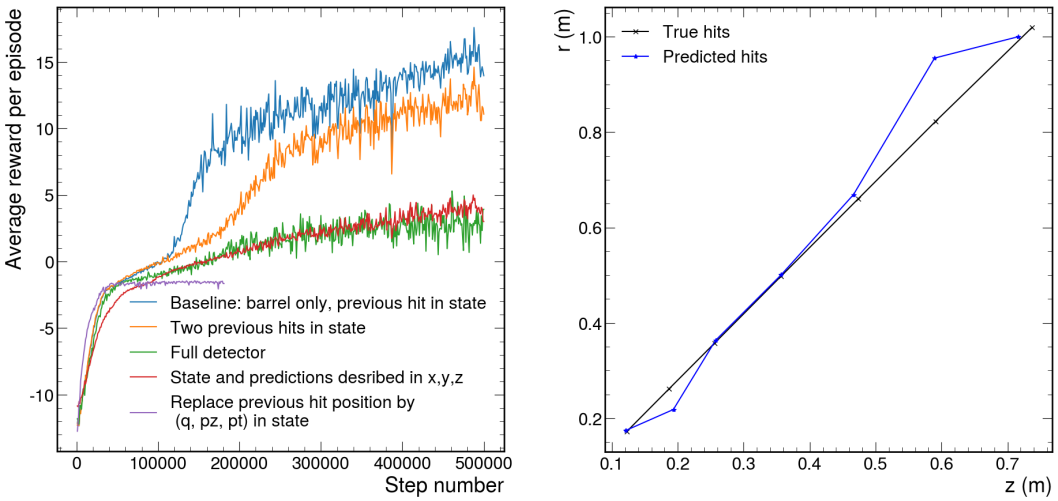
Figure 5.5: Learning how to reconstruct a single track. Rewards (left) and path taken (right).

surprisingly, this is the case with PPO as well. The hyperparameters were not tuned for these algorithms, and standard implementations were used. It is very likely that the clipping region of PPO should have been constrained more to avoid a catastrophic update. TRPO and TD3 both seem to reconstruct fairly good hit positions in their out-of-the-box implementations.

Learning the trajectories of multiple particles

To learn the trajectories of multiple particles, the state description needs to be changed. Given that an agent is at a certain hit, where it should go next could vary depending on the track. The simplest way to extend the state is to include the position of the previous hit. This assumes that where the track will go next is only dependent on the current and previous hit position. This is a simplification, but should hold true to a rough approximation. This approach has the very large benefit that it does not require any information about the detector geometry or particle tracking. Both should be implicitly learnt. A good RL agent will implicitly learn the location of the layers by observing its rewards. It does have a large action space of "forbidden" actions that it will have to learn to eliminate.

To simplify the problem, we start by considering particles that have a $p_T > 2$ and have at least



(a) Learning to reconstruct barrel tracks (b) Example of performance on a test track

Figure 5.6: Fig (a) and (b) demonstrate how an algorithm can achieve decent performance when predicting next hit positions in the barrel. The algorithm generally seems to be better at predicting hit positions closer to the starting point. The errors of the predicted points in (b) are not displayed, as they simply reflect the learnt standard distribution of the action sampled, as described in Section 5.1.1. The uncertainty is therefore a product of the training time and hyperparameters, and is not of particular interest at this point.

seven hits after removing double hits in any one layer. We will predict five hit positions for each track and end the episode after this. We also start by considering tracks where all the hits are in the barrel of the detector. This was first tried using a TD3 algorithm. Though this seemed to learn, it quickly got stuck in a local optimum. It achieved an accuracy of about 25%, but the correct hits were always the first two hits. This clearly shows that the exploration/exploitation strategy was insufficient. Though this might be abated with appropriate hyperparameter tuning, there are many models to choose from, so a TRPO model was chosen. From the experiments in Section 5.5.1, this was very stable and reconstructed the track well. This showed promising results when tracking for one particle. The results are shown in Fig. 5.6. Based on the results in the previous section, a reward of 5 was given when the prediction was within 1 cm of the true hit position. The maximum reward is therefore 25.

Fig. 5.6 shows that the TRPO algorithm can learn a decent approximation of where the hits in a track are based on the previous hit position. Fig. 5.6 (a) shows a few variations of the experiment. Including more information about the track, in the form of track momentum, including two previous hits instead of one, or describing the hit positions in x,y,z-dimensions

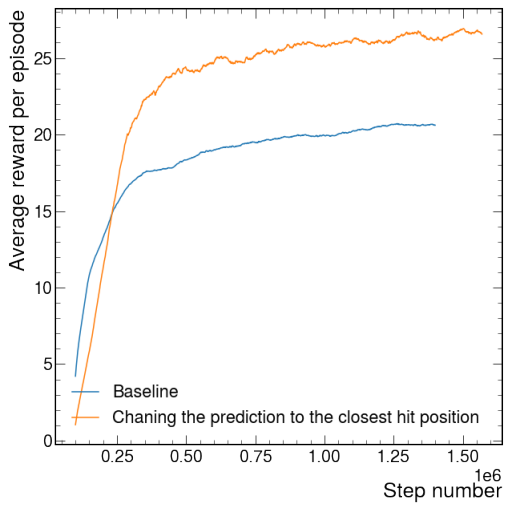


Figure 5.7: Learning to reconstruct tracks in the TrackML barrel while changing the prediction to the closest hit. The baseline refers to the same TRPO structure used in Fig. 5.6. The orange line has two modifications. Firstly, the hit prediction is corrected to the closest hit position. There are also extra rewards given as the agent gets closer to the correct hit.

seems to hinder learning. This is likely because there is more information to learn, and the algorithm is not at a sufficient precision where this extra information is useful. Reconstructing tracks in the entire detector instead of just the barrel is naturally also much more difficult. Fig. 5.6 (b) shows an example of a reconstructed track.

If we run the algorithm for a long time, the learning saturates. It is difficult to encourage it to get close enough to the correct hits when it does not know where the observed hits are. This could be addressed by reward shaping; there could be several bonus points added to the reward as the predictions get closer to the correct hits. An extra bonus of 10 is given if the agent is within 0.1 cm of the correct hit. We could also try to correct the predicted position to the closest observed hit position. This is a common way to deal with large, discrete action spaces. The result is shown in Fig. 5.7. Though these measures improves the performance, it does not meet the precision requirements we outlined in Section 5.5. We might be able to improve the model by using different models, hyperparameter tuning and describing the state differently. It nevertheless seems that it would be a challenge to get to achieve the required precision since we are quite far away from it, and the learning seems to saturate.

5.6 Discrete action space

We have seen that RL has the potential to predict an approximate hit position, but it struggles to achieve the required precision. So far, the RL algorithms have not used any information about the observed hit positions when it makes its predictions. This is clearly not a good use of the available information. Perhaps the RL algorithm could instead distinguish between possible next hits. This could be used in conjunction with e.g. a Kalman filter; when there are several possible next hits, the RL agent could choose the best one, avoiding the combinatorial problem altogether. Since we are now choosing between a set of actions, we have a discrete action space. Discrete action spaces typically have a fixed set of actions it can choose between, and it evaluates all the possible actions at any given time. The actions could be something simple, like navigating left/right, or something more complex, like chess. Our problem might seem similar to chess since there is a large set of possible actions, and not all actions are allowed at any given time. In chess, there are 64 squares, so the are 64 places a piece can be picked up from, and 64 it can be placed, giving an action space of size $64 \times 64 = 4096$. Because of the resolution of the CMS detector, however, we have on the order of a hundreds of million of possible states. Environments like this are known as Large Discrete Action Spaces (LDAS) [161]. The most common approach to such spaces is to do as we explored in the previous section - implement a continuous action space, and choose the discrete action closest to the predicted action. Some approaches group actions to make them easier to learn. This is not an option here, since fine granularity is what we are trying to achieve. Another approach is to learn lower dimensional embeddings for the possible actions. This would add a lot of computation, going against our goal of a fast solution. There is a small amount of research on related problems, but we have a few special constrains. Not only is our action space large, the possible actions change drastically with each event. We can also exclude almost the entire action space at any given point, since we know to a good approximation where the track will go next. These constraints are also what makes the problem less suited to a model-based approach, like doing a Monte Carlo tree search of the possible options. We are therefore in uncharted waters. Though we will use known models, we will implement RL in a way that is, to the best of our knowledge,

novel.

We will explore two main methods, as illustrated in Fig. 5.8. Both methods have a state description that includes the current hit position in z, r and a few track parameters. The methods differ by how they include information about the possible next hits. The first method includes the position of *one* potential next hit. The state's quality is predicted, and this is repeated for all the compatible hits. The chosen next hit comes from the highest scoring state. The second method instead has a state with *all* the possible next hit positions included at once. It predicts quality scores for n different categories, where n is the number of compatible hits. The categories represent the position of the sorted list of hits. If e.g. category 0 has the highest quality score, the next hit will be the hit at position 0 in our data frame of compatible hits. This is unusual in an RL algorithm, since the actions do not always have the same effect. Instead of action 0 e.g. always being a step to the right, it could be either to the left or right, depending on where the compatible hits are. The first method has the benefit that there is a direct link between action and quality. The second method has the benefit that all the compatible hits are seen at once before making a choice on the action. We therefore consider both methods worth exploring. Using the previous hit position, as in Section 5.5 was explored, but showed very poor performance. This is because we are now distinguishing between something finely spaced instead of predicting a general position. Predicting a position benefits much more from knowing that the next step will be similar to the last, where we here instead need information about what type of track we have.

Since we are deviating from common RL practices, it is wise to start simple, as in Section 5.5. We will start with a simple DQN. Starting by learning one track is not very useful, because the task is simple to overfit. It can quickly be seen that a DQN algorithm can learn one track very quickly. We start with the same simplifications as used in Section 5.5. We will also estimate the track parameters in the state using a truth-level seed of three hits, since three hits is the minimal requirement to fit helix parameters. Though the algorithm will be trained on parameters estimated from truth-level seeds, it is applicable to a scenario without the truth-level information. The only change is that the algorithm will have to be repeated for each of the track seeds. This is similar to what CMS already does, as described in Section 2.7.3. We will

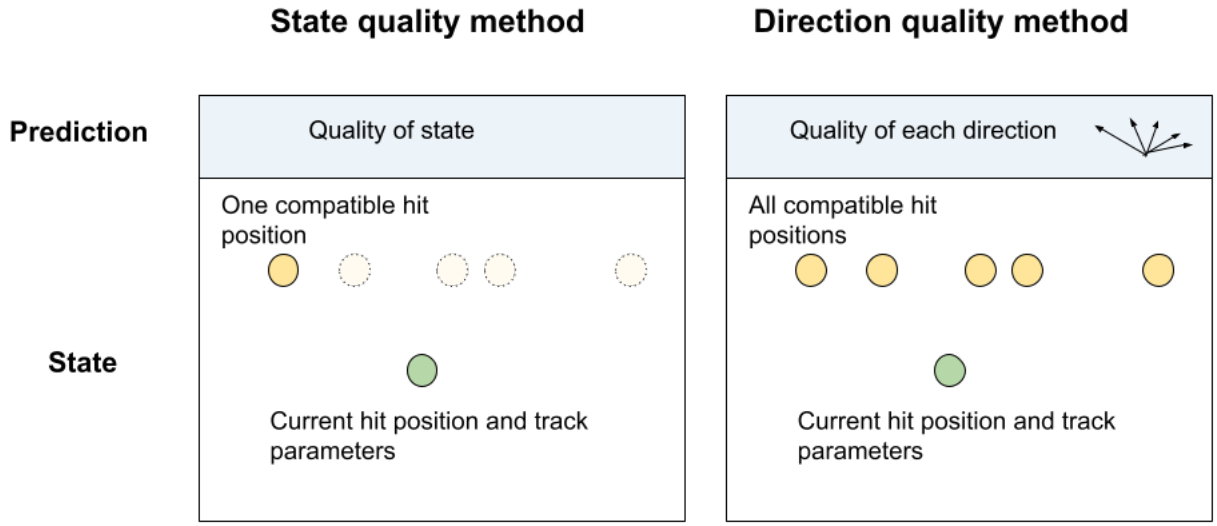


Figure 5.8: Two approaches to learning the quality of possible next track hits.

start by choosing to end the episode after a fixed number of steps to keep the implementations simple. In keeping with the findings from Section 5.5.1, we will give a bonus of five when the correct hit is identified. The reward is otherwise the negative distance between the correct hit and the predicted hit.

5.6.1 Learning a state quality for each possible next hit

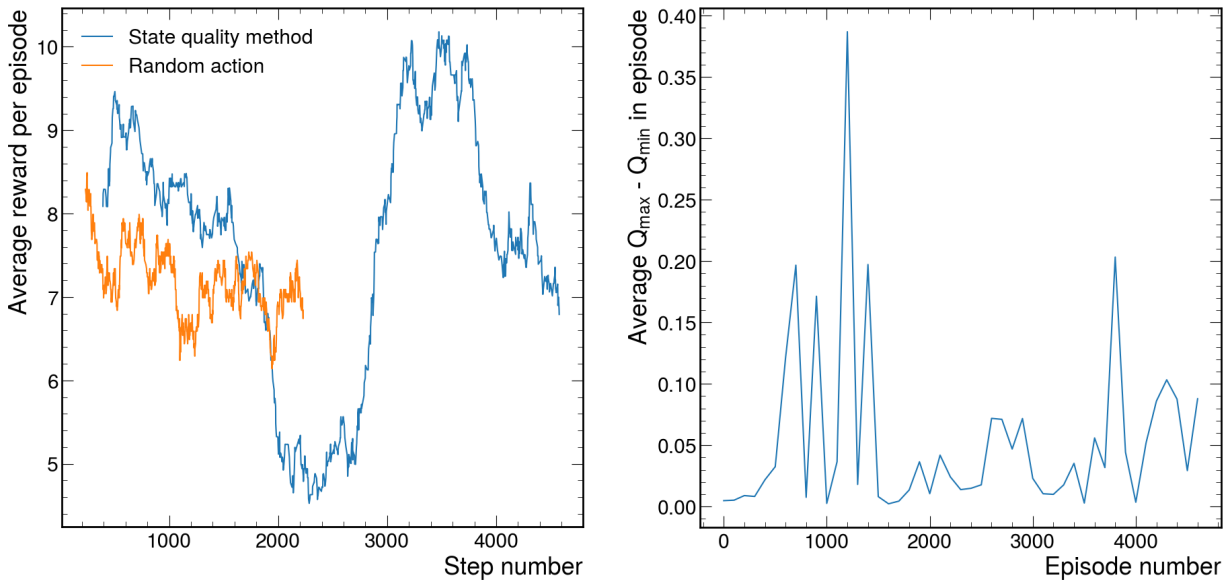
The first problem we are faced with is how to pre-select the hits that the RL agent can choose from. Since CMS tracking usually uses up to five possible hits at any given time, we will give the agent five possible hits to choose from. It is worth noting that neural nets expect a constant input size. The number of candidates will therefore have to be pre-selected. If there are fewer potential hits in some cases, the input would need to be padded. As a starting point, we could let the agent choose between the correct next hit and the n hits that are closest to it. This makes sure that the agent always has the option of choosing the right hit, and it tests its capability of distinguishing between very close hits. Always having the right hit available when training does not mean this needs to be the case when the model is implemented, it is just an easy way to teach the agent. A potential problem is that the agent will then always see the same next state, no matter which action it chooses, so we are violating a core tenet of RL. The next possible maximum reward will always be the same, so we are altering Equation 5.6. The

main effect is that the agent will only ever consider the current step, not the possible future rewards. A few experiments showed that the benefit of always having access to the correct hit outweighed the benefit of being able to consider future steps. It is common in RL that the agent struggles to learn until it sees several examples of "success" - in this case choosing a right hit. We therefore choose to start with this implementation, keeping in mind that this might prove to be a limitation.

We then implement the "state quality" method where we let a DQN learn the quality of the state as described by a hit position, track parameters, and one potential next hit position. We start with a simple neural net with two layers of 32 neurons. The quality is predicted for five different states, each containing one compatible hit. The best hit is chosen to update the track, and the process repeats. The results are shown in Fig. 5.9. The result when taking a random action is also shown for reference. This was run for a short amount of time, but shows an average reward of around 7. The RL agent does not seem to outperform taking a random action. A potential explanation is shown to the right in the same plot. For each starting point, we evaluate five distinct states, each containing one possible hit. All of these five will look numerically very similar. The average spread in Q-value between the states will inevitably also be very similar. We could choose to exaggerate the differences in position, but this would wash out the physical connection to the space the agent is navigating in, likely leading to a poor performance.

5.6.2 Learning the quality of two possible actions

If all the possible hits are present in the state, we will not run into the problem of similar Q-values that we do in the state-quality approach. We will therefore implement the "direction quality" method. We sort the compatible hits by the r and then by the z coordinate. We will now learn the quality of two different actions. Choosing e.g. action 0 will always mean choosing the hit with the lowest r-position out of the compatible hits. We shall use a small neural net, starting with only four layers containing either 32 or 64 neurons. We will start with the standard simplifications that were also used in Chapter 4;



(a) Learning the quality of state-hit pairs

(b) Spread in Q-values

Figure 5.9: The graph to the left shows how there is no clear learning curve when learning the Q-values of state-hit pairs. One of the challenges is shown to the right. The states for the five possible hits have very close Q-values, hindering learning.

- Remove all hits that do not originate from a track of $p_T > 2$ GeV
- Only allow one hit per track per layer
- Use the truth-level first three layer hits as seeds to estimate track parameters
- Only predict five hits for each track. This means we only reconstruct tracks that have 8 or more hits
- Use the truth-level correct hits in training

5.7 Learning tracks in the barrel

To start simple, we will begin by only learning tracks that traverse the barrel of the detector. We also start by letting the agent choose between the right hit and the hit that is closest to it. This is a hard task in the sense that the hits will be very close to each other, but an easy task in the sense that there are only two possible actions. The results are shown in Fig. 5.10. The DQN uses buffers, as described in Section 5.2, but the accuracies reported here are all

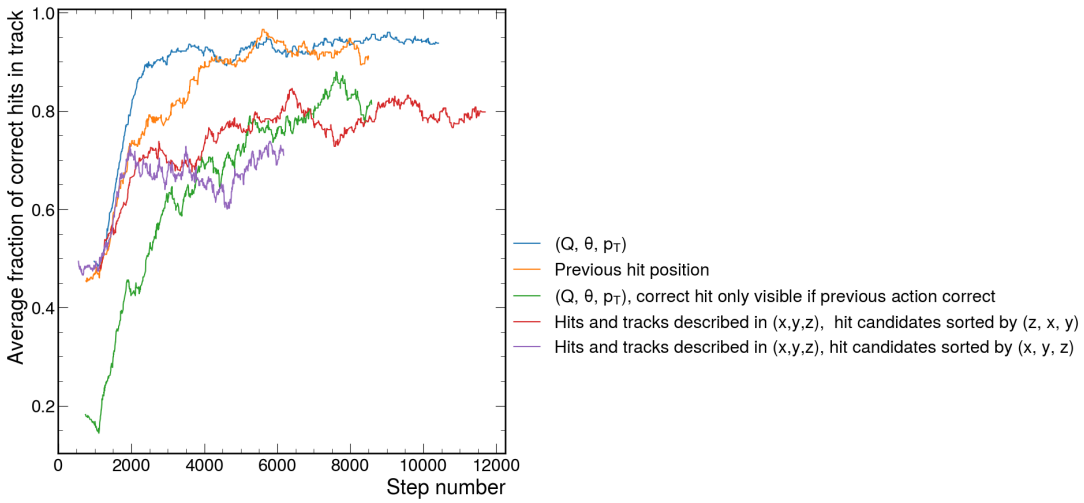


Figure 5.10: Learning tracks in the barrel of the TrackML detector when describing the state in different ways.

evaluated on tracks that the agent has never seen before. We therefore do not have to worry about overfitting in the plots that we consider in this chapter. For extra caution, any plots that do not show the training have been made with data that is not used in training.

Fig. 5.10 shows that we are able to learn how to select the hits in a track quite well under the simplifications we have used. Describing the track by track parameters or by the previous hit positions seem to perform quite similarly. This illustrates the point made in Section 4.6, that under these simplifications, only information about the previous hit position is needed. The fact that using the track parameters perform just as well, if not better, means that the agent is able to use more abstract information to learn the hit positions. The track parameters (Q, θ, p_T) were chosen as they represent the minimal information needed to decide where the track will go next from its current state. Many different variations of track parameters were tried, such as including momentum in x or y z-space, including other angles like ϕ, or including vertex positions. None seemed to outperform the ones shown here. We also trained the agent in a situation where the correct hit is always presented as one of the options. This makes it easier for the agent to learn, as shown. It does not mean that the correct hit always has to be available during inference. A few experiments were also done where the hit positions were described by x, y, z and the track parameters were described by the momentum in these directions. This also performs well, but struggles to learn as well as the r, z space. This is partially because there is much more information. The observation includes the position of the current hit and

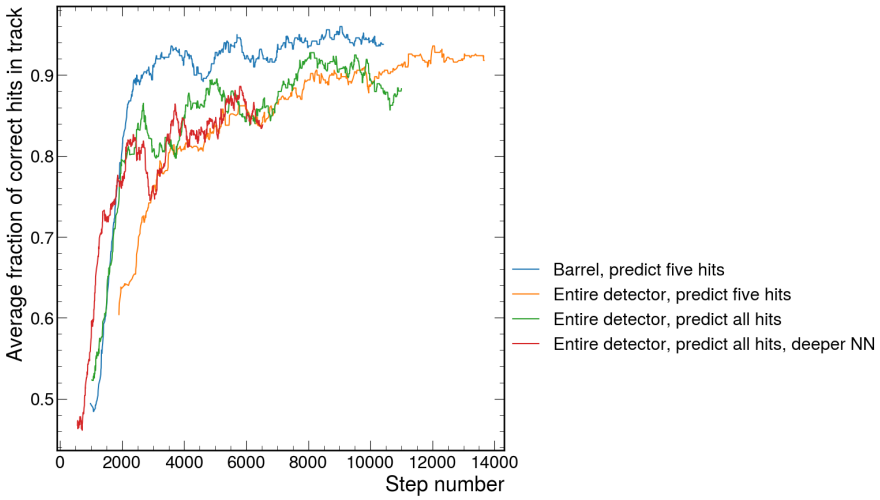


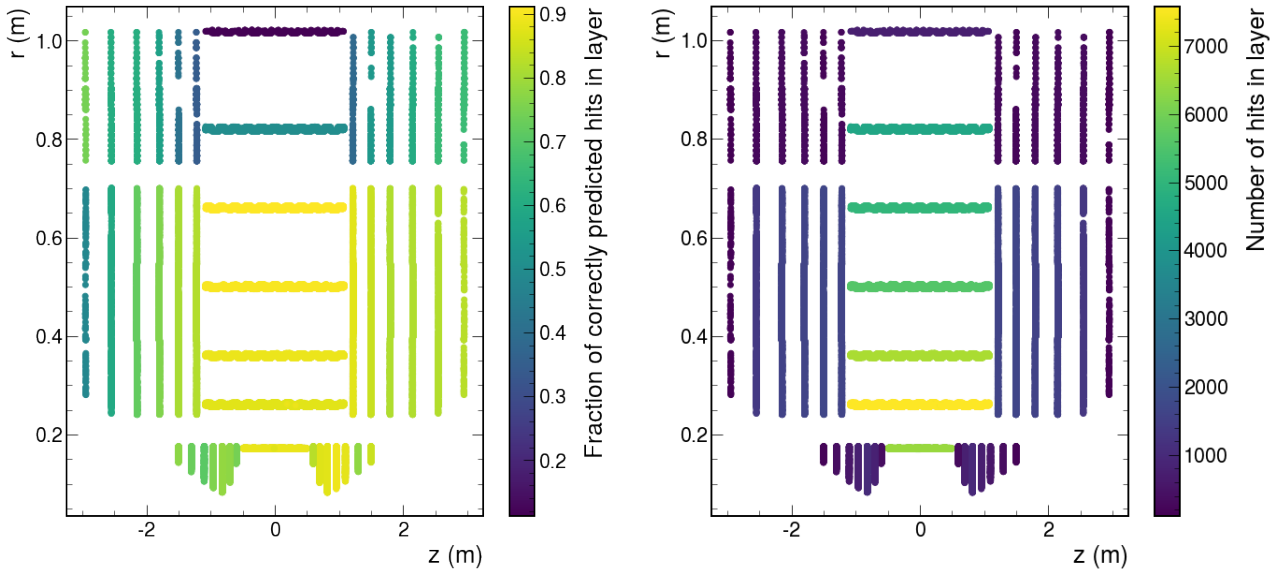
Figure 5.11: Learning tracks in the TrackML detector. The number of hits refer to how many hits we predict for each track.

two compatible hits, so the dimensionality of this part of the observation increases from six to nine. Most importantly, the track dynamics is more complex, since the track does not move in a straight line in this space. We will now extend the problem to the full detector to see if the agent is capable of holding even more information.

5.8 Learning tracks from the entire detector

Having seen good performance for tracking in the barrel, we are ready to extend the problem to the full detector. There will be more information to learn, so it expected that the agent spends slightly longer to achieve a good performance. To see the performance in the entire detector, we will also try to lift the simplification that we only predict five hits for each track. We will now use the truth information to know when to stop predicting hits for a track. In a real scenario we would not know where the end of the track is. We might instead be able to stop the track when the learn quality of the compatible hit is below a certain threshold. This is discussed further in Section 5.10.1. The results are shown in Fig. 5.11.

The Figure shows that learning tracks in the entire detector performs almost as well as just learning tracks from the barrel. As expected, it learns slightly slower. Predicting hits until the end of the track also shows a very similar accuracy. A few different neural net architectures



(a) Hit prediction accuracy in different detector layers

(b) Hit occupancy in the detector

Figure 5.12: Measuring the prediction accuracy and number of hits aggregated over 10 000 tracks. The predictions were made when distinguishing between two hits, one of which is the correct hit.

were also explored, but they did not show an improved performance. The learning curves are very similar, suggesting that the standard and the deeper neural net learn in very similar ways. This reinforces the idea presented in Section 4.6, that the problem is vastly simplified by removing many of the hits. We can see the reason for the very slight performance drop when extending to the full detector in Fig. 5.12. The barrel has a high number of hits, so the RL agent sees many examples of hits here. The endcaps have fewer hits, so the agent sees fewer examples. Note that this hit occupancy is somewhat specific to the TrackML data because of how it was simulated. For the CMS Phase 2 tracker, the hit occupancy is much more constant in psuedorapidity. This can bee seen e.g. in [81]. One might expect that the agent struggles in the more densely populated part of the detector since the compatible hits would be closer to each other. The fact that this is not the case would either suggest that the agent does not have a problem with distinguishing very close hits, or that the simplifications we have made make it an easy task. This will be explored in Section 5.9. Somewhat surprisingly, the accuracy is not symmetric. The agent seems to perform slightly better in the positive z-space. This could easily be explained by small biases or the agent getting stuck in local minima.

We could address the sparsity of hits in the endcap and the bias at the same time by taking

advantage of the symmetry of the detector. This could be done by simply taking the absolute value of the z-coordinate, so the agent has half the phase-space to learn. This would not work well for e.g. displaced tracks that might e.g. go from the positive to the negative side of the detector. These events were excluded from training. This did not show increased performance. Another way to increase performance for the less commonly seen end layers is to oversample, which is a common technique in ML. We simply increase the probability that a track that goes to the rarely seen layers is given to the agent during training. This led to a higher performance in the oversampled layers, but not to an overall increase in performance. As it focused more on one task, is seems that the performance might have dropped in other, less visible ways. The fact that oversampling improves the accuracy in these layers shows that it is mainly not due to the randomness introduced by multiple scattering, which would increase as the track goes further into the detector. This might still be an important effect when we include all the hits in the event, since it would only affect our ability to distinguish closely spaced hits.

5.8.1 Including all the hits in the hit selection pool

So far, we have only included hits from tracks that have a p_T over 2 GeV, while also removing noise hits and hits from tracks that leave two or more hits in a layer. We will go back to a situation where we only predict five hits for the barrel to start addressing this simplification. If we include all the hits in the event, the agent struggles to learn, as shown by the blue line in Fig. 5.13. Because we have more hits in the event, the distance between the hit candidates will be smaller. This is clearly more difficult for the agent to learn. Much like it is ill advised to start a physics A-level course by teaching QCD calculations, it can be ill advised to teach an RL agent by showing it the full complexity of a problem at once. Instead, we can create a curriculum. This is known as curriculum learning, and is a common RL technique [162]. Instead of having the agent learn to distinguish very close hits from the start, we can let it learn how to distinguish hits further apart before gradually increasing the complexity. In Fig. 5.13, we start with our standard simplifications. At iteration ten thousand, we start introducing hits from lower p_T tracks. Every 500 steps, the hit pool changes to include hits from tracks that are

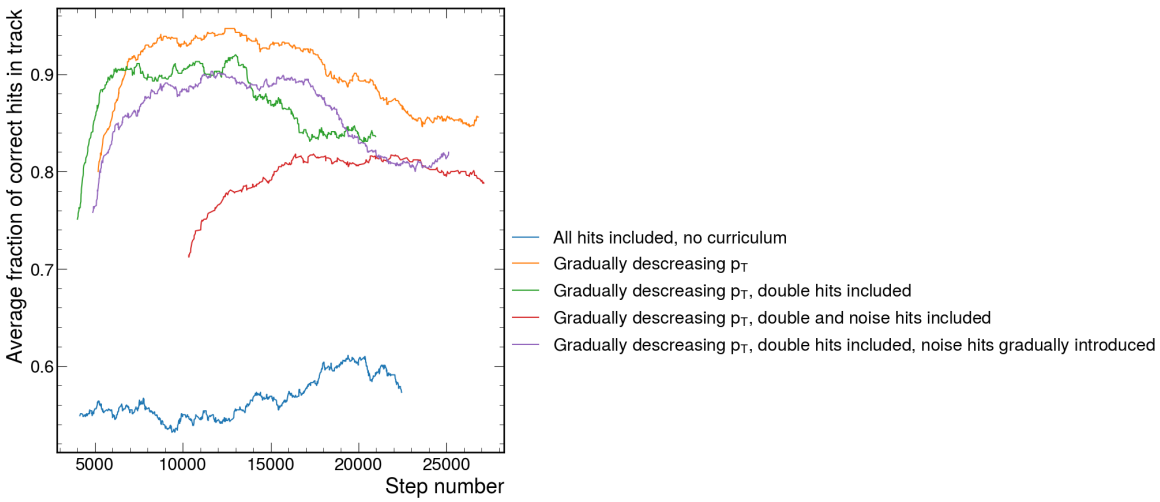


Figure 5.13: Curriculum learning in the barrel of the detector. The hit selection pool gradually increases so that the agent chooses between hits that are gradually closer together.

0.1 GeV lower than the previous iteration. Since we start with a p_T cut of 2 GeV, all the track hits are included by step twenty thousand. The figure shows that this is an effective strategy. We also need to include the noise hits and the hits from when there are two or more hits per track per layer.

5.9 Choosing between n hits

Ideally, we would like to choose between more than two hits at once. As mentioned in Chapter 2, CMSSW tracking considers up to five hits at once, while three hits is the default. We could always pass hits two at a time through our neural net and compare them, but with latency in mind, we would prefer if we could compare many hits at once. Since our agent can distinguish between the two closest hits, it should in principle be able to eliminate hits that are even further away. Choosing between five hits is, however, a much harder task to learn than choosing between two hits. Experiments showed that a higher number of hits required a lower learning rate, and the agent would otherwise be prone to get stuck in local minima. A lower learning rate encourages more exploration since the policy updates slowly. Instead of jumping to conclusions and getting stuck, the agent slowly explores the phase space.

Fig. 5.14 shows the result of learning to distinguish between n hits. All of the models shown

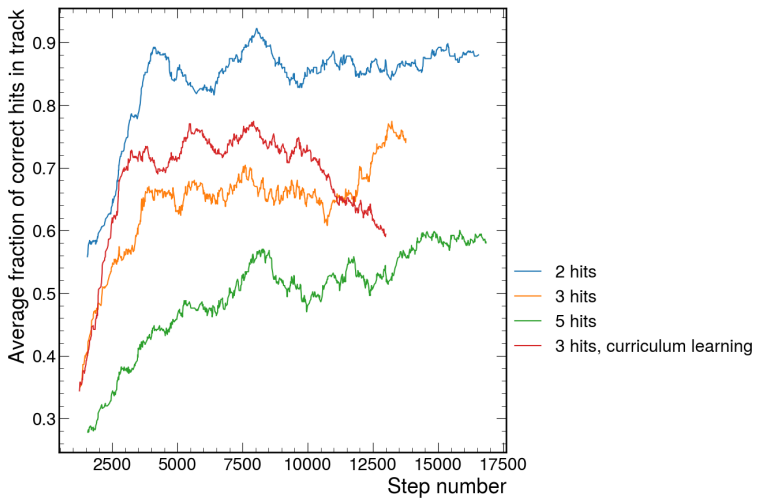


Figure 5.14: Learning to choose between a selection of n hits. At each step of the tracking, n hits are shown to the agent, which can take n actions in response.

here use the full detector and all except one use the simplifications we have discussed, such as using a reduced hit pool. It is clear that our agent is indeed able to distinguish between a larger number of hits, though it takes it longer to learn. We also want to see how well it performs when the hit pool increases. Choosing between a larger number of closely spaced hits is a fairly hard task. It is tempting to use curriculum learning to let the agent learn to slowly choose between an increasing number of hits. This is not an easy thing to do since neural nets expect a fixed input size. We could let some of the input vector be empty and slowly fill it, but this is not likely to do well since the agent will quickly learn that e.g. only two actions matter, and then need to unlearn it. Instead, we can have a set number of choices and let the task gradually increase in complexity. We will do this in a different way to that in the previous section. Now, we will use the full complexity of the problem, but create an offset in our hit selection. Initially, we will let the agent choose between the correct hit and some hits that are quite far away. These hits will then move closer and closer to the correct hit, until we are predicting between the correct hit and its closest neighbours. This is shown by the read line in Fig. 5.14. The agent uses the full hit pool at step 13 000, and we can clearly see that the performance drops as it gets close to this limit.

5.10 Towards a complete RL tracking implementation

Having seen that an RL agent can learn to do tracking under certain simplifications, we are eager to remove our simplifications to see if the performance still holds. In a real tracking scenario, we do not have access to truth-level hits. We use truth level hits when we select the hit candidates at each step and when we estimate the track parameters from the first three truth-level hits. The first point can be addressed by doing track propagation. Instead of looking for the true hits and the hits closest to it, we can use the track parameters to propagate the track to new layers and find the hits that are closest to the propagation point. We could still train an agent using truth-level information, but when we run inference, we can't use any truth-level information. This is explored in Section 5.10.1. In order to not use any truth-level information at all, we also need to implement a track seeding method with which to estimate the initial track parameters. This is explored in Section 5.10.2. To test the performance, and also to show the true power of not needing much detailed detector information, we shall do this for the TrackML data, CMS Phase 2 MC data, and for Run 3 CMS data.

5.10.1 Implementing a simple track propagation algorithm

Because the tracks will change depending on the decisions of our RL agent, we need to be able to perform track propagation during training as the agent makes decisions. The RL codebase is in Python, so it is easiest to implement a track propagation method in Python as well. A simple track propagation based on mkFit's algorithm was implemented [119]. Our implementation uses helix propagation equations, but does not use a Kalman filter or propagate uncertainties or material effects. The track parameters are not updated after it is estimated from the seeds. A few technicalities are worth noting. Similarly to how track propagation in CMSSW and mkFit works, tracks are propagated to compatible layers. In this implementation, a dictionary of common layer paths was created. A sample of 200 events with track p_T over 1 GeV was used to populate a dictionary containing rounded p_T and θ as keys and the corresponding possible layer paths as values. Only very common layer patterns were stored to avoid having mappings

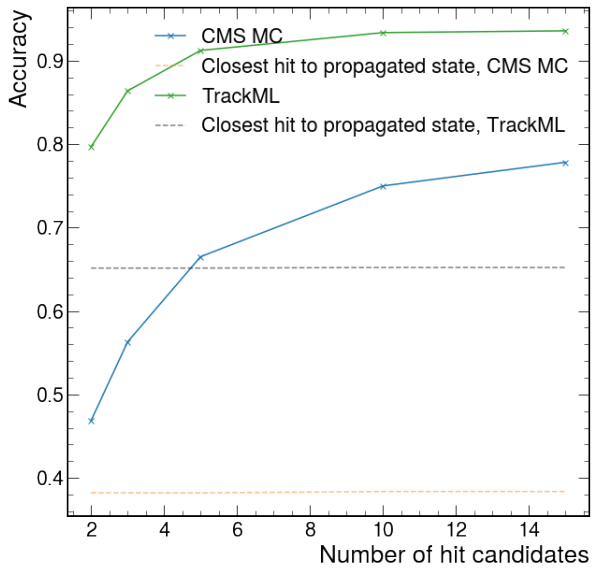


Figure 5.15: Performance of a simple tracking algorithm. The solid line shows how often the correct hit is available in our candidate pool given that the current hit is correct. The dashed line shows how often we choose the right hit if we always pick the hit closest to the track propagation point.

of tracks with e.g. missing hits. When selecting tracks to start from, only tracks that follow an accepted pattern and had a p_T above 1 GeV were considered. When doing track propagation, we look for paths within the $p_T - \theta$ dictionary that matches that of our seed. Only paths that have crossed the same exact layers previously are considered. If there are more than one compatible next layer, the track will be propagated to all the possible next layers. The potential next hits given to the RL agent will be the n that are closest to the track propagation points. So far, a hit has been considered correct if it is in the next layer in the track. Now, we have the possibility of skipping a layer, because the path dictionary allows it in certain situations. We will therefore slightly change our definition of finding the right hit to now include any hit that belongs to the track. We will give a reward if our agent picks any of the correct track hits. The tracking stops when the path ends, or there are no hits near the point we propagated to. The performance of this algorithm as a stand-alone algorithm is shown in Fig. 5.15.

Fig. 5.15 shows that our simple tracking algorithm performs reasonably well. The performance for CMS MC data is lower than for TrackML, which is to be expected. CMS events contain more hits and has a more complex detector geometry. The takeaway from this graph is that an RL agent should be able to at the very least do better than the dashed lines, which is the

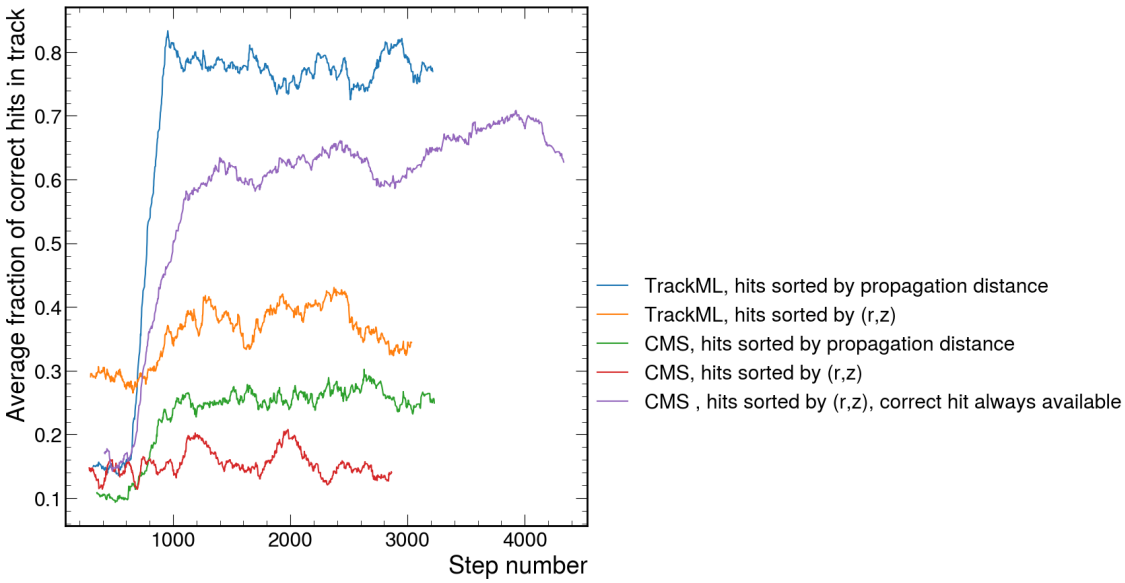


Figure 5.16: Track propagation with an RL filter that helps the track choose between three compatible hits at each stage. The sorting of the hit candidates is important because it directly links to the action prediction. If they are e.g. sorted ascending by a distance, selecting action 0 will always mean choosing the hit with the shortest distance measurement.

accuracy if we always choose the hit closest to our propagation point. The solid lines represent close to a theoretical maximum of what the RL agent could achieve. If the probability of seeing the right hit in the hit pool given that the previous his is correct is e.g. 60%, then this is close to the maximum of what the RL agent could learn with this tracking algorithm.

5.10.2 Track propagation with an RL filter

We are now ready to combine our tracking algorithm with an RL agent. We will start with truth-level seeds and only propagate tracks where the seed p_T is above 2 GeV and the track has hits in at least seven detector layers. All the event hits are included in the hit selection pools. We will let the agent choose between 3 compatible hits at each step, to reflect the common CMSSW CKF scenario. If the track propagation finds less than three compatible hits, the hit closest to its track propagation point is repeated in the state description so that the neural net receives a constant input size. The episode ends when there are no more compatible hits or the agent has reached the end of its list of layers to which to propagate. The results are shown in Fig. 5.16.

Fig. 5.16 shows that this algorithm can be applied to both CMS and TrackML data, with some important caveats. We might be tempted to sort the hits by their distance to the point the track was propagated to. This would mean that e.g. action 0 will always represent choosing the hit that was closest to the track propagation point. We are implicitly giving the agent extra information about the track and the hits without expanding the state description. This method is shown in blue and green for TrackML and CMS, respectively. The TrackML agent quickly picks up on the way the hits are sorted, and almost exclusively predicts that the action is 0. It then hovers around this point, never quite improving. This is a classic example of the exploration/exploitation issue that is endemic to RL. The CMS agent also quickly seems to pick up on a pattern, but then struggles. If we instead sort the hits by (r, z), as we have done so far in this chapter, the performance falls. The agent is now learning about the physics of the problem, but struggles to learn. The main element that stops these agents from learning further is the availability of "correct" examples. It is widely acknowledged in RL that the more sparse the rewards are, the more difficult it is for an RL agent to learn. When the correct hit isn't an option, the agents *has* to choose an incorrect option, steering the track on the wrong course and creating rippling effects for the remains of the episode. In the interest of teaching the agent faster, we could instead make sure it always has the option of choosing the right hit. This will make the "correct" examples more common, and hasten the learning. This is shown by the purple line in the Figure. Note that the performance exceeds what we predicted to be the theoretical maximum using Fig. 5.15 because we are now including the truth-level hit.

The CMS agent in purple in Fig. 5.16 seems to achieve a decent performance. We will now see how well it can perform when the correct hits are not always available. A sample of 1000 tracks was used to create Fig. 5.17. In order to reflect the quality of the RL agent rather than the quality of the tracking algorithm, the truth level hit was included as an option in each of the states. The different categories in this plot are defined in the same ways as they were in Fig. 4.5. We also introduce an additional metric useful in this situation, which we shall call a double match. The definitions are;

- Perfect match: The track contains all the hits in the track, and all the hits belong to the

same track

- Double-majority match: More than 50% of the hits come from the same track, and more than 50% of the track's hits are included
- LHC match: At least 75% of the hits belong to the same track
- Double match: At least 50% of the hits belong to the same track

The red line in Fig. 5.17 shows that we identified hits that belong to the right track at least half the time for around 70 % of the tracks. For half the tracks, we are able to select the right hit 75% of the time. When we consider how many of the particle's hits we correctly identified, the performance falls. This is mainly because the track propagation usually stopped before all the layers were explored. This is means that the agent picked layer path that was too short, or that there were no compatible hits later in the track. It is possible that multiple scattering has played a role in this, but it is difficult to say without further studies. There are no very clear performance patterns in either p_T or η. The uncertainties of the data points are not included because they are very large and makes it difficult to read the plot. This shows that we need a larger sample to see consistent patterns. This would require a larger data sample than we had available, and preferably a speedup of the track propagation implementation. We are therefore not quite at a point where we can reasonably compare the details of this tracking efficiency to that of CMS or our GNN implementation.

5.10.3 Seeding algorithm

The final step to use the RL agent without any truth level information is to create a track seeding algorithm. We will implement a very simple road search algorithm [81]. First, we identify five combinations of three layers in the inner part of the pixel detector that will act as our seeding layers. We will iterate over each of these layer combinations to find seeds. Starting with one of these layer combinations, we create a straight line fit for each hit in the first layer by assuming that each track originated perfectly from the beamline. This line is used to estimate

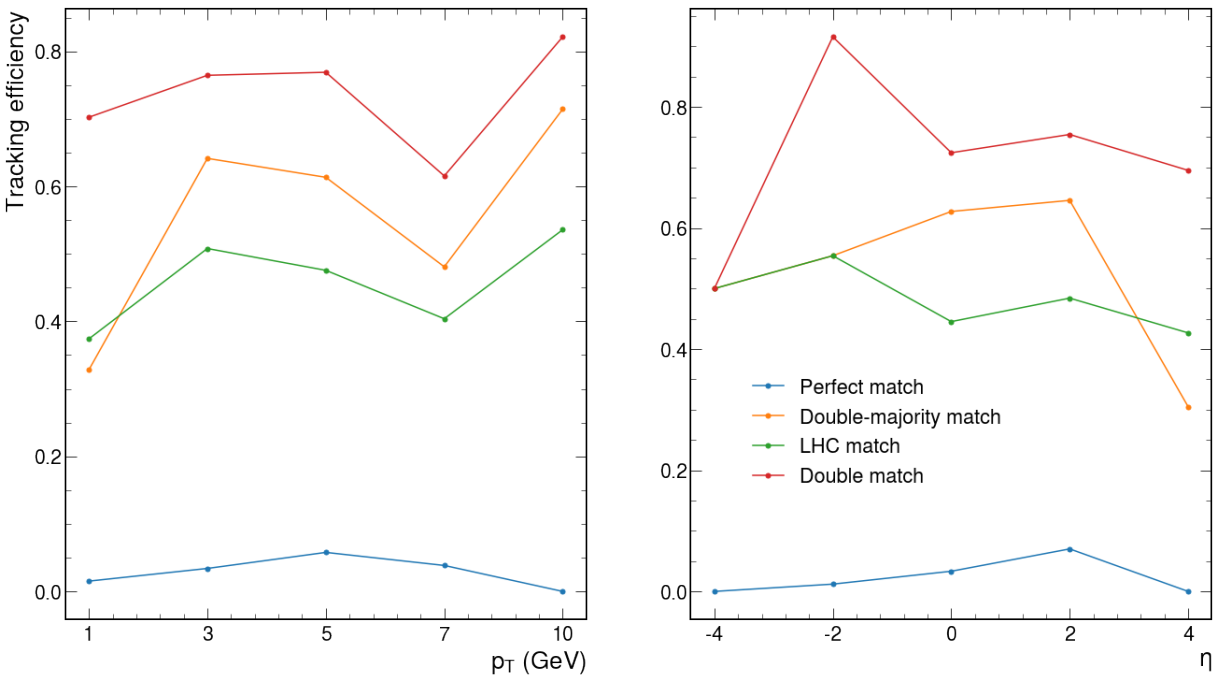
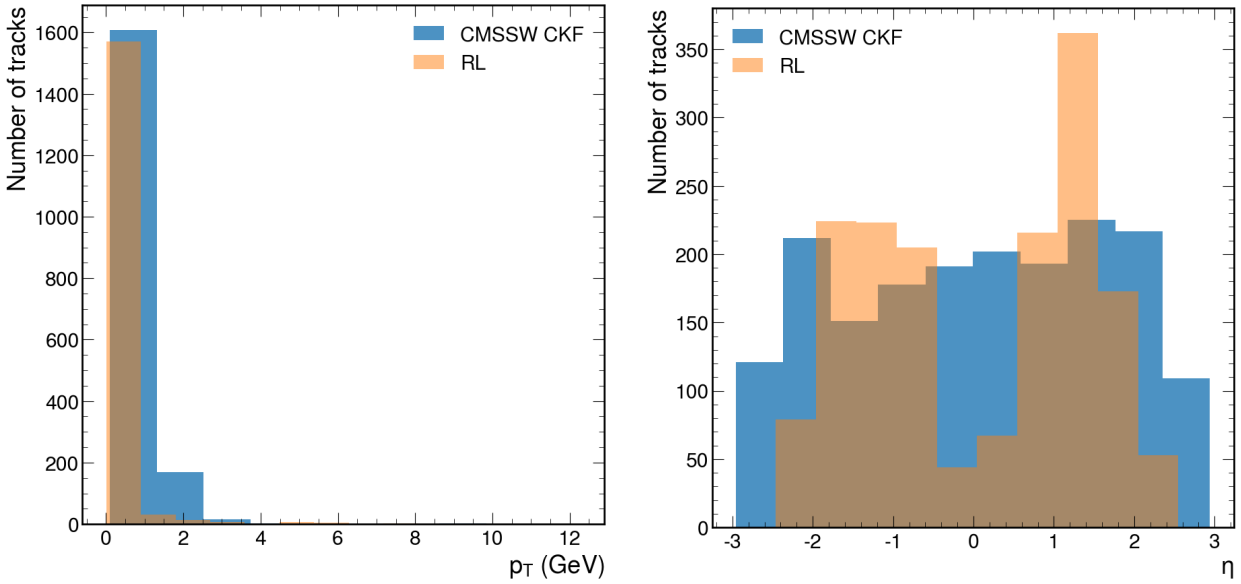


Figure 5.17: Tracking efficiency of an RL assisted track propagation algorithm.

the intersection with the next layer, and potential hits in this layer are identified. If there are no close hits, the seed is discarded. The track is then extrapolated to the final seed layer. The straight line is refitted, and a χ^2-measurement is done to estimate the quality of the seed. Seeds that are below a certain threshold is discarded. There are several assumptions here that would not lead to a sufficient tracking performance in a real implementation, but we are only looking for a very simple proof of concept. Armed with these simple seeds, we are ready to implement an RL assisted tracking algorithm.

We now apply the seeding, track propagation and RL methods in together for CMS Run 3 data. For each event, seeds are made. For each seed, we propagate the track. As the track propagates, the hit with which to update the track is chosen by a pre-trained RL agent. Though we are using the Run 3 geometry, our RL agent was trained on CMS MC Phase 2 data, since that is what we have studied in this chapter. Since the agent was able to learn to distinguish hits quite well during training, we would hope that it has learnt general patterns and rules of selecting hits given a previous hit, track parameters and a few compatible hits. It should therefore hopefully be applicable to a slightly different geometry. The tracking stopped once there were no more compatible hits, or five hit predictions had been made. If there were fewer than 7 hits in the



(a) p_T distribution

(b) Hit occupancy in the detector

Figure 5.18: Reconstructing tracks in an event from Run 3 data.

track, the track was rejected. The major drawback of our implementation is that it is very slow. This is not a feature of reinforcement learning, but rather of making implementations of seeding and tracking very quickly, and in a highly suboptimal way. For this reason, only a sample event is shown in Fig. 5.18. For the same reason, the tracks were not refitted after the track reconstruction, and the track parameters shown in the Figure are the parameters of the accepted seed. It must be emphasised that this implementation is not intended to serve as a fully mature tracking alternative, but rather as a proof of concept that one could use RL assisted tracking without using truth-level information, and without necessarily re-training for a different detector geometry.

Fig. 5.18 shows a fairly good agreement with the CMSSW CKF. Different events were used as spot-checks to confirm that the distributions shown here hold true for many of the events. The number of tracks in the event agreed very well between the two implementations throughout. The number of tracks can be fine tuned by a choice of χ^2 limit of the seeds and similar parameters, but the implementations seemed to agree well without any fine-tuning. The RL agent seems to consistently underestimate the p_T. This is likely to be because the track parameters were not refitted after the track reconstruction. The η distributions also differ a fair amount. The RL tracking method has very few tracks reconstructed in the barrel of the detector. This

is where our RL agent usually performs best, as shown in Fig. 5.12. There is also a surprising asymmetry, with many more tracks reconstructed in the positive η space. This is solely because of the seeding algorithm. If one makes a similar plot of the η of the seeds, one gets an almost identical distribution, as more seeds are found going out to the endcaps than continuing on in the barrel. A seeding algorithm more similar to the CKF seeding algorithm is therefore needed for a more realistic comparison. Though our implementation has many limitations, it seems like a promising first step in using RL for tracking.

5.11 Summary

In this chapter, we have introduced reinforcement learning for particle tracking. Starting from simply learning to navigate to a point, levels of difficulty were added to the problem until we arrived at a full RL-assisted tracking algorithm. Two main methods were presented. One tried to approach tracking in a continuous space, predicting the position of the next track hit without any knowledge of the hits in an event. This performed well, but it was difficult to achieve a precision fine enough to distinguish between two close hits. The major advantage of this method is that it requires no track propagation and therefore no use of detector geometry. This makes it very quick and easy both to run and to implement.

There are several steps one can make to further evaluate the potential of this algorithm. One of the main points would be to demonstrate that it does provide a computational speedup. Because it uses a conventional tracking algorithm, it will rely on some of the matrix multiplications and code branching studied in Chapter 3. The propagation does not need to be very precise, which might eliminate some of the time consuming error propagation and control logic. Another interesting study would be to use the continuous and discrete algorithms together. The continuous algorithm could predict an approximate next hit position, and the discrete algorithm could hone in on the precise hit position. To improve the tracking performance, we would need to consider whether the agent needs more information. From Chapter 4, it might seem like learning hits far apart is easier, but distinguishing close hits is a more difficult task,

requiring information about the neighbouring tracks. This could be drawn into the RL agent by creating a more complex state description. An even more attractive idea would be to use Multiple Agent Reinforcement Learning (MARL) [163]. This is a paradigm where multiple agents either collaborate or compete, and it could be very applicable to tracking. Instead of trying to evaluate each hit as they appear in our track propagation, we could recognise that the quality of a hit must be seen in context of the other tracks nearby. This is the same logic that is used in multi-hop GNNs, and MARL could achieve something similar without having to build time-consuming graphs.

Chapter 6

Conclusion

This thesis has considered ways of preparing particle tracking methods for the HL-LHC. Chapter 3 showed that the HL-LHC's increased luminosity poses a large challenge for the HLT. If we keep the HLT as it is, it would take twenty times as long to run as it currently does. Around 50-70% of this slowdown can be attributed to particle tracking. The pileup increase from around 50 to 200 creates a combinatorial explosion in the number hit candidates when propagating tracks. The computational performance of the combinatorial Kalman filter tracking was studied in detail. There were several areas that could clearly be improved, including increasing the degree of parallelisation and decreasing branching. Some of this has been addressed by the mkFit collaboration. They have largely kept the same tracking logic as the CMSSW, but parallelised it more and made simplifications. A performance study of this algorithm was done. It showed that mkFit code is much more performant than the CMSSW, but still falls short of the HL-LHC needs.

An appealing alternative to CKF particle tracking is machine learning. It could take advantage of past experience to reduce the combinatorial problem. The favoured ML method at the moment seems to be graph neural nets. They can encapsulate advanced tracking information and implicitly reconstruct tracks in the entire detector simultaneously. Chapter 4 studied these techniques in detail. It was shown that building graphs is a very challenging element. It is hard to include all the hits in an event without running into computational performance issues. A

few methods were explored to mitigate this, including using tracking stubs or module mappings. They bring the problem back into a manageable domain, but there are not yet any conclusive results that show that GNNs are capable of meeting the stringent timing requirements of the HL-LHC. The physics performance was also considered by applying the GNN methodology to CMS Phase 2 Monte Carlo and CMS Run 3 data for the first time. Both of these implementations showed encouraging performance.

To address the time consuming graph building of GNNs, a completely novel approach was considered by introducing reinforcement learning for tracking. Chapter 5 explored RL for tracking in a continuous and in a discrete action space. Both methods showed a capability of learning how to reconstruct tracks. The former made predictions that were consistently within a centimetre of the correct hit position. It was difficult to make it learn something more precise than this. The discrete action space showed more promise. An agent was given between two to five hits to choose from at each track propagation step. This was shown to perform well under a variety of conditions. Using data both from TrackML, CMS Phase 2 Monte Carlo and CMS Run 3, the performance was studied. The hit classifications score was around 65% in a scenario similar to that of a realistic CMS Phase 2 scenario, and between 80-90% when simplifications similar to those commonly used for GNNs were applied. There is a lot of room for improvement here, but it is a promising first approach. One of the large appeals of this algorithm is that the neural net was very small, at only four layers with 32-64 neurons in each layer. This makes it a prime candidate for FPGAs. Combined with the massively reduced branching one would have if one could filter out hit candidates, it might provide a large speedup for tracking.

Accelerating track reconstruction using both conventional methods and using graph neural nets are active fields of research with large collaborations behind them. Reinforcement learning would require a more concerted effort to continue as it is a very experimental approach. It could potentially be a best of both worlds solution, combining the finely tuned tracking propagation methods while leveraging the power of ML.

Bibliography

[1] O. S. Brüning *et al.*, "LHC design report," *CERN*, vol. 1, 2004. DOI: 10.5170/cern-2004-003-v-1. [Online]. Available: https://cds.cern.ch/record/782076.

[2] S. Weinberg, The discovery of subatomic particles. Cambridge University Press, 2003, ISBN: 9780521823517.

[3] A. W. Rose, "The level-1 trigger of the CMS experiment at the LHC and the Super-LHC," Dec. 2009. [Online]. Available: http://cds.cern.ch/record/1272460.

[4] CMS collaboration, "The CMS experiment at the CERN LHC," *Journal of Instrumentation*, vol. 3, S08004, 08 Aug. 2008, ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/08/s08004. [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-0221/3/08/S08004.

[5] A3D3, AI to accelerate scientific discovery, 2023. [Online]. Available: https://a3d3.ai/about/.

[6] R. Amann *et al.*, "A brief history of the LEP collider," *Nuclear Physics B - Proceedings Supplements*, vol. 109, pp. 17–31, 2-3 Jun. 2002, ISSN: 0920-5632. DOI: 10.1016/s0920-5632(02)90005-8.

[7] CMS collaboration, "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC," *Physics Letters B*, vol. 716, pp. 30–61, 1 Jul. 2012, ISSN: 03702693. DOI: 10.1016/j.physletb.2012.08.021. [Online]. Available: https://arxiv.org/abs/1207.7235v2.

[8] ATLAS collaboration, "Observation of a new particle in the search for the standard model Higgs boson with the ATLAS detector at the LHC," *Physics Letters B*, vol. 716, pp. 1–29, 1 Sep. 2012, ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.020.

[9] N. A. Bahcall, "Dark matter universe," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 112, pp. 12 243–12 245, Oct. 2015, ISSN: 1091-6490. DOI: 10.1073/pnas.1516944112. [Online]. Available: www.nasonline.org/programs/sackler-colloquia/completed_colloquia/dark-matter.

[10] N. Aghanim *et al.*, "Planck 2018 results - VI cosmological parameters," *Astronomy Astrophysics*, vol. 641, A6, Sep. 2020. DOI: 10.1051/0004-6361/201833910. [Online]. Available: https://www.aanda.org/articles/aa/full_html/2020/09/aa33910-18/aa33910-18.html.

[11] A. B. Balantekin and B. Kayser, "On the properties of neutrinos," *Annual Review of Nuclear and Particle Scienc*, vol. 68, pp. 313–338, Oct. 2018. DOI: 10.1146/annurev-nucl-101916-123044/cite/refworks. [Online]. Available: https://www.annualreviews.org/content/journals/10.1146/annurev-nucl-101916-123044.

[12] D. Campi *et al.*, "Commissioning of the CMS magnet," *IEEE Transactions on Applied Superconductivity*, vol. 17, pp. 1185–1190, 2 Jun. 2007. DOI: 10.1109/tasc.2007.897754.

[13] L. Arnaudon *et al.*, Linac4 technical design report, 2006. [Online]. Available: https://cds.cern.ch/record/1004186.

[14] E. Lopienska, The CERN accelerator complex, layout in 2022, Feb. 2022. [Online]. Available: https://cds.cern.ch/images/CERN-GRAPHICS-2022-001-1.

[15] CERN, Pulling together: Superconducting electromagnets, 2019. [Online]. Available: https://home.web.cern.ch/science/engineering/pulling-together-superconducting-electromagnets.

[16] O. Aberle *et al.*, High-luminosity large hadron collider (HL-LHC): Technical design report, 2020. DOI: 10.23731/cyrm-2020-0010. [Online]. Available: https://cds.cern.ch/record/2749422.

[17] Q. Wu, "Crab cavities: Past, present, and future of a challenging device," U.S Department of Energy, May 2015. DOI: 10.18429/jacow-ipac2015-thxb2. [Online]. Available: https://www.osti.gov/biblio/1183862.

[18] ATLAS collaboration, "The ATLAS experiment at the CERN large hadron collider," *Journal of Instrumentation*, vol. 3, S08003, 08 Aug. 2008, ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/08/s08003. [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-0221/3/08/S08003.

[19] M. Spiropulu and S. Stapnes, "LHC's ATLAS and CMS detectors," in. 2022, pp. 25–53. DOI: 10.1142/9789812779762_0003. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/9789812779762_0003.

[20] LHCb collaboration, "The LHCb detector at the LHC," *Journal of Instrumentation*, vol. 3, S08005, 08 Aug. 2008, ISSN: 1748-0221. DOI: 10.1088/1748-0221/3/08/s08005. [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-0221/3/08/S08005.

[21] S. Chen *et al.*, "Heavy flavour physics and CP violation at LHCb: A ten-year review," *Frontiers of Physics 2023*, vol. 18, pp. 1–69, 4 Mar. 2023, ISSN: 2095-0470. DOI: 10.1007/s11467-022-1247-1. [Online]. Available: https://link.springer.com/article/10.1007/s11467-022-1247-1.

[22] I. Bigi and A. I. Sanda, CP Violation. 2000. [Online]. Available: https://books.google.ch/books?hl=no&lr=&id=ZUANziLsV-MC&oi=fnd&pg=PR17&dq=cp+violation+&ots=EkSMfZ8zFk&sig=6Lp9O5ZiOiwaN_KBK7PwiRLMFmo&redir_esc=y#v=onepage&q=cp%20violation&f=false.

[23] A. D. Dolgov, "CP violation in cosmology," vol. 163, 2006, pp. 407–438, ISBN: 1586036289. DOI: 10.3254/1-58603-628-9-407. [Online]. Available: https://books.google.com/books/about/CP_Violation_From_Quarks_to_Leptons.html?hl=no&id=8hvvAgAAQBAJ.

[24] ALICE collaboration, "The ALICE experiment at the CERN LHC," *Journal of Instrumentation*, vol. 3, S08002, 08 Aug. 2008, ISSN: 1748-0221. DOI: 10.1088/1748-

0221/3/08/s08002. [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-0221/3/08/S08002.

[25] CERN, New state of matter created at CERN, 2000. [Online]. Available: https://home.cern/news/press-release/cern/new-state-matter-created-cern.

[26] P. Braun-Munzinger and J. Stachel, "The quest for the quark–gluon plasma," *Nature*, vol. 448, pp. 302–309, 7151 Jul. 2007. DOI: 10.1038/nature06080. [Online]. Available: https://www.nature.com/articles/nature06080.

[27] S. Chattopadhyay, "High-energy heavy ion collisions in the ALICE experiment at LHC-CERN: An overview," *The European Physical Journal Special Topics*, vol. 232, pp. 2831–2840, 17 Nov. 2023. DOI: 10.1140/epjs/s11734-023-01007-z. [Online]. Available: https://link.springer.com/article/10.1140/epjs/s11734-023-01007-z.

[28] J. Butler *et al.*, Technical proposal for the phase-II upgrade of the CMS detector, Jun. 2015. DOI: 10.17181/cern.vu8i.d59j. [Online]. Available: https://cds.cern.ch/record/2020886.

[29] The CMS Collaboration, Report on the physics at the HL-LHC, and perspectives for the HE-LHC, 2019. DOI: 10.23731/cyrm-2019-007. [Online]. Available: https://cds.cern.ch/record/2703572.

[30] M. Mangano *et al.*, "The physics potential of HL-LHC," 2018. [Online]. Available: https://cds.cern.ch/record/2650176.

[31] A. Rajantie, "Higgs cosmology," *Philosophical Transactions of the Royal Society A*, vol. 376, 2114 Mar. 2018. DOI: 10.1098/rsta.2017.0128. [Online]. Available: https://royalsocietypublishing.org/doi/10.1098/rsta.2017.0128.

[32] G. P. Salam *et al.*, "The higgs boson turns ten," *Nature*, vol. 607, pp. 41–47, 7917 Jul. 2022, ISSN: 1476-4687. DOI: 10.1038/s41586-022-04899-4. [Online]. Available: https://www.nature.com/articles/s41586-022-04899-4.

[33] CMS Collaboration, "Evidence for Higgs boson decay to a pair of muons," *Journal of High Energy Physics*, vol. 2021, 1 Sep. 2020. DOI: 10.1007/jhep01(2021)148. [Online]. Available: http://arxiv.org/abs/2009.04363.

[34] CMS collaboration, "A portrait of the Higgs boson by the CMS experiment ten years after the discovery," *Nature*, vol. 607, pp. 60–68, 7917 Jul. 2022, ISSN: 1476-4687. DOI: 10.1038/s41586-022-04892-x. [Online]. Available: https://www.nature.com/articles/s41586-022-04892-x.

[35] M. Gouzevitch and A. Carvalho, "A review of Higgs boson pair production," *Reviews in Physics*, vol. 5, p. 100 039, Nov. 2020, ISSN: 2405-4283. DOI: 10.1016/j.revip.2020.100039.

[36] J. R. Espinosa, "Vacuum stability and the Higgs boson," *Proceedings of Science*, vol. 29-July-2013, Nov. 2013, ISSN: 1824-8039. DOI: 10.22323/1.187.0010. [Online]. Available: https://arxiv.org/abs/1311.1970v1.

[37] ATLAS Collaboration and CMS Collaboration, Physics with the phase-2 ATLAS and CMS detectors, Apr. 2022. [Online]. Available: http://cds.cern.ch/record/2805993.

[38] R. S. Gupta *et al.*, "How well do we need to measure Higgs boson couplings?" *Physical Review D*, vol. 86, p. 095 001, 9 Nov. 2012, ISSN: 15507998. DOI: 10.1103/physrevd.86.095001. [Online]. Available: https://journals.aps.org/prd/abstract/10.1103/PhysRevD.86.095001.

[39] N. Craig, "Naturalness hits a snag with Higgs," *Physics*, vol. 13, Nov. 2020. DOI: 10.1103/physics.13.174.

[40] G. Gustavino, "Beyond-the-standard-model searches at HL-LHC," 2018. [Online]. Available: https://pos.sissa.it/.

[41] P. Nath, "Supersymmetry unification, naturalness, and discovery prospects at HL-LHC and HE-LHC," *The European Physical Journal Special Topics*, vol. 229, pp. 3047–3059, 21 Dec. 2020. DOI: 10.1140/epjst/e2020-000021-4. [Online]. Available: https://link.springer.com/article/10.1140/epjst/e2020-000021-4.

[42] CMS collaboration, Search for supersymmetry with direct stau production at the HL-LHC with the CMS phase-2 detector, 2019. [Online]. Available: https://inspirehep.net/literature/1704284.

[43] F. Chadha-Day *et al.*, "Axion dark matter: What is it and why now?" *Science Advances*, vol. 8, p. 3618, 8 Feb. 2022. DOI: 10.1126/sciadv.abj3618/suppl_file/sciadv.abj3618_sm.pdf. [Online]. Available: https://www.science.org/doi/10.1126/sciadv.abj3618.

[44] D. Hooper and S. Profumo, "Dark matter and collider phenomenology of universal extra dimensions," *Physics Reports*, vol. 453, pp. 29–115, 2-4 Dec. 2007. DOI: 10.1016/j.physrep.2007.09.003.

[45] A. Robertson *et al.*, "What does the Bullet Cluster tell us about self-interacting dark matter?" *Monthly Notices of the Royal Astronomical Society*, vol. 465, pp. 569–587, 1 Feb. 2017, ISSN: 0035-8711. DOI: 10.1093/mnras/stw2670. [Online]. Available: https://dx.doi.org/10.1093/mnras/stw2670.

[46] T. M. Undagoitia and L. Rauch, "Dark matter direct-detection experiments," *Journal of Physics G: Nuclear and Particle Physics*, vol. 43, 1 Sep. 2015. DOI: 10.1088/0954-3899/43/1/013001. [Online]. Available: http://arxiv.org/abs/1509.08767.

[47] L. Lee *et al.*, "Collider searches for long-lived particles beyond the standard model," *Progress in Particle and Nuclear Physics*, vol. 106, pp. 210–255, May 2019, ISSN: 0146-6410. DOI: 10.1016/j.ppnp.2019.02.006.

[48] Muon g-2 collaboration, "Measurement of the positive muon anomalous magnetic moment to 0.20 ppm," *Physical Review Letters*, vol. 131, 16 Aug. 2023. DOI: 10.1103/physrevlett.131.161802. [Online]. Available: http://arxiv.org/abs/2308.06230.

[49] CDF collaboration, "High-precision measurement of the W boson mass with the CDF II detector," *Science*, vol. 376, pp. 170–176, 6589 Apr. 2022, ISSN: 1095-9203. DOI: 10.1126/science.abk1781. [Online]. Available: https://www.science.org/doi/10.1126/science.abk1781.

[50] LHCb collaboration, "Measurement of lepton universality parameters in $b^+ \to k^+\ell^+\ell^-$ and $b^0 \to k*0\ell^+\ell^-$ decays," *Physical Review D*, vol. 108, 3 Dec. 2022, ISSN: 2470-0029. DOI: 10.1103/physrevd.108.032002. [Online]. Available: https://arxiv.org/abs/2212.09153v1.

[51] ALICE collaboration, "The ALICE experiment - a journey through QCD," *The European Physical Journal C*, Nov. 2022. [Online]. Available: https://arxiv.org/abs/2211.04384v1.

[52] R. Gozalo-Brizuela and E. C. Garrido-Merchan, "ChatGPT is not all you need. A state of the art review of large generative AI models," Jan. 2023. [Online]. Available: https://arxiv.org/abs/2301.04655v1.

[53] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210–229, 3 Jul. 1959, ISSN: 0018-8646. DOI: 10.1147/rd.33.0210. [Online]. Available: http://ieeexplore.ieee.org/document/5392560/.

[54] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 67–82, 1997, ISSN: 1089-778X. DOI: 10.1109/4235.585893.

[55] HEP ML Community, A living review of machine learning for particle physics, 2020. [Online]. Available: https://iml-wg.github.io/HEPML-LivingReview/.

[56] I. Goodfellow *et al.*, Deep Learning. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[57] A. Amini *et al.*, "Spatial uncertainty sampling for end-to-end control," May 2018. [Online]. Available: http://arxiv.org/abs/1805.04829.

[58] Wikimedia commons, Neural network, 2013. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network.

[59] C. Banerjee *et al.*, "An empirical study on generalizations of the relu activation function," *ACMSE 2019*, pp. 164–167, Apr. 2019. DOI: 10.1145/3299815.3314450. [Online]. Available: https://dl.acm.org/doi/10.1145/3299815.3314450.

[60] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, pp. 930–945, 3 1993, ISSN: 0018-9448. DOI: 10.1109/18.256500.

[61] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314, 4 Dec. 1989, ISSN: 0932-4194. DOI: 10.1007/bf02551274/metrics. [Online]. Available: https://link.springer.com/article/10.1007/BF02551274.

[62] Y. Bengio *et al.*, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1798–1828, 8 2013. DOI: 10.1109/tpami.2013.50.

[63] X. Zhao *et al.*, "A review of convolutional neural networks in computer vision," *Artificial Intelligence Review*, vol. 57, pp. 1–43, 4 Apr. 2024. DOI: 10.1007/s10462-024-10721-6/figures/33. [Online]. Available: https://link.springer.com/article/10.1007/s10462-024-10721-6.

[64] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from Electronics, volume 38, number 8, april 19, 1965, pp.114 ff.," *IEEE Solid-State Circuits Society Newsletter*, vol. 11, pp. 33–35, 3 Feb. 2009, ISSN: 1098-4232. DOI: 10.1109/n-ssc.2006.4785860.

[65] R. H. Dennard *et al.*, "Design of ion-implanted MOSFET's with very small physical dimensions, reprinted from IEEE journal of solid-state circuits, vol. sc-9, october 1974, pp.256-268," *IEEE Solid-State Circuits Newsletter*, vol. 12, pp. 38–50, 1 Feb. 2009, ISSN: 1098-4232. DOI: 10.1109/n-ssc.2007.4785543.

[66] J. Shalf, "The future of computing beyond Moore's law," *Philosophical Transactions of the Royal Society A*, vol. 378, 2166 Mar. 2020, ISSN: 1364-503X. DOI: 10.1098/rsta.2019.0061. [Online]. Available: https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0061.

[67] V. Eijkhout *et al.*, Introduction to High Performance Scientific Computing. 2016. [Online]. Available: https://zenodo.org/records/49897.

[68] K. Rupp, 42 years of microprocessor trend data, Feb. 2012. [Online]. Available: https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/.

[69] V. A. Pedroni, Circuit Design with VHDL, Third. MIT Press, 2020. [Online]. Available: https://books.google.ch/books?hl=en&lr=&id=aVzbDwAAQBAJ&oi=fnd&pg=PR15&dq=circuit+design+with+vhdl&ots=T54AEb8OdW&sig=2wYYiv4bd3bRd2dRWJQUZwuyIYk&redir_esc=y#v=onepage&q=circuit%20design%20with%20vhdl&f=false.

[70] V. Taraate, Digital Logic Design Using Verilog. 2016, ISBN: 9789811631986. DOI: 10.1007/978-981-16-3199-3. [Online]. Available: https://doi.org/10.1007/978-981-16-3199-3.

[71] J. Kodosky and C. Lopes, "Labview," *Proceedings of the ACM on Programming Languages*, vol. 4, HOPL Jun. 2020. DOI: 10.1145/3386328. [Online]. Available: https://dl.acm.org/doi/10.1145/3386328.

[72] P. Coussy and A. Morawiec, High-level synthesis: From algorithm to digital circuit. Springer Netherlands, 2008, pp. 1–297, ISBN: 9781402085871. DOI: 10.1007/978-1-4020-8588-8/cover.

[73] The FastML Team, HLS4ML, 2023. DOI: 10.5281/zenodo.1201549. [Online]. Available: https://github.com/fastmachinelearning/hls4ml.

[74] J. Duarte *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," Apr. 2018. DOI: 10.1088/1748-0221/13/07/p07027. [Online]. Available: http://arxiv.org/abs/1804.06913.

[75] A. Heintz *et al.*, Accelerated charged particle tracking with graph neural networks on FPGAs, 2020. [Online]. Available: https://arxiv.org/abs/2012.01563.

[76] M. Kozuma *et al.*, "Demonstration of FPGA acceleration of Monte Carlo simulation," *Journal of Physics: Conference Series*, vol. 2438, p. 012 023, 1 Feb. 2023, ISSN: 1742-6596. DOI: 10.1088/1742-6596/2438/1/012023. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/2438/1/012023.

[77] T. S. Virdee, "The LHC detector challenge," *Physics World*, vol. 17, p. 35, 9 Sep. 2004, ISSN: 2058-7058. DOI: 10.1088/2058-7058/17/9/41. [Online]. Available: https://iopscience.iop.org/article/10.1088/2058-7058/17/9/41.

[78] T. Sakuma and T. McCauley, "Detector and event visualization with SketchUp at the CMS experiment," *Journal of Physics: Conference Series*, vol. 513, pp. 22–32, Nov. 2013. DOI: 10.1088/1742-6596/513/2/022032. [Online]. Available: http://arxiv.org/abs/1311.4942.

[79] I. Neutelings, CMS coordinate system, 2021. [Online]. Available: https://tikz.net/axis3d_cms/.

[80] CMS collaboration, "Description and performance of track and primary-vertex reconstruction with the CMS tracker," May 2014. DOI: 10.1088/1748-0221/9/10/p10009. [Online]. Available: http://arxiv.org/abs/1405.6569.

[81] CMS collaboration, The phase-2 upgrade of the CMS tracker, Jun. 2017. DOI: 10.17181/cern.qz28.flhw. [Online]. Available: https://cds.cern.ch/record/2272264.

[82] F. Hartmann, Evolution of Silicon Sensor Technology in Particle Physics. Springer International Publishing, 2017, ISBN: 978-3-319-64434-9. DOI: 10.1007/978-3-319-64436-3. [Online]. Available: http://link.springer.com/10.1007/978-3-319-64436-3.

[83] Tracker group of the CMS collaboration, "The CMS phase-1 pixel detector upgrade," *Journal of Instrumentation*, vol. 16, 2 Dec. 2020, ISSN: 1748-0221. DOI: 10.1088/1748-0221/16/02/p02027. [Online]. Available: https://arxiv.org/abs/2012.14304v1.

[84] F. Reiss, "Study of b-hadron properties with semileptonic b-hadron decays," vol. ICHEP2020, SISSA, Feb. 2021, p. 415. DOI: 10.22323/1.390.0415.

[85] G. Abbiendi *et al.*, "Tau decays with neutral kaons," *European Physical Journal C*, vol. 13, pp. 213–223, 3 2000. DOI: 10.1007/s100520000317/metrics. [Online]. Available: https://link.springer.com/article/10.1007/s100520000317.

[86] E. Tournefier, "The preshower detector of CMS at LHC," *Nuclear Inst. and Methods in Physics Research, A*, vol. 461, pp. 355–360, 1-3 Apr. 2001, ISSN: 0168-9002. DOI: 10.1016/s0168-9002(00)01243-2.

[87] CMS collaboration, The phase-2 upgrade of the CMS endcap calorimeter, Nov. 2017. DOI: 10.17181/cern.iv8m.1jy2. [Online]. Available: https://cds.cern.ch/record/2293646.

[88] ATLAS collaboration and CMS collaboration, LHC experiments present new Higgs results at 2019 EPS-HEP conference, Jul. 2019. [Online]. Available: https://home.cern/news/press-release/physics/lhc-experiments-present-new-higgs-results-2019-eps-hep-conference.

[89] CMS collaboration, The phase-2 upgrade of the CMS muon detectors, Sep. 2017. [Online]. Available: https://cds.cern.ch/record/2283189.

[90] CMS collaboration, The phase-2 upgrade of the CMS level-1 trigger, Apr. 2020. [Online]. Available: https://cds.cern.ch/record/2714892.

[91] K. Bunkowski, "The algorithm of the CMS level-1 overlap muon track finder trigger," *Nuclear Inst. and Methods in Physics Research, A*, vol. 936, p. 368, Jun. 2018, ISSN: 0168-9002. DOI: 10.1016/j.nima.2018.10.173. [Online]. Available: https://cds.cern.ch/record/2629839.

[92] M. Dordevic, "The CMS particle flow algorithm," *EPJ Web of Conferences*, vol. 191, Jan. 2014, ISSN: 2100-014X. DOI: 10.1051/epjconf/201819102016. [Online]. Available: https://arxiv.org/abs/1401.8155v1.

[93] CMS collaboration, CMS offline software. [Online]. Available: https://github.com/cms-sw/cmssw.

[94] H. Sert, "CMS high level trigger performance in run 2," 2019, pp. 58–64. [Online]. Available: https://ceur-ws.org/Vol-2507/58-64-paper-9.pdf.

[95] CMS collaboration, "CMS trigger system," Sep. 2016. DOI: 10.1088/1748-0221/12/01/p01020. [Online]. Available: http://arxiv.org/abs/1609.02366.

[96] G. Giurgiu *et al.*, "Pixel hit reconstruction with the CMS detector," Aug. 2008. [Online]. Available: https://arxiv.org/abs/0808.3804.

[97] T. Ceccherini-Silberstein and M. Coornaert, Cellular Automata. Springer, 2010, pp. 1–36, ISBN: 978-3-642-14034-1. DOI: 10.1007/978-3-642-14034-1_1. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-14034-1_1.

[98] M. Hansroul *et al.*, "Fast circle fit with the conformal mapping method," *Nuclear Inst. and Methods in Physics Research, A*, vol. 270, pp. 498–501, 2-3 Jul. 1988, ISSN: 0168-9002. DOI: 10.1016/0168-9002(88)90722-x.

[99] V. Völkl *et al.*, "TrickTrack: An experiment-independent, cellular-automaton based track seeding library," 2018. [Online]. Available: https://indico.cern.ch/event/658267/contributions/2813731/.

[100] D. Emeliyanov, Track finding techniques in experimental particle physics, 2017. [Online]. Available: https://www.ppd.stfc.ac.uk/Pages/ppd_seminars_170215_talks_dmitry_emeliyanov.pdf.

[101] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, pp. 35–45, D 1960. [Online]. Available: https://www.cs.unc.edu/~welch/kalman/kalmanPaper.html.

[102] A. Strandlie and W. Wittek, Propagation of covariance matrices of track parameters in homogeneous magnetic fields in CMS, 2006. [Online]. Available: https://www.star.bnl.gov/~fisyak/star/TrackingReferences/Propagation%20of%20Covariance%20Matrices%20NOTE2006_001.pdf.

[103] R. Faragher, "Understanding the basis of the Kalman filter via a simple and intuitive derivation," *IEEE Signal Processing Magazine*, vol. 29, pp. 128–132, 5 2012, ISSN: 1053-5888. DOI: 10.1109/msp.2012.2203621.

[104] J. C. Butcher, "A history of Runge-Kutta methods," *Applied Numerical Mathematics*, vol. 20, pp. 247–260, 3 Mar. 1996, ISSN: 0168-9274. DOI: 10.1016/0168-9274(95)00108-5.

[105] CMS collaboration, The phase-2 upgrade of the CMS barrel calorimeters, Sep. 2017. [Online]. Available: https://cds.cern.ch/record/2283187.

[106] CMS collaboration, A MIP timing detector for the CMS phase-2 upgrade, Mar. 2019. [Online]. Available: https://cds.cern.ch/record/2667167.

[107] A. L. Rosa, "The CMS outer tracker for the high luminosity LHC upgrade," *Journal of Instrumentation*, vol. 15, p. C02029, 02 Feb. 2020, ISSN: 1748-0221. DOI: 10.1088/1748-0221/15/02/c02029. [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-0221/15/02/C02029.

[108] B. Kreis, "Particle flow and PUPPI in the level-1 trigger at CMS for the HL-LHC," Aug. 2018. [Online]. Available: https://arxiv.org/abs/1808.02094v1.

[109] CMS collaboration, The phase-2 upgrade of the CMS data acquisition and high level trigger technical design report, 2021. [Online]. Available: https://cds.cern.ch/record/2759072/files/CMS-TDR-022.pdf.

[110] L. Vage, Cms hlt tracking performance model, 2020. [Online]. Available: https://github.com/livcms/CMS_HLT_Tracking_PerformanceModel.

[111] D. R. Gene and M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *AFIPS Conference Proceedings - 1967 Spring Joint Computer Conference, AFIPS 1967*, pp. 483–485, Apr. 1967. DOI: 10.1145/1465482.1465560. [Online]. Available: https://dl.acm.org/doi/10.1145/1465482.1465560.

[112] Intel Vtune, Top-down microarchitecture analysis method. [Online]. Available: https://www.intel.com/content/www/us/en/docs/vtune-profiler/cookbook/2023-0/top-down-microarchitecture-analysis-method.html.

[113] Github - CMSSW13$HLTX$. [Online]. Available: https://github.com/cms-sw/cmssw/tree/CMSSW_13_0_HLT_X.

[114] A. Bocci, Scripts and web pages to visualise CMSSW resource usage with carrotsearch circles, 2020. [Online]. Available: https://github.com/fwyzard/circles/tree/master.

[115] N. T. Q. Hoa *et al.*, "The iterative clustering framework for the CMS HGCAL reconstruction," *Journal of Physics: Conference Series*, vol. 2438, p. 012 096, 1 Feb. 2023, ISSN: 1742-6596. DOI: 10.1088/1742-6596/2438/1/012096. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/2438/1/012096.

[116] P. Ram and K. Sinha, "Revisiting kd-tree for nearest neighbor search," *Proceedings of the ACM SIGKDD*, pp. 1378–1388, Jul. 2019. DOI: 10.1145/3292500.3330875. [Online]. Available: https://dl.acm.org/doi/10.1145/3292500.3330875.

[117] A. Bocci *et al.*, "Heterogeneous reconstruction of tracks and primary vertices with the CMS pixel tracker," *Frontiers in Big Data*, vol. 3, p. 601 728, Dec. 2020, ISSN: 2624-909X. DOI: 10.3389/fdata.2020.601728.

[118] S. Summers and A. Rose, "Kalman filter track reconstruction on FPGAs for acceleration of the high level trigger of the CMS experiment at the HL-LHC," *EPJ Web of Conferences*, vol. 214, p. 01 003, 2019. DOI: 10.1051/epjconf/201921401003. [Online]. Available: https://www.researchgate.net/publication/335862755_Kalman_Filter_track_reconstruction_on_FPGAs_for_acceleration_of_the_High_Level_Trigger_of_the_CMS_experiment_at_the_HL-LHC.

[119] S. Berkman *et al.*, "Speeding up the CMS track reconstruction with a parallelized and vectorized Kalman-filter-based algorithm during the LHC run 3," Apr. 2022, ISBN: 2304.05853v1. [Online]. Available: https://arxiv.org/abs/2304.05853v1.

[120] Y. Gu *et al.*, Line segment tracking - a highly parallelizable algorithm for track building, Mar. 2023. [Online]. Available: https://indico.cern.ch/event/1247039/contributions/5296624/subcontributions/415810/attachments/2608478/4506056/Line%20Segment%20tracking%20Mar%209%202023%20v3.pdf.

[121] J. M. Zhang *et al.*, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, 1 Jan. 2022. DOI: 10.1109/tse.2019.2962027.

[122] C. S. Amrouche *et al.*, "Machine learning techniques for charged particle tracking at the ATLAS experiment," Mar. 2021. DOI: 10.13097/archive-ouverte/unige:152041. [Online]. Available: https://archive-ouverte.unige.ch/unige:152041.

[123] S. Farrell *et al.*, "Novel deep learning methods for track reconstruction," 2018. [Online]. Available: https://arxiv.org/abs/1810.06111.

[124] M. Spiropulu *et al.*, Exa.trkx: HEP tracking at the exascale, 2019. [Online]. Available: https://exatrkx.github.io/.

[125] F. Scarselli *et al.*, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, pp. 61–80, 1 Jan. 2009. DOI: 10.1109/tnn.2008.2005605.

[126] T. Zhong *et al.*, "Probabilistic graph neural networks for traffic signal control," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2021-June, pp. 4085–4089, 2021, ISSN: 1520-6149. DOI: 10.1109/icassp39728.2021.9414829.

[127] Z. Li and A. B. Farimani, "Graph neural network-accelerated Lagrangian fluid simulation," *Computers and Graphics*, vol. 103, pp. 201–211, Apr. 2022, ISSN: 0097-8493. DOI: 10.1016/j.cag.2022.02.004.

[128] B. Sanchez-Lengeling *et al.*, "A gentle introduction to graph neural networks," *Distill*, Sep. 2021, ISSN: 2476-0757. DOI: 10.23915/distill.00033. [Online]. Available: https://distill.pub/2021/gnn-intro.

[129] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, pp. 1–159, 3 2020. [Online]. Available: https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf.

[130] G. DeZoort *et al.*, "Graph neural networks at the large hadron collider," *Nature Reviews Physics 2023 5:5*, vol. 5, pp. 281–303, 5 Apr. 2023, ISSN: 2522-5820. DOI: 10.1038/s42254-023-00569-0. [Online]. Available: https://www.nature.com/articles/s42254-023-00569-0.

[131] P. W. Battaglia *et al.*, "Interaction networks for learning about objects, relations and physics," 2016. [Online]. Available: https://arxiv.org/abs/1612.00222.

[132] M. Kiehn *et al.*, "The TrackML high-energy physics tracking challenge on Kaggle," *EPJ Web of Conferences*, vol. 214, p. 06 037, 2019, ISSN: 2100-014X. DOI: 10.1051/epjconf/201921406037. [Online]. Available: https://www.epj-conferences.org/articles/epjconf/abs/2019/19/epjconf_chep2018_06037/epjconf_chep2018_06037.html.

[133] G. DeZoort *et al.*, "Charged particle tracking via edge-classifying interaction networks," *Computing and Software for Big Science*, vol. 5, pp. 1–13, 1 Dec. 2021, ISSN: 2510-2044. DOI: 10.1007/s41781-021-00073-z/tables/3. [Online]. Available: https://link.springer.com/article/10.1007/s41781-021-00073-z.

[134] A. Salzburger *et al.*, TrackML particle tracking challenge, 2018. [Online]. Available: https://kaggle.com/competitions/trackml-particle-identification.

[135] T. Sjöstrand *et al.*, "A brief introduction to PYTHIA 8.1," *Computer Physics Communications*, vol. 178, pp. 852–867, 11 Oct. 2007. DOI: 10.1016/j.cpc.2008.01.036. [Online]. Available: http://dx.doi.org/10.1016/j.cpc.2008.01.036.

[136] X. Ai *et al.*, "A common tracking software project," *Computing and Software for Big Science*, vol. 6, 1 Jun. 2021, ISSN: 2510-2044. DOI: 10.1007/s41781-021-00078-8. [Online]. Available: https://arxiv.org/abs/2106.13593v1.

[137] S. Amrouche *et al.*, The tracking machine learning challenge: Accuracy phase, 2019. [Online]. Available: https://arxiv.org/abs/1904.06778.

[138] S. Amrouche *et al.*, "The tracking machine learning challenge: Throughput phase," *Computing and Software for Big Science*, vol. 7, pp. 1–19, 1 Dec. 2023, ISSN: 2510-2044. DOI: 10.1007/s41781-023-00094-w/figures/13. [Online]. Available: https://link.springer.com/article/10.1007/s41781-023-00094-w.

[139] L. Vage, Cmsgnn, 2021. [Online]. Available: https://github.com/livcms/cmsgnn.

[140] A. Heintz *et al.*, Accelerated charged particle tracking with graph neural networks on FPGAs, 2020. [Online]. Available: https://arxiv.org/abs/2012.01563.

[141] M. Ester *et al.*, A density-based algorithm for discovering clusters in large spatial databases with noise, Mar. 1996. [Online]. Available: https://www.osti.gov/biblio/421283.

[142] H. Torres *et al.*, "Physics performance of the ATLAS GNN4ITk track reconstruction chain," 2023. [Online]. Available: http://cds.cern.ch/record/2882507/files/ATL-SOFT-PROC-2023-047.pdf.

[143] L. H. Vage, CMS ML hackathon: GNN-4-tracking, 2021. [Online]. Available: https://indico.cern.ch/event/1041335/overview.

[144] M. Mieskolainen, "HyperTrack: Neural combinatorics for high energy physics," Sep. 2023. [Online]. Available: https://arxiv.org/abs/2309.14113v1.

[145] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction Second edition, in progress. MIT press, 2018. [Online]. Available: http://incompleteideas.net/book/RLbook2020.pdf.

[146] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and {g}o through self-play," *Science*, vol. 362, pp. 1140–1144, 6419 Dec. 2018, ISSN: 10959203. DOI: 10.1126/science.aar6404. [Online]. Available: https://www.science.org/doi/10.1126/science.aar6404.

[147] {OpenAI}, Part 2: Kinds of {rl} algorithms - spinning up documentation, 2018. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[148] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 3-4 May 1992, ISSN: 0885-6125. DOI: 10.1007/bf00992696.

[149] J. Schulman *et al.*, "Trust region policy optimization," *32nd International Conference on Machine Learning, ICML*, vol. 3, pp. 1889–1897, Feb. 2015. [Online]. Available: https://arxiv.org/abs/1502.05477v5.

[150] J. Schulman *et al.*, "Proximal policy optimization algorithms," Jul. 2017. [Online]. Available: https://arxiv.org/abs/1707.06347v2.

[151] C. J. C. H. Watkins and P. Dayan, "{q}-learning," *Machine Learning 1992 8:3*, vol. 8, pp. 279–292, 3 May 1992, ISSN: 1573-0565. DOI: 10.1007/bf00992698. [Online]. Available: https://link.springer.com/article/10.1007/BF00992698.

[152] V. Mnih *et al.*, "Playing {a}tari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602.

[153] W. Dabney *et al.*, "Distributional reinforcement learning with quantile regression," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, pp. 2892–2901, 1 Apr. 2018, ISSN: 2374-3468. DOI: 10.1609/aaai.v32i1.11791. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11791.

[154] M. Andrychowicz *et al.*, "Hindsight experience replay," *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 5049–5059, Jul. 2017, ISSN: 10495258. [Online]. Available: https://arxiv.org/abs/1707.01495v3.

[155] H. V. Hasselt *et al.*, "Deep reinforcement learning with double {q}-learning," *30th AAAI Conference on Artificial Intelligence*, pp. 2094–2100, Sep. 2015, ISSN: 2159-5399. DOI: 10.1609/aaai.v30i1.10295. [Online]. Available: https://arxiv.org/abs/1509.06461v3.

[156] D. Silver *et al.*, "Deterministic policy gradient algorithms," *International Conference on Machine Learning*, 2014. [Online]. Available: https://proceedings.mlr.press/v32/silver14.pdf.

[157] S. Fujimoto *et al.*, "Addressing function approximation error in actor-critic methods," *35th International Conference on Machine Learning, ICML*, vol. 4, pp. 2587–2601, Feb. 2018. [Online]. Available: https://arxiv.org/abs/1802.09477v3.

[158] T. Haarnoja *et al.*, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *35th International Conference on Machine Learning, ICML*, vol. 5, pp. 2976–2989, Jan. 2018. [Online]. Available: https://arxiv.org/abs/1801.01290v2.

[159] A. R. Mahmood *et al.*, "Benchmarking reinforcement learning algorithms on real-world robots," Sep. 2018. [Online]. Available: https://arxiv.org/abs/1809.07731v1.

[160] Y. Duan *et al.*, "Benchmarking deep reinforcement learning for continuous control," *33rd International Conference on Machine Learning, ICML*, vol. 3, pp. 2001–2014, Apr. 2016. [Online]. Available: https://arxiv.org/abs/1604.06778v3.

[161] F. Akkerman *et al.*, "Dynamic neighborhood construction for structured large discrete action spaces," May 2023. [Online]. Available: https://arxiv.org/abs/2305.19891v4.

[162] Y. Bengio *et al.*, "Curriculum learning," *ACM International Conference Proceeding Series*, vol. 382, 2009. DOI: 10.1145/1553374.1553380. [Online]. Available: https://dl.acm.org/doi/10.1145/1553374.1553380.

[163] K. Zhang *et al.*, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Studies in Systems, Decision and Control*, vol. 325, pp. 321–384, 2021, ISSN: 21984190. DOI: 10.1007/978-3-030-60990-0_12/figures/3. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-60990-0_12.