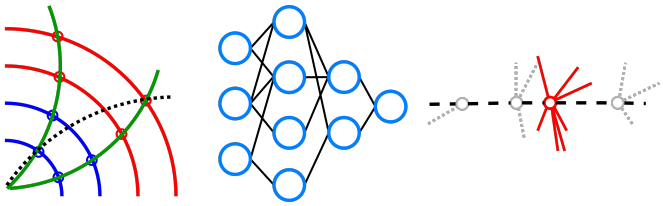
Imperial College London

Fast Machine Learning in the CMS Level-1 Trigger for the High-Luminosity LHC

by

Christopher Brown



A thesis submitted to Imperial College London for the degree of
Doctor of Philosophy

Supervisors: Professor Alexander Tapper and Doctor Mark Pesaresi
High Energy Physics
School of Physics
Imperial College London
June 2023

Abstract

The Large Hadron Collider at CERN will undergo an upgrade in 2027 to increase the integrated luminosity delivered to its associated experiments by a factor of 10 over its lifetime. The higher luminosity will allow CMS to broaden its physics programme with the Higgs boson self coupling being a key measurement that will be accessible with this upgrade. The upgrade will also increase the number of simultaneous proton-proton interactions (pileup) to between 140 and 200. To maintain physics selectivity in this high pileup the CMS Level-1 trigger will be upgraded to use charged particle tracks from the silicon tracker for the first time and more sophisticated algorithms, including those based on machine learning, will be implemented on an all-FPGA architecture.

Level-1 track reconstruction is an essential part of the Level-1 trigger upgrade; without it the trigger would be unable to cope with the increased pileup and will lose sensitivity. Work on the final module of the reconstruction algorithm is presented. A track quality boosted decision tree was designed, implemented and also shown to outperform basic cuts on track fitting parameters and has been run in hardware tests. Demonstrations of reduced versions of the track reconstruction algorithm were performed along with integration tests with the wider trigger system.

The global track trigger uses the reconstructed tracks to perform primary vertex finding. The location of the hard scatter in an event is essential for reducing background pileup contributions to energy sums and reduces the number of candidates being used in downstream algorithms. An end-to-end neural network-based approach to primary vertex finding and track-to-vertex association was developed and shown to outperform the baseline approach. This model was successfully implemented in firmware, running within the required latency and resource usage, and initial hardware testing results are shown.

The use of machine learning models in a changing environment such as the Level-1 trigger is an open topic in trigger development. Different strategies such as robust learning, uncertainty quantification, and continual learning are proposed and their suitability for the different time scales of the changing environment in CMS is explored. A study into the use of continual learning in the trigger demonstrates that machine learning models are at risk of performance degradation in a changing environment, but a continual learning approach can prevent this from happening and produces more robust models; an essential feature for algorithms which decide whether data is kept.

Statement of Originality

I declare that the work presented in this thesis is my own, with contributions from others in my collaboration. Where a figure or result is not my own, the original work is acknowledged and referenced.
Chapter 4: I wrote the Kalman filter output module emulator and firmware. I co-designed the track quality boosted decision tree and was responsible for integrating and testing its firmware within both the emulation and firmware of the Kalman filter output module.
Chapter 5: I performed the additions to the baseline FastHisto approach and characterised their performance. I adapted the original end-to-end neural network found in the CMS technical report from 2020, describing the Phase-2 Upgrade of the CMS Level-1 Trigger, for use in firmware as well as rewriting the convolutional neural network part and developing the differentiable Argmax network. I characterised and improved the neural network with updated track reconstruction datasets. I implemented the neural network in firmware with guidance from the hls4ml developers and performed the testing of this network in hardware, which led to the publication of multiple of my own conference proceedings.
Chapter 6: I performed all the tests shown in this chapter myself using a mixture of my own and other's firmware. The track reconstruction skinny chain firmware was written by others as was the original FastHisto vertex finding firmware. All other parts of these tests were performed by me including the fibre optic setup, the development of automatic testing scripts and all of the hardware, emulation and firmware simulation comparison tools.
Chapter 7: I conducted the study of continual learning in the Level-1 trigger, the degraded tracker datasets were provided to me and the package used for continual learning was the Avalanche package. The rest of the study was performed by me with the feedback of others.

Copyright Statement

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.

Acknowledgements

I would like to thank my supervisors Alex Tapper and Mark Pesaresi for their consistent help throughout my PhD, always providing feedback and ideas, and proofreading various talks, papers and this thesis. I would also like to thank Sioni Summers for introducing me to the world of fast ML and constant help debugging. I thank Benjamin Radburn-Smith for his endless knowledge of CMS procedure and teaching me the art of getting plots approved. I would also like to thank my friends and colleagues at both Imperial and CERN who made the pandemic days bearable, and the Geneva days enjoyable. To my family for the unwavering support throughout my years at university and for the free bed and board during much of my PhD. And finally, friends for supporting me throughout my PhD, particularly to Ewan for listening to me try to explain this research and for painstakingly proofreading it. Special thanks go to Skype and Zoom for your invaluable contribution to this work, for being there for every meeting, every conversation, and every social event for at least half of my PhD.
This work was supported by a studentship from the Science and Technology Facilities Council.

Contents

Abstract **3**

Statement of Originality **5**

Copyright Statement **7**

Acknowledgements **9**

1 Introduction **21**
1.1 The Standard Model . . . 21
1.1.1 Pre-LHC Physics Status . . . 21
1.2 The Large Hadron Collider . . . 23
1.3 The Compact Muon Solenoid Experiment . . . 24
1.3.1 Physics Goals . . . 25
1.3.2 Magnet System . . . 28
1.3.3 Tracker . . . 28
1.3.4 Calorimetry . . . 29
1.3.5 Muon Systems . . . 30
1.3.6 Trigger and DAQ . . . 31
1.3.7 Physics Contributions and Open Questions . . . 33
1.4 The High-Luminosity LHC . . . 35
1.5 Field Programmable Gate Arrays . . . 35

2 The Phase-2 Upgrade of CMS **39**
2.1 Phase-2 Upgrades . . . 39
2.1.1 Physics Programme . . . 39
2.1.2 Tracker Upgrade . . . 44
2.1.3 HGCAL . . . 46
2.1.4 Other Upgrades . . . 48
2.1.5 Trigger Upgrade . . . 49

3 Introduction to Machine Learning **57**
3.1 Core Concepts . . . 57
3.1.1 Supervised Learning . . . 58
3.1.2 Unsupervised Learning . . . 59
3.2 Decision Trees . . . 60
3.2.1 Bagging and Boosting . . . 61
3.3 Neural Networks . . . 61

3.3.1 Convolutional Neural Networks . . . 63
3.4 Fast Machine Learning . . . 66
3.4.1 Architectural Considerations . . . 66
3.4.2 Quantisation . . . 66
3.4.3 Pruning . . . 67
3.4.4 Implementation . . . 67

4 Track Reconstruction **69**
4.1 System Design . . . 69
4.2 Track Reconstruction . . . 70
4.2.1 Output to the Level-1 Trigger . . . 74
4.3 Track Finder Firmware Implementation . . . 76
4.3.1 Track Transformation . . . 78
4.3.2 η Routing . . . 78
4.3.3 Track Buffering . . . 79
4.4 Track Quality . . . 80
4.4.1 Fake Tracks . . . 80
4.4.2 Boosted Decision Trees for Track Quality . . . 85
4.4.3 Firmware Implementation . . . 89
4.5 Conclusions . . . 92

5 The Global Track Trigger **95**
5.1 System Design . . . 95
5.2 Vertex Reconstruction and Track-to-Vertex Association . . . 96
5.2.1 Motivation . . . 96
5.2.2 Baseline Approach . . . 97
5.2.3 Extensions to the Baseline Approach . . . 100
5.3 End-to-End Vertex Reconstruction . . . 104
5.3.1 Architecture and Learning . . . 105
5.3.2 Firmware Implementation . . . 108
5.3.3 Comparison to Baseline . . . 114
5.4 Track-Based Missing Transverse Energy . . . 116
5.4.1 Track Quality . . . 118
5.4.2 End-to-End Performance . . . 119
5.5 Conclusions . . . 119

6 Demonstrators **123**
6.1 Introduction . . . 123
6.2 Hardware . . . 125
6.2.1 Apollo . . . 125
6.2.2 Serenity . . . 126
6.2.3 Extensible, Modular, Data Processor Framework . . . 126
6.3 Testing Setup . . . 127
6.4 Track Finder Single Board Testing . . . 128
6.4.1 Subchain Testing . . . 128
6.4.2 Skinny Chain Testing . . . 130
6.4.3 Track Quality Testing . . . 130

6.5 GTT Single Board Testing . . . 132
6.5.1 End-to-End Vertexing Testing . . . 132
6.6 Integration Testing . . . 134
6.7 Conclusions . . . 137

7 The Deployment of ML in Changing Environments **141**
7.1 Changing Environments and the Implications for ML algorithms . . . 141
7.2 Solutions . . . 142
7.2.1 Robust Machine Learning . . . 143
7.2.2 Retraining and Redeployment . . . 143
7.2.3 Uncertainty Quantification . . . 144
7.2.4 Continual Learning . . . 147
7.3 Application of Continual Learning to the Level-1 Trigger . . . 150
7.3.1 CMS as a Changing Environment . . . 150
7.3.2 Level-1 Trigger ML in a Changing CMS . . . 153
7.4 Conclusions . . . 160

8 Conclusion **163**

References **167**

List of Acronyms **175**

List of Figures

1.1 Pre-LHC Higgs boson mass exclusion limits from combined electroweak data. 22
1.2 The LHC schedule for 2011–2029. 24
1.3 A slice of the CMS detector in the r-ϕ plane. 25
1.4 The CMS coordinate system, in both the r-z and r-ϕ planes. 26
1.5 Feynman diagram of the Higgs boson vector boson fusion production process. 26
1.6 Predicted branching fraction as a function of the Higgs boson mass for a variety of decay modes. . 27
1.7 One quarter of the CMS Phase-1 tracker in the r-z plane. 29
1.8 The original CMS Level-1 trigger schematic. 32
1.9 The Phase-1 CMS Level-1 trigger schematic. 33
1.10 Higgs boson couplings to SM particles as measured by CMS. 34
1.11 Main features of a field programmable gate array. 36

2.1 CMS event displays for a high pileup (100 PU) fill of the LHC. 40
2.2 Feynman diagrams for same sign vector boson scattering. 40
2.3 Feynman diagrams for di-Higgs production. 41
2.4 Likelihood scan as a function of the Higgs boson self coupling. 42
2.5 Predicted Higgs boson coupling uncertainties for up to run 3 and up to the end of the HL-LHC. . . 43
2.6 Feynman diagram of the $\tau \rightarrow \mu\mu\mu$ decay via a neutrino oscillation. 43
2.7 One quarter of the CMS Phase-2 tracker in the r-z plane. 44
2.8 Sketch of the outer tracker p_{T} module concept. 45
2.9 Schematics of the PS and 2S outer tracker modules. 46
2.10 p_{T} and d_0 resolution versus η for the CMS tracker. 47
2.11 Schematic of the high granularity calorimeter. 48
2.12 Phase-2 CMS vector boson fusion jet reconstruction efficiency versus $|\eta|$. 49
2.13 The Phase-2 CMS Level-1 trigger schematic. 50
2.14 The pileup per particle identification algorithm steps. 52
2.15 Jet finding and H_{T} reconstruction performance of Phase-2 Level-1 trigger algorithms. 53
2.16 Lepton p_{T} sensitivity for various samples with and without Level-1 track reconstruction. 53
2.17 Key Phase-2 Level-1 trigger signal sensitivity with and without the use of particle flow. 54

3.1 Diagram of a decision tree. 60
3.2 Diagram of a perceptron. 62
3.3 Diagram of a single neural network node. 63
3.4 Diagram of a neural network. 64
3.5 Diagram of a convolutional neural network. 65

4.1 One quarter of the CMS Phase-2 tracker in the r-z plane. 70

4.2 The tracklet stage of the L1 track reconstruction. . . . 71
4.3 Kalman filter coordinate definition. . . . 72
4.4 Level-1 track reconstruction efficiency versus η and p_{T}, and non-genuine fraction vs. p_{T} 74
4.5 Level-1 track reconstruction efficiency versus p_{T} split by particle type. . . . 75
4.6 Flow diagram of the KF output module 77
4.7 Track χ^2 distributions split by particle type 81
4.8 Track stub number distributions split by track type. . . . 82
4.9 Track helix parameter distributions split by track type 84
4.10 Fake track z_0-η distribution. . . . 85
4.11 Track $\tan(\lambda)$ distribution split by particle type 87
4.12 Track quality BDT output split by track type 88
4.13 Track finding efficiency versus non-genuine fraction for the track quality BDT. . . . 89
4.14 Track finding efficiency after quality cuts vs BDT cut threshold split by particle type 90
4.15 Level-1 track reconstruction parameters with and without quality cuts. . . . 91
4.16 BDT performance and resource usage versus fixed-point precision and pruning parameter. . . . 92

5.1 Global track trigger system diagram 96
5.2 A single $t\bar{t}$ event p_{T}-weighted, z_0 histogram 98
5.3 The effect of high η tracks on the FastHisto algorithm. . . . 99
5.4 Reconstructed track z_0 resolution as a function of $|\eta|$ 100
5.5 The effect of η on track z_0 resolution. . . . 101
5.6 Vertex residual plots for different additional weighting scenarios. . . . 102
5.7 Vertex finding performance versus BDT threshold 103
5.8 Vertex finding performance for the combined extension of FastHisto 103
5.9 End-to-end neural network architecture. . . . 106
5.10 Differentiable Argmax processing steps. . . . 107
5.11 Learnt track weights as a function of input track features. . . . 109
5.12 End-to-end network weight as a function of p_{T} and η for BDT output values 0–3. . . . 110
5.13 End-to-end network weight as a function of p_{T} and η for BDT output values 4–7. . . . 111
5.14 End-to-end network parameter distributions before and after pruning and quantisation. . . . 113
5.15 Neural network primary vertex finding residuals 116
5.16 End-to-end neural network receiver operating characteristic curve 117
5.17 Two-dimensional primary vertex residual plots for the baseline and neural network approaches . . 117
5.18 Track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ trigger rate in a minimum-bias sample. . . . 119
5.19 Track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ trigger turn-on curve in a SUSY sample. . . . 120
5.20 Track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ resolution plots for the baseline and end-to-end network. . . . 121

6.1 Apollo and Serenity ATCA boards. . . . 127
6.2 Track finder single board testing diagram 129
6.3 Hardware–emulation agreement for all KF chain track parameters. . . . 131
6.4 Skinny Chain floorplan on a VU7P 132
6.5 Hardware–emulator matching for $\frac{1}{R}$ in the Skinny Chain. . . . 133
6.6 GTT single board testing diagram 133
6.7 End-to-end neural network implementation VU9P floorplan. . . . 134
6.8 KF output to GTT input; single board, loopback testing diagram 135
6.9 KF output to GTT input; multiple board, single link testing diagram 136

6.10 KF output to vertexing; multiple board testing diagram 137
6.11 Floorplan of the nine KF outputs on one VU7P 137
6.12 Floorplan of the GTT vertexing 138
6.13 Photograph of the TF to GTT integration testing setup 138

7.1 Diagrams for different model retraining scenarios. 145
7.2 Standard NN versus stochastic NNs. 147
7.3 The Avalanche library for continual learning. 148
7.4 Level-1 trigger reconstructed track η and p_{T} distributions for different tracker degradation samples. 154
7.5 Level-1 trigger reconstructed track number of stubs and track quality BDT distributions for different tracker degradation samples. 155
7.6 Correctly regressed vertex histogram of track parameters 156
7.7 Incorrectly regressed vertex histogram of track parameters 156
7.8 ROC plot for the no retraining fake vertex ID model 157
7.9 ROC plot for the top-up fake vertex ID model 158
7.10 Diagram for the CL replay training scheme 159
7.11 ROC plot for the CL top-up fake vertex ID model 160

List of Tables

1.1 FPGAs used for different phases of the CMS Level-1 trigger 38

4.1 96-bit track word content 77
4.2 96-bit track word to 64-bit packets splitting scheme 79
4.3 KF output resource usage and latency 80
4.4 Track χ^2 cuts used for track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ 85

5.1 FastHisto resource usage for different implementations 99
5.2 End-to-end resource usage, split into the three subnetworks 115
5.3 Track quality parameter thresholds for track-based $E_{\mathrm{T}}^{\mathrm{miss}}$. 118

6.1 Track finder subchain testing 130

Chapter 1

Introduction

1.1 The Standard Model

The Standard Model (SM) of particle physics describes most of the fundamental particles and forces that make up the known universe. The strong force governs quark interactions mediated by the eight gluon force carriers and is described by the quantum chromodynamics (QCD) field theory [1]. The weak force describes radioactive decay, and is mediated by the massive[1] $W^{\pm}$ and Z bosons. The long-range electromagnetic (EM) force that describes electric and magnetic interactions of charged particles is mediated by the massless photon as is described by the quantum electrodynamics (QED) field theory [2]. At low energies the weak and EM forces appear separate with massive bosons for the weak and a massless boson for the EM. At high energies, such as those found in particle colliders or the early universe, the interactions of the quarks and leptons can be described more generally by the electroweak theory, with four massless bosons [3] [4].

To transition from this high energy theory to the observed broken symmetry of weak and EM forces, a mechanism is needed to break the symmetry and give the $W^{\pm}$ and Z bosons mass. The theory of spontaneous symmetry breaking developed by Brout, Englert and Higgs [5] [6] postulates a new complex scalar field with a non-zero vacuum expectation value. At high energies this introduces four new massless bosons along with the four electroweak bosons which do not interact with the new field, leaving them massless. As energy drops, three of the degrees of freedom from the additional scalar field are absorbed by the electroweak bosons which combine to give the massive $W^{\pm}$ and Z bosons and massless photon. The final degree of freedom is left as a scalar massive particle, the Higgs boson. In the vacuum the Higgs field has a non-zero value meaning other fermions interact with this field to give them mass in a separate mechanism to the process that gave the bosons mass. This theory can predict the masses of all the fundamental particles, barring the Higgs boson, which is left as a free parameter in the field. Neutrino masses are also not predicted by the SM, and what mechanism gives them mass is an open question in particle physics [7].

1.1.1 Pre-LHC Physics Status

Before the Large Hadron Collider (LHC) the theory of electroweak physics and the spontaneous symmetry breaking needed for the bosons to acquire their observed mass was well understood, but the existence, and mass, of the additional Higgs boson was unknown. Previous results from the Large Electron Positron (LEP) collider experiments put a lower exclusion of 114 GeV for the Higgs mass but with an upper limit of ~1 TeV for the SM to

[1]The weak force is observed as being short range, necessitating massive boson mediators.

remain consistent [8]. Proving this particle existed and finding its mass in part motivated the creation of the LHC. Using all previous electroweak data, assuming only the SM exists below the TeV scale, the Higgs mass has an upper limit of 186 GeV, shown in figure 1.1. This low mass (114–186 GeV) is a problem in the SM as the mass of the Higgs is predicted to be the sum of a bare mass term and a loop correction term; for a low mass Higgs these terms need to nearly perfectly cancel and requires a high degree of fine-tuning with no obvious reason for this to occur [9].

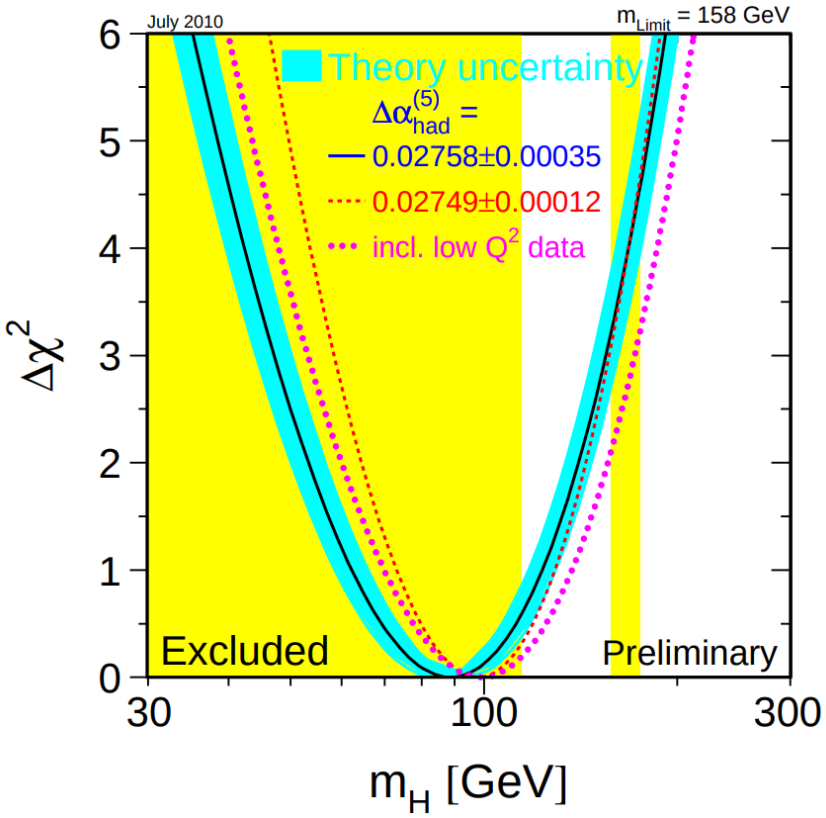


Figure 1.1: Pre-LHC Higgs boson mass exclusion limits from combined electroweak data. $\Delta\chi^2 = \chi^2 - \chi^2_{\text{min}}$ versus Higgs boson mass m_H. The yellow 95% exclusion region 0–114 GeV was obtained by direct LEP searches of $e^+e^- \rightarrow HZ$ [10]. The exclusion region between 158 and 175 GeV was obtained by the Tevatron [11]. The fit on the Higgs boson mass using all electroweak data sets an upper limit of 186 GeV with a 95% confidence level [8].

Other open questions in physics included the presence of dark matter which was predicted using astrophysical observations [12]. This dark matter has to be massive, so could potentially interact with the predicted Higgs field making the Higgs boson a useful tool for trying to find dark matter. Any additional particles discovered by the LHC would help to explain these astrophysical observations.

A theoretically elegant solution to these problems would be the existence of Supersymmetry (SUSY) [13], which introduces an additional symmetry to all SM particles with a new set of supersymmetric particles. Each bosonic SM particle receives a fermionic SUSY particle and vice versa, which would allow the corrections that lead to the Higgs mass fine-tuning problem to cancel out, negating the need for this unphysical fine-tuning. These additional particles could also be the missing dark matter observed in the universe. However, SUSY introduces $O(120)$ additional free parameters all of which would need to be measured making the experimental testing of SUSY a huge task. Most natural SUSY scenarios exist at the TeV scale to avoid additional fine-tuning, so a 10s of

TeV hadron collider, such as the LHC, was likely to find them.

Finally, if the SM Higgs exists what does this mean for unified theories of strong, weak and EM interactions? Does the SM remain consistent at 1 TeV or is there new physics to describe what is beyond this scale? How does gravity fit into this picture, and why is it so much weaker than the other forces?

1.2 The Large Hadron Collider

The LHC [14] was built in the LEP collider tunnel at CERN, Switzerland, starting construction in 1998 and finishing in 2008. It is a 26.7 km proton-proton collider with an ultimate collision energy of 14 TeV, and can also accelerate other hadrons, such as lead ions. The significant difference between the LHC and its predecessor LEP is the acceleration and collision of hadrons instead of electrons. While this allows for a higher collision energy due to the higher mass hadrons with less synchrotron radiation, it has the disadvantage of hadronic background from many simultaneous proton-proton interactions (hadronic pileup), proton remnants, and multiple parton interactions within the same proton-proton interaction. The majority of proton-proton interactions at the LHC interaction points will be low energy elastic collisions with soft hadronic interactions. These lead to low energy forward jets and hadrons, but because of the many interactions, significant energy is deposited in the detectors surrounding the interaction points. These events are known as minimum bias and are collected by the LHC experiments to measure trigger efficiencies. A trigger pathway is set to fire based on some energy deposited in the forward regions of the detector[2]. In some collisions there will be a high-energy, inelastic interaction where the protons are fragmented and large amounts of energy is transferred, leading to electroweak interactions; Higgs bosons, high mass quarks, new physics, etc. The interaction that takes place depends on the energy in the collision and the probability of certain interactions happening, called the branching ratio. Likewise, the final decay products that are seen depends on the probability of a certain decay happening, called the branching fraction. In these inelastic cases the LHC experiments want to record all the data from these collisions to reconstruct as much of the hard interaction as possible to understand the underlying physics in the event.

There are two general purpose experiments situated at opposing sides of the LHC ring; ATLAS [15] and the Compact Muon Solenoid (CMS) [16], both designed for a range of physics goals—most notably the Higgs discovery. ATLAS and CMS have similar design philosophies being cylindrical, nearly hermetic detectors with general purpose subdetectors designed to reconstruct charged particle trajectories, EM and hadronic energy deposits, and muons. The experiments have differing technologies in almost all of their subsystems, allowing the experiments to cross-check their results with independent measurements of the same underlying physics. The other two main experiments; LHCb [17] and ALICE [18], have specific physics motivations and designs. LHCb focuses on b-quark physics and is a single-arm forward spectrometer, whereas ALICE focuses on heavy ion physics with multiple tracking detectors for the dense heavy ion event topologies.

The LHC began stable physics collisions in 2011 with an initial instantaneous luminosity of $4 \times 10^{33}\text{cm}^{-2}\text{s}^{-1}$, which was produced with bunches of protons colliding every 50 ns (a rate of 20 MHz) with around 10^{11} protons per bunch. As the cross-section for proton-proton interaction is small, the number of observable interactions in the interaction points was around 25 per bunch crossing—referred to as a pileup (PU) of 25. Over the last decade, the LHC has delivered proton beams in 3–4 year running periods in which the experiments tried to collect as much data as possible for further analysis. Figure 1.2 shows these running periods

[2]Zero-bias events are also used where no energy is required to be deposited in the detector, and the trigger is fired based on a random clock. These events could contain no proton-proton interactions at all

along with the current run 3 and the future run 4. Between these runs there are long shutdowns of the LHC where the collider and experiments are upgraded for the next running period. The LHC aims to increase both its centre-of-mass (CoM) energy and instantaneous luminosity over each of the long shutdowns. Between the first run (run 1) to now (run 3), the CoM energy has risen from 7 TeV to 13.6 TeV and the luminosity has increased to over $2 \times 10^{34}\text{cm}^{-2}\text{s}^{-1}$, with a 25 ns bunch spacing (a rate of 40 MHz). The increase in luminosity has increased the pileup to up to an average of 60 interactions per bunch crossing for the LHC experiments, but provides far more data for the experiments, enriching their physics programmes. With the increased pileup compared to run 1 and changing physics focuses, the LHC experiments have also been upgraded over the previous decade of LHC running with incremental upgrades being made to all the experiments during the shutdown periods. However, this is the limit of the current LHC and without significant upgrades to the magnets; the luminosity, and data for the experiments cannot increase. For this reason, run 4, starting in 2029, will see the start of the High Luminosity LHC (HL-LHC), which marks a significant change to the LHC and the associated experiments. There will also be a modest increase in collision energy to 14 TeV; however this is the limit of the bending dipole magnets and any further energy increase would need improved magnet technology and a larger radius collider.

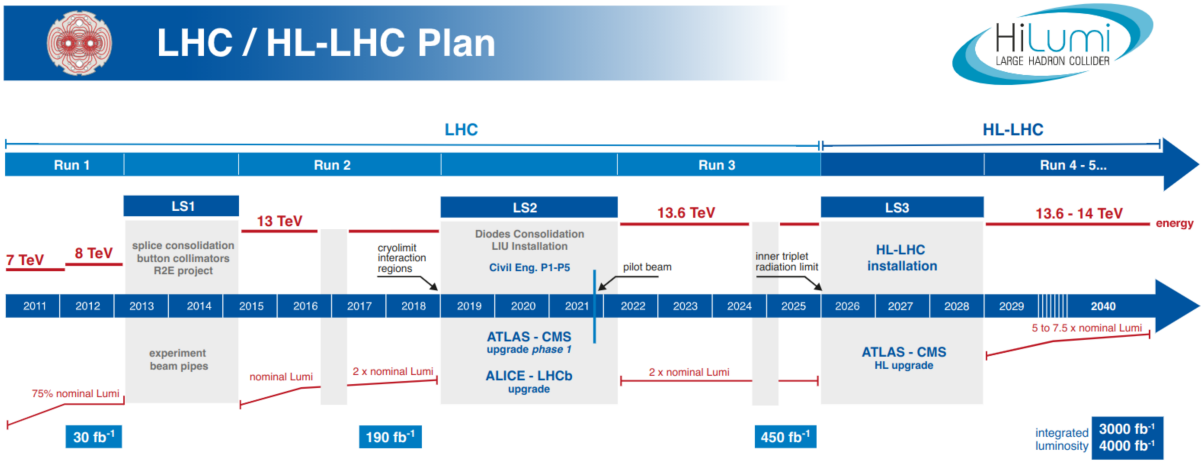


Figure 1.2: The LHC schedule for 2011–2029. Included is the targeted luminosity (in red at the bottom) and energy (in red at the top), physics runs and technical stops. When the current run ends in 2025 CMS will undergo its Phase-2 upgrade to be ready for the HL-LHC which will start its first run in 2029. Current as of January 2022 [19].

1.3 The Compact Muon Solenoid Experiment

The CMS experiment is one of the two multipurpose detectors and is situated at one of the LHC interaction points. It is a 28.7 m long, 17 m high, cylindrical detector designed to accurately reconstruct proton-proton collisions in high pileup environments. CMS uses different layers of detector subsystems, each specialising in different particle reconstruction; these subsystems are embedded in a 3.8 T superconducting solenoid that curves charged particle tracks to give momentum measurements. The position of each of these subdetectors is shown in figure 1.3 and described in detail below.

CMS operates a right-handed coordinate system originating at the nominal[3] interaction point: the centre of the detector, as shown in figure 1.4. Typically, the r-z and r-ϕ planes are used: the panels on the left and right of

[3]In normal LHC operation the bunches of protons are ~7.5 cm wide leading to a spread in the proton-proton interactions along the beampipe around z = 0. The interaction point itself can vary with the LHC beam optics.

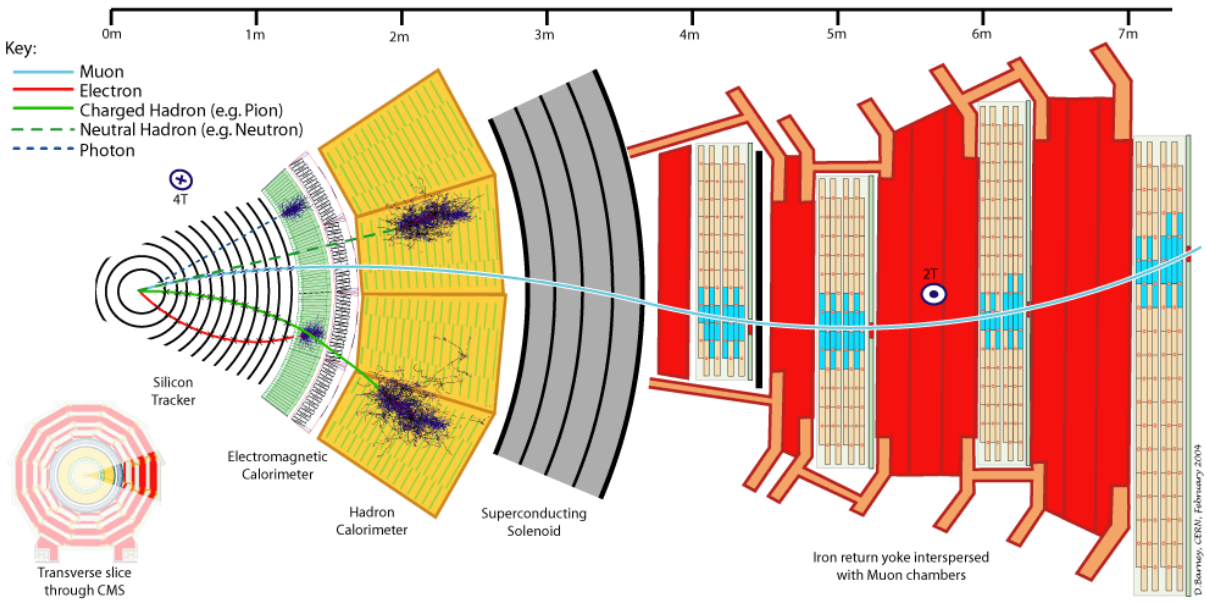


Figure 1.3: A slice of the CMS detector in the r-ϕ plane showing the individual subdetectors and their distance away from the interaction point at $r = 0$. The detector response of the main expected particles passing through the detector is also shown, as is the double curving of a muon track in the magnetic field, a key feature of the CMS detector [20].

figure 1.4 respectively. Also used throughout CMS analysis and reconstruction is the pseudorapidity $\eta = -\ln(\tan(\frac{\theta}{2}))$. Labelled is the angle λ, which is used specifically in the track reconstruction described in chapter 4. Transverse energy, E_{T}, and momentum, p_{T}, are typically used over the total momentum and energy and are defined as the total energy or momentum in the r-ϕ plane. These are used because of the complete coverage of the CMS detector in this plane versus the r-z plane where there is poor coverage at high η due to the beam pipe and very high radiation doses. With the complete coverage in the r-ϕ, CMS analyses can use momentum conservation to identify the presence of particles that have not interacted with the detector, such as neutrinos. Additional quantities, such as the missing transverse energy $E_{\mathrm{T}}^{\mathrm{miss}}$, can also be defined.

1.3.1 Physics Goals

One of the primary goals of the CMS experiment was the discovery of the Higgs boson [21]. When considering the expected branching fraction of the Higgs boson in various decay channels in figure 1.6, a low mass Higgs boson was most likely to be found in the bb channel, but because of the hadronic background from additional pileup interactions it was discounted as a major discovery channel, as is the case with the gg and cc channels. The WW channel results in multiple neutrinos requiring very good $E_{\mathrm{T}}^{\mathrm{miss}}$ resolution which is a difficult signal to accurately reconstruct. The "golden" decay channel was considered to be the ZZ channel owing to the clean signature of four leptons — CMS particularly saw the $ZZ \rightarrow 4\mu$ channel as a major discovery channel across the mass spectrum of the Higgs. This makes high efficiency muon reconstruction essential as this channel needs the simultaneous measurement of four muons with any inefficiency being raised to the fourth power. The four muon signature also gives a key trigger signal that is independent of hadronic pileup so flexible muon triggering with redundancy was also deemed essential. The other key decay channel was $H \rightarrow \gamma\gamma$, which has a low branching fraction but a very clean signature of just two isolated photons, making a high granularity calorimeter and performant tracker to veto electron and pion conversion another key design choice for CMS. The ability to

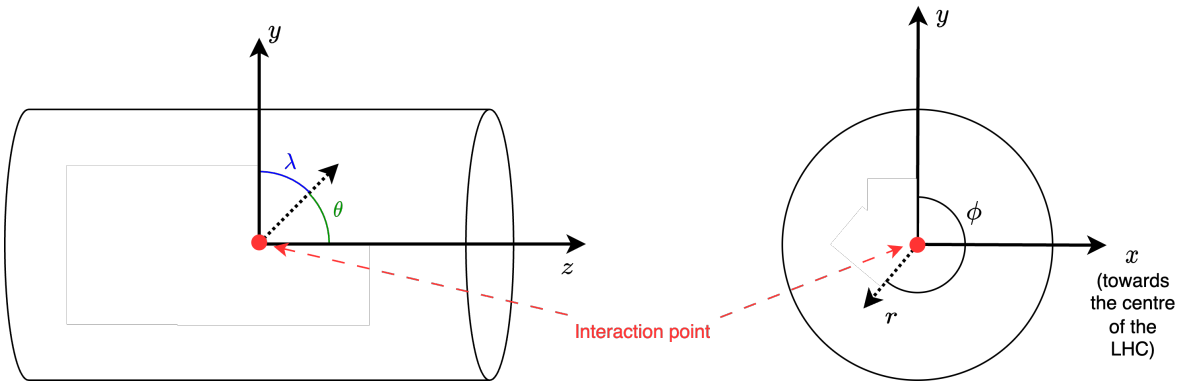


Figure 1.4: The CMS coordinate system. On the left is the y-z plane and on the right the x-y plane with the approximate interaction point at $x = y = z = 0$. Shown are the polar angle, θ, which measures angle from the beampipe, and the angle λ, used in track finding. This is the angle from the centre line of the detector, where $\lambda = \frac{\pi}{2} - \theta$. The angle ϕ, on the left, is the polar coordinate redefining the x-y plane as the r-ϕ plane, more natural for the cylindrical detector. With this r the y-z plane is redefined as the r-z plane, with r being the distance radially outwards from the interaction point.

accurately determine a primary vertex is a major part of this photon isolation so a high granularity pixel detector for vertex finding is also needed. If the vertex is located to within 1 cm the dominant contribution to the mass resolution of the Higgs becomes the ECAL energy response which also needs to be as good as possible. Another key channel was the $H \rightarrow \tau\tau$ decay via vector boson fusion (VBF) (shown in figure 1.5), but it needed a high efficiency for tagging forward jets. This makes high granularity, high pseudorapidity calorimetry a strong requirement for efficient triggering and reconstruction of this decay.

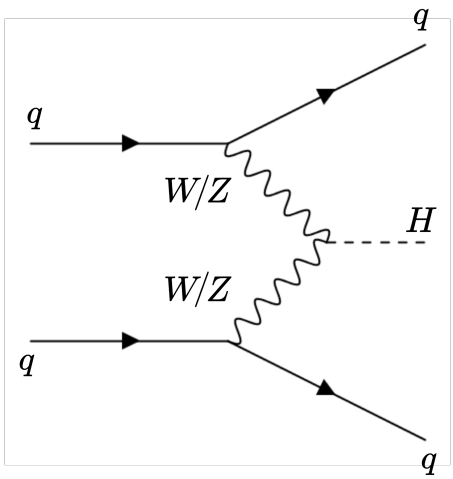


Figure 1.5: Feynman diagram of the Higgs boson vector boson fusion production process. While this is the process with the second-largest cross-section, it features two forward jets from the outgoing quarks making it a useful topology for triggering without having to trigger on the Higgs decay products.

Another key goal was SUSY discovery or placing limits on possible SUSY particles. It was expected that SUSY decays would always contain a light, weakly interacting SUSY particle which would lead to $E_{\mathrm{T}}^{\mathrm{miss}}$ and the decays of the other SUSY particles to b-jets and τ-jets. Such decays require good, full-coverage calorimetry reconstruction and good b and τ-tagging performance which relies on reconstructing transverse displacements of

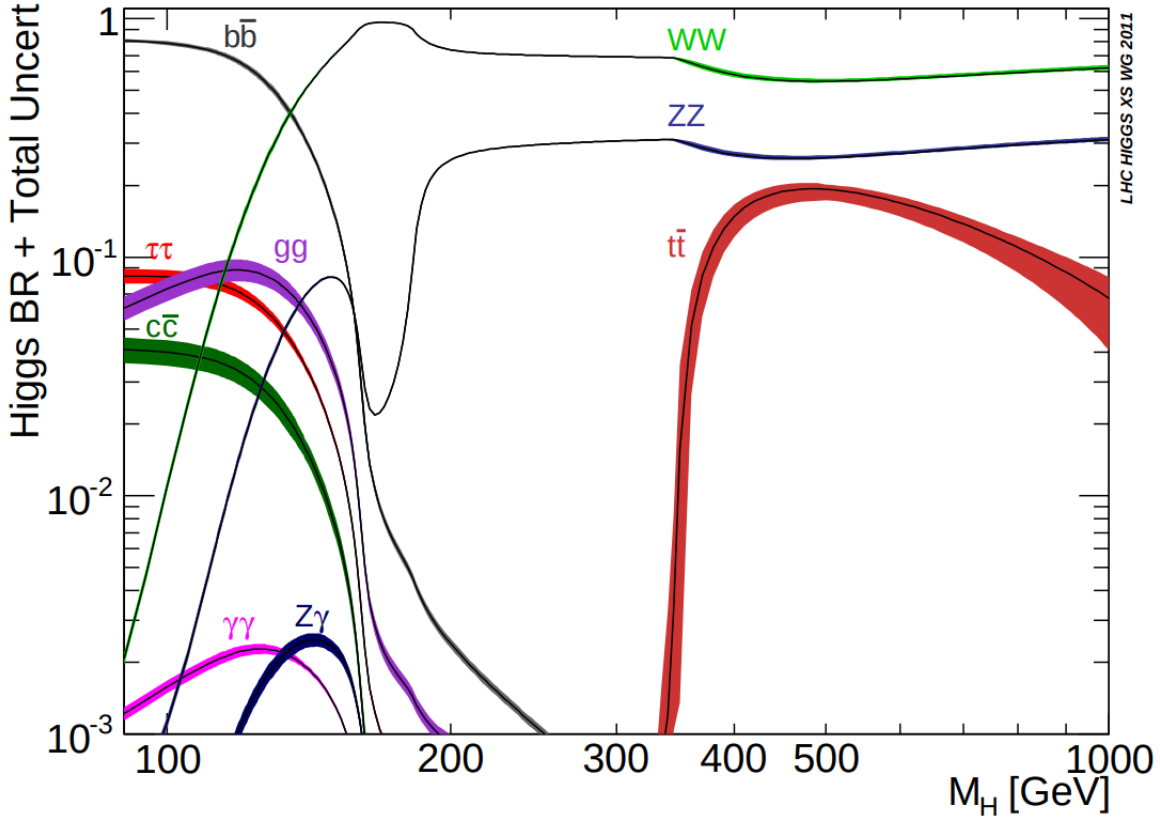


Figure 1.6: Predicted branching fraction as a function of the Higgs boson mass for a variety of decay modes [22]. When considering the Higgs boson mass of 125 GeV it is clear the bb channel dominates with significant contributions from gg, $\tau\tau$ and cc, all of which are difficult to separate from hadronic backgrounds. The major discovery channels; ZZ and $\gamma\gamma$ have far lower branching fractions but are far cleaner signals for both triggering and reconstruction.

O 10 μm with the inner tracker. Due to the differing lifetimes of the b and τ, 1.5 ps versus 0.2 ps their displacements also differ; millimetres versus 10s of micrometres, meaning the inner tracker needs to be able to reconstruct this displacement with O 10 μm resolution.

Furthering the understanding of SM physics was also a key part of the CMS physics reach with precision measurements targeted across the range of QCD, electroweak and flavour physics. At the LHC the top-quark production rate would be high making top-quark measurements a key area for CMS with high b-tagging efficiency required for reconstruction of top-quark decays. Various SM violating processes were also considered, such as $\tau \rightarrow 3\mu$, which requires well reconstructed muons across the detector η coverage at lower p_{T}s.

Finally, the LHC is a hadron collider; therefore a large background from hadronic pileup was expected so the ability to reconstruct jets to high efficiency and perform highly accurate vertex reconstruction was essential. In the initial run of the LHC the average distance between vertices was 20 mm which has since fallen to 9 mm for run 3. Separating these interactions requires high granularity and well time-resolved detectors. It also meant having robust muon triggering systems in case the calorimetry was blinded by the pileup.

1.3.2 Magnet System

The central part of the Compact Muon Solenoid, is the solenoid. This 13 m long, 6 m wide, 3.8 T superconducting solenoid has the primary purpose of bending charged particle tracks as they traverse the detector. Shown in figure 1.3, within the bore of the magnet (in grey) sits the tracker, which reconstructs the curved charged particle tracks, allowing their momentum to be determined. The return yoke of the magnet (in red) is integrated within the muon stations. This means muon tracks curve one way inside the tracker and the other way inside the muon stations giving twice the measurements on the muon momentum, improving the muon momentum resolution. The double curving of muons in the detector occurs up to an η of 1.5 after which the muon trajectories are too parallel to the magnetic field to experience the full bending. The magnetic field strength allows CMS to reconstruct muons with unambiguous charge up to 1 TeV. The large internal field of the magnet also means low energy charged particles will be unable to leave the tracker volume, reducing the overall background in the calorimeters.

1.3.3 Tracker

The CMS tracker [23] is an all-silicon, two-part detector, 5.8 m long and with a diameter of 2.5 m. The CMS physics requirements mean the tracker needs to be high granularity with a fast response time but also needs to be radiation hard to cope with the harsh detector environment. Silicon trackers meet all these requirements but need efficient cooling and readout electronics. It is preferable to have low material budgets for the tracker to reduce multiple scattering of charged particles in non-active detector material. Scattering makes the trajectories more difficult to reconstruct, and it reduces particle energy meaning total charged particle energy is mismeasured in the calorimeters. Another issue, particularly for electrons, is bremsstrahlung where the trajectories of the charged particles spiral inwards as they interact with the tracker material in the CMS magnetic field. This effect necessitates dedicated electron track reconstruction that also takes into account the bremsstrahlung photons deposited in the calorimeter [24]. Finally, it is also important to reduce photon conversion in the tracker; a key requirement of the $H \rightarrow \gamma\gamma$ search was to keep the photon conversion rate below 50%.

The innermost detector, the inner tracker (shown in green in figure 1.7, the additional layer of the inner tracker was part of the Phase-1 upgrade), consists of three layers of pixel modules [4] in the barrel and two endcap layers with a total active area of 1 m^2. The inner tracker provides precise 3D reconstruction of charged particle trajectories and vertices with a point spatial resolution of 10 μm in r-ϕ and 15–20 μm in z for the barrel layers.

The outer tracker, also known as the silicon strip tracker (shown in red and blue in figure 1.7), consisting of ten layers of silicon microstrip sensors in the barrel and eleven in the endcaps uses cheaper strip detectors as the granularity does not need to be as high. With the high magnetic field and the fact particles are radiating outwards, the occupancy in the outer tracker is reduced making the use of pixel detectors in these layers unnecessary. The strip tracker is further subdivided into an inner and outer barrel tracker and inner and outer endcap tracker. The barrel region has a 23–52 μm resolution in r-ϕ and 230–530 μm resolution in z. The total strip tracker has nearly 10 million silicon strips with a total area of over 200 m^2. Both the inner and outer trackers are unable to contribute to the hardware trigger decisions due to the large data-rates being produced by the detectors. Instead, they use analogue read out chips to amplify and buffer signals pending a trigger accept signal.

A major issue in hadron colliders is pileup and high granularity tracking allows for this pileup to be vetoed versus the high p_T "interesting" interactions. Vertex location down to the mm level and secondary vertex tagging

[4] A module refers to a structure containing the silicon sensor, the readout chip for the sensor and other infrastructure such as power for the sensor and chip. These modules are organised in ladders in the barrel and wedges in the endcaps

is important and high granularity pixel detectors make this possible. The CMS inner tracker has a reconstructed vertex resolution of 50μm [25], far better than the average 20 mm spread of vertices in the 60 PU of run 3. High efficiency for b and τ-tagging is must for a variety of physics signatures including top quark decay modes, Higgs decays, as well as possible SUSY stop decays, which also requires high efficiency secondary vertex finding.

During the second long shutdown, between 2018 and 2022 (shown in figure 1.2), CMS underwent its Phase-1 upgrades. Part of this was a replacement of the inner tracker [26] because of the radiation damage from the previous LHC running period and the planned doubling in luminosity for the next run. This upgrade saw an additional inner tracker layer added at a closer radius to the beam pipe and a reduced overall material budget. The structure of the Phase-1 tracker is shown in figure 1.7 though the only difference between this layout and the original design is the addition of the extra pixel layer closest to the interaction point.

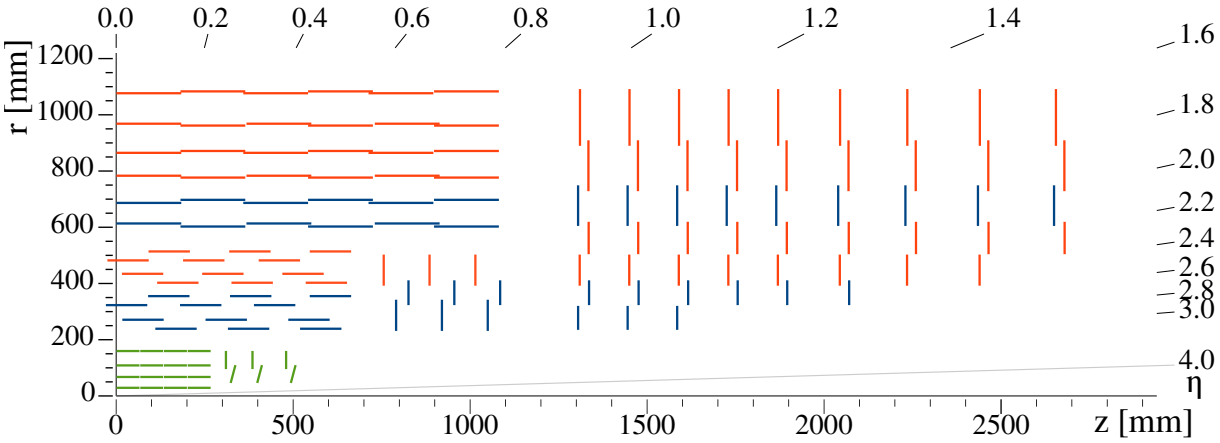


Figure 1.7: One quarter of the Phase-1 CMS tracker in the r-z plane [26]. The inner tracker is shown in green with the additional layer introduced in Phase-1 closest to the beamline at $r = 0$. The outer tracker is shown in blue and red. In red are the doubled sided strip modules which use strip detectors with two sets of strip modules and the single sided modules in blue [26].

1.3.4 Calorimetry

ECAL

The electromagnetic calorimeter (ECAL) [27] is a homogeneous calorimeter consisting of 75,848 high-density lead tungstate crystals read out by avalanche photodiodes in the barrel and the more radiation tolerant vacuum phototriodes in the endcaps with a coverage of $|\eta| < 3.0$. It is designed to have a high granularity (2.2 cm), fast response time[5] (100 ps) and to be radiation hard. The crystals measure $22 \times 22 \times 230$ mm^3 with a Molière radius of 2.2 cm meaning most EM showers are constrained to a few crystals. Because of optical damage in the crystals from the radiation, the ECAL has a monitoring system that measures the light response over time. This allows the ECAL to maintain performance over the many years of CMS running by adding correction factors to the reconstructed energy deposits in the crystals. In the endcap region there is an additional preshower detector consisting of interleaved lead absorbers and silicon sensors. This is used to reject $\pi \rightarrow \gamma\gamma$ decays where the granularity of the endcap ECAL is not fine enough to distinguish the two photons. The fake single isolated photons from this decay are the main background of the $H \rightarrow \gamma\gamma$ decay.

[5]The time taken for a signal to be registered after a particle has deposited energy in the crystal.

The ECAL is designed to capture electromagnetic showers from electrons and photons by absorbing the EM decay chain of these particles. The ECAL has a high energy resolution of 0.4% for electrons and photons of 100 GeV and is fast with a crystal response time[6] of 25 ns and can achieve a 100 ps time resolution across the detector for large energy deposits [28]. It will also absorb energy from charged hadrons and muons as they traverse the detector adding to the reconstruction of these objects. One of the main physics driving cases for the high granularity of the ECAL was the detection of $H \rightarrow \gamma\gamma$, which required a mass resolution of ~1% in the ECAL as well as effective trackless cluster isolation with precision vertex finding [29].

HCAL

The hadronic calorimeter (HCAL) [30] is a sampling calorimeter with brass and steel absorbers and plastic scintillator active detectors. Due to the magnet solenoid, the HCAL is relatively thin and so an additional tail catching calorimeter is placed outside the solenoid to capture very high energy hadronic decays. The HCAL spans a high pseudorapidity region up to $|\eta| = 3$, with additional hadronic forward (HF) detectors with coverage up to $|\eta| < 5$, useful for the high occupancy forward regions of CMS. Ultimately, the HCAL energy resolution is poor; the energy resolution is almost $\frac{100\%}{\sqrt{E}}$, however, CMS employs a reconstruction technique called Particle Flow (PF) [31] to improve reconstructed object energy resolution. PF uses combined information from across the detector to link objects in different subdetectors, giving multiple measurements of each object. PF is performed both in the HLT and in offline analysis, but not in the hardware-level trigger (discussed below) because the tracker cannot be read out at the full 40 MHz. It allows different energy and momentum measurements in different parts of the detector to be combined to give an improved energy reconstruction by removing energy deposits from overlapping particles.

The HCAL is designed to capture hadronic showers from jets and hadrons and is the only subdetector that detects neutral hadrons. Their reconstruction is essential for accurate $E_{\mathrm{T}}^{\mathrm{miss}}$ calculations which are needed for neutrino reconstruction or additional heavy neutral SUSY particle reconstruction.

1.3.5 Muon Systems

As the name implies, muon reconstruction is a key part of the CMS detector, with muon identification, muon measurement, and triggering on these muons all essential parts of the CMS muon systems [32]. There are three complementary systems for detecting muons in CMS. In the low muon flux, uniform magnetic field barrel region, drift tube (DT) are used. These are organised into four stations inside the layers of iron magnet yoke. These stations are further subdivided in the first three stations into eight chambers, four measuring r-ϕ coordinates and four measuring z coordinate. The final muon station only has an r-ϕ measurement. The drift tubes have a 260 μm hit resolution and 1.8 mrad resolution in the r-ϕ plane [33].

In the endcap regions, with high muon flux and a non-uniform magnetic field, cathode strip chamber (CSC) are used. These have fast response time, better granularity and increased radiation resistance compared to the DTs. Each endcap has four stations interleaved with the yoke return. The cathode strips run radially outwards giving a precise measurement in the r-ϕ plane with a 150 μm resolution offline and 2 mm resolution for the Level-1 trigger. The anode wires of each chamber runs perpendicular to the cathode strip to give an η measurement [34].

[6] The time taken for the photon activity in the crystal to return to zero.

Both these technologies offer good p_T resolution of less than 10% for muon p_T up to 1 TeV. When combined with the opposing bend inner tracker measurement, high performance at low p_T, with resolutions of 1%, is achieved for muon p_T < 100 GeV. Together these muon systems provide muons to the Level-1 trigger system with a > 98% efficiency.

With possible high luminosities being reached over the first three runs an additional muon station consisting of resistive plate chamber (RPC) was added in both the barrel and endcaps. RPCs are fast (2 ns response time versus the 25 ns bunch crossing interval) but with lower granularity than the CSCs and DTs making them a suitable detector for muon triggering in the event that the other two systems cannot deal with the pileup-induced background. The RPCs are embedded inside the DT stations with two in each of the first two stations and one in the remaining layers. In the endcap, planes of RPCs are inserted within the first three CSC stations. With p_T> 25 GeV muons the addition of the RPCs give a 2% efficiency increase in the overlap regions between the barrel and endcap [35].

Accurate muon reconstruction and triggering was a key part of the search for the $H \rightarrow 4\ell$ signal and good reconstruction of the four muon final state was essential. In general muons are a key element to many reconstructions as they are minimally ionising and do not suffer as much from the large radiative losses as electrons do, making them easy to detect. It also means they can have the best momentum resolution of any of the detected particles in CMS. As the muons are minimally ionising, they appear strongly over the hadronic background from the pileup so are essential for CMS triggering as pileup increases.

1.3.6 Trigger and DAQ

In one bunch crossing of the LHC, CMS creates around 1 MB of data from all the subsystems; at the 40 MHz bunch crossing rate this results in a data-rate of around 40 TBs^{-1}. This cannot be stored and analysed so rate reduction is needed in a way that keeps "interesting" physics signatures, such as those coming from the Higgs and discards soft hadronic background. CMS achieves this with a two-stage trigger.

The first trigger, the Level-1 (L1) trigger [36], is a hardware-based trigger implemented on field programmable gate arrays (FPGA) and application specific integrated circuits (ASIC) connected by electrical cables, and is close to the detector—in the adjacent cavern. When originally designed, the trigger had a 3.2 μs total latency, of which less than 1 μs was trigger decision time[7], and a 100 kHz readout rate. While low granularity information from the muons and calorimeter systems are used to decide to keep an event, the full granularity detector information is stored in buffers ready to be readout if the trigger is fired. The L1 trigger is divided into local and global components shown in figure 1.8. The local components of the muon trigger used local DT and CSC segment and track finders to reconstruct muons. These, along with RPC pattern triggers, are combined in the global-muon trigger (GMT) to create muon objects. The calorimeter generates trigger primitives from super (5×5 crystal) clusters of cells in the ECAL, HCAL, and HF, which give single trigger tower E_T. These TPs are combined in the regional calorimeter triggers that create electron and photon candidates, as well as energy sums, a τ veto and some information for the GMT. The global calorimeter trigger (GCT) combines all of this to create global level energy sums and the ranking of calorimeter trigger objects. The GT receives information from the GMT and GCT and combines it in a set of 128 complex decisions based on a tailored trigger menu that evolves depending on the physics needs of the experiment.

[7]The rest is due to the latency of sending information to and from the adjacent cavern.

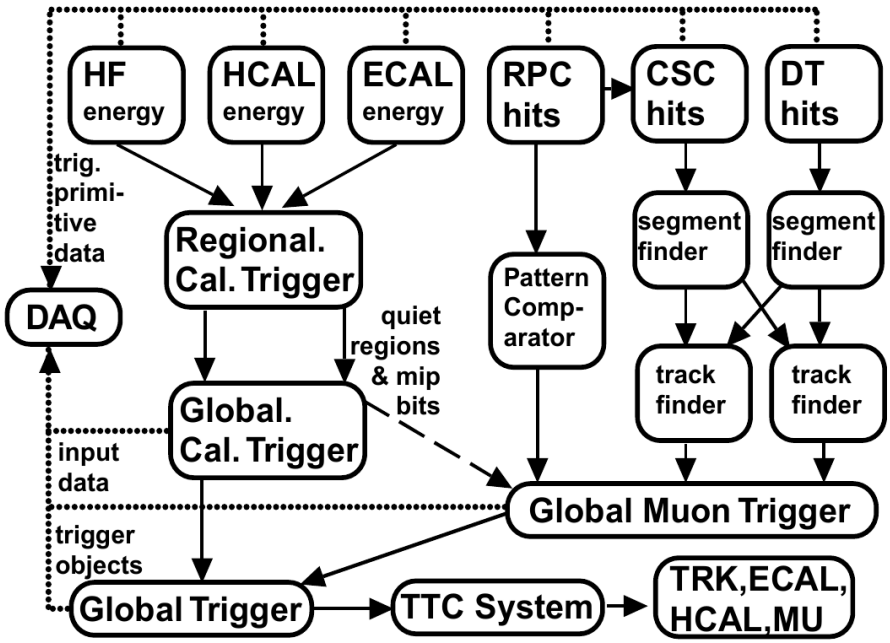


Figure 1.8: The original CMS L1 trigger system schematic showing the trigger primitives at the top and layers of control before reaching the global trigger. Only information from the calorimeters and muon systems are included and undergo their own local reconstruction before a final trigger decision is made in the global trigger. The global trigger interfaces with the data acquisition (DAQ) and timing, trigger and control (TTC) systems to read out the entire detector buffers if the trigger is fired.

The data acquisition (DAQ) and timing, trigger and control (TTC) system [37] of the detector reads out at the rate of the L1 accept, 100 kHz, generating a data flow of around 100 GB/s from over 650 data sources. These are read from the detector front ends into the event farm, high level trigger (HLT). The HLT performs a fast analysis of the entire, full-granularity, detector data in a faster version of an offline reconstruction and reduces the readout rate further to around 1 kHz utilising an off-detector CPU farm. The remaining events are written to disk for offline reconstruction and analysis.

The L1 trigger was upgraded in the CMS Phase-1 upgrades [38] to deal with the increased pileup in the second run of the LHC. Nominal pileup in run 1 was around 25 events per bunch crossing, which increased to 50 in run 2 and an increased energy of 13 TeV. The original L1 trigger was ill-equipped to deal with this increased background while maintaining electroweak sensitivity. The choice was made to move to an all FPGA architecture for the L1 trigger with high bandwidth optical links, and an increased latency of 3.8 μs. A summary of the FPGAs used between the original design and the Phase-1 upgrade is shown in table 1.1. The large increase in available resources made the trigger more flexible and allowed it to evolve as the LHC conditions and physics requirements of the experiment rapidly changed; at this time SUSY had not been found where it was expected so new triggering strategies were needed. The overall structure of the trigger remained the same, demonstrated by figure 1.9, but there were improvements to all of the trigger electronics with a complete replacement of the processing cards. There were also various improvements to the algorithms, taking advantage of the far larger FPGAs available, with examples being a machine learning boosted decision tree for muon track quality implemented in the global muon trigger and the use of dynamic clustering in the calorimetry triggers [39]. This shift to more advanced and flexible algorithms using machine learning is a significant advantage of the FPGA-based trigger and motivates the continued use of even larger FPGAs with more machine learning algorithms for the Phase-2 upgrade — discussed

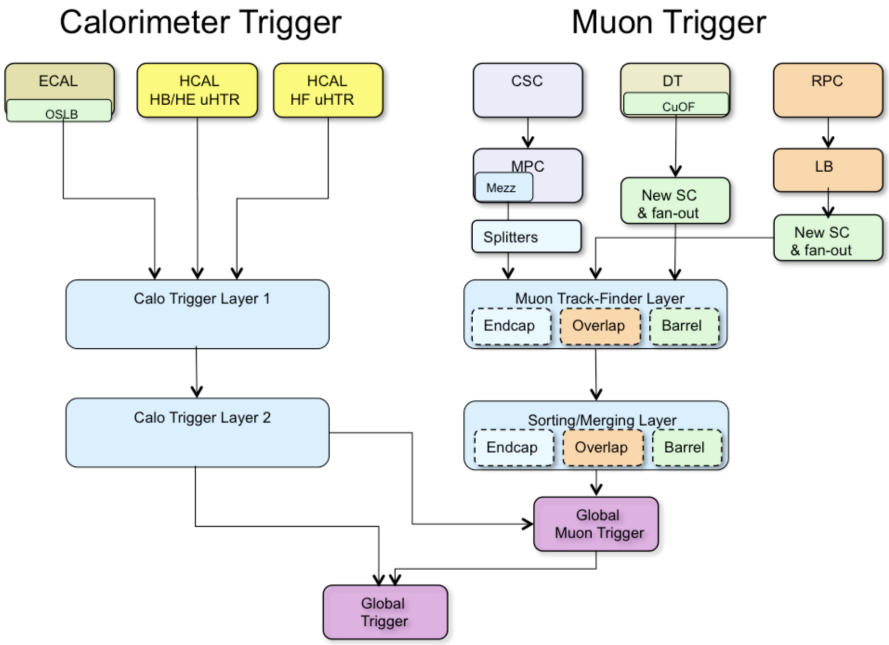


Figure 1.9: The Phase-1 CMS L1 trigger system schematic showing the trigger primitives at the top and layers of control before reaching the global trigger. This trigger has the same basic architecture as the original trigger shown in figure 1.8 but with improved algorithms and replaced hardware.

in the next chapter.

1.3.7 Physics Contributions and Open Questions

The main physics contribution of CMS (so far) is the discovery of the 125 GeV Higgs boson in 2012 [40]. The main channels, as predicted, were the $H \rightarrow \gamma\gamma$ and $H \rightarrow ZZ \rightarrow 4\ell$ channels. In the years proceeding this discovery CMS has built a detailed picture of the Higgs and its decays [41]. The coupling of the Higgs to all but a few of the fermions has been measured; Higgs to the lightest quarks and electron have yet to be observed, though are very difficult to accurately reconstruct because of the LHC's hadronic background and low branching fractions. Figure 1.10 summarises the results from CMS for a number of decay channels, including $H \rightarrow \mu\mu$, a result that was thought unobtainable in the standard LHC configuration without the high luminosity upgrade. However, there remains many open questions in Higgs physics. A key measurement that is expected to be reachable with the LHC energies and enough data, is the Higgs self coupling. This is a key test of electroweak theory and gives insight to the shape of the Higgs potential. Another possible measurement is the Higgs to c quark pair that is so far elusive because the high hadronic background; with highly efficient jet tagging and enough data this measurement could be possible.

The Higgs mass was determined to be 125 GeV, which is very light and requires a large fine-tuning to explain why it is so light in comparison to the scale at which the SM is valid. Natural answers to this came in the form of SUSY, and in part motivated the LHC. SUSY was expected below 1 TeV for a light Higgs, which has not been observed. Many different searches [42] over the last three runs of the LHC have not yielded the set of supersymmetric partners to the SM particles which would have neatly prevented this fine-tuning problem as well as giving a viable candidate for dark matter. The extent to which SUSY has been disproved in numerous channels

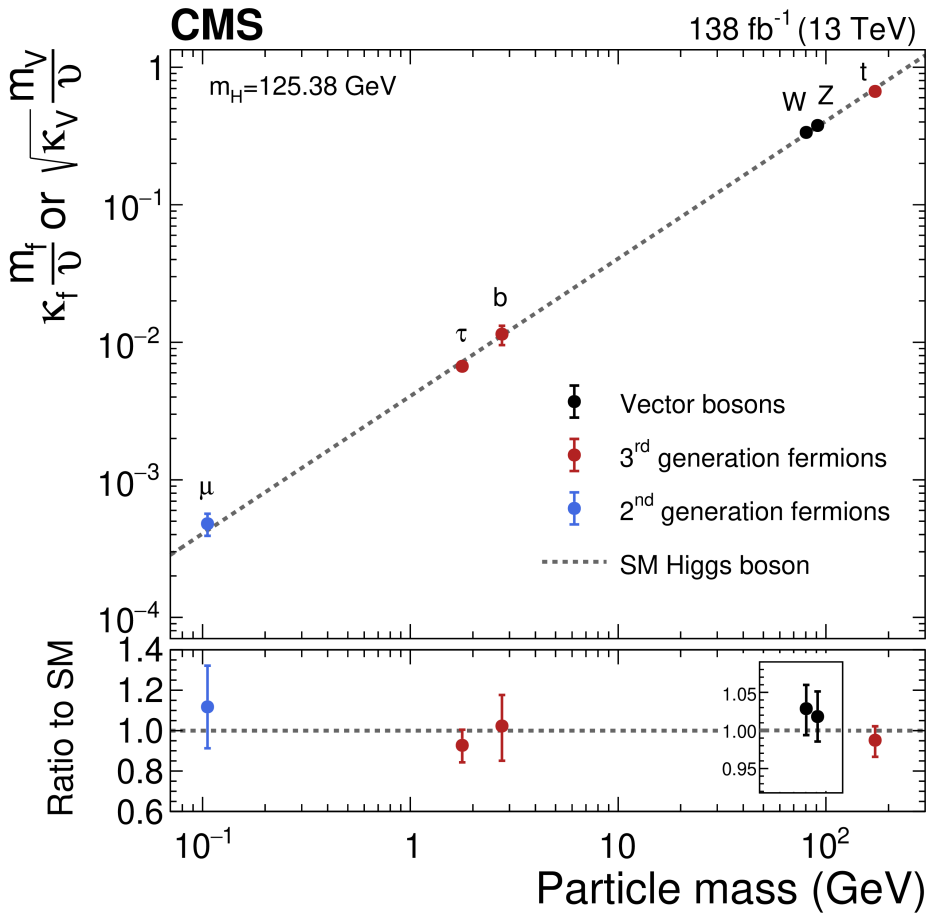


Figure 1.10: The measured coupling of the Higgs boson to fermions and heavy bosons as a function of their mass. Shown is the expected line given an SM Higgs boson and the high degree of agreement between the found Higgs boson and observed couplings. The fact that the measurements for vector bosons and multiple generations of fermions agree so closely with the SM theory is a demonstration of the merits of the SM and ability of CMS to accurately measure it. These results were produced with 138 fb^{-1}, 20 times lower than what the HL-LHC will provide [41].

is a significant result of CMS and ATLAS and leaves open many questions for theoretical physicists and has motivated more sophisticated analyses techniques. Searches are ongoing with different forms of SUSY that introduce extensions to the SM with hints of additional Higgs bosons being present at different mass ranges. However, more data is needed to claim observation of these extensions.

CMS has also performed high precision measurements of the SM, such as the top mass with a 30% improvement versus the previous best measurement [43]. So far these kinds of measurements have very closely followed the expectations of the SM theory with some hints of possible new physics, with large enough datasets it is possible to either confirm or deny these hints. For example a goal of CMS in the coming years is a precise measurement of the W mass to help to explain the tension with the SM observed by the CDF II experiment [44].

1.4 The High-Luminosity LHC

In order to provide more data to the experiments, the LHC is being upgraded during the third long shutdown between 2026 and 2029 to the high-luminosity LHC [45]. The schedule for these upgrades are shown in figure 1.2. The aim of the upgrade is to increase the luminosity of the beams in each of the collision points by 5 to 7.5 times the LHC design luminosity. More protons will be interacting than ever before meaning the number of rare events occurring increases, with the potential of 170 million Higgs bosons and 120,000 Higgs boson pairs being produced in the next era of the LHC. The HL-LHC will feature more powerful quadrupole magnets used for focusing the beams, an increase in the number of protons per bunch crossing by a factor of two, and an upgrade of the beam focusing optics used to create the interaction region in the LHC experiments. These upgrades mean the average number of proton-proton collisions in each bunch crossing will rise from the LHC's original number of 25 in run 1 to 140 for run 4 with possibility to rise to up to 200 within the HL-LHC's lifetime. The increased instantaneous luminosity means the integrated luminosity over the HL-LHC lifetime could be as high as 4000 fb^{-1}, which is 10 times higher than the data collected up to the end of run 3.

With this increased dataset, CMS can achieve a number of its physics goals, notably the Higgs self coupling observation [46]. However, this is only possible if CMS remains sensitive to the same physics and is not blinded by the increase in pileup the increased luminosity entails. This, in part, necessitates upgrades to CMS, notably in the trigger, again to deal with the increased pileup. Another issue with the increased luminosity is higher radiation; the current tracker will be replaced to deal with this radiation as will the endcap calorimetry. The other major reason for the CMS upgrade is to expand the physics capabilities of the detector and trigger system. CMS has not found the physics that was expected in the first three runs and the physics goals of the experiment are constantly adapting meaning the detector needs to constantly adapt and be ready for unconventional signatures and analyses. The upgrades being made to CMS and the potential improvements they can make to the CMS physics programme are detailed in chapter 2.

1.5 Field Programmable Gate Arrays

FPGAs are matrices of configurable logic which can be reprogrammed depending on the use case [47]. As they are reconfigurable logic they are fast and far more flexible than conventional processing architectures such as CPUs or GPUs. This makes them very desirable for triggering systems where the requirements of the trigger might change over time and very fast processing is a must. While CPUs or GPUs have a factor 10 higher clock speeds, GHz compared to FPGAs' 100s MHz, FPGAs are arbitrarily parallelisable; all data passing through the FPGA can be processed simultaneously within the constraints of the resources on the chip. Resources refer to the overall number of specific logic blocks on the FPGA which have different uses. There is also routing fabric that links logic blocks together and allows data to flow across the FPGA.
The four main resources on an FPGA are:

- Look-up table (LUT): small memory units that determine an output given an input. These can be seen as a truth tables and are used to implement combinatorial logic such as AND, OR, NOT gates.
- Flip flop (FF): maintain the state of a given signal in an FPGA between clock cycles, also known as registers.
- Block random access memory (BRAM): a more efficient way of storing large arrays and act as a memory in the FPGA.
- Digital signal processor (DSP): dedicated processing units for multiplication and accumulation.

These resources are organised in stripes in the FPGA, as is shown in the bottom part of figure 1.11. LUTs and FFs are packaged together in units called logic cells which form the majority of the processing on the FPGA. Rows of these logic cells are interspersed with alternating rows of DSPs and BRAMs so that the path length from processing blocks is minimised. The routing fabric populates the dark area between these rows and forms tree like structures to reach all processing elements needed by an algorithm.

In larger FPGAs logic is organised in super logic regions (SLRs). In the physical FPGA the silicon is organised in a two-dimensional plane (the SLR); in order to increase the logic without the physical size, multiple two-dimensional planes are stacked on top of each other. An unrolled schematic of an FPGA with the SLRs highlighted is shown in the top part of figure 1.11. Communication within these planes is fast, but communicating between the planes is slower because of fewer interconnections between the planes. The data input and output for the FPGA is organised along its edges where the physical electrical connections between the processing board and FPGA are located, shown by the small boxes along the edges of the schematic in figure 1.11.

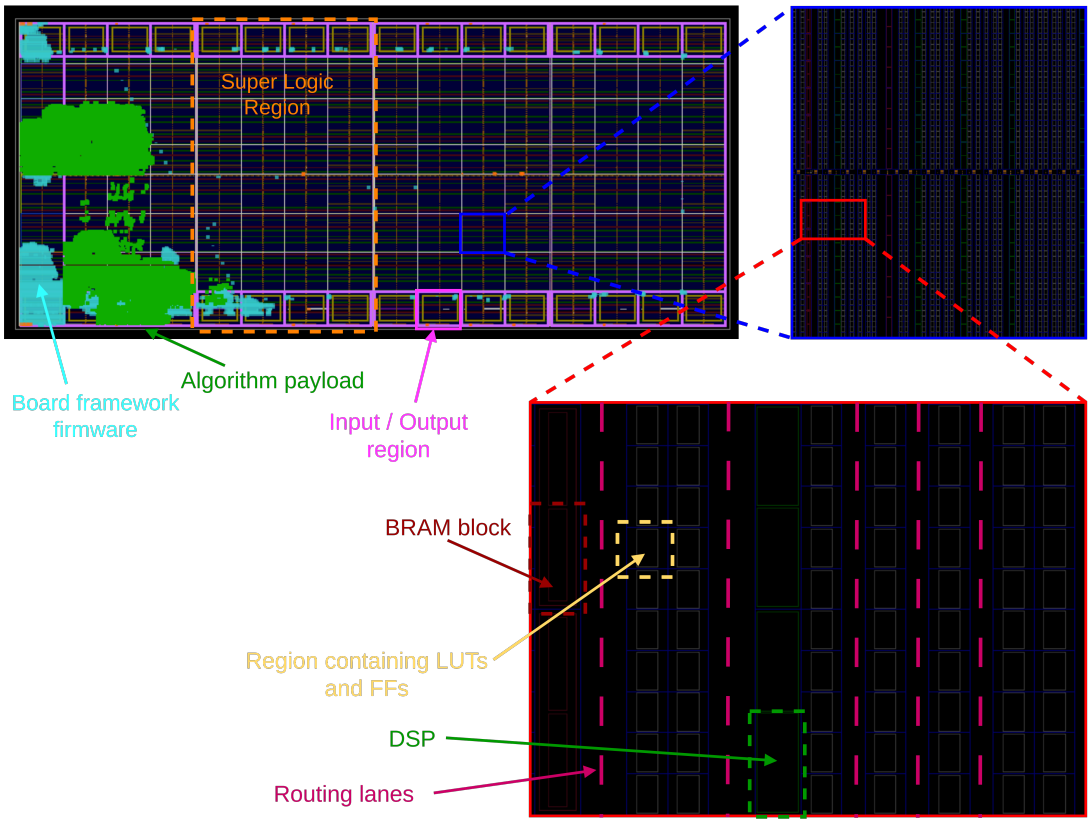


Figure 1.11: Floorplan of the Virtex Ultrascale+ VU13P [48] field programmable gate array, the chip of choice for the CMS Phase-2 upgrade. The global view of the chip at the top shows the overall placement of the four super logic regions, in reality these regions are stacked, and the chip has a square form factor. Also shown is the algorithm highlighted in green, the board framework (see chapter 6, section 6.2.3) in blue and the location of the FPGA input and output around the edge of the FPGA in pink. The zoomed in region shows the internal structure of the FPGA with rows of BRAMs and DSPs interspersed by LUT and FF regions. The lanes between these blocks are used for routing fabric where the routing forms a tree-like structure to transfer data between the processing areas.

Logic resources can be combined to create highly parallel algorithms across the FPGA through the use of a hardware description language (HDL). These languages are used to define the signals and logic operations in an FPGA. Commonly used are VHDL and Verilog, which are low-level HDLs. In more recent years high level synthesis (HLS), which takes C++ and compiles it into an HDL, is becoming popular as a design tool as it does

not require knowledge of difficult-to-understand HDLs. Once an algorithm for the FPGA has been written, known as firmware, it is synthesised, which maps the algorithm to sets of logic gates. At this point the algorithm can be simulated; a process where the flow of data across the algorithm is simulated one clock cycle at a time. These simulations are used to verify the synchronisation of the algorithm and check the output is as expected. Parallel to the firmware development, an emulation of the firmware is also written (typically in C++), which allows algorithms to be developed quickly in emulation and gives a reference for the firmware simulation and hardware implementation to be compared against.

Once correct, this synthesised mapping is then implemented on a target FPGA, where the actual resources defined in the synthesis are routed and placed on the chip. This process can take multiple hours for complex designs on large FPGAs. A process called floorplanning is used to physically place logic blocks on the silicon where consideration of the placement of logic within SLRs is important because of the relatively slow communication between SLRs in the chip. In the implementation step a target clock frequency is also decided which tells the FPGA at what clock speed to run at, something that can lead to issues if a certain operation on the FPGA cannot be completed in time because of delays moving the data through the logic gates.

The total time taken for data to flow through the algorithm is called the algorithm's latency and can be measured in clock cycles or total time (clock cycles divided by clock frequency). The time taken for an algorithm to be ready to receive new data is called the initiation interval (II).

When programming FPGAs there are some key considerations:

- Is the operation possible? Certain mathematical operations are not directly possible using bare FPGA components, for example, division. Care is needed to split an operation into sets of multiplications, additions, and bit shifts.
- Fixed point mathematics: floating point mathematics is not possible in FPGAs because of the variation in bitwidth of the numbers, instead fixed-point mathematics is used where the integer and decimal bits are predetermined before synthesis. A consideration of how many bits are needed for each internal signal is also necessary as more bits require more resources.
- Synchronisation: certain operations take certain numbers of clock cycles, e.g. using a DSP needs three clock cycles so care needs to be taken when synchronising signals.
- Timing: signals traversing long path lengths in FPGAs take time and may not arrive at the next logic block before the next rising clock edge. Path lengths and internal complexity of FPGA logic need to be managed otherwise the desired clock speed might not be possible leading to incorrect outputs and a design that fails to meet timing.

There are a number of advantages to using FPGAs in a trigger system. They have the low latency and data throughput focus of ASICs but with the flexibility of CPUs and GPUs. They can be reprogrammed as the physics requirements or detector performance evolves. FPGAs have always been used in the L1 trigger for these reasons but in the 30 years since CMS was proposed the FPGA technology in use has improved, as is demonstrated in table 1.1. Between the original design and Phase-1 upgrade, where CMS fully embraced the use of FPGAs across the trigger system, the number of logic cells on the FPGA used has increased by 20 times and the DSPs by nearly 40 times. This large increase in available resources allowed the L1 trigger to extend its physics reach and start to introduce machine learning algorithms to the trigger for the first time. The Phase-2 upgrade will see another six times as many logic cells, four times as many DSPs, and nine times as much BRAM available in a single FPGA which opens up a range of possibilities for the trigger, as is described in the next chapter.

Table 1.1: Specifications for the FPGAs used by the CMS L1 trigger system for the original design, Phase-1 upgrade and future Phase-2 upgrade. These FPGAs were specifically used for the calorimeter trigger but are typical of the FPGAs used across the wider system. Logic cells are a combined LUT and FF with additional carry logic.

CMS Era	Years	FPGA	Logic Cells	BRAM (Mb)	DSPs
Original	2008–2016	Virtex-II	32,000	1.7	96
Phase-1	2016–2025	Virtex 7	693,000	52.9	3,600
Phase-2	2028–Onwards	Virtex Ultrascale+ VU13P	3,780,000	455.0	12,288

Chapter 2

The Phase-2 Upgrade of CMS

In order to maintain sensitivity to electroweak processes and expand the physics capabilities in the HL-LHC environment, the CMS detector is being upgraded. It is planned that the upgrades will be commissioned in 2029 and mark the Phase-2 era of the CMS experiment [49]. Key upgrades are being made across the detector and trigger system, taking advantages of decades of research in high energy physics (HEP) detector and computing technology. The endcap calorimetry will be replaced and consolidated with the high granularity calorimeter (HGCAL), the tracker will be replaced and will provide tracks to the Level-1 trigger for the first time, and the L1 trigger will be completely replaced, taking advantage of upgraded, higher granularity detectors and the latest FPGAs with a readout rate of 750 kHz. An outline of the physics programme for CMS during the HL-LHC is provided as well as the upgrades being made to CMS in order to reach these physics goals.

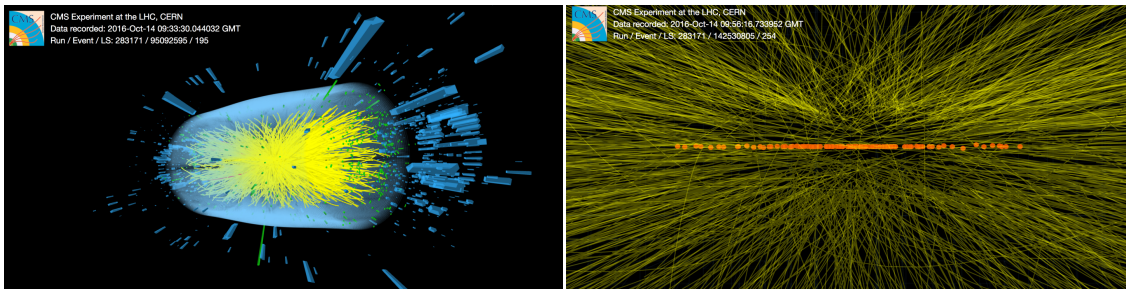
2.1 Phase-2 Upgrades

The HL-LHC will produce unprecedented peak instanteneous luminosities reached in the CMS detector. Peak instantaneous luminosities of 7.5×10^{34} cm^{-2}s^{-1} will lead to an integrated luminosity of up to 4000 fb^{-1} by the end of the HL-LHC lifetime. Compared to the integrated luminosity at the end of run 2 of 200 fb^{-1} this far larger dataset allows the LHC experiments to increase their precision in numerous SM measurements, as well as probing new physics otherwise unavailable to these experiments. This increase in instanteneous luminosity comes with an increase of simultaneous proton-proton interactions (pileup) with the average from run 3 of 60 moving to up to 200 in the highest specification of the HL-LHC. This will result in higher radiation in the CMS detector as well as motivating a complete replacement of the trigger and DAQ system to deal with the background pileup, and maintain physics sensitivity. Figures 2.1a and 2.1b show the scale of this challenge. These event displays, from a high pileup fill of the LHC during run 2, only have a pileup of 100; the HL-LHC could have double this and the detector needs to be able to consistently and accurately reconstruct these high multiplicity events. There are many specific physics analyses being considered in the HL-LHC era; some key measurements are listed below however a more complete list can be found in [46].

2.1.1 Physics Programme

Precision Standard Model Measurements

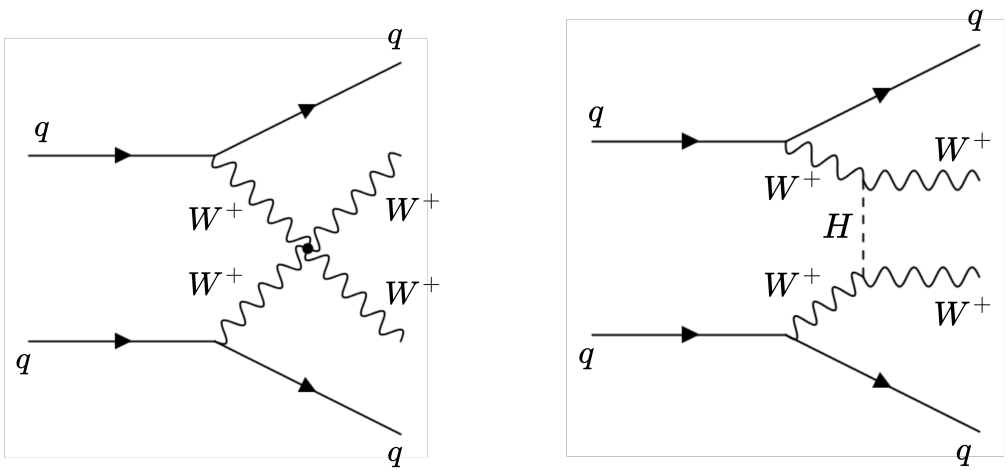
An important test of the SM is in vector boson scattering (VBS), characterised by the presence of two W-bosons with the same electric charge and two forward jets. This process, first observed in 2017 [51] [52] and shown in



(a) (b)

Figure 2.1: CMS event displays for a high pileup fill of the LHC where isolated bunches were collided to give an average pileup of 100. Figure (a) shows the total number of reconstructed tracks being produced with very high densities of tracks across the tracker, shown by the yellow tracks in the image. Figure (b) shows the 86 reconstructed vertices in the event, again showing the high density of reconstructed tracks in yellow and subsequent vertices in orange. This event is only a single 100 PU event in a dedicated LHC fill; in the HL-LHC the pileup could be up to double this and CMS will have double the tracks and vertices to reconstruct at the full 40 MHz [50].

figure 2.2, probes the self interaction between the weak gauge bosons. It is a rare process (with a cross-section of a few fb) and the larger dataset of the HL-LHC era allows for a more precise characterisation of electroweak physics. Due to this process' sensitivity to extended Higgs models through a diagram similar to 2.2b, it can add evidence for or against multi-Higgs extensions to the SM. An important signature of this process are high pseudorapidity jets, therefore precise reconstruction from the forward calorimetry and high tracker acceptance is needed.



(a) (b)

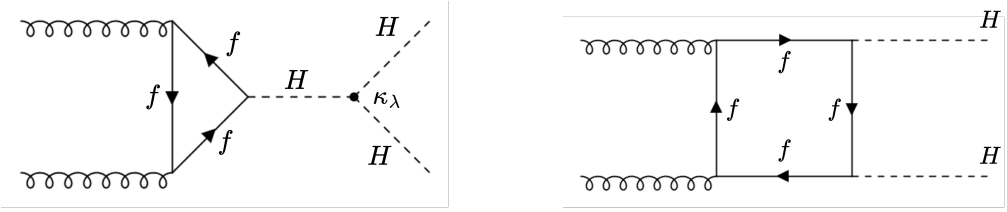
Figure 2.2: The Feynman diagrams for the same sign vector boson scattering. On the left is the four W boson vertex and on the right the process via an intermediate Higgs boson. There is an equivalent diagram for the process via an intermediate Z or gamma boson. In order to prevent a continuous increase in production rate as scattering energy increases the diagrams cancel and any deviation in this cancellation allows the Higgs interaction with the other bosons to be explored up to high energies.

Precision top quark measurements from single top and top-quark pair production are useful for probing flavour changing neutral currents in the SM [53]. Typically, the top quark will decay to a W boson and a b quark, and does so before hadronisation, giving a clean measurement of the weak decay, if the reconstruction of the W and b

are precise enough. In the SM, top decays to the s and d quarks are suppressed and to the c and u quarks forbidden at tree level, so precise characterisation of these rare decays can prove or disprove various theories that boost flavour changing neutral current decays. Top quark decays lead to high multiplicity events due to the heavy top, b, and W hadronic decays. This makes high granularity calorimetry and effective triggering in a high multiplicity environment a must, especially with the hadronic background from the pileup. Events with large hadronic activity such as the top decay will be particularly difficult to reconstruct in the high pileup environment as the event signal is so similar to the background. These events will also test the readout system by providing far more activity in the detector to reconstruct at the trigger level. It is for this reason that top, anti-top production events are used as benchmarks for triggering algorithm development as they can be seen as the "worst-case" scenario for the trigger reconstruction with no "easy" trigger pathways such as high-p_{T} muons to trigger on.

Higgs Physics

The number of Higgs bosons produced in CMS in the first three runs of the LHC will be $O(10$ million$)$; the HL-LHC will have up to 10 times this number, leading to an expansive Higgs physics programme for the CMS and ATLAS detectors. Higgs boson pair production will be a key measurement in the HL-LHC as it allows determination of the Higgs self-coupling (shown in figure 2.3) which can characterise the Higgs potential as well as offering a portal to new physics models with multiple Higgs bosons, as is predicted by many SUSY models. Figure 2.4 shows the likelihood of obtaining certain values of the SM Higgs self coupling in various decay channels. The figure assumes that the SM processes are the only ones contributing to the di-Higgs production meaning κ_λ is expected to be one. In any specific channel, given its rarity and difficulty to trigger on, CMS expects to find a value for κ_λ inside the corresponding line with a certain likelihood. After the HL-LHC CMS predicts to have a 95% confidence that if κ_λ is one then the analysis will see a range of κ_λ between -0.18 and 3.6 when combining all channels. When using all channels it is expected that CMS will have a significance of 2.6σ in observing di-Higgs production, which rises to 4.0σ when combined with ATLAS [54]. Also shown in the figure, the $HH \rightarrow \tau\tau bb$ and $HH \rightarrow \gamma\gamma bb$ decay modes are significant contributions to the Higgs self coupling but need highly sensitive single lepton triggers for the τ decay and effective γ isolation, which can be achieved with charged particle tracking in the L1 trigger.



(a) (b)

Figure 2.3: The Feynman diagrams for the leading di-Higgs production mechanisms as the LHC. On the left is the leading production mechanism of the Higgs boson—gluon fusion. With the produced Higgs decaying to two further Higgs bosons. This gives a direct measurement of the Higgs self coupling κ_λ but is rare because the intermediate Higgs must be far from its on-shell mass to produce two subsequent Higgs. The diagram on the right shows the interfering diagram of di-Higgs production via a fermion loop, again rare due to the low production cross-sections for the Higgs boson at each of the fermion-fermion vertices. These diagrams interfere destructively making the observation of the Higgs self coupling very rare.

Another key Higgs boson measurement is in the $H \rightarrow \tau\tau$ channel because of its sensitivity to multiple Higgs models; the high mass of the τ means a stronger coupling to the Higgs and a sensitivity to additional couplings.

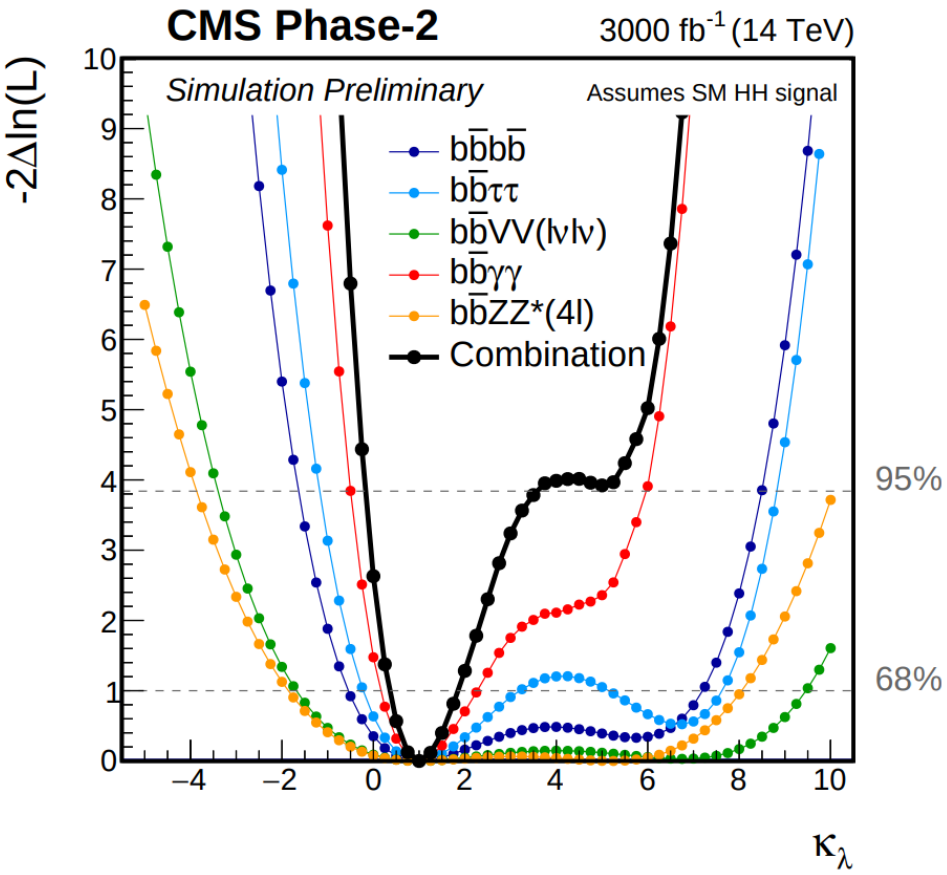


Figure 2.4: Expected likelihood as a function of the ratio of observed Higgs self coupling to SM expected Higgs self coupling, $\kappa_\lambda = \lambda_{\mathrm{HHH}}/\lambda^{\mathrm{SM}}_{\mathrm{HHH}}$. Each decay channel is shown separately with the combined likelihood in black. This scan assumes an SM HH signal and shows that CMS, by the end of the HL-LHC, will have a limit of the coupling of 0.35–1.9 at the 68% confidence limit and -0.18–3.6 at the 95% confidence limit. If a coupling outside this 95% range is observed, there will be evidence of new physics beyond the SM [54].

However, this decay relies on good performance across the CMS detector because of the variety of the decay modes of the τ and precise triggering to distinguish pileup jets from hadronic τ jets with their differing structure. With additional tracking information in the L1 trigger, the acceptance of τ final states could increase up to 5.5 times versus the Phase-1 trigger.

Precision Higgs boson measurements are a key part of the CMS Phase-2 physics programme. Figure 2.5 shows the decrease in the uncertainty for Higgs couplings for the end of run 3 on the left and the end of the HL-LHC on the right. Assuming no improvement to systematic uncertainty quantification (in green), there will be a significant decrease in the uncertainty down to 1% across the Higgs boson couplings compared to 10% at the end of run 3, but only if the triggering efficiency of CMS remains the same or better in the higher pileup.

Beyond the Standard Model

Another key measurement in the HL-LHC programme is the lepton flavour violating process of $\tau \rightarrow \mu\mu\mu$ (shown in figure 2.6) which, while theoretically possible in the SM with neutrino oscillations, is very rare with a branching fraction $\approx 10^{-55}$ [56]. Extensions to the SM increase the branching fraction of this decay, making it a key probe of beyond SM physics. This decay is characterised by high pseudorapidity, low p_{T} muons which means good forward muon reconstruction is necessary, as is effective triggering on these signals.

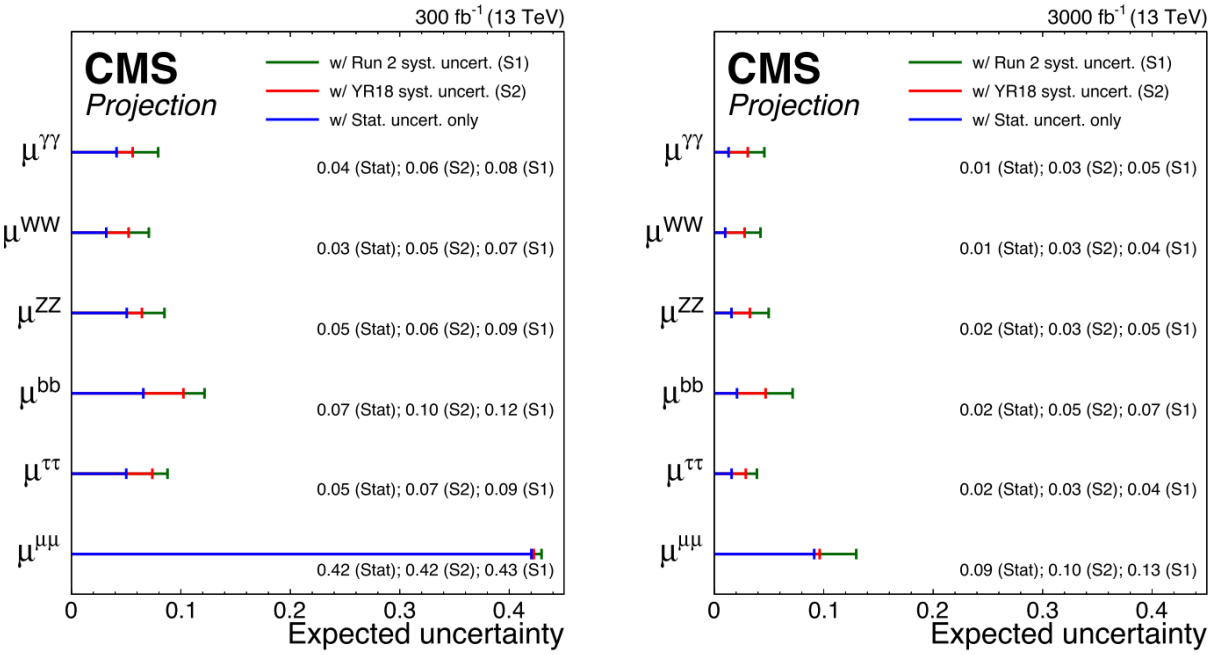


Figure 2.5: Predicted Higgs boson coupling uncertainties for up to run 3 on the left and up to the end of the HL-LHC on the right. In green is shown the uncertainties assuming run 2 systematic calculations. In red are the predictions for the systematic uncertainties at the time of the HL-LHC which are lower because of predicted improvements in analysis techniques. In blue are the predictions if the systematic uncertainties were zero, so is driven only by the amount of data collected. One of the largest potential improvements between the end of run 3 and the HL-LHC is the coupling to muons which is simply a result of the rarity of this decay [55].

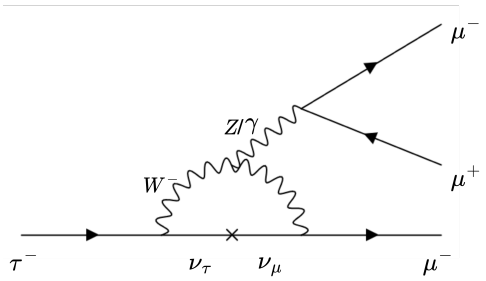


Figure 2.6: Feynman diagram of the SM process of $\tau \rightarrow \mu\mu\mu$ via the oscillation of a τ neutrino to a muon neutrino. Due to the small mass differences between the neutrinos this oscillation typically occurs at km distances making the SM path for this decay incredibly rare with a branching fraction of 10^{-55}. Any additional lepton flavour violating processes beyond the standard model will increase the branching fraction making this process a key probe for new physics.

A key aim of the original LHC and associated experiments was in the observation of Supersymmetry. While this has not happened in the first three runs of the LHC, there is potential in the HL-LHC to continue the search. Due to the range of SUSY physics under consideration, the entire CMS detector needs to at least maintain its physics performance with respect to Phase-1 but in the higher pileup environment; necessitating high radiation tolerance from the innermost detectors and highly sensitive triggers that can veto the additional soft pileup.

2.1.2 Tracker Upgrade

A key part of all the physics goals of the HL-LHC is precise tracking in a high radiation environment. This necessitates a complete replacement of the CMS tracker, both inner tracker (IT) and outer tracker (OT), to more radiation tolerant materials and to have a lower overall material budget. Also, increased angular coverage in both the inner and outer tracker will boost sensitivity to forward physics. Another requirement to maintain, and potentially improve, the physics performance of the CMS L1 trigger is the use of reconstructed tracks in the L1 trigger, a new, and integral part of the Phase-2 L1 trigger. The layout of one quarter of the upgraded tracker is shown in figure 2.7. Compared to the layout of the Phase-1 tracker in figure 1.7, one of the largest upgrades is to the pixel detector which will have far more endcap layers which increase the pseudorapidity range up to 4 which, when combined with the high granularity endcap calorimeters, results in far better particle flow reconstruction at high η, shown in figure 2.12. The outer tracker has fewer layers but with higher granularity which maintains the overall track resolution with respect to the Phase-1 tracker and reduces the overall readout bandwidth, essential for the use of the tracker in the Level-1 trigger. The new tracker will also have a reduced material budget with a particle seeing approximately half the amount of material at the thickest point in the new tracker, greatly reducing the chance of electron conversion and multiple scattering.

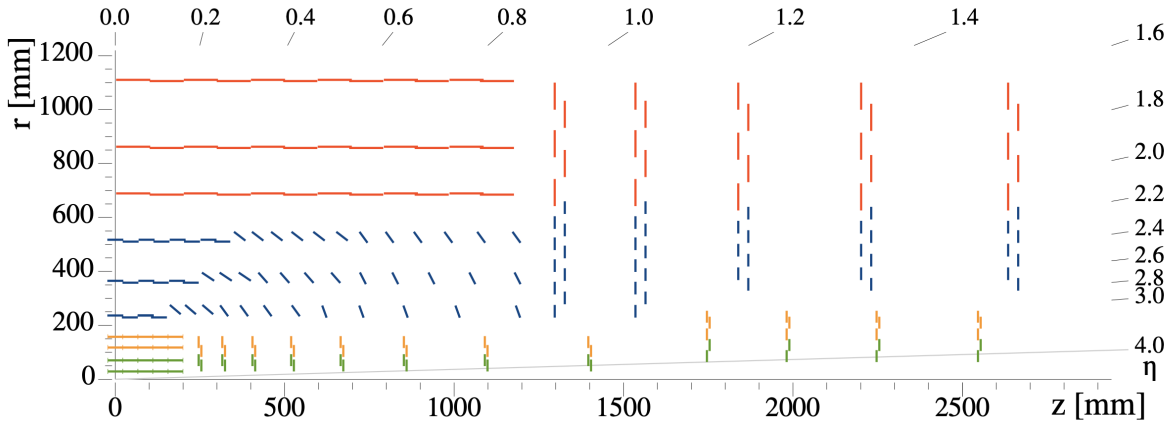


Figure 2.7: One quarter of the Phase-2 CMS tracker in the r-z plane [57]. The inner tracker is shown in green and yellow, with the different modules having different pixel sizes; smaller pixels closer to the interaction point. The outer tracker is shown in blue and red. In blue are the pixel strip modules which use both pixel and strip detectors. In red are the two strip module detectors which only use strip detectors.

Inner Tracker

The IT is to be replaced with a higher granularity pixel detector with pixel sizes of 25 $\times$ 100 μm^2; a pixel area six times smaller and width twice as thin as compared to the previous tracker. This high granularity means the impact parameter d_0 (displacement in the x-y) resolution of the tracker will be $\sim$ 70 μm, which is 3 times better than the Phase-1 tracker (the full d_0 resolution as a function of η is shown in figure 2.10b). The η coverage will be increased up to $\approx$ 4, compared to a maximum $|\eta| = 2.5$ in the Phase-1 tracker. The yellow and green modules in figure 2.7 show the IT, with the yellow indicating tracker modules with four pixel readout chips and the green those with two, which have been developed with larger buffers for the increased L1 latency. Because of the high data-rate from the IT modules, the IT will not contribute to L1 decisions but can be read out at a rate of 750 kHz. The IT DAQ will also be replaced with new data, trigger and control (DTC) boards with larger buffers and FPGAs for processing.

Outer Tracker

The main feature of the OT in Phase-2 is the ability to send information to the L1 trigger for track reconstruction. If the current Phase-1 tracker design were used, the data rate from the OT would be far too high for the L1 trigger to process, so a new type of module for the OT is used. These p_T modules consist of two closely spaced layers of silicon with a tunable window allowing for an on-module cut for charged particles with a $p_T > 2$ GeV. This concept is shown in figure 2.8 with the window shown in green. As a charged particle traverses the CMS detector in the r-ϕ plane it curves in the magnetic field, with the radius of curvature inversely proportional to the particle's p_T. A high p_T particle passing through the two silicon layers has a straight path that falls within the window and is registered as a "stub" for L1 reconstruction; the leftmost track in figure 2.8. Low p_T, curved particle trajectories register outside this window and are not used for L1 reconstruction; the rightmost track in figure 2.8. The hits in the detector layers are stored regardless of their p_T so that if an L1 accept is received, the full p_T spectrum of tracks can be readout and reconstructed in the HLT and offline.

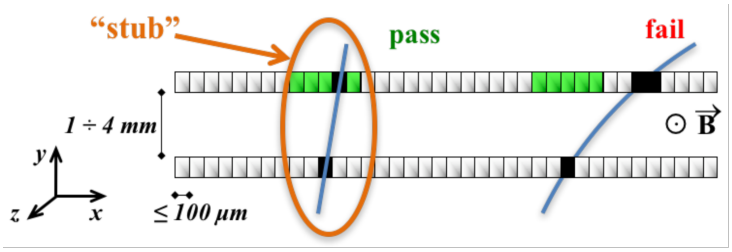
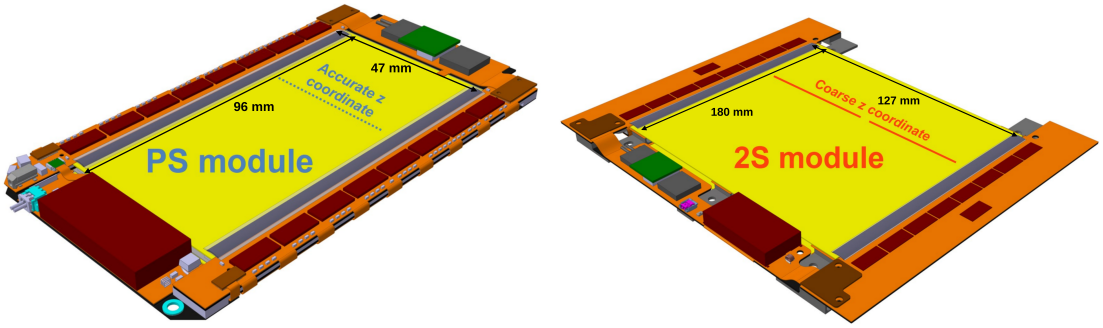


Figure 2.8: The outer tracker p_T module concept, where a high p_T track is within the stub window on the left forming a stub for online processing at L1 and a low p_T track that is outside the window and is not sent to the trigger on the right. These windows can be tuned to track p_T, depending on the requirements, and constraints, of the L1 trigger [58] [59].

A standard approach to two layer silicon strip sensors is to have one strip in one plane and the other orthogonal, giving a stereo strip measurement and precise reconstruction in both the z and ϕ planes. However, in order for the p_T concept to work both strips need to be parallel; an issue that motivates the use of two different types of module for the OT. The innermost layers of the OT consist of pixel-strip (PS) modules which have one strip (PS-s) and one macro-pixel (PS-p) sensor; these are shown in blue in figure 2.7 and their schematic in figure 2.9a. The strip sensor have a strip pitch of 100 μm giving a 29 μm resolution for hits in ϕ. The pixel sensors, with 1.47 mm long, 100 μm wide pixels, give a constraint on the resolution of the z_0 of the tracks of 1 mm, which is enough for primary vertex reconstruction (discussed in chapter 5). The outermost layers consist of two strip (2S) modules with a 90 μm pitch and 26 μm resolution in ϕ. With no stereo strip offset there is poor resolution in z (and r in the endcap) but still acceptable when combined with PS stubs. The lower cost and lower bandwidth of the 2S modules and the fact the occupancy in these modules is lower motivates their use in the outermost layers of the OT.

Another key aspect of the p_T modules is interconnectivity between the two layers, which is performed at both ends of each module. Due to cost and complexity there is no communication between the halves of each module, resulting in an inefficiency in the modules for high incident angle tracks that pass across both halves of the modules. To improve efficiency in the innermost layers at the edges of the barrel where this effect is most significant, the PS modules are tilted, as shown in figure 2.7 between z = 200 mm, and 1200 mm. Because of the varying curvature of tracks at different radii in the OT, every p_T module has a different stub window tune depending on its position in the tracker.



(a) (b)

Figure 2.9: On the left, the schematic of the outer tracker PS module. The strips in the first layer run parallel the shorter axis with a split in the middle. Each side of the silicon sensor has 960 strips are read out by the readout chips along the edges of the sensor. The macro-pixels lie underneath the strips with 32 pixel rows across the shorter axis. On the right, the schematic of the outer tracker 2S module consisting of two layers of silicon with strips running parallel the short axis split in the middle. Each side of the sensor has 1016 strips which are read out at the edges of the module.

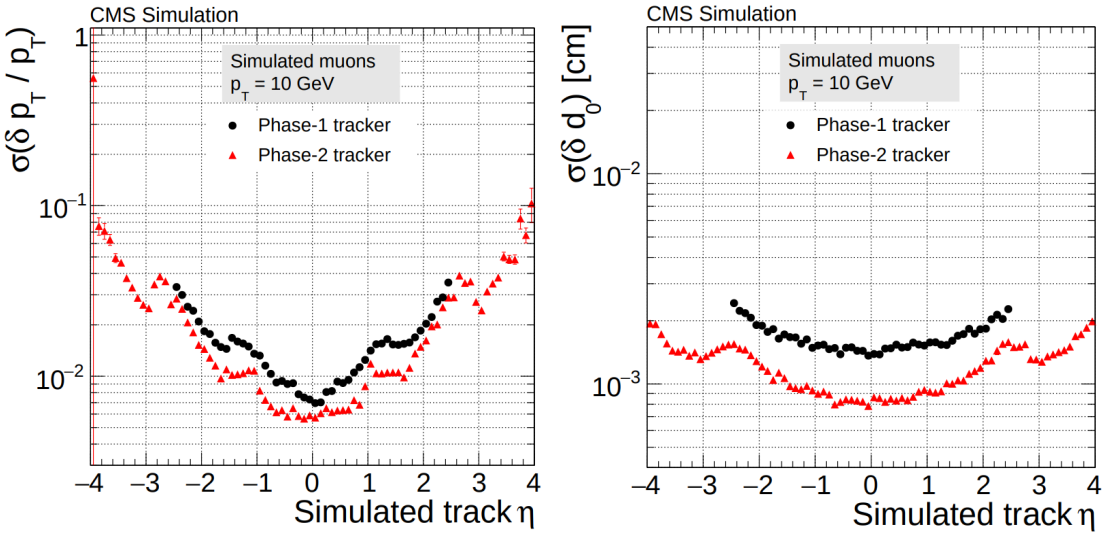
Once registered as stubs in the OT, the modules are read out through a DTC board, with 24 DTCs per nonant of the OT, and 72 modules per DTC board. These stubs are then routed to the track finder processor (TFP) boards which perform the track reconstruction, detailed in chapter 4.

Together the IT and OT will have a reduced material budget with respect to the Phase-1 tracker and the improvements to the granularity of the modules mean the track finding performance improves over the Phase-1 system. Figures 2.10a and 2.10b demonstrate this with an improvement seen in the resolution of the track parameters. The increased η coverage also means tracks of a higher η can be reconstructed which is needed to fully exploit the higher granularity endcap calorimeters, as described below.

2.1.3 HGCAL

Because of radiation damage the current endcap calorimetry is not suitable for the HL-LHC, so it is to be replaced with the HGCAL [60]. As the total radiation dose after the full running period of the HL-LHC will be very large, silicon sensors were chosen as the active material because of their reliability after higher radiation doses and increased granularity (cell size of 1 cm^2) compared to the currently used scintillating crystals. This granularity leads to better separation of objects in the calorimeter for example, in a VBF event where the majority of the energy in each jet is carried by two pions and a photon all within 1 cm of one another. VBF jets are a key tag for rare Higgs decays where the decay products of the Higgs cannot be triggered on, for example Higgs to low mass quarks. The HGCAL will see both the ECAL and HCAL components of the calorimetry merged into one subdetector with 28 EM layers of silicon sensors sandwiched between WCu baseplates and stainless steel absorbers and 22 hadronic layers using stainless steel absorbers and silicon sensors. Finally, plastic scintillators will be used for the outer HCAL layers where the occupancy is lower. The organisation of these layers are shown in figure 2.11. The ECAL layers have a predicted energy resolution of under 2% for photons energy > 200 GeV with the effect of high pileup being negligible. The HCAL layers are predicted to have an energy resolution of 10% for pion energy > 100 GeV with the high pileup worsening the resolution below this energy threshold.

The use of silicon sensors increases the granularity and time resolution of the detector, with half the EM cells



(a) (b)

Figure 2.10: Figure (a) on the left shows the p_{T} resolution for isolated p_{T} = 10 GeV muons with the Phase-1 and Phase-2 trackers. Not only does the resolution improve across the barrel region because of the reduced material budget and smaller pixel sizes but the Phase-2 tracker extended η coverage means it can reconstruct tracks with an η up to 4. Figure (b) on the right shows the transverse impact parameter d_0 resolution as a function of η where a similar improvement is seen for the Phase-2 tracker making displaced vertex tagging more efficient [58].

and all the hadronic cells being read out to the L1 trigger. This level of granularity for the L1 trigger allows complex algorithms such as particle flow to be considered. The limiting factor in the performance of the particle flow algorithm is the ability to match tracks to calorimeter objects meaning high granularity is essential to non-ambiguously match tracks to energy deposits, as is effective track finding. Part of this matching is the finding of a primary vertex in the Level-1 trigger, again only possible with tracks and vital for using particle flow with the HGCAL. The use of the HGCAL in the Level-1 trigger was a large factor in the decision to focus on transverse granularity versus pure energy reconstruction performance. Another feature of the HGCAL that can be exploited at Level-1 is the ability to point back to vertices. High granularity in both r and η of 1 cm and lateral shower sampling means calorimetry objects can be traced back to their production vertex meaning displaced signatures can be identified with the calorimeter; a feature that has never been seen in the Level-1 trigger. While no trigger algorithms are currently using this feature it opens up many possibilities for future algorithms with the highly flexible Level-1 trigger architecture and could be used to improve calorimeter object to primary vertex association.

The choice of high density absorbers in the calorimeter results in laterally compact showers which gives good shower separation, another key aspect for PF reconstruction, especially in the high pileup environment with many overlapping energy deposits. With the use of silicon sensors with fast readout rates, high timing precision is possible as showers progress through the HGCAL layers. This will give 25 ps resolution for shower development which, compared with a single bunch crossing that takes $O(100)$ ps, can give pileup identification for individual HGCAL showers. This replacement of the endcap calorimetry in conjunction with the upgraded tracker allows far better reconstruction of forward objects, especially important in the VBS and VBF event topologies that are significant parts of the Phase-2 CMS physics programme, a feature that is clearly shown in figure 2.12.

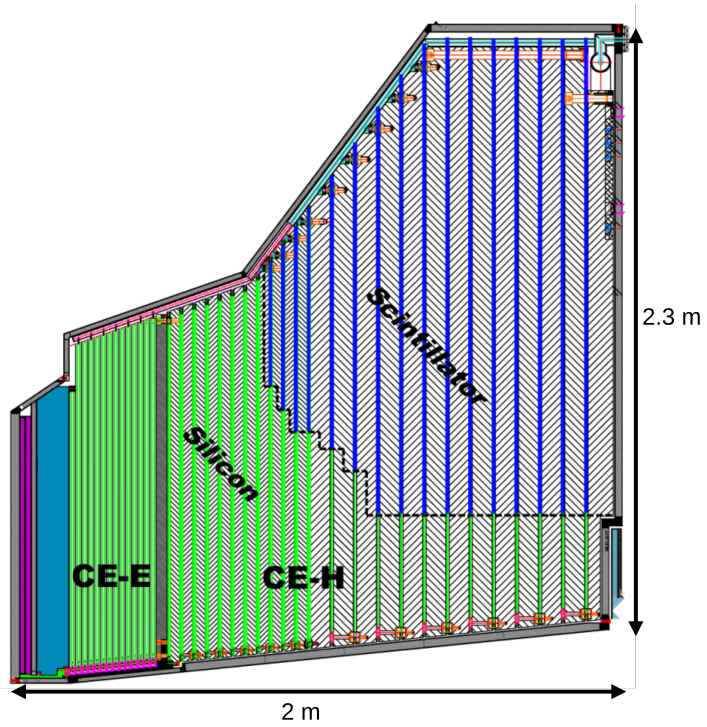


Figure 2.11: A slice in r-z of one half of one endcap of the high granularity calorimeter split into the two compartments, the electromagnetic compartment (CE-E) and hadronic compartment (CE-H). The silicon parts of the CE-E and CE-H are shown in green with the scintillator component of the CE-H in blue [61]. The far higher density of sensors in the ECAL is clear and is necessary to completely capture the electromagnetic energy deposits. At higher η, where the particle flux is lower, the hadronic part of the calorimeter is far less dense and can use cheaper scintillator activate materials.

2.1.4 Other Upgrades

Because of the relatively low radiation doses received by the barrel calorimeters and effective methods for improving their performance in high radiation environments, the barrel calorimeters will not be replaced. Instead, the backend boards for both the ECAL and HCAL will be replaced allowing for the full detector granularity to be read out to the L1 trigger. In order to reduce detector noise, the amplifier circuitry is being redesigned and the ECAL operating temperature is also being lowered [62] [63], giving the barrel ECAL 30 ps timing resolution and improved anomalous spike rejection.

The muon systems will also have a replacement of backend and frontend electronics to allow for increased buffer sizes for the increased L1 latency as well as faster processing for the higher pileup environment [64]. There will also be extended coverage up to $\eta \approx 3$ to give better reconstruction of highly boosted objects in line with the increased tracker coverage, particularly important for Higgs boson to multi-lepton final states. The extended coverage will be provided by new RPC detectors and a set of gas electron multipliers (GEM) [65] with a 75 μm spatial resolution which add a redundant system to the high η regions which were only previously covered by CSCs.

The HLT and DAQ systems will also be upgraded, with the HLT rate increased from 1 kHz to 7.5 kHz in line with the increased L1 rate. The physics sensitivity of the HLT will be maintained and extended in the higher pileup HL-LHC environment and maintain a manageable computing resource expenditure. A move to heterogeneous architectures taking advantage of coprocessors, for example GPUs and FPGAs, is a key part of

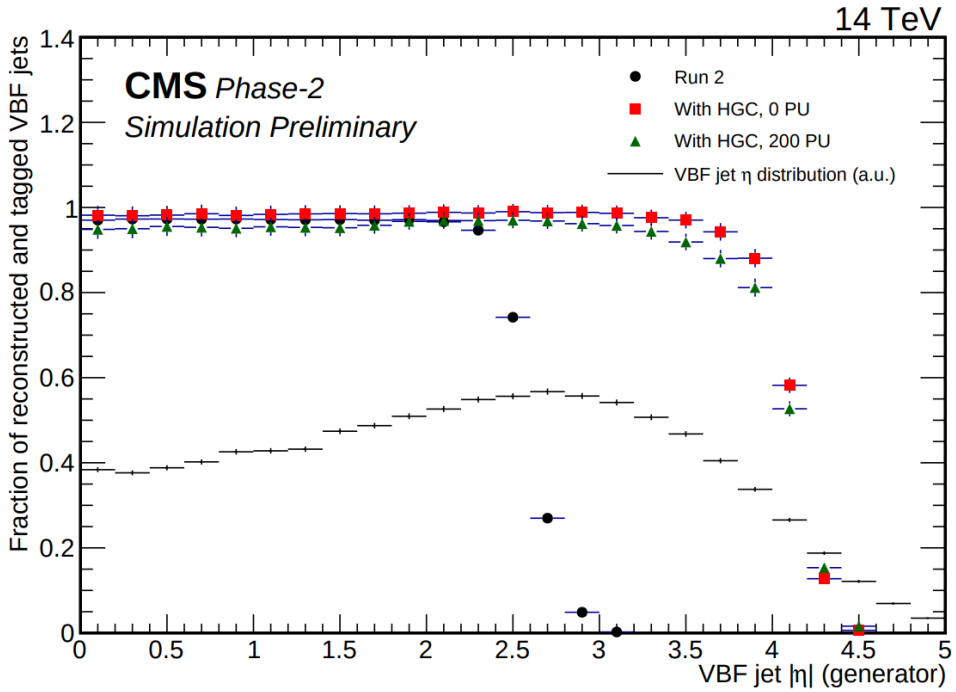


Figure 2.12: Phase-2 CMS vector boson fusion jet reconstruction efficiency versus $|\eta|$. In the barrel region, $|\eta|$ <1.5, there is no difference between run 2 CMS with 25 PU (black dots) and Phase-2 CMS (red squares) as the barrel calorimetry is unchanged, there is an impact due to pileup with the 200 PU scenario (green triangles) reducing the jet finding efficiency. There is a clear improvement from the HGCAL at higher $|\eta|$ where the majority of the VBF jets are generated; the HGCAL is able to efficiently reconstruct these jets up to $|\eta| < 3.5$ and still offers some level of reconstruction up to $|\eta| < 4.5$, whereas the run 2 endcap drops off at $|\eta| = 2.5$. This allows CMS to capture more of the VBF jet distribution, shown by the black lines in the figure [60].

future HLT developments [66].

2.1.5 Trigger Upgrade

If the current Phase-1 trigger system that only uses calorimetry and muon systems were to remain in place in the high pileup environment of the HL-LHC, the trigger rate would have to rise from 100 kHz to 4 MHz to maintain the same physics selectivity. This rate is too high for the HLT system to process and would result in an unmanageable amount of data needing to be stored. This issue necessitates a complete replacement of the trigger system, which not only aims to maintain the current L1 trigger efficiency, but also enhance it with the upgraded detector subsystems. A flexible, modular architecture with access to a more complete range of subdetectors allows the L1 trigger to be adaptable to new physics searches across the lifetime of the HL-LHC. The L1 trigger will also have an increased rate to 750 kHz (from 100 kHz) to maintain efficiency and an increased buffer latency to 12.5 μs so that higher granularity detector information and track finding can be used in the L1 trigger. The L1 trigger will be implemented entirely on FPGAs; specifically the Xilinx Virtex Ultrascale+ [48] range of FPGAs with the VU13P being the FPGA of choice across much of the system. The use of FPGAs makes the trigger reprogrammable if algorithm improvements are made, but it means that all algorithms need to be written for FPGA architectures, which is not a trivial process. Careful consideration of the overall latency and resource usage of algorithms needs to be considered throughout the design process and with the high data rates and specific requirements of the trigger system, custom processing boards have also been designed. The trigger will be implemented across a variety of different processing boards, with the Apollo [67], APx, Serenity [68], and X20 boards all in development, as well as custom Samtec firefly optics for data transfer [69] [70].

The trigger design, shown in figure 2.13, is divided into individual subdetector pathways which then combine in the correlator trigger and global trigger (GT). The calorimeter trigger, in red, uses trigger primitives (TPs) from the barrel calorimeter (BC), hadron forward (HF) and HGCAL. The BC sends crystal information to the barrel calorimeter trigger (BCT), which creates clusters of energy deposits in the calorimeter and the global calorimeter trigger (GCT) uses information from the calorimetry subsystems to create trigger objects: electrons, photons, jets, hadronically decaying taus, and energy sums [71]. These are used in the correlator trigger for PF. The HGCAL, in accordance with its differing design, performs its own cluster reconstruction which is sent directly to the correlator layers as well as the GCT. Calorimeter only jets are also reconstructed in the GCT as an alternative algorithm to the full PF reconstruction and sent to the GT.

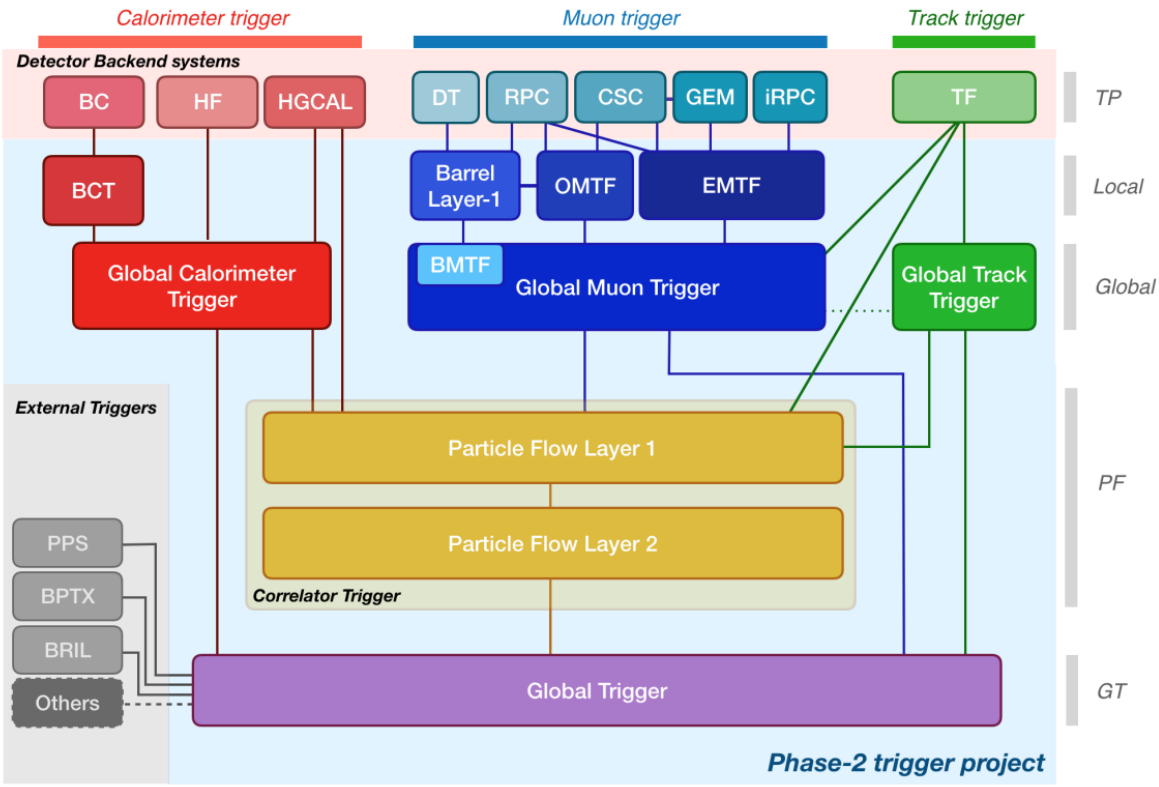


Figure 2.13: Schematic of the Phase-2 trigger system showing the calorimeter pathway in red, the muon pathway in blue and the tracker pathway in green. These are combined in the correlator trigger in yellow and a final decision is made by the global trigger in purple. The major difference between this design and the original design in figure 1.8 is the track trigger pathway and the new correlator trigger that runs particle flow. The trigger primitives (TPs) are shown at the top and come from each of the subdetectors, for a more detailed explanation of each of the trigger primitives and their subsequent processing in the local trigger stages, see [57].

The muon systems, shown in blue in figure 2.13, use TPs from the drift tubes (DT), resistive plate chambers (RPC), cathode strip chambers (CSC), and gas electron multipliers (GEM). These different detector technologies span different areas in pseudorapidity and send their TPs to the barrel, overlap, and endcap muon track finders: BMTF, OMTF, EMTF. The track finders create muon candidates which are then collected in the global-muon trigger where track matching to tracker tracks is performed. The track matched muons and standalone muon candidates are sent to the correlator layers as well as the GT for muon only triggering.

The tracker pathway (in green) creates tracks from the OT stubs with $p_T > 2$ GeV in the track finder (TF) (see chapter 4) which are sent (see chapter 6) to the global track trigger (GTT) (see chapter 5), where the primary vertex is found as well as tracker only jets and energy sums. With the strict latency and lack of reconstruction using the IT pixel detectors it is not possible to accurately reconstruct all vertices in an event, as is found in the HLT and offline reconstruction. Instead, the hardest scatter (or primary vertex) in the event is located, defined offline as the vertex with the highest $\sum p_T^2$. This is a useful object to find in an event as it allows downstream algorithms to associate other objects to the highest energy interaction in an event, and veto the hadronic pileup. Not only does this improve energy sums by removing background pileup and allow multi-object isolation, but it reduces the total number of objects being processed downstream, making more complex algorithms possible within the overall trigger latency.

Trigger objects are sent to the correlator trigger (in yellow) which combines information from the tracker, calorimeters, and muon systems. The correlator is further split into two subsections. The first, correlator layer 1, performs PF where objects in different subdetectors are linked [72]. For this algorithm to be possible, tracker tracks are needed as is high granularity calorimetry; it is the use of PF at L1 that motivates tracker track reconstruction and high granularity calorimeter information being made available at L1. In order for the algorithm to meet the latency requirements of the trigger it is regionalised in η and ϕ so that different regions can be processed in parallel; a significant advantage of a local reconstruction algorithm.

The PF algorithm proceeds as follows:

- Promote any calorimeter objects outside the tracker $|\eta|$ acceptance to a neutral hadron or photon (depending on their energy deposition pattern).
- Match muons to tracker tracks to create muon objects and remove these tracks from resulting processing.
- Match tracker tracks to calorimeter objects. In the barrel this follows a three-step process; firstly, tracks are matched to EM deposits to create photon and electron candidates. Secondly, remaining EM clusters are linked to hadronic clusters. Finally, hadronic clusters are linked to tracks to create charged and neutral hadrons.

The next step in the correlator trigger is to reduce the background pileup contribution, in an algorithm called pileup per particle identification (PUPPI), shown in figure 2.14. This algorithm uses the GTT vertex to match previously found objects. Any objects with charged particle tracks not associated to the vertex are removed while any neutral particles can be removed or down weighted depending on their distance to the vertex. An advantage of this algorithm is that it is inherently local; PF candidates within an η–ϕ region can be processed with the vertex and a weighting returned without cross communication with other regions, making this algorithm possible within the L1 trigger. Accurate vertex reconstruction and high track to vertex association efficiency is a key part of this algorithm that directly impacts the jet reconstruction and missing energy calculations in the trigger. The use of PUPPI in the Level-1 trigger is one of the major reasons the vertex needs to be reconstructed to a high degree of accuracy.

The second correlator trigger layer uses the PUPPI candidates from the previous layer to compute physics objects in a global event reconstruction with multiple algorithms running in parallel. The first being jet reconstruction where high energy collimated streams of particles are identified and linked together. There are currently multiple approaches to jet finding: a seeded cone approach and histogramming approach, each with their own advantages but both with high efficiency for jets with $p_T > 100$ GeV. Global energy sums including total missing transverse energy E_T^{miss}, useful for neutrino identification or invisible decays, and total hadronic

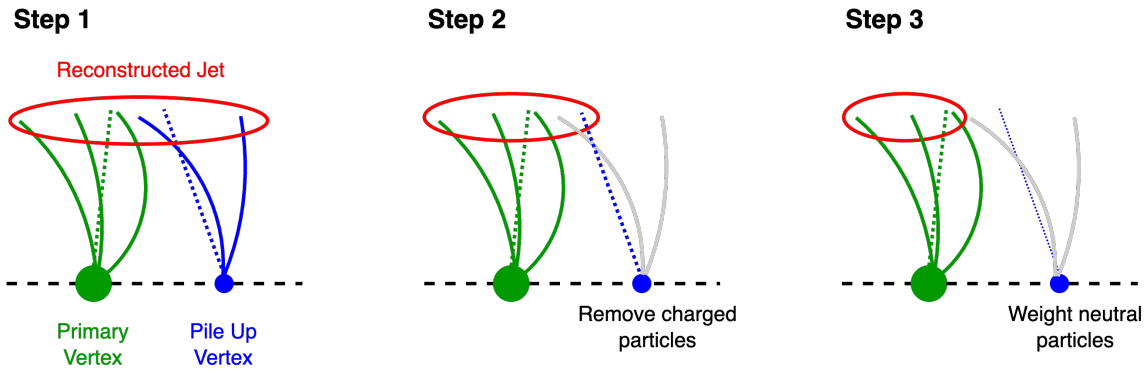


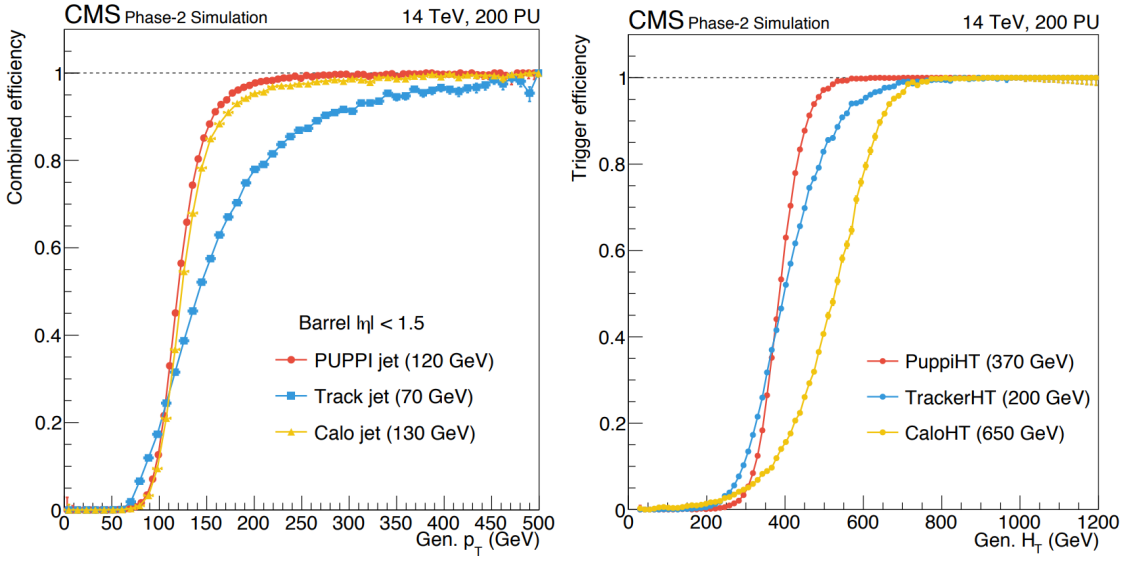
Figure 2.14: The pileup per particle identification algorithm. The reconstructed jet shown in red is for illustrative purposes as it is reconstructed after the PUPPI algorithm has been completed but is included here to show the impact of the PUPPI algorithm on the jet reconstruction. Step 1 is to take all PF objects and match them to their vertices with one primary vertex being provided by the GTT. Step 2 is to remove any charged PF candidates not associated to the primary vertex. Step 3 is to down-weight neutral PF candidates based on their distance in z away from the primary vertex.

transverse energy H_{T} are also calculated from the full range of PUPPI candidates and reconstructed jets respectively. Figure 2.17a shows that in samples with significant $E_{\mathrm{T}}^{\mathrm{miss}}$ the use of PF can improve the trigger threshold by 200 GeV which results in a factor 10 or more improvement in the number of rare decays such as $H \rightarrow$ inv. or SUSY triggered on in the L1 trigger.

The performance impact of using the PUPPI algorithm for jet reconstruction can be seen in figures 2.15a and 2.15b. Figure 2.15a demonstrates that using full detector inputs for jet finding outperforms subdetector only algorithms, with tracker only jet finding being significantly worse than calorimetry only. This is due to neutral hadronic energy and any tracks under $p_{\mathrm{T}} = 2$ GeV not being used by the tracker only algorithm. There are also issues with jet merging, where the jets have not spread radially outwards enough in the tracker to be differentiated. PUPPI slightly outperforms the calorimetry only algorithm; the reason for this modest improvement is that this is jet finding efficiency so accurate reconstruction of jet energy, where using PF is beneficial, does not matter. Figure 2.15b shows a far better reconstruction of overall hadronic energy in the event from the PUPPI algorithm. Where the calorimetry only algorithm suffers from poor energy reconstruction and pileup background, the PUPPI algorithm not only has better energy reconstruction because of the use of PF but also PU rejection making its efficiency far higher. Tracker only H_{T} performs reasonably well with an implicit cut of PU due to the track reconstruction p_{T} cut, but is unable to isolate individual jets with the same power as the PUPPI algorithm.

Also performed in the correlator layer-2 is e/γ and muon reconstruction which are calculated with and without tracker isolation. The impact of the track isolation can be seen in figure 2.16. When considering various rare decays that lead to muons, such as $HH \rightarrow bb\tau\tau_h$, where one τ decays leptonically, the use of electron isolation can improve electron p_{T} sensitivity by 20 GeV. Similarly, muon p_{T} sensitivity improves by 10 GeV, which is less pronounced because of the relatively high muon reconstruction efficiency in the standalone muon chambers.

Finally, dedicated τ reconstruction is performed. There are multiple algorithms for this, focusing on hadronic decays and leptonic decays of the τ. The main (hadronic) τ_h reconstruction takes advantage of a neural network-based approach that uses global event level quantities and information from the input candidates to reconstruct the τ_h. Again, the use of L1 track reconstruction and PF is essential for this τ reconstruction. Figure



(a) (b)

Figure 2.15: Figure (a) on the left shows jet finding efficiency as a function of true jet p_{T} for Phase-2 L1 trigger algorithms. Figure (b) on the left shows H_{T} trigger efficiency as a function of true H_{T} for the same algorithms. Shown in blue is tracker only jet reconstruction, in yellow calorimetry only, and in red the full PUPPI reconstruction after PF and PUPPI weighting. The highest performing algorithm in both metrics is PUPPI as it is able to isolate jets effectively for the jet isolation in (a) but has the advantage of removing background pileup for the overall H_{T} calculation in (b) [57].

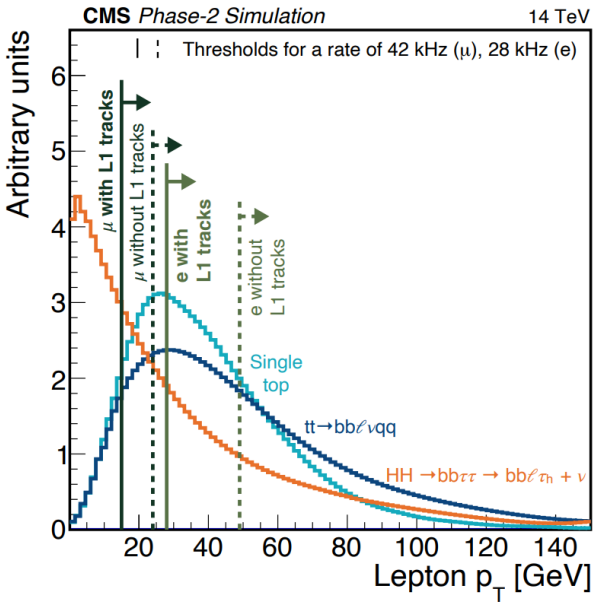
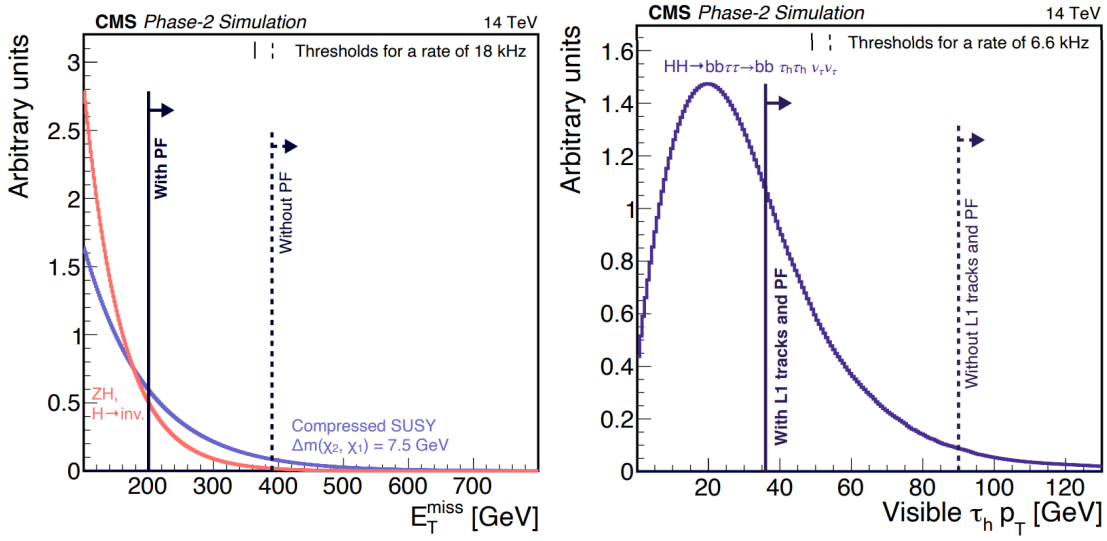


Figure 2.16: Lepton p_{T} sensitivity for various samples with and without L1 track reconstruction. A 20 (10) GeV improvement in electron (muon) p_{T} sensitivity is seen across a range of samples. This improvement means that at a constant trigger rate the lepton p_{T} threshold is lower because background events with similar signatures are not being triggered on [57].

2.17b shows that in an $HH \rightarrow bb\tau_h\tau_h$ decay, the τ_h p_{T} trigger threshold can be reduced by over 50 GeV, leading to a 10 times improvement in triggering on these events versus the background.



(a) (b)

Figure 2.17: Figure (a) on the left shows $E_{\mathrm{T}}^{\mathrm{miss}}$ distributions for two key signal samples, $H \rightarrow$ inv. and a SUSY decay with the thresholds with and without the use of PF shown. There is a 200 GeV improvement in $E_{\mathrm{T}}^{\mathrm{miss}}$ trigger threshold when using PF, demonstrating how large a difference the PF calculation makes to the overall energy reconstruction in CMS. Figure (b) on the right shows hadronic τ p_{T} sensitivity for the $HH \rightarrow bb\tau_h\tau_h$. The use of tracker tracks and PF in this reconstruction decreases the trigger threshold by 50 GeV, meaning far more di-Higgs events can be collected because of better background rejection [57].

The final step of the L1 trigger is the GT, where all the previously reconstructed objects are combined and a decision is made whether to trigger and readout the entire detector based on a carefully constructed menu. To maintain sensitivity to a wide range of physics signals and adapt to changing requirements from downstream physics analyses, the GT must be flexible and reprogrammable. There are $O(1000)$ different algorithms in use in the GT which also computes multi-object correlations, such as $\Delta\eta$ or $\Delta\phi$, making it a highly complex system. The classic GT algorithms consist of simple cut-based triggers where various different trigger conditions are evaluated and if met the trigger is fired. There are different combinations of single object, multi-object and correlational triggers that are combined and, because of the large number of input objects, these come with a large combinatorial cost resulting in complex time-multiplexed architectures on multiple FPGAs.

Another possibility in the GT is the use of machine learning (ML) algorithms. Due to the wealth of information input to the GT at both a global and single object level, highly complex ML algorithms can be designed to take advantage of this information. One example currently proposed for the L1 trigger are neural network autoencoders for new physics detection. These NNs could learn the probability density of SM physics from the inputs and output a score based on how close a single event matches these probability densities. Any anomalous events that depart from SM expectation can be triggered on as they could contain new physics that the cut based approaches are not optimised to trigger on. An advantage of an NN approach to these kinds of complex trigger decisions is that the firmware implementation is relatively straight forward; recent advances in tools such as hls4ml [73] that allow NNs to be transpiled to an HDL create a single, reusable firmware core that can be inserted within the GT firmware. Tuning the algorithm is as simple as retraining the network and rebuilding the firmware, thus removing the need for complex rewriting of firmware—this makes highly flexible algorithms possible. Due to this flexibility, much work is ongoing in exploring NN-based approaches to GT algorithms and

general algorithms across the L1 trigger.

Another parallel part of the standard trigger pathways will be the 40 MHz scouting system [74]. This will see all the trigger primitives from the subsystems being readout at 40 MHz (the LHC event rate) using spare optical links from each of the trigger subsystems. The scouting allows for monitoring of the GT performance without the bias that comes from the fact that any non-triggered data from the GT is lost. This is invaluable in maintaining high performance in the trigger and spotting data inconsistencies as they occur. To facilitate this monitoring, the GT will also send its trigger decision to the scouting system. Another key feature of the scouting system is the ability to test new algorithms in parallel with the trigger system. With information from the GT about how the trigger has fired, new algorithms can be compared to the previous ones and developed without losing physics data. ML-based algorithms will be a key testing area for the L1 scouting system where global event-level ML classifiers or anomaly detection systems can be taken advantage of to find events previously unreachable for conventional triggers.

The L1 trigger follows a similar design pattern to previous triggers where the different levels of processing logic start with low-level trigger primitive creation from the detector subsystems that lead onto local and then global algorithms. A mixture of local time-multiplexed reconstruction (different parts of the trigger processing different events at the same time), local regional reconstruction and global reconstruction is leveraged to ensure optimal use of all the FPGAs in the system and reduce the interconnections between every part of the system. This approach has the advantage of being understandable from a physics perspective making issues easier to spot and efficiencies easier to calculate — essential for physics analysis. The underlying trigger architecture will remain constant but the structure described above leaves large amounts of processing space and capability open for future developments. Systems such as the global trigger are designed to be flexible and the selection of very large resource abundant FPGAs mean there is space to experiment, especially with the machine learning approaches described above. While ML cannot be used for the entire reconstruction chain it can be used to enhance existing reconstruction and improve algorithmic performance. This approach of ML-enhanced algorithms has the advantage of interpretability that is essential for trigger applications and the performance gains that ML can bring. The L1 trigger will be an ML-enhanced system that will only improve as new techniques and models are developed.

The Phase-2 L1 trigger system sees a complete replacement of the previous L1 trigger with HLT and offline techniques being realised at the full 40 MHz collision rate. Information from the fully replaced outer tracker and HGCAL, and higher granularity information from the calorimetry and muon systems allow PF and PUPPI to be run using the primary vertex. This level of reconstruction across the detector allows the L1 trigger to maintain and extend its physics reach compared to the previous trigger, despite the increase in pileup to a possible 200 additional interactions. Extensive use of ML in the trigger also allows for highly complex algorithms and reconstruction to be carried out and careful consideration of the use of this ML in the critical L1 reconstruction chain is required (see chapter 7).

Chapter 3

Introduction to Machine Learning

Machine Learning is a term that can refer to any algorithm that makes predictions on new data based on something learnt from old data. In its simplest form this encompasses a linear fit to a two-dimensional set of points where the resulting line equation can be used to make a prediction about a new point in the space. Modern machine learning has evolved from this as computational power has increased exponentially, with decision trees and deep neural networks becoming some of the most popular techniques. These techniques rely heavily on large datasets being freely available, and it is the ability to collect and process this level of data that has pushed ML forwards in the last 30 years.

High energy physics is one of the most data intensive sciences, with the LHC alone producing 10s of petabytes a month with overall storage of nearly 500 petabytes. Traditional analyses of this data, where custom algorithms are developed manually and deployed to analyse this data have become increasingly difficult and time-consuming. ML offers an alternative, simpler methodology of training a customisable model on this data to make predictions, and can outperform simpler tailored algorithms. ML is currently frequently used in physics analyses for these reasons but careful consideration of exactly what these models are learning and the biases this might involve has to be taken [75]. More recently, it has become possible to deploy ML in online analyses for triggering and data reduction purposes. This is possible because of improved computational power in online systems with large FPGAs or GPUs able to hold complex models, and advances in common tools able to transform ML models to hardware specific languages. In the Phase-2 upgrade of CMS ML will become an integral part of the L1 trigger taking advantage of the highly complex tasks ML can perform, as well as the simplicity of the development process compared to custom algorithm development and implementation.

This short section introduces the core concepts, especially those used in later chapters, and outlines methodologies for achieving sub micro-second inference times on FPGAs, as is required in the L1 trigger upgrade.

3.1 Core Concepts

While there are many types of ML, the two most common classes of learning are supervised and unsupervised learning. Supervised learning has an input data set $x_{1..N}$ with corresponding labels $y_{1..N}$ and some learning paradigm will map x to y. This class of learning covers regression and classification problems where the answer is known at learning time. Unsupervised learning aims to find structure within an input dataset where there is no set answer to this problem. This covers a range of applications including clustering of data, anomaly detection or dimensionality reduction where a highly complex multidimensional dataset can be reduced to a simpler

representation for further processing [76].

3.1.1 Supervised Learning

The first requirement of supervised learning is a dataset of N samples with a target label y for each sample. For each sample there is a corresponding feature vector $\mathbf{x}$. The learning model is aiming to predict y using the vector of features. This y could be continuous, in which case the aim is to perform regression, or categorical, which would be a classification task. This dataset is split (typically 70/30) into a training and testing dataset where an ML strategy can be trained, and its performance evaluated on the unseen testing dataset. The splitting of data into training and testing datasets is very task dependent with no set rules on how large the datasets should be and normally depends on how much data is available. However, there are specific cases where a test dataset is different to the training dataset in order to understand how a model could react to unseen data to tests its robustness.

A simplistic but nevertheless useful way of understanding supervised ML is as a function estimation problem. Ultimately, there is some function $f(\mathbf{x})$ that maps the input data $\mathbf{x}$ to the target y. The ML strategy is to find a model, or selection of models that best estimates this function. For a regression task this model could be as simple as a linear regression with two parameters, or be a multi-billion parameter NN. Careful choice of model, and its complexity, is needed depending on what the input data is and what features are available. For classification tasks there are an equally wide variety of models that could be used; from a logistic regression that finds some boundary in the feature space to split apart data or to multi-layered NNs that find multidimensional decision boundaries. Each model has a set of parameters or weights that need to be determined to best estimate the desired function.

The optimal set of parameters for a model is determined through training on successive batches of training data with some loss function L that describes how far from the correct answer the model's prediction is. Regression tasks use loss functions such as the mean squared error or mean absolute error between the prediction and target. Classification tasks use loss functions such as the binary cross entropy [77] between prediction and target. To train, the parameters of the model are optimised in a way that minimises this loss. When making highly complex models it is possible to almost exactly fit to the training data, and these models are said to be "overfitting" to their training data. These models would not be able to make accurate predictions in the test dataset so model complexity needs to be controlled. This is typically performed using weight regularisation where additional terms are added to the loss function. For example, Lasso regression [78] sums all model parameters; the larger the parameters the higher the loss. Alternatively, ridge regression [79] sums the square of all model parameters; the more parameters there are, the higher the loss. Under-complex models can have the opposite problem of "underfitting" to the training data meaning they are not expressive enough to map the features to the target which can occur if the model itself is not large or complex enough, e.g. a linear regression for a polynomial fit. Complex models can also be underfit if they are not trained on enough data.

The final step in supervised learning is the minimisation of the loss function with respect to the model parameters. The subject of function minimisation is very broad and an entire subcategory of mathematics, so the most common approach to ML loss minimisation will be introduced. Gradient descent is an iterative loss minimisation that starts at a point in the loss function space and finds the derivative of the loss surface with respect to the current model parameters. The next step is to change the model parameters in such a way to follow the maximum negative gradient direction. This process is carried out iteratively until a loss minimum is found. The size of each step is controlled by a learning rate; too large and the gradient descent step might step over the minimum, too small and the model training will never converge. As typical training datasets are very large, and

computing the loss across all samples is computationally complex, a random sample or subsample is taken, and a common process called stochastic gradient descent [80] is used.

There are a number of parameters that control how well a model might train which cannot be optimised in a gradient descent manner due to the non-differentiability of the loss function with respect to these parameters. These external parameters that control the learning and general model architecture are called hyperparameters of the ML problem. These are optimised through multiple trainings of the same model, often in an ad-hoc way, to find some optimal set of parameters for a specific model and dataset.

There are four main ingredients to supervised ML with some typical questions asked when setting up the ML solution:

- Dataset: what are x and y? What features of x are best? Is y categorical?
- Model: what base function maps x to y? Linear regression, logistic regression, decision tree, etc.? In section 3.2 onwards, some common supervised learning models are described.
- Loss: how are the model's predictions evaluated versus the targets? Are there additional regularisation terms to control the model complexity?
- Minimisation: how is the model updated in order to minimise the loss? Are there additional external parameters to optimise?

3.1.2 Unsupervised Learning

Unsupervised learning predominately deals with creating a model that maps its input data back to the input data space but having recovered structure in the data. For example, a clustering algorithm that returns the position of its clusters in the feature space, or anomaly detection which maps $\mathbf{x}$ to $\mathbf{x}$ via some latent feature space which has encoded the overall distributions of $\mathbf{x}$, allowing anomalies to be detected. It is possible to recast the four main ingredients of supervised ML to unsupervised learning.

- The dataset is still some set of feature vectors $\mathbf{x}$, but there is no set of targets to find. Instead, $\mathbf{x}$ is mapped back to $\mathbf{x}$. When testing an unsupervised algorithm it can be descriptive to insert known "labelled" data into testing to evaluate how well the model performs, for example in anomaly detection a known anomalous x can be inserted to see if the model flags it as anomalous.
- The models involved are again very diverse; clustering algorithms can be as simple as a k-nearest neighbours [81] or as complex as autoencoder NNs [82], and are selected based on the overall aim of the algorithm.
- The loss functions involved in training are typically those that describe how far the input is to the output. This can be as simple as the least-squares loss and ultimately looks to answer the question "how well has the model recreated the input data"?
- The optimisation procedure tends to be more complex than the gradient descent methods used in supervised learning. While gradient descent can be used, there are often more complex approaches due to non-differentiable loss functions.

3.2 Decision Trees

A decision tree (DT) is a simple algorithm that has a set of calculations at which one of the input features is evaluated versus a threshold [83] to create a set of decisions (or nodes) that cascade to give a final answer (a leaf node). A simple DT is shown in figure 3.1. These can be trained with the input feature and threshold being parameters of the training process with both regression and classification trees being possible, the difference being the output of the leaf nodes. There are multiple algorithms that can be used to govern the tree splitting, but the underlying concept is that at each node the variable and threshold which reduces the entropy (for classification) or variance (for regression) in the data sample is calculated. For example, in the particle ID task of figure 3.1, if the dataset has a 50-50 split in high and low mass particles a threshold of 100 MeV on particle mass will decrease the entropy of the output subsamples as each will now only contain a subset of the particle categories. This process of finding optimal features and thresholds occurs at each splitting until some stopping criteria is reached, which might be a simple depth stopping criteria (that stops when a certain number of splits is reached) or the overall entropy in each of the final leaf nodes. Due to their low complexity, simple DTs will be biased, or underfit, and deep trees will overfit to the training data and have high variance when evaluated on a test dataset. This issue can be seen in the particle ID task; the very simple tree is not sensitive to electrons versus positrons so is underfitting and another tree or a deeper tree is needed to provide this information. To solve this issue, ensemble methods are typically used with DTs.

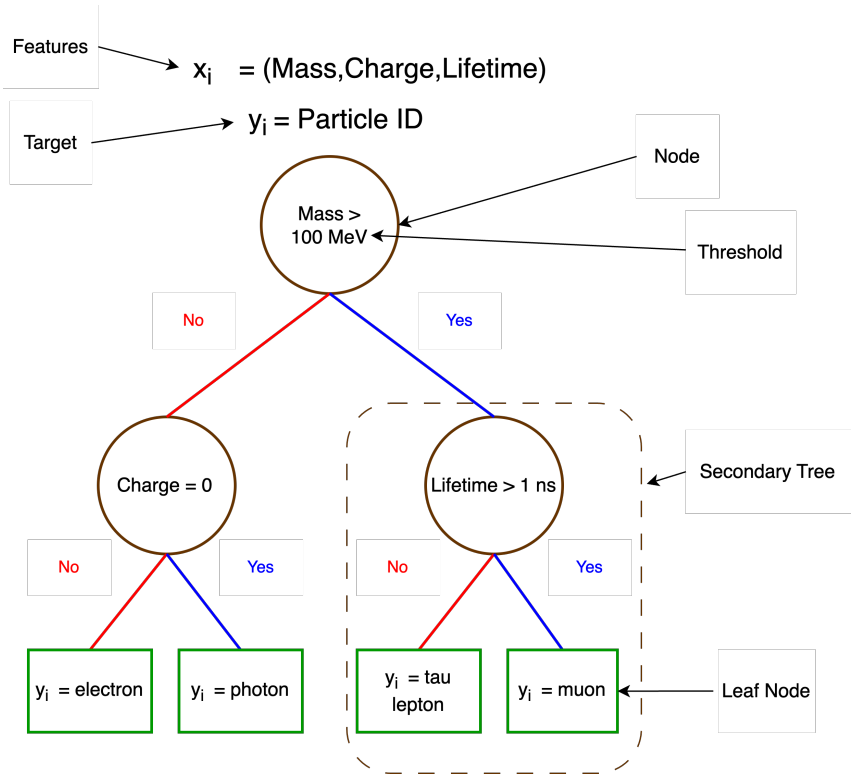


Figure 3.1: A decision tree for a simple particle ID task. Each data point consisting of a mass, charge and lifetime are evaluated at each node in the tree. Each node compares one feature to a threshold and decides which split the data point follows. After many nodes the data point reaches a leaf node that returns a final prediction, in this case particle ID. The thresholds and which feature to inspect can be changed to give different results labels and are optimised to give the best answer for many data points.

3.2.1 Bagging and Boosting

Bagging is a very simple form of ensemble DT learning where the training dataset is randomly split into smaller subsamples and a DT overfit to each subsample [84]. When evaluating on the test dataset all the trees are evaluated and their average prediction is used which has a lower variance than a single tree trained on all the data. An improvement to reduce the overfitting in each subtree uses a random selection of features in every random split of the dataset—an algorithm called random forests.

An issue in bagging is that each tree is trained separately, so they could make similar mistakes. A way of avoiding this is to use a boosting approach where the error of the previous tree is used to correct the next tree [85]. This is performed by training a simple tree on a random split of data, evaluating its performance on a different split of the data and up-weighting the samples it incorrectly predicted. This re-weighted split of data is then used to train the next decision tree. A fundamental part of this process is to use very simple trees that overfit to their subsample but underfit to the entire dataset. These models are generally referred to as boosted decision trees (BDT).

A final extension to this boosting approach is to calculate an overall loss function. When a simple tree is fit, the loss is calculated and the gradient of that loss with respect to the tree parameters calculated. A new tree is then created which aims to correct the previous tree, updating the new tree parameters to reduce the loss. Each tree is weighted by their overall contribution to minimising the loss so that when they are combined to make a prediction, the best possible prediction is made. This gradient boosting method is very flexible as it allows any differentiable loss function to be used as the target for learning, allowing for regression and classification tasks to be optimised [86]. These models are referred to as gradient boosted decision trees (GBDT).

There are some key advantages of DT techniques:

- Simple: decision trees can be printed out, and every threshold examined making understanding the algorithms choices and possible biases relatively easy.
- Agnostic to feature scaling: as the underlying operation is a set of thresholds these thresholds can be scaled as the features scale.
- Automatic feature pruning: if a feature does not contribute to improving the tree it will not be used
- Fast and lightweight: because of their low complexity, DTs are fast at training and inference.

There are disadvantages:

- Predictions are not continuous: as the overall model is made of many discrete trees the resulting output in a regression problem is discrete, meaning they do not extrapolate well in out-of-distribution samples which differ from the training sample.
- Simple: they cannot create complex functions between features, for example multiplying two features together.
- No internal representations: each tree, when being trained, cannot use internal representations created by previous trees, they are each naively following a gradient descent.

3.3 Neural Networks

A very popular algorithm for ML is the neural network (NN). NNs have their roots in a computational unit called a perceptron [87] where a set of inputs are multiplied by a weight and summed, i.e. $y = \sum_{i=0}^{N} w_i x_i + b$ where w_i

are the weights of the sum and b is a bias term. This sum then passed through a threshold function to produce a 1 or 0. The simple perceptron unit is shown in figure 3.2.

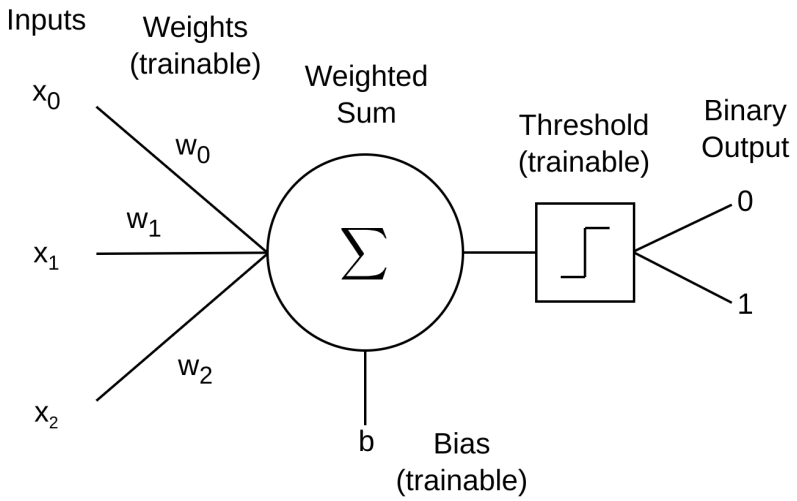


Figure 3.2: A perceptron with the trainable weights labelled w_i, bias (b), and step function threshold. Inputs x_i are weighted and summed, and their output passed through the step function, if the sum is greater than a threshold a one is returned, if not a zero is returned.

Many of these units can be stacked in a network to create a multilayered perceptron with each perceptron node connected to every other node. Once again, to train the model the derivative of the loss with respect to the model parameters is found and the model parameters updated in the direction that minimises the loss. This can be performed iteratively for a multilayer perceptron but is computationally costly. A faster way to compute the loss gradient is to use an algorithm called back-propagation [88], but this relies on the entire network being differentiable with respect to the loss so the simple threshold function on the output of every node is not usable. These thresholds are replaced with differentiable non-linear functions called "activations" and it becomes possible to calculate the derivative of the loss with respect to the activation, called the error of the node. Non-linear activations are important as without this the network could be simplified to a single matrix multiplication which is no more complex than a weighted linear sum.

There are many activation functions used but one of the most common is the Rectified Linear Unit [89] that returns 0 under a threshold and returns the input above the threshold. The replacement of the threshold with a non-linear activation is shown in figure 3.3 with the interconnected version, the full NN, shown in figure 3.4. A fully connected NN, as is shown in the figure, is known as a dense neural network (DNN).

The gradient of the entire network is found by propagating the errors (differentials of the activations) backwards through the network, multiplying them by the outputs at each node though a chain-rule multiplication of derivatives. The errors and outputs are calculated in a forward pass of the network, the gradients are calculated in the backwards pass, so the weights and biases of the network can be updated for the next forward pass. Again, to reduce computational complexity, the process is not carried out on a signal input $\mathbf{x}$ but on a batch of inputs and averaged across them. One full cycle of batches that includes the entire training dataset is called an epoch. Typically, many epochs of many batches are performed to iteratively train an NN.

NNs are a very powerful technique in ML because of the hidden representation of features being passed

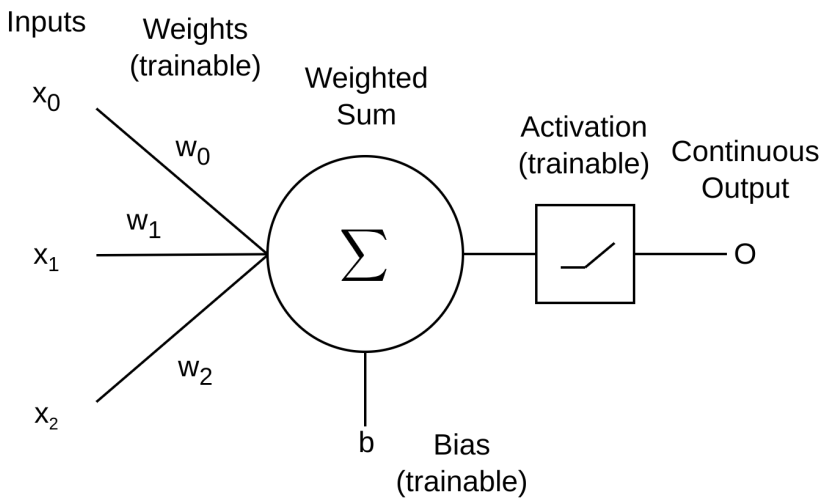


Figure 3.3: A neural network node showing the trainable weights (w_i), bias (b), and activation function. Inputs x_i are weighted and summed, and their output passed through the activation function that returns a continuous output depending on what the input is. This node's output is differentiable with respect to the weights, biases, and parameters of the activation function, making back-propagation possible.

into internal layers—so-called "latent" features. This allows complex functions of input features to be created. They can also be trained with different loss functions for different tasks, making them flexible for almost any ML scenario. However, there are many issues; there are a seemingly infinite number of external parameters to control when creating NNs with network architecture, depth, learning rate, learning scheme being a small set of examples. Typically, these parameters are either decided ad-hoc or some ideal set is found through hyperparameter optimisation. There is also a lack of understanding over exactly how the network is learning because of the hidden representations, which leads to explainability and reliability issues.

3.3.1 Convolutional Neural Networks

When using high-dimensionality information such as an image, a fully connected DNN is unsuitable because of the millions of potential parameters needed to use every pixel of an image as an input feature. This scales poorly with the size of image and leads to unsuitably slow training. A method to avoid this issue is to introduce a convolutional kernel on the set of features [90], to create a convolutional neural network (CNN). Convolutions are a way of using a single set of parameters across an entire set of input features. For example, using an input of 256 bins, a 3-bin sliding window is moved across the bins one step at a time with each of the 3 bins being multiplied by the window parameters and passed through an activation to give a new convolved set of 256 bins. When training, only the 3-bin window parameters need to be optimised rather than 256×256 different weights and biases. This simple example is shown in figure 3.5 with a number of parameters that can be tuned:

- Kernel Size: the size of the filter used; larger kernels have more parameters but can find larger patterns in an image.
- Channels: how many kernels per input image? Each channel will output an output array of possible image features, but more kernels are more expressive.
- Step size: how many elements of the input array are stepped by the filter? Step sizes larger than 1 will see a smaller output array and is a method of reducing the size of the inputs.

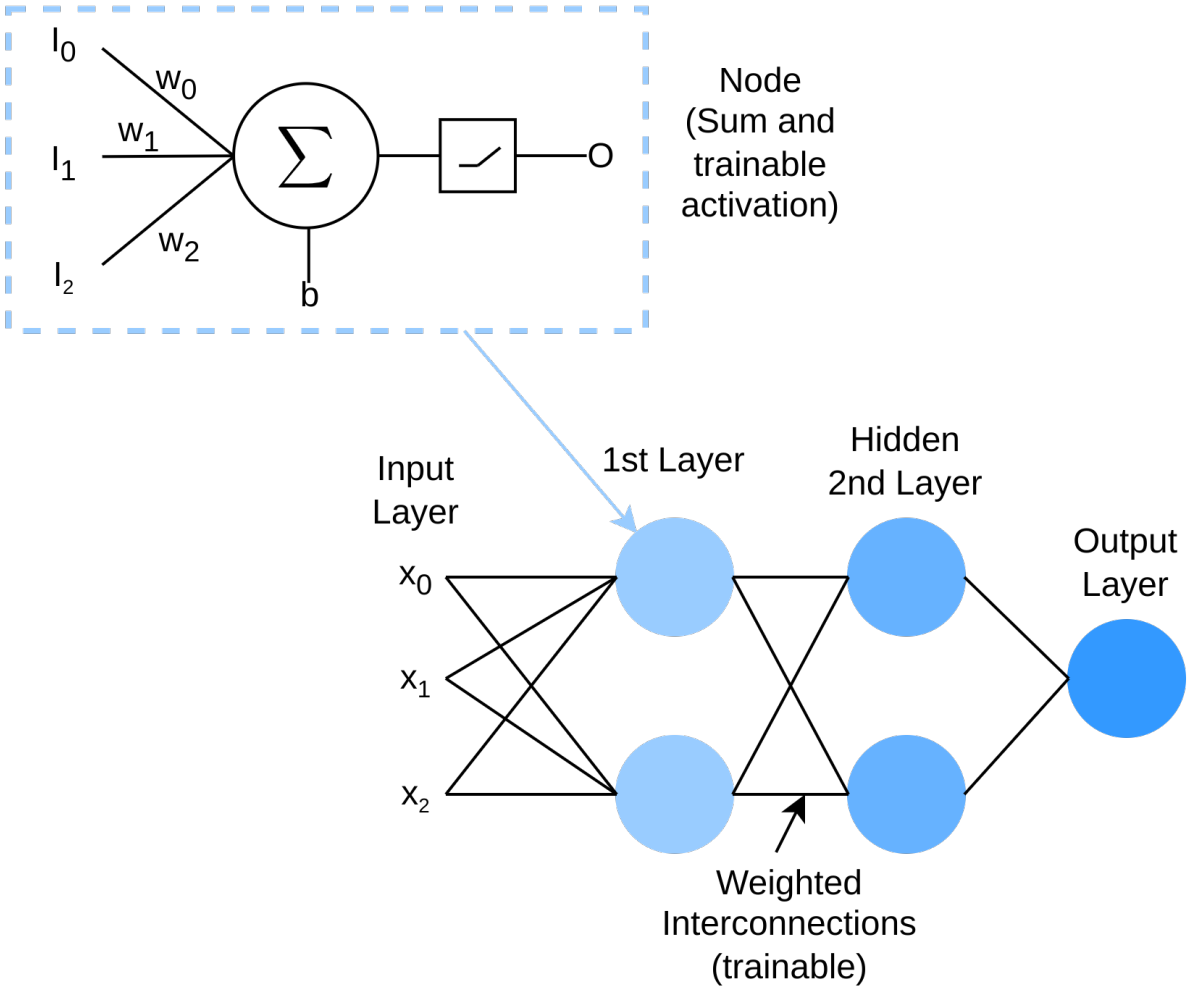


Figure 3.4: A neural network. A set of nodes make up each layer with each node being the neural node shown in figure 3.3, with inputs I_i leading to outputs O. The input layer inputs each of the features into all of the nodes in the first layer. The outputs of each of the nodes of the first layer are output into the nodes of the second layer, and then the outputs of these nodes are combined in the output layer. The second layer is a "hidden layer" meaning the inputs and outputs are unseen and are entirely dependent on the training of the network.

CNNs can be used for a variety of input shapes, with multiple dimensions. The most common CNNs are colour image CNNs that use two-dimensional kernels for images with different channels for each colour, each channel is a different kernel that acts on the image. CNNs typically use multiple layers of convolutions, non-linear activations, and pooling layers which first convolve to extract a feature and then pool (aggregate the feature) to compress images in their x and y dimensions but increase the number of channels. This process continues until the resulting tensor is flattened and provides input to a small fully-connected DNN with some output node for the task being trained for.

In the case of CNNs, the input data is a Cartesian grid of numbers where a single pixel is linked only to each pixel that surrounds it. This can be seen as a simplification of a wider representation of information; an image is just a graph but with the only node information allowed being one number and the only edges being to the four surrounding pixels. If instead these conditions are relaxed and each graph node (or pixel) can be a variety of features with any number of edge connections to other nodes there are a set of more general NN techniques that are used. Graph neural networks [91] work on graph-structured data using graph "convolutions" on these graph

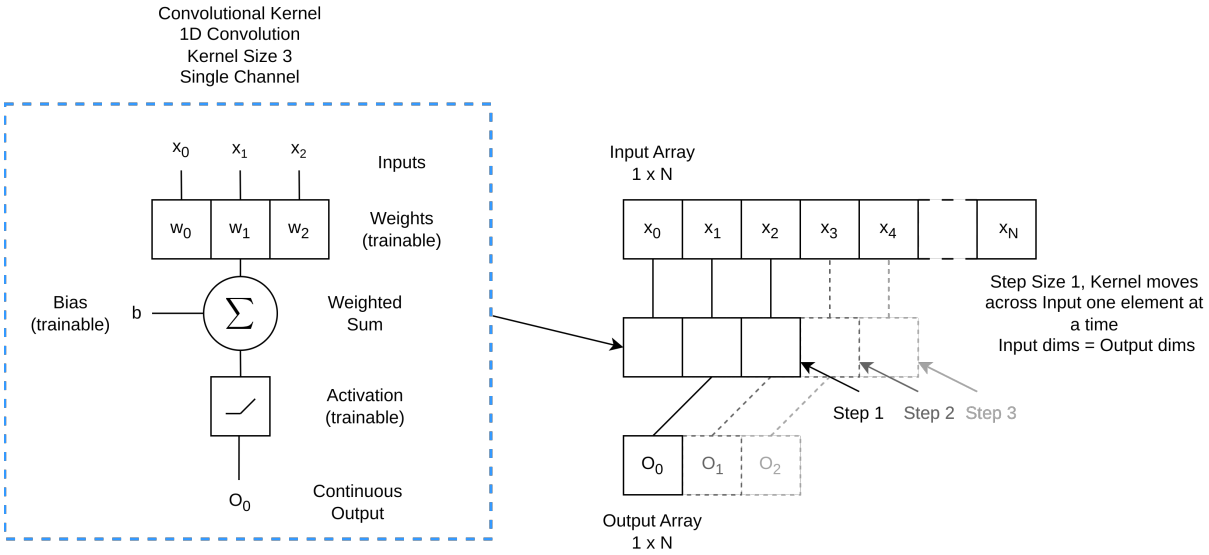


Figure 3.5: A convolutional neural network with the convolutional kernel shown separately. The kernel includes a weighed sum and activation function with trainable weights and activations. The kernel itself is no different to the neural node shown in figure 3.3, however instead of there being N inputs to the node only three inputs are input. The kernel is stepped across the data to create one output per step building up another array as output.

structures. These styles of NN are very powerful at finding complex links between feature points in many dimensional, making them suitable to complex tasks and are commonly used in situations where the dataset does not exist in a simple Cartesian grid.

NNs are a core part of modern ML with key advantages:

- High-level feature extraction: NNs can produce non-linear combinations of features that can accurately model the desired function.
- Adaptable: networks made of the same building blocks of interconnected nodes have the flexibility to perform a wide range of tasks with differing loss functions or multiple tasks simultaneously.
- Easy to set up: because of the popularity of these techniques, setting up and training NNs is simple, with boilerplate code readily available. Whether this creates effective NNs or even if NNs are appropriate for the task remains open.

But they also have disadvantages:

- Many parameters: with multiple interconnected layers the number of parameters in a network becomes very high, while methods such as CNNs combat this they can still be over parameterised, overfit to data, and can be computationally costly.
- Interpretability: the hidden layers in a network create internal representations meaning subsequent predictions cannot be easily understood.
- Hyperparameters: with the variety of NN techniques and possible parameters to tune the network, an optimal model for a given task is ad-hoc and results can be difficult to reproduce if small changes to the data occur.

3.4 Fast Machine Learning

From a firmware programming point of view, ML techniques are simple and effective ways of creating highly complex functions without the difficulty of writing these functions in viable firmware. Modern FPGAs are large and resource abundant and modern tools for generating firmware from NNs and BDTs are easy to use and create packaged firmware blocks that can be easily inserted into existing firmware. This design philosophy speeds up the firmware development process and can result in highly effective algorithms, if the ML task is thoughtfully set up. In real-time applications the inference time of ML needs to be fast with 100–1000 ns inference time. In order to make use of the parallelism of the FPGAs and increase the amount of data that can be processed ML models are replicated multiple times and evaluated in parallel, making their overall size critical. When designing ML models for firmware there are some key areas that need to be addressed.

3.4.1 Architectural Considerations

When designing fast ML, the first step is to consider what architectures are possible. Simplicity is key in designing fast ML where very deep and very complex models will not be able to run in the desired latency. Some models that act on series of data in order to learn a structure within the data, for example predicting words in a sentence, will have very long latencies. This is due to the need to feed the output of a network acting on one data point into the same network acting on the next point. Graph NNs, because of the complex graph structure of their input data, suffer from high latencies simply due to the graph building stage as the data is organised. The L1 trigger currently only plans to have only DNNs, CNNs and BDTs because of their simplicity, but research into implementing more complex algorithms in firmware is ongoing [92]. The choice over which algorithm to use depends on input data. If there are high-level features that are expressive of the target variable then BDTs are highly performant and far less computationally expensive compared to NNs. Whereas if only low level features are available or a non-linear combination of features are needed, DNNs are appropriate. CNNs are useful if the data is image-like with translational symmetry, but the number of input values is too high making the use of a DNN costly.

When designing a fast ML algorithm, the size of the model must be small. While state-of-the-art DNNs can have up to billions of parameters, this is entirely impossible for a low-latency, resource constrained environment and instead 1000s of parameters need to be targeted (even this is too large and techniques are needed to reduce this further). A balance must be made between model expressiveness, needing more parameters, and model resource usage, requiring fewer parameters. CNNs are a good option for reducing parameters but can invoke high latencies if an entire image needs to be processed by moving a convolutional kernel over it before being passed onto the next stage of an algorithm. While there are techniques for evaluating all parts of an image in parallel this comes at a high resource cost. BDTs are an ensemble method that typically have low depth trees but with up to 1000s of trees which again leads to many total parameters and total operations needed in firmware, so fewer trees are better.

3.4.2 Quantisation

FPGAs work in fixed-point mathematics. This means each internal representation of a number has a fixed number of bits with a set number of bits for the integer part of a number and set number of bits for the decimal. This number of bits is optimised before the firmware is implemented and the larger the number of bits the more resources are required in the FPGA in order to represent the numbers. An NN or BDT needs to have each of its parameters restricted to that set number of bits before it is implemented in firmware. Typical CPU training of a BDT or NN uses 32-bit floating point mathematics meaning there is a variable number of bits for integer and decimal so long as together they add to 32. The simplest method for restricting the parameters is with post-training quantisation where after training is complete the model has its parameters recast to a fixed-point set

of numbers. If this fixed-point set of numbers does not encompass the model parameters, truncation can occur, leading to poor performance. The larger the discrepancy between target fixed-point numbers and model parameters the worse this truncation can be.

In BDTs, quantisation is performed by finding all the parameters in the trees and inspecting the total distribution, making a cut that is reasonable. Performance does not tend to be heavily impacted by this quantisation, especially if the inputs used for training are already in their quantised representations. With NNs it is possible to perform post-training quantisation, but there is a more robust alternative. Quantized-aware training restricts the bitwidths of the network parameters while the training is running, which means the network learns weights and biases that fit within the fixed-point representations [93]. This method is so effective that single-bit parameter networks can be trained making very lightweight networks [94].

3.4.3 Pruning

Pruning is a process of iteratively removing unimportant parts of a model [95] [96]. For BDTs this can be achieved by examining the overall effect on the loss function of a particular node split and if that effect is below a threshold, remove the node. This is a simple process to remove any parts of the overall model that are not making large contributions and can make the model smaller. Pruning can be performed while training with each tree being evaluated as it is grown. For NNs the process is more complicated. Firstly, regularisation terms are introduced to the network to have smaller weights such as ridge and lasso regularisation that penalise the absolute sum of the weights and absolute sum of the weights squared respectively. Together these cause the network to have fewer, smaller weights as it trains. During training or at the end of a training cycle the network weights can be evaluated and the parameters closest to zero within a threshold set to zero permanently. This removes the interconnections between nodes in the network resulting in a less complex, and less overfit network which in some cases is more robust than a fully-connected network. This can be a very effective technique in reducing network size by up to 90% while maintaining the same task performance. Typically, the NN starts large with many nodes and is allowed to learn for multiple epochs before pruning occurs which allows the network to fully explore the feature spaces and create important connections. Starting from a smaller network and pruning it less, while effective at giving a small overall network, will not have the expressiveness of a larger network that is then pruned back as it has not had the chance to fully explore the feature space and possible higher level latent features.

3.4.4 Implementation

The final step of transferring a quantised and pruned model to an FPGA is the use of a transpiler such as hls4ml [97] [73] or conifer [98]. These tools take NNs and BDTs and create the firmware block or direct HDL (in the case of conifer) to be implemented with the firmware. This can then be wired up to existing firmware and tested. There are parameters within hls4ml to consider. Firstly, the reuse factor allows firmware blocks to be used by multiple parts of the same NN, for example a weight and bias operation is the same across the network so can be reused resulting in lower resource usage. However, this means a higher latency as multiple operations need to pass through the same firmware block before the entire network is evaluated. Again, this choice is a trade-off between resource usage and total latency. Another consideration in CNNs is the parallelisation factor of the input [99]. The CNN has a kernel that moves across the image a step at a time meaning each step has to occur before the next step resulting in a high latency, known as a serial input stream. Alternatively, the kernel can be replicated and work on the entire image at once in multiple parallel streams, known as a fully parallel input stream. A final consideration when implementing firmware is the inputs to the model; in firmware these inputs are also bit-restricted or in integer representation, making training with these bit-restricted numbers important. Typically, these features will be of

different scales with different bit widths and digitisations. For NNs this poses a problem, as different scaled inputs can cause vanishing gradients when training. Implementing lightweight feature scaling is therefore an important part of developing NN firmware and care must be taken to ensure the training features are identical to those being provided in firmware.

Chapter 4

Track Reconstruction

The function of the track finder (TF) is to reconstruct charged particle tracks with a $p_T > 2$ GeV using the outer tracker p_T modules [58]. These tracks are used in a variety of ways downstream. One major use is in vertex finding in the global track trigger, which reconstructs the primary vertex (PV) and is used in associating tracks and other trigger objects to the hard scatter in an event [100] (see chapter 5). This is particularly important for reducing pileup for the correlator trigger which uses the vertex to weight candidates in the PUPPI reconstruction. This gives the Level-1 trigger high efficiency for key physics signatures in spite of the large additional pileup. Another use of the tracks is for muon isolation and improving their p_T resolution which is useful for distinguishing pileup interactions from the hard scatter processes and increasing the efficiency of muon triggers. A more detailed discussion of the use of L1 tracks in key physics analyses is found in chapter 2.

The following plots and discussion of the track finding and track quality use an inclusive top-quark pair production sample. This was decided based on the high multiplicity of the event, i.e. many tracks will be created from the underlying top quark production and decay due to the high mass of the top quark. These kinds of high multiplicity events, when overlaid by soft hadronic pileup with many tracks originating in these decays, can be seen as a 'worse-case' scenario for the track finding algorithm with $O(10{,}000)$ stubs[1] in the detector. For the L1 trigger these events have no other trigger paths other than accurate reconstruction of the jets originating from the top quarks through the correlator trigger, so efficient tracking is a necessity.

4.1 System Design

The TF needs to be able to produce tracks from across the outer tracker with high efficiency (> 95%) for both the barrel and endcap regions, with a total coverage of $-2.4 \leq \eta \leq 2.4$ and a $p_T > 2$ GeV. It needs to have flat efficiency across ϕ and reconstruct tracks for the beam spot region of $-15 \leq z_0 \leq 15$ cm. It needs to have a low number of fake tracks, which are defined as tracks not originating from true particles in the detector but instead created out of a complex combination of stubs that match the shape of a curved track[2]. Finally, it needs to run on Xilinx VU13P FPGAs with a total latency of 4 μs.

In order to be able to provide tracks downstream for the full detector in a low enough latency, the TF is split regionally and in a time-multiplexed manner. There are nine ϕ regions or "nonants" and a time-multiplexing period of 18 meaning each TF board will receive a new event every 450 ns with multiple events being processed in parallel

[1] See chapter 2, section 2.1.2

[2] For the studies presented below fake tracks are found using Monte Carlo truth matching of true charged particles to reconstructed tracks based on which stubs are common between the two.

on different boards. The planned board for the TF will be an Apollo board hosting two VU13P FPGAs, with a total of 162 boards for the full outer tracker. As input, each TF board will receive 48 fibre optic links providing stubs from the DTC, which are duplicated at nonant boundaries to eliminate any cross communication between processor boards, meaning each nonant, for any given bunch crossing (BX), is independent. As output, each TF board will provide two fibre optic links duplicated six times for the various downstream consumers containing reconstructed tracks to downstream algorithms which will receive a total of 18 links for the full set of tracks across the detector for a given BX.

4.2 Track Reconstruction

The reconstruction of tracks at Level-1 is one of the most computationally demanding for the system. With a $p_T >$ 2 GeV and an average pileup of 200, in a fully hadronic top quark pair production event, the average number of stubs created in the detector is 15,000 at 40 MHz, which need to be processed to create ~ 200 tracks. There have been a number of algorithms suggested for the track reconstruction but ultimately a hybrid mixture of two — the tracklet [101] and TMTT [102] approaches — will be used for the trigger. It begins with the tracklet pattern matching step [101] which finds tracks based on seed stubs, followed by a Kalman filter (KF) that performs the track fit. After these two steps a BDT is used to evaluate the quality of the tracks, as described in section 4.4.2, and the tracks are routed and formatted before being sent to the rest of the trigger.

The tracklet stage of the algorithm uses combinations of the six barrel layers (L1–L6) and five disk layers (D1–D5) as seeds to create tracklets, these layers are shown in figure 4.1. The seeds used are L1+L2, L3+L4, L5+L6, L1+D1, L2+D1, D1+D2, D3+D4 as well as L2+L3 for specifically the transition region. This selection of seeds is used to have a wide number of options across the entire outer tracker in order to maximise efficiency for this seed finding stage. However, a compromise is made as more seeds need more firmware to process them. Seeds are used to create initial "tracklets" in the tracker, a process show in step 1 of figure 4.2.

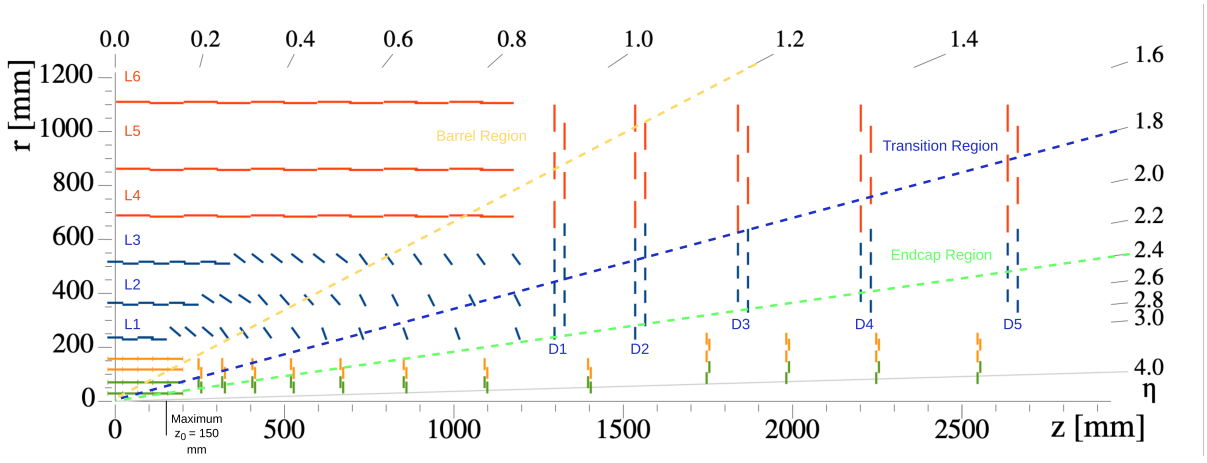


Figure 4.1: One quarter of the tracker [57] in the r-z plane. The inner tracker is shown in green and yellow, the outer tracker PS modules are shown in blue and 2S modules shown in red. The barrel layers are labelled L1 to L6 increasing as the radius increases and the disks are labelled D1 to D5 in increasing z. The barrel region ($0 \leq |\eta| < 1.2$) refers to the area where tracks will only be reconstructed from barrel stubs, similarly for the endcap region ($1.8 \leq |\eta| < 2.4$), which will mostly be endcap stubs. The transition region ($1.2 \leq |\eta| < 1.8$) can have both barrel and endcap stubs which can cause issues for the track reconstruction.

The next stage of the reconstruction process (step 2 in figure 4.2) is to project track candidates both outwards and inwards (if using layers other than the two closest to the beam line) from the tracklet seed to the other layers

with a constraint that the track candidate originates from $x, y = 0$ (a beam spot constraint). These projections form a window in each layer in which stubs are searched. The windows are created based on predetermined calculations for each seed type and can be tuned to maximise efficiency. In each layer matching stubs are added to the track candidate (step 3 in figure 4.2) and in the case of multiple matches the stub with the smallest residual (distance between the stub and track candidate in a specific layer) is kept. A minimum of four stubs and a maximum of six stubs is required to create a track. Again, this is a compromise between efficiency and firmware resources where fewer minimum stubs would lead to high efficiency but high fake rate and many candidates for the KF. A higher number of maximum stubs would lead to a better fit for the track but would require a KF able to deal with the higher combinatorics associated with having many stubs per track. At this stage, any track candidates with shared stubs are merged prior to the KF in order to reduce the number of candidates going to the KF and to reduce the overall duplicate rate of the algorithm. The tracklet stage of the chain is written as an emulation for comparison to firmware using bit accurate information throughout the algorithm.

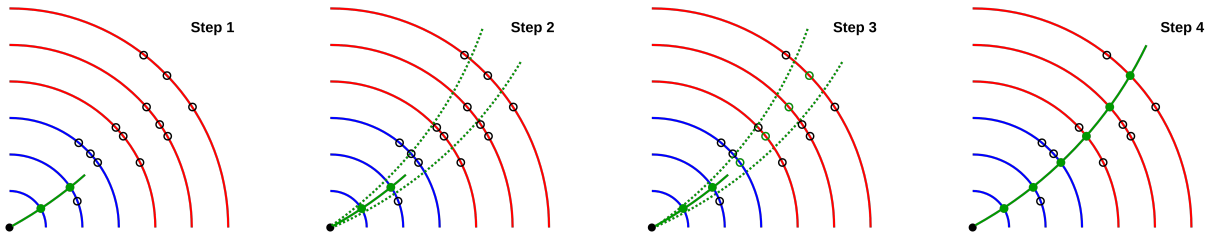


Figure 4.2: The tracklet stage of the L1 track reconstruction. Step 1 shows the initial seed stubs of L1 and L2, together these form the tracklet. Step 2 shows the projection of the tracklet's parameters out to the other layers with an uncertainty. Step 3 shows the selection of other tracker hits based on their position in the tracklet's projection window. Step 4 shows the final fitted track with all stubs associated to it.

The Kalman Filter receives the track candidates along with their residuals; these form the KF state and covariance matrix respectively. The stubs are also sent from the tracklet stage and iteratively added in the KF formalism which is an algorithm that iteratively adds measured information to a state to improve the estimation of the state [103]. There are two steps to a KF; a propagation step and then a state update. In the propagation step the state and covariance matrix (the error of the state) are multiplied by a state transition matrix to give an estimate of the state in the next step[3]. The update step then uses measured data and its own uncertainties to improve the estimate of the state. The amount that the measured data influences the state depends on the relative uncertainty of the measured data, called the Kalman gain. This iterative process continues for as many steps as needed.

In the track fitting context the state x is the four track parameters:

$$x = (\frac{1}{2R}, \phi_0, \cot(\theta), z_0), \tag{4.1}$$

where R is the radius of curvature of the track, $R = p_T/(0.003qB)$, with q being the charge of the particle and B being the CMS magnetic field strength. ϕ_0 and z_0 are the track azimuthal angle and longitudinal impact parameter of the track at the beam line. θ is the polar angle, with $\cot(\theta) = \tan(\lambda)$ (λ is defined in figure 1.4). The measurements added to the KF state are the stubs in each layer with the stub residuals forming the uncertainty in the measurement. The state projection equations come in the form:

$$\phi = \frac{r}{2R} + \phi_0, \tag{4.2}$$

[3]While it is natural to assume the KF steps in time, the step could be any stepping parameter.

$$z = \cot(\theta)r + z_0, \tag{4.3}$$

with r being the radius of the next stub. ϕ and z are the azimuthal angle and impact parameter of the track at this given r. These equations are linear in r making it simple to use stub radius r as the stepping parameter in the state projection. This feature motivates the use of these track parameters instead of the more common p_{T} and η. In each iteration the track is projected outwards from the beamline to the next detector layer. At this layer the stub associated to the layer is added to the state as the measurement, with a Kalman gain depending on the stub residuals. The state is updated using this stub. Once all the detector layers are iterated over, a final track fit is given.

Within this formalism it is possible to remove stubs if the Kalman gain is too large, meaning the KF can internally reduce the number of incorrectly matched stubs or fake tracks being created. A key motivation for using the KF in this track fitting scenario is the small matrices involved; with this four parameter fit the maximum matrix size is 4×4, and only one of these small matrices needs to be inverted.

However, there are some modifications that can be made to the KF to reduce its resource usage. In this form the KF needs to be able to handle full ranges of z, ϕ and θ. Also, for a given track there are far more possible stub combinations at small r as opposed to larger r as tracks spread out from the beamline. This motivates two changes. The first is to redefine track parameters in terms of a cylinder of radius T. This cylinder has an equal number of stubs inside and out and tracks are defined by where they cross this cylinder with the new coordinates z_T, ϕ_T and r_T — shown in figure 4.3. The track fitting space is also split into ϕ and η regions so that the number of stubs in each region is equal. This means that for any given track fitting region the number of stubs to be processed is even and there is no situation where certain detector regions take longer to process than others.

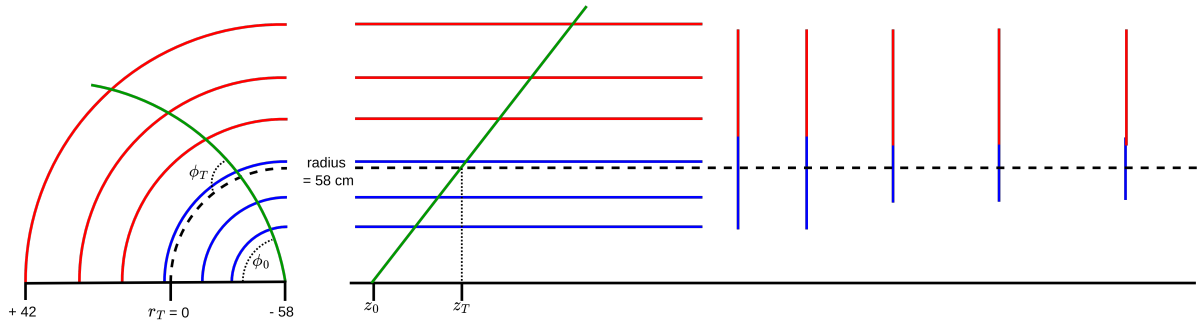


Figure 4.3: Definition of the coordinates used by the Kalman filter for the r-ϕ plane on the left and r-z plane on the left, with an example track shown in green. A cylinder (the black dashed line) is created inside the tracker where the density of stubs inside and outside the cylinder is equal. The cylinder has radius T and allows the Kalman filter to redefine its tracks based on this radius with z_T, ϕ_T and r_T. With the balanced density of stubs the Kalman filter has a more equal processing load when creating tracks.

Multiple KF workers (parallel processors running the KF) are envisaged depending on the number of track candidates being provided by the duplicate removal stage, again, a balance between performance and resource usage will be found. There are currently two versions of the KF, one purely simulation with floating point calculations throughout and one with fixed-point calculations and truncation. This second fixed-point version represents the emulation of the firmware KF and is used to directly compare to firmware run on an FPGA, as is described in chapter 6. The simulated KF has internal χ^2 cuts which are used to remove states with too high uncertainties as a handle on fake tracks whereas the emulation KF does not, and relies only on the Kalman gain to

reduce mismatched stubs. This decision was made based on the development of the track quality BDT (described below) which is able to discriminate real from fake tracks effectively. Allowing downstream users to select tracks based on their quality increases flexibility as different users require different track purity. The final tracks and associated stubs are passed downstream to the KF output module, which has also been written in simulation and emulation. The performance of both the emulated and simulated KFs are shown below.

The hybrid algorithm has flat efficiency across η demonstrated by figure 4.4a, where efficiency in this case is defined as number of reconstructed tracks matched to a generator particle divided by the number of reconstructable[4] generator particles passing through the outer tracker in a given η bin. This definition of efficiency does not change as fake rate increases, it can be seen as the true positive rate of the track finding algorithm. The efficiency is around 95% with no geometrical dependence on η, reducing bias for downstream algorithms. The emulated KF has slight reduction in efficiency in high η endcap regions due to a truncation of tracks outside the algorithm acceptance of $\eta = 2.4$. This occurs in the emulation which has set fixed digitisation ranges of found track parameters whereas the simulation version does not and performs a cut on track η after it has finished calculation.

The p_T efficiency seen in figure 4.4b demonstrates that efficiency increases as p_T increases rapidly from $p_T = 2$ GeV, but begins to tail off at higher p_T. Both simulated and emulated KFs perform similarly. The low efficiency at low p_T is simply an effect of tracks being on the boundary of 2 GeV and the reconstructed resolution of track p_T not being perfect. At higher p_T the efficiency drops, which is not seen in electrons and muons (shown in figure 4.5); electron efficiency is low at lower p_T due to multiple scattering and plateaus to 90%, lower than other particles because of the bremsstrahlung losses causing the electron tracks to deviate from perfect helices. Muon efficiency is > 95% for all p_T bins and is very high because muons are minimally ionising at these ~GeV energies (not low enough p_T to undergo multiple scattering and not high enough p_T to undergo radiative losses [104]) making their tracks very clean and easy to reconstruct. The drop in efficiency at higher p_T in hadrons is due to the presence of high energy jets. Jets contain many hadrons in a collimated stream in the detector leading to a high multiplicity of stubs in one region of the detector. This increases the difficulty of track finding due to the increase likelihood of incorrect stubs falling within the tracklet projection windows. The high stub density also leads to truncation where the processing of the front end electronics and DTC cannot pass all the stubs downstream, so some are lost.

As p_T increases, the fake rate increases—as shown in figure 4.4c. The fake rate or fraction of non-genuine tracks is defined as the number of tracks not matched to a genuine particle divided by the number of reconstructed tracks, and can be seen as the false discovery rate of the track finding algorithm. As p_T increases the number of fake tracks falls in line with the number of real tracks, but not as quickly, which can be seen in figure 4.9d. This again is linked to the presence of high p_T jets with many stubs in a single region making incorrectly linking stubs more likely for the track finding algorithm. The emulated KF sees slight troughs in the fake rate at certain p_T, e.g. $p_T = 8, 12, 24$ GeV which is due to the duplicate removal. The current emulation of the duplicate removal performs its calculations in p_T bins, matching tracks in these bins and removing the duplicates. At the region between these bins there is a drop in performance for the duplicate removal. This leads to many duplicate tracks at the boundaries of the bins which, with the definition of non-genuine fraction being the number of fake tracks / number of reconstructed tracks, leads to an apparent reduction in the fake rate as shown by these troughs in the fake rate. The number of fakes has not changed, the number of reconstructed tracks is higher, The fake rate for the emulated KF is significantly higher, due to the lack of internal cuts on tracks, but this large increase in fake rate

[4]Only tracks $p_T > 2$ GeV and in the set z_0 and η boundaries are reconstructed by the algorithm so only the generator particles in the same kinematic regions are considered.

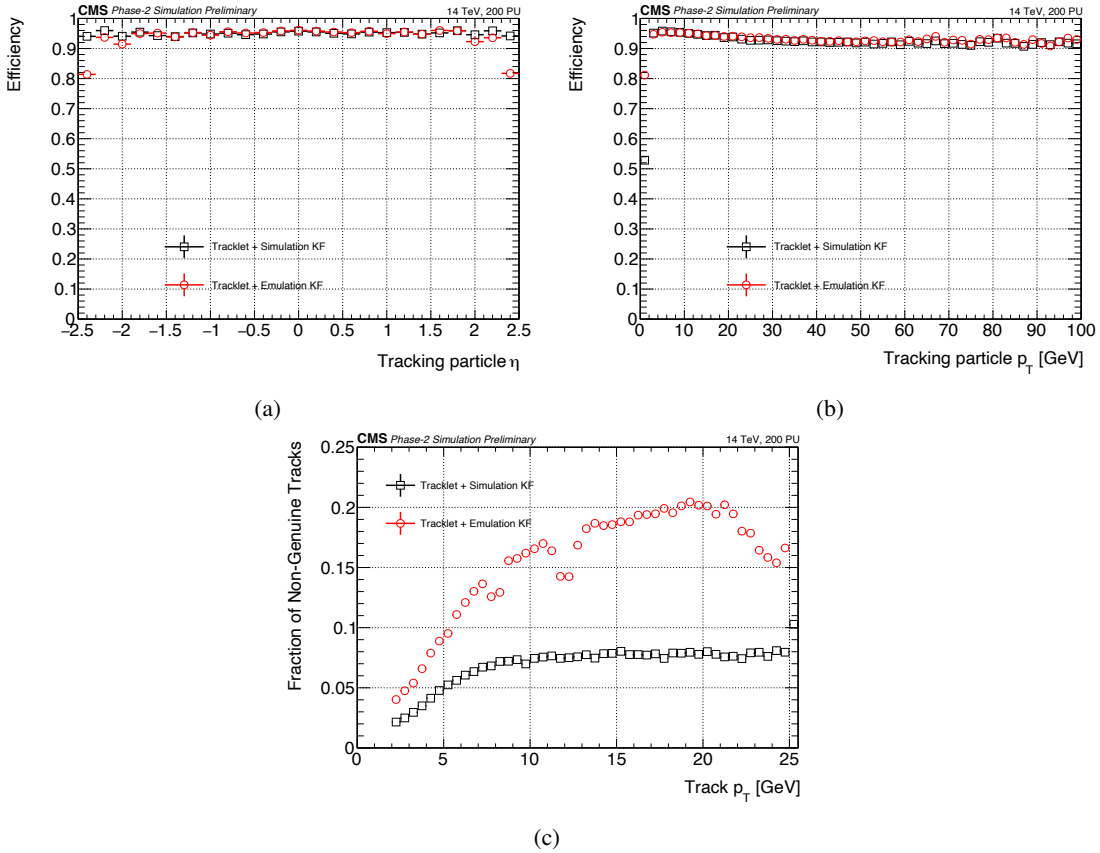


Figure 4.4: Level-1 hybrid track parameters. Figures (a) and (b) show track finding efficiency as a function of η and p_{T}. Figure (c) shows the rate of tracks not matched to generator particles as a function of p_{T}. In black is the floating point simulated KF, in red the fixed-point emulated KF. All plots are generated from a top quark pair production sample with over 180,000 events. Figure (a) shows flat efficiency across η and similar performance for simulation and emulation apart from at extreme η. Figure (b) also shows similar simulation and emulation performance. Figure (c) shows a far larger fraction of non-genuine tracks in the emulation due to the lack of χ^2 cuts. The dips at $p_{\mathrm{T}} = 8, 12, 24$ GeV are due to the duplicate removal creating more tracks rather than there being fewer non-genuine tracks at these points.

can be dealt with by track quality algorithms described below.

An extension to the baseline hybrid track finding is to include displaced track finding, i.e. removing the beamspot constraint in the tracklet seeding stage and allow the KF to also fit for the d_0 of the track. This would see triplets of layers used to seed the tracks and would be useful for reconstructing long-lived particles of d_0 up to 5 cm with reasonable efficiency. There is also an improvement to the reconstruction of electrons which, because of bremsstrahlung effects, leave distorted tracks in the tracker. Work is ongoing to understand the firmware feasibility of this approach as there is a large overhead associated with including the additional triplet seeds and associated processing needed for the displaced tracking.

4.2.1 Output to the Level-1 Trigger

The output from the TF to the Level-1 trigger consists of track words of 96 bits each, outlined in table 4.1. The values in this table are calculated in the KF output step which takes fitted KF tracks and their associated stubs and outputs these track words. The calculations in simulation for each of the track quality and helix parameters in

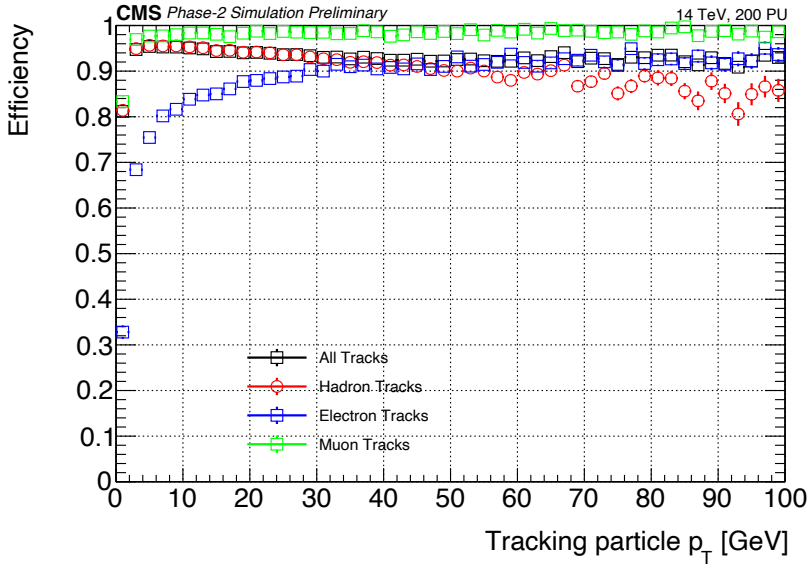


Figure 4.5: Level-1 hybrid track p_{T} efficiency split by particle type. Muon efficiency in green shows very high reconstruction efficiency across the p_{T} range because of the clean tracks left by muons. The electron efficiency in blue is low for low p_{T} because of the multiple scattering and remains low because of bremsstrahlung effects. The hadron efficiency starts reasonably high at low p_{T} but drops because these tracks are part high energy jets with many stubs in specific detector regions which makes the track reconstruction inefficient and stub truncation likely. The overall track rate in black shows a midpoint between hadron and muon efficiency.

table 4.1 are listed below. A detailed description of the firmware implementation is outlined in section 4.3 and differences between simulation and firmware discussed.

In simulation, the $\chi^2_{r\phi}$ (χ^2_{rz}) variables are calculated based on stub ϕ (z) positions squared, multiplied by a weight that depends on; the module ID of the stub, additional scattering terms, and additional corrections based on the pitch of the module. This weighting is labelled v0 (v1) and is calculated for each stub and multiplied by the ϕ (z) squared and summed. To ensure that tracks with higher numbers of stubs do not have higher χ^2, the sum is normalised by degrees of freedom which is proportional to the number of stubs in the track. The sum is then digitised based on the bins given in table 4.1 which are optimised for downstream cuts on χ^2 quality.

The χ^2_{bend} is based on the bend of the stub within the p_{T} modules and how well-matched the bend in the module is versus the final calculated p_{T} of the track. It uses the stub bend, which is calculated in the p_{T} modules and sent from the DTC to the TF as part of the stub word. Corrections are made to account for the module tilt and sensor spacing. Again this is digitised in a way to optimise for downstream track quality cuts.

The multivariate analysis (MVA) track quality is a variable used to describe the quality of a track. This is planned to be filled by a BDT which is described in detail in section 4.4.2.

The hitpattern is a 7-bit variable with one bit for each detector layer up to a maximum of seven layers for a track. It is calculated by inserting a 1 in every place a stub is found when iterating through the maximum allowed number of stubs of a track. The hitpattern allows simple variables to be calculated such as number of stubs for a track (how many 1s in the hitpattern) or number of missing internal layers (how many 0s between the 1s in the hitpattern). As there are a total of eleven different layers and disks in the outer tracker a conversion is needed to understand exactly which modules the hitpattern corresponds to. This can be calculated based on the track η, i.e. a

low η track has stubs from the barrel layers so each of the seven hitpattern bits correspond to barrel layers. So far, no downstream users are using this level of complexity in their track quality calculations, but it is a possibility.

The other MVAs are currently unfilled but various possibilities of either displaced track quality or some electron-based track quality have been explored, giving better handles on downstream track quality for different types of tracks.

In general, the helix parameters are easy to calculate. The KF provides a track that is calculated with respect to its ϕ sector centre and η sector centre using the different coordinate system in the KF, described in section 4.2. This means there needs to be a conversion back to the global CMS coordinate system, but remaining in a local ϕ for the nonant.

- 1/R remains the same between the KF word and track word: it is the radius of curvature of the track, which is used throughout the TF algorithm.
- ϕ_0 is the azimuthal angle of the track w.r.t the nonant centre: it is calculated from the KF ϕ_T by subtracting a factor based on the curvature of the track.
- $\tan(\lambda)$ is calculated from the KF $\cot(\theta)$ with the addition of a factor based on which η sector the track was calculated in.
- z_0 is calculated from the KF z_T subtracting a factor based on the $\cot(\theta)$ of the track.
- d_0 is a placeholder variable that will be filled by the track's displacement from the beamline for extended tracking.

The track helix parameters are digitised across the ranges given in table 4.1 which represent the full ranges of the parameters, i.e. tracks will always be reconstructed within these ranges because of the p_{T}, η and z_0 constraints of the track finding. The granularity is simply the digitisation range divided by the total number of bits given for each variable, which are calculated to ensure the overall reconstruction precision of the track parameter is larger than the granularity.

4.3 Track Finder Firmware Implementation

The firmware implementation of the TF algorithm is divided into two stages; the first is the tracklet firmware implemented using a mixture of Vivado HLS and VHDL, the second is the KF implemented directly in VHDL. The tracklet firmware consists of processing blocks and memory blocks for storing the calculated tracklets. The KF consists of an input formatting stage, the KF processor itself and then the KF output step. There are a variable number of KF workers that process tracks in parallel depending on the number of input tracks from the tracklet stage. This is necessary to ensure that the throughput of tracks is high enough to meet the overall algorithm latency requirement. The tracklet and KF firmware implementations are ongoing projects and are not discussed in detail.

The KF output firmware is made up of several distinct steps, as shown in figure 4.6.

1. Track transformation,
2. Track routing in η,

Table 4.1: 96-bit track word content divided into track quality handles and track helix parameters. Priority is given to the track helix parameters, so they have large numbers of bits with full digitisation ranges across their possible outputs. The granularity is always smaller than the resolution of the track parameters to avoid loss of information. The rest of the available track word bits are split between quality variables. These variables are used for cuts downstream so do not need fully digitised ranges, and there would not be enough bits to give useful quality cuts so binned versions are given, optimised for downstream quality cuts.

Track Parameter [unit]	Number of Bits	Granularity	Approximate Digitisation Range
Valid	1		
Track Quality			
χ^2_{rz} / dof	4	{0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 6, 8, 10, 20, 50, inf}	
$\chi^2_{r\phi}$ / dof	4	{0, 1, 1.5, 2, 2.5, 3, 3.5, 4, 5, 6, 10, 15, 20, 35, 60, 200, inf}	
χ^2_{bend} / dof	3	{0, 0.75, 1.0, 1.5, 2.25, 3.5, 5.0, 20.0, inf}	
MVA Track Quality	3	{0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0}	
Hit Pattern	7	{To Be Determined}	
Other MVAs	6	{To Be Determined}	
Track Helix Parameters			
$1/R$ [1/cm]	15	2.602e-07	−0.0085/0.0085
ϕ_0 [rad]	12	0.00034	−0.785 /0.785
$\tan(\lambda)$	16	0.00024	−8.0/8.0
z_0 [cm]	12	0.00999	−20.469/20.469
d_0 [cm]	13	0.00038	−15.4/15.4

3. Track quality (section 4.4.3),

4. Track buffering,

5. Link formatting.

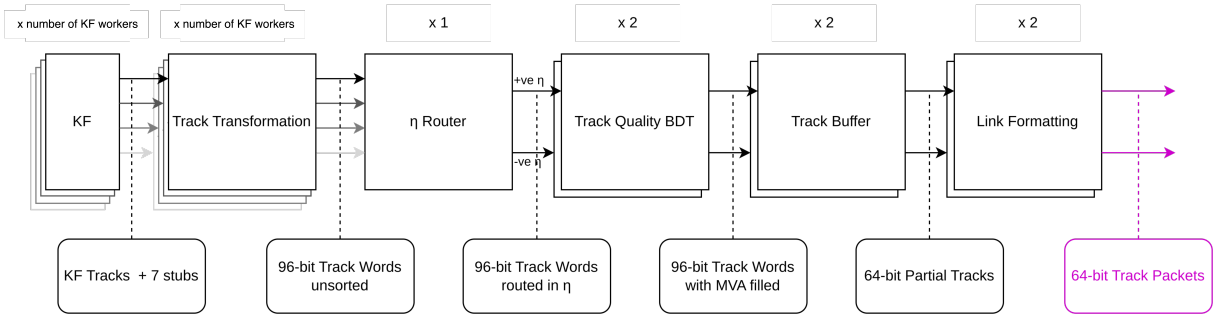


Figure 4.6: Flow diagram of the KF output module. The individual processing steps are shown with their expected parallelism above and the output of each module into the next module below. The order of modules was chosen to ensure the correct information is available at every step and there are no unnecessary replications of firmware. For example, the η router needs global η that is only calculated in the track transformation. While the track quality could be placed before the router this would lead to extra replications of the module if the number of KF workers increases, with no improvement in latency.

4.3.1 Track Transformation

The track transformation takes KF tracks and outputs the 96-bit track words described above in section 4.2.1. All the calculations of the track parameters are performed in parallel to save latency, with buffers of each variable used to store finished parameters as others are continuing their calculations.

- 1/R is taken directly from the KF word so is stored until the other calculations are complete.
- $\tan(\lambda)$ is calculated by taking the $\cot(\theta)$ from the KF and adding a constant stored in a LUT, selected based on the η sector of the track. This is then re-scaled to match the width of the $\tan(\lambda)$ in the track word.
- z_0 needs to be both re-scaled and shifted based on $\cot(\theta)$. The calculation begins by multiplying $\cot(\theta)$ by a fixed constant depending on the geometry and the KF z_T by a re-scaling factor. Both of these calculations use DSPs which are buffered on the input and output. The outputs are combined and bit-shifted to give a final z_0 which is correctly digitised for the output.
- ϕ_0 is calculated in a similar manner to z_0 but using 1/R as the shifting factor. There is also a sector correction based on if the track has originated from the positive or negative KF ϕ sector. Again these factors are scaled using DSPs and then the output bit-shifted to give the final ϕ_0 for the track word.

The calculation of the $\chi^2_{r\phi}$ (χ^2_{rz}) variables is more complex. It begins by iterating through stubs and calculating stub ϕ (z) squared. This is performed by using $4x^2 + 4x + 1$ (the expansion of $(x + \frac{1}{2})^2$ multiplied by 4 to remain in integer mathematics) and then later bit-shifting to act as a division by 4, ensuring a repeatable rounding occurs. As the simulation v0 (v1) weightings depends on information that is not accessible from the stub words in firmware, a simplification is made. The v0 (v1) weightings are correlated with $\Delta\phi$ (Δz) of the stubs, so a LUT is created giving an approximate v0 (v1) for each of the $\Delta\phi$ (Δz) across their full ranges giving a 2^9 (2^{13}) entry LUT. Once these weightings are found they are multiplied by the ϕ^2 (z^2) and stored in an array. This array is summed over and then the corresponding $\chi^2_{r\phi}$ (χ^2_{rz}) bin is found between 0 and 15 using a LUT. This calculation takes 7 clock cycles making it the highest latency step in the track transformation.

The hitpattern is simply created from the stubs, 1 where a stub is found, else 0. The χ^2_{bend} is currently not implemented in firmware as the stub bend information is not passed downstream of the DTC. It also needs the p_{T} of the track to correctly calculate the χ^2 but p_{T} is not used anywhere in the TF algorithm and requires a conversion. This poses a problem for downstream users as the χ^2_{bend} is an important handle on track quality; an approximation similar to what is used for the $\chi^2_{r\phi}$ and χ^2_{rz} will need to be made. However, as is described below, a track quality boosted decision tree can give better discrimination between real and fake tracks making the use of χ^2_{bend} less essential. d_0 is also not implemented but will be once the displaced tracking is completed. Additionally, in this transformation step the η sign is calculated based on the η region of the track and stored as the track's sort key for routing.

4.3.2 η Routing

One of the requirements of the TF is that the tracks are passed downstream in two links; one for positive η and one for negative η. This is needed by the correlator layers performing PUPPI which is regionalised in η. As the correlator is already resource constrained, it is simpler to route tracks earlier in the track finder itself. The routing needs to be flexible to allow any number of inputs (as there are any number of KF workers) leading to two outputs,

but the most likely number of KF workers and inputs will also be two.

The routing uses sets of RAM buffers for storing the tracks and two parallel processes; one deals with inputting tracks into the RAMs and one deals with reading them out. For each input stream there are two first in first out (FIFO) RAM buffers, one for +ve η and one for $-$ve η. In the case of two input streams this results in four separate FIFOs. In each clock cycle the tracks on the input streams are read and their sort key extracted. The tracks are sent to either the +ve or $-$ve FIFO for that input stream with a write address used to keep track of how full the FIFOs are. On the output tracks are read out of the FIFOs with two parallel processes, one controls +ve η and one controls $-$ve η. The +ve η process iterates through the +ve FIFOs in a circular fashion, reading out a track if it finds one, skipping to the next FIFO if it does not. Read addresses are used to keep track of where the process is as it checks through the FIFOs. This process occurs until all tracks are read out and resumes again when the next event's set of tracks start filling the FIFOs again. A similar process is run for $-$ve η.

The track quality is performed after this routing, the firmware implementation of this is described in section 4.4.3. Its input is two streams of 96-bit track words as is its output meaning it can be updated or changed without impacting the wider KF output firmware. This allows the BDT model to be updated without the rest of the firmware being impacted, a useful feature for an ML firmware implementation.

4.3.3 Track Buffering

The track word is 96-bits long but a data packet being transferred from the TF is a 64-bit packet per clock cycle, so the tracks must be split. Data packets are 64-bits long because of the specification of the link protocol between trigger boards, the link protocol is constant across all trigger subsystems. Again, tracks are stored in a buffer as bit vectors, 96-bits long and outputted as per table 4.2.

Table 4.2: 96-bit track word to 64-bit packets splitting scheme. In a three clock cycle loop different track word bits are read out. A flag in the start of the event tells downstream users which clock cycle in the loop they are currently on and which track word bits they are currently reading.

Clock cycle 1	64-bits = track 1 63–0	FW state 1
Clock cycle 2	first 32 bits = track 1 bits 96–64	FW state 2
Clock cycle 2	second 32 bits = track 2 bits 31–0	FW state 2
Clock cycle 3	64-bits = track 2 bits 96–32	FW state 3
Clock cycle 4	64-bits = track 3 bits 63–0	FW state 1
...	(repeating)	

The table shows how two tracks are fully transmitted every three clock cycles. The firmware simply keeps track of which of the three states it is currently in and reads the track buffer appropriately. Reset flags based on the event start are used to reset the state.

The KF output firmware has been tested in emulation, simulation of firmware and hardware testing. These tests are detailed in chapter 6. Other tests including sending the output tracks over fibre optics to other boards and processing algorithms are also described. The overall resource and latencies of each step are listed below; the most resource intensive section is the track transformation because of the multiple parallel calculations it needs to perform. The track quality has the highest latency due to the depth of the BDT; every tree split requires multiple calculations. Overall, the KF output runs in under 80 ns at 360 MHz and makes up a very small fraction of the

VU7P it was implemented for. As the final project is targeting a VU13P, these resource usage percentages will drop significantly, making this KF output module a very lightweight step of the TF algorithm chain.

Table 4.3: Resource usage and latencies of a Xilinx VU7P running at 360 MHz for the KF output implementations assuming two KF workers. These resource usages are estimates from a Vivado synthesis of the networks and the latencies are from a C-Simulation of the logic using ModelSim. If more KF workers are necessary the track transformation module resources would scale linearly with the number of workers and the track routing would also increase, with the BRAM usage scaling linearly. The other modules are constant with the number of KF workers as is the latency for all modules.

Module	Latency (clock cycles)	II (clock cycles)	LUTs%	DSPs%	BRAMs%	FFs%
Track Transformation	8	1	1.10	1.32	0.00	0.13
Track Routing	4	1	0.04	0.00	0.20	0.26
Track Quality	13	1	0.25	0.00	0.00	0.19
Track Buffering and Link Formatting	4	1	0.17	0.00	0.00	0.32
Totals	29	1	1.56	1.32	0.20	0.90

The full TF firmware has been implemented in various substages in order to test the viability of the overall algorithm. The first is the "skinny chain" which sees a single tracklet seed (L1L2) used and only one η direction implemented along with a single KF worker. This chain has been fully tested in emulation, firmware simulation and hardware running, as detailed in section 6. Other subchains include the barrel-only chain that only uses barrel layers which has only been implemented for the first half (tracklet) part of the chain but has been successfully run in firmware simulation. The full project implementation has yet to be realised in firmware because of a number of issues related to the timing of the firmware; work is ongoing to solve these issues.

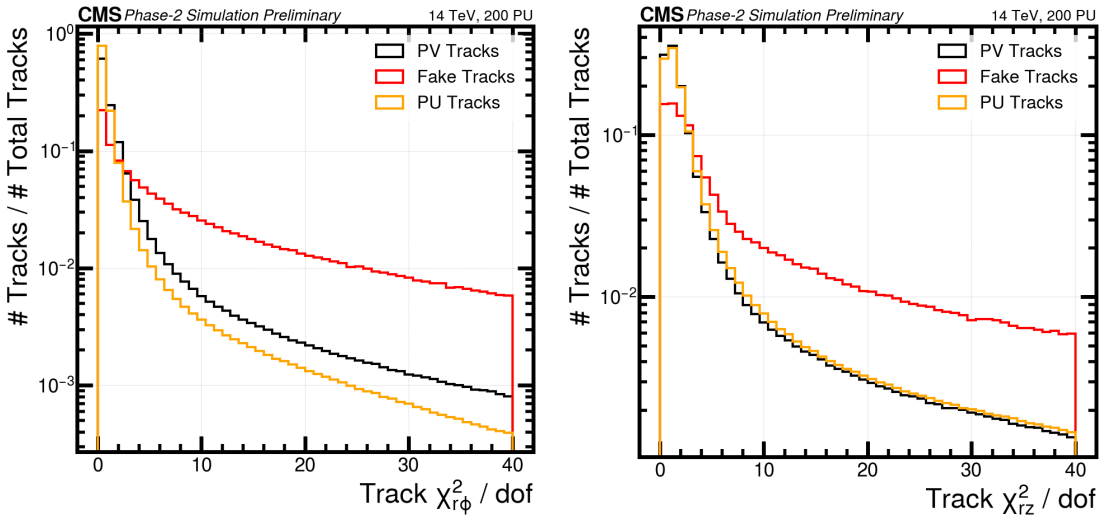
4.4 Track Quality

Algorithms downstream of the track finder rely on well-reconstructed tracks but in the trigger there is no way of knowing if a track is an accurate reconstruction of a true particle passing through the detector or even if the reconstruction of a track corresponds to a real particle. For the following discussion on poorly reconstructed tracks, the emulated KF is used as the input. This was selected as there are no internal track cuts made by the algorithm, so the role of track quality is more important as it needs to deal with all range of fake tracks being produced by the KF. It also allows for full comparison to the track quality firmware using the KF firmware as input.

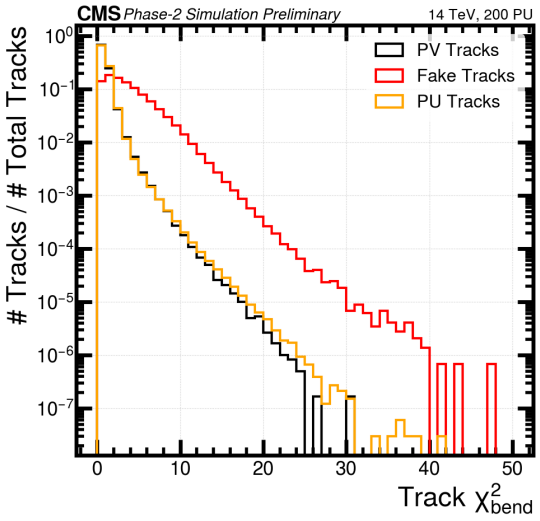
4.4.1 Fake Tracks

An inevitable part of any track reconstruction in a high multiplicity environment such as the CMS tracker is the selection of hits and reconstruction of a track that does not correspond to the hits left by a real particle. This can occur when many particles overlap, leaving tens of thousands of stubs in the tracker. Typically, these fake reconstructed tracks will have a lower fit quality than a real track which is understandable as the algorithm has had

to fit a line to a set of points which do not necessarily lie on the expected helix of a charged particle. This lower quality can be observed in figures 4.7a–4.7c where larger values of all the χ^2 fit variables can be seen for fake tracks in red. The χ^2 quality of pileup (PU) tracks, in orange, is roughly the same as primary vertex (PV) tracks which demonstrates a low correlation of p_T to the χ^2 variables, a desirable feature in the track algorithm. A small variation is seen in the $\chi^2_{r\phi}$ variable between PV and PU tracks due to the lower curvature of higher p_T tracks in the r-ϕ plane where particles are perpendicular to the CMS magnetic field.



(a) (b)



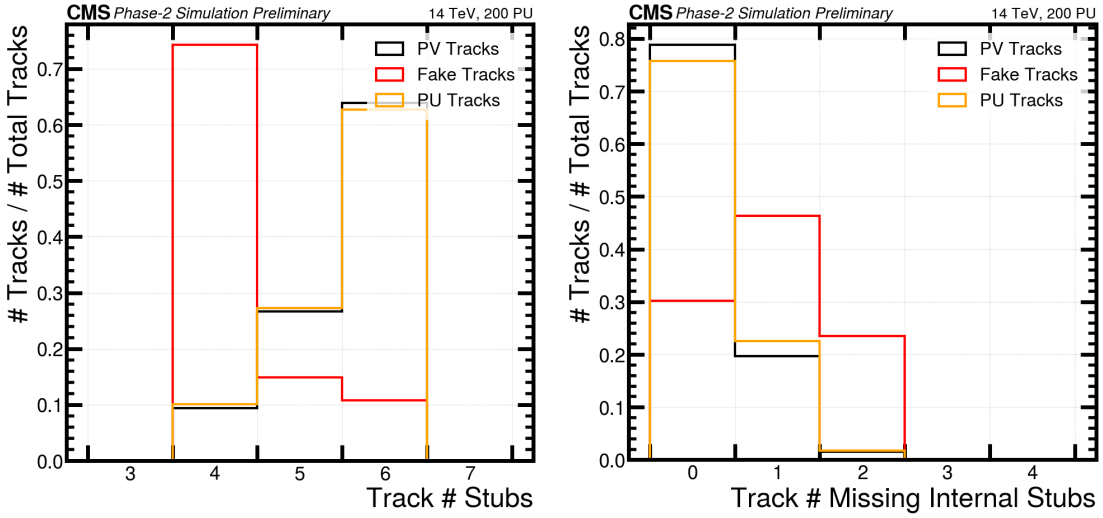
(c)

Figure 4.7: Track χ^2 distributions split into tracks originating from a true particle associated to the primary vertex in the event (black), tracks originating from true pileup particles (orange) and those reconstructed without matching to a true particle (red). Relative numbers of tracks are shown to highlight the difference between the track types, in reality the true number of fake tracks is around 20% that of the real tracks and the PU tracks significantly outnumber the PV tracks. In all cases a higher proportion of fake tracks have higher values of χ^2s showing a worse quality in these tracks.

Another feature of the fake reconstructed tracks is the number of stubs used to reconstruct them. The

minimum required stubs for reconstruction is four which means tracks can be reconstructed without all the barrel or endcap layers, giving greater overall efficiency especially in the transition region where the complex geometry complicates the track reconstruction. However, the minimum of four stubs leads to a greater possibility of fake tracks being reconstructed where only four stubs are required to line up in a helix for the reconstruction algorithm to create the track. Tracks with higher number of stubs are more likely to be real as shown in figure 4.8a.

It is useful to inspect the stub number in a more detailed way by considering which stubs are missing. When considering the stub distribution in the tracker, the innermost layers have a higher density of stubs and each subsequent layer has a lower density as the particles fan out from the interaction point. This means that there are many compatible tracks which have a stub close to the beamline in layer one and then subsequent stubs in outer layers which can lead to many combinations of stubs that form tracks which miss internal layers. These tracks with missing internal stubs are more likely to be fake, as shown by figure 4.8b.



(a) (b)

Figure 4.8: Track stub number distributions split into tracks originating from a true particle associated to the primary vertex in the event (black), tracks originating from true pileup particles (orange) and those reconstructed without matching to a true particle (red). Figure (a) is the total number of stubs used to create the track and figure (b) is the number of layers skipped in the track reconstruction. It is very clear that fake tracks are typically reconstructed with fewer stubs and as a result more stubs are missing from internal layers.

Finally, there are certain geometric features of fake tracks that arise because of the shape of the tracker and the transition from barrel layers to endcap layers. Firstly, considering just PV and PU tracks, we expect a flat distribution in ϕ, as particles are produced isotropically from proton collisions. This is seen in figure 4.9a in the flat black and orange lines.

An η dependence is seen in PU and PV tracks. PV tracks are more central because of the underlying event topology of the non-boosted $t\bar{t}$ sample where more central tracks are produced in the hard scatter. High momentum transfer occurs and a large deflection of the protons leads to energy being transferred outwards from the collision. The PU tracks originate from softer elastic collisions of protons where a small deflection and little momentum transfer occurs between the protons leading to lower energy particles continuing in the direction of the protons at higher η. This will change for different underlying events such as VBF processes where a higher η skew

on the primary vertex tracks would be expected.

As the tracks in the detector are produced by two overlapping proton bunches each with a Gaussian shape, we expect the z_0 distribution of tracks to be Gaussian in both PV and PU tracks, seen in figure 4.9c.

Finally, the p_T distribution of tracks will follow a complex distribution depending on the underlying event and additional pileup. We expect the hard scatter primary vertex tracks to be of a higher p_T than the pileup, but both will peak at the lowest reconstructable track p_T of 2 GeV and tail-off rapidly. We expect this as in reality there are far more tracks with a $p_T < 2$ GeV with the distribution rising from O 10 MeV to an O 100 MeV peak and then falling. The track finder captures the high (> 2 GeV) tail of this distribution. This is seen in figure 4.9d with a far lower p_T distribution in PU tracks.

The fake tracks follow a different distribution due to the geometry of the tracker and features of the track reconstruction. In ϕ, the fake tracks follow a pattern which oscillates, which is due to the nonant structure of the track finding algorithm where there are cross-over regions and ϕ dependent processing. There is also duplication of stubs on boundaries which means that these additional stubs can be combined in erroneous ways if the real track is in one region but some of its stubs are duplicated in another.

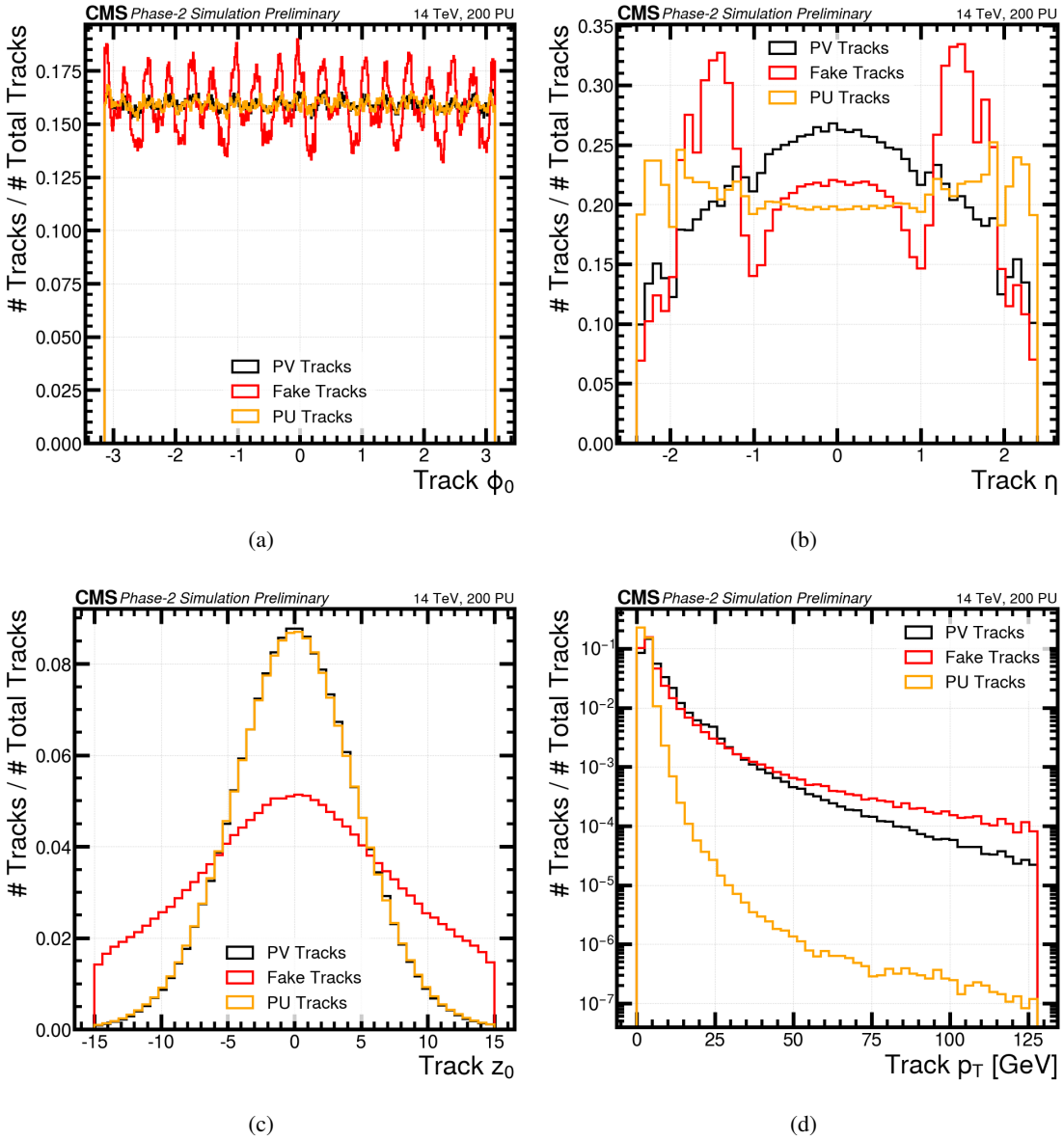
There is a large difference in η due to the transition region where stubs in both the barrel and endcaps can lead to poorly reconstructed tracks. This leads to the peaks in fake tracks at $|\eta| \approx 1.5$.

The z_0 distribution of fakes sees a far higher number of fakes at high $|z_0|$ because of their non-physical origin. When considering both η and z_0 together in figure 4.10a, it can be seen that these high z_0 tracks are the same fake tracks creating the peaks at $|\eta| = 1.5$. From this it can be seen that the track finding algorithm can fit these transition region fake tracks more easily with higher z_0 leading to a wider distribution in z_0 for fake tracks.

The p_T distribution shows that fake tracks are of a higher p_T than PU tracks and follow a similar distribution to the PV tracks. This follows from the combination of stubs to generate fake tracks. For low p_T, tracks have higher curvature and the stubs in the detector layers will follow this curved line, it is less likely that random combinations of stubs will form a curved line in the detector leading to a low p_T fake. A higher p_T tracks will be a straighter line in the detector layers making it more likely that a random selection of stubs will fall on this line, especially when internal stubs are missed out, as is the case with fake tracks.

The baseline method for dealing with fake tracks is to impose quality cuts on various parameters. This is performed downstream of the track finder with these cuts being tuned for a given application. For example, the track-based E_T^{miss} calculation, as described in section 5.4, is a key example of this where the presence of a fake track can make a large difference to the final calculation. Table 4.4 outlines the cuts used for E_T^{miss}.

While these cuts are effective at removing fakes, shown in figure 4.15b, there is an impact on overall track finding efficiency with a 10% drop. The true tracks that have been removed are of a lower quality so have worse parameter resolutions compared to well reconstructed, low χ^2, tracks. This method of simple cuts is easy to understand and interpret with only three variables being used. These cuts do need re-tuning for different versions of the track finding algorithm but a single working point of x% fake rate or x% track finding efficiency can be set, and the cuts changed to meet this working point. Another advantage of a χ^2 cut is the low firmware usage and latency, all that is required is a value comparison which is a one clock cycle operation.

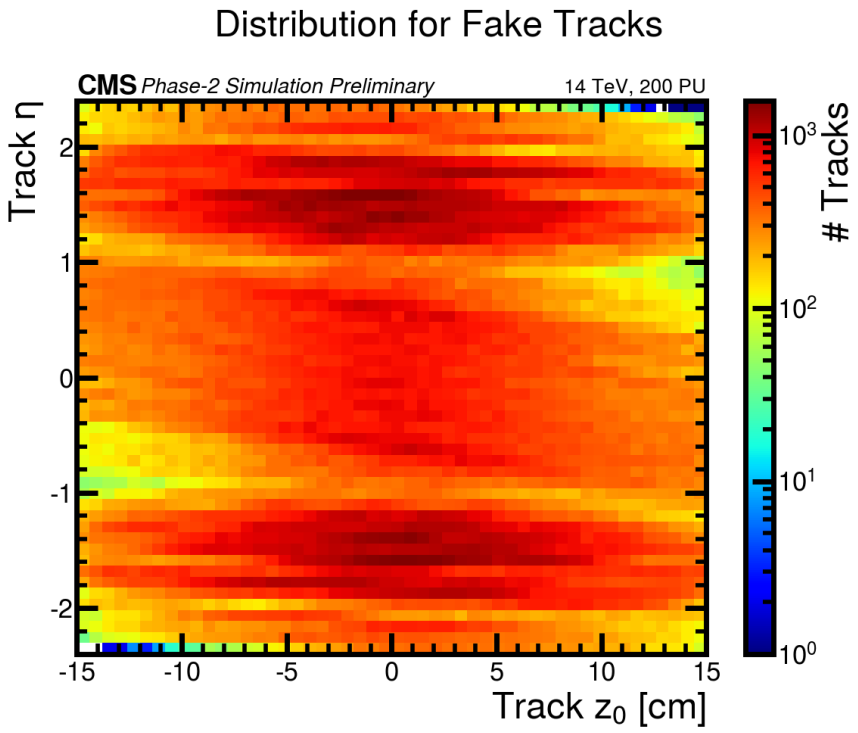


(a) (b)

(c) (d)

Figure 4.9: Track helix parameter distributions split into tracks originating from a true particle associated to the primary vertex in the event (black), tracks originating from true pileup particles (orange) and those reconstructed without matching to a true particle (red). In figure (a) track ϕ_0 with a finer binning, used to show the underlying oscillation in the fake track distribution. In figure (b) the track η shows peaks in the fake track distribution due to the transition region. Figure (c) shows track z_0 with more fake tracks at higher z_0 because of erroneous reconstruction of transition region tracks. Finally, figure (d) shows track p_T where fake tracks follow a similar curve to the PV tracks due to the track reconstruction fitting straight lines of mismatched stubs more easily.

This cut-based approach does not take into account the more complex relationship between track parameters and the presence of fakes. As has been clearly shown, the geometry of the tracker causes a higher number of fakes in specific regions. Simple cuts cannot be used to remove these specific regions e.g. with an η cut, as this would drop track finding efficiency entirely in these regions so a more complex approach for using this geometric information is needed. It can also be seen from figure 4.15a that a bias in certain η regions is introduced by these cuts where there is a significant drop in efficiency in specific η region, which is due to the apparently worse quality tracks being reconstructed.



(a)

Figure 4.10: Fake track z_0-η distribution. In the barrel region, $1.0 < \eta \leq 1$, there is a negative linear dependence between the two track parameters which is due to the higher probability of fitting a track with random stubs if the track originates at a high (low) z_0 and points back towards −ve (+ve) η. The peaks at $|\eta| = 1.5$ due to the transition region fitting have wide distributions in z_0 leading to the wider distribution of z_0 fake tracks seen in figure 4.9c.

Table 4.4: Track quality parameter thresholds for track-based $E_{\mathrm{T}}^{\mathrm{miss}}$. Other cuts on the tracks are enforced by the track algorithm itself, e.g. $|z_0| \leq 15$ cm, $|\eta| \leq 2.4$, and number of stubs ≥ 4.

Parameter	Threshold
χ^2_{bend}	≤ 4
χ^2_{rz} / dof	≤ 9
$\chi^2_{r\phi}$ / dof	≤ 12

4.4.2 Boosted Decision Trees for Track Quality

While it would be possible to manually search for optimal cuts that find regions in track parameter space where fake tracks are more likely by looking at different parameter distributions, it is simpler to automate the approach to finding and optimal set of cuts with an ML approach. The advantage of such an approach is the ability of an ML classifier to search n-dimensional parameter spaces for an optimal decision boundary for a certain classification task without external input. However, it is important to correctly frame the parameter space for the ML model to avoid bias and to ensure useful features are being used. This approach has the added advantage of being easy to optimise on different data, e.g. if the track finding algorithm is updated a large dataset can be created and used to

train the ML model. Another advantage is the ability to pass downstream a flexible score which algorithms can cut on based on how much tracking efficiency is optimal to lose versus how many fakes remain.

An ML approach does have disadvantages; the main one being a lack of interpretability, when using complex ML systems such as multiple hidden layer NNs, a lack of understanding is inherent in the model. While it is possible to view correlations between a model output and the input features this can only give a general trend of the data. This issue ties into the problem of bias in ML models where there could be a feature within the data that the model is picking up on, biasing it to certain types of inputs. However, it is still very easy to bias simple cuts as well, for example, a cut on $\chi^2_{r\phi}$ will have a bias against PV tracks as they typically have larger $\chi^2_{r\phi}$ and so it can be advantageous to use ML models with multiple variables to avoid this kind of simple bias.

In order to train an effective classifier, the features require careful consideration. Having many features leads to a more complex, larger model which is unsuitable for the low latency, low resource application of the L1 trigger. Having many correlated features that do not impact model performance despite having high individual classification power is unnecessary.

To design the features used for a track quality ML model, the parameters in the track word were considered using the analysis detailed above for each track variable. Firstly, the χ^2 variables have clear discriminatory power for real versus fake tracks. Also, the number of stubs and number of missing internal layers is useful, which again have a clear dependence on real and fake. For the track helix parameter ϕ_0, there is only a small discrimination in narrow bands of ϕ_0 which could lead to an uneven loss of tracking efficiency in ϕ_0 which is very undesirable for downstream algorithms, so it is not used. Track p_T is also not used as a feature because of the lack of discrimination between PV tracks and fake tracks; while apparent tracking efficiency might remain high when cutting on a p_T trained classifier, as efficiency is measured including PU, the tracking efficiency for PV tracks would drop posing large issues downstream where PV tracks are the most desirable. Track z_0 is used with the high discrimination power at high z_0, but the absolute value is used to avoid a bias to one half of the detector. Finally, instead of η the $\tan(\lambda)$ variable is used as it is available in hardware; figure 4.11 shows that there are still clear regions in $\tan(\lambda)$ where more fakes are expected, meaning it is still a useful variable for identifying fake tracks. Again, absolute $\tan(\lambda)$ is used to avoid asymmetry in the output of the classifier.

A GBDT was selected as the ML algorithm because of its simplicity. It has been shown previously that the GBDT approach is successful at identifying fakes based on track parameters [105]. The features being supplied to the algorithm are already high-level features with a large degree of discrimination, so the aim of the ML is to find an optimal decision boundary in a large parameter space, making a decision tree a good solution. A more complex solution, such as an NN, would be useful for determining a complex function used to discriminate fakes from low-level track information. While this could give a high level of discrimination, the firmware cost of an NN versus a BDT is high and in the track finder, where resources and latency are very tight, a BDT is more suitable. In order to keep firmware usage down, a very small BDT was designed; 60 estimators with a depth of 3 giving a very shallow BDT, but because of the complex high-level track parameters being used as input, a large depth is not necessary.

It is common in classifier-based ML to balance datasets by the target variable, i.e. to have the same number of real tracks as fake tracks in the training dataset in order to reduce bias in the training towards the class that is seen most frequently. However, when looking at the real track distribution a further complication is introduced when considering the particle type the track originates from. For the events used for training, inclusive $t\bar{t}$, the majority of tracks are kaons and pions ($\sim 70\%$). This can be understood from the decay of the top into b-quarks and light

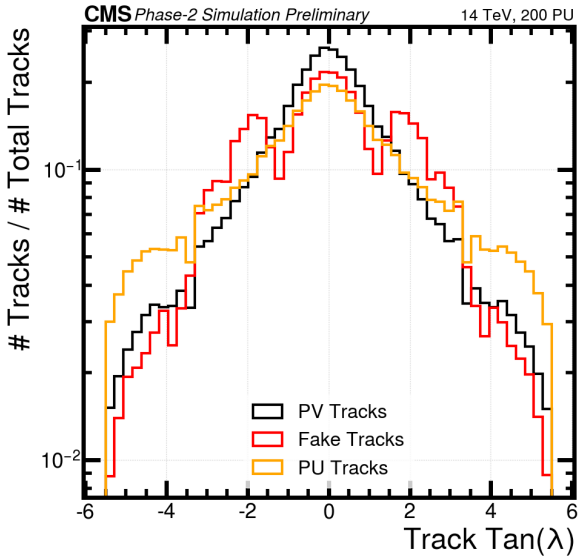


Figure 4.11: Track tan(λ) distribution split into tracks originating from a true particle associated to the primary vertex in the event (black), tracks originating from true pileup particles (orange) and those reconstructed without matching to a true particle (red). There are still clear regions in tan(λ) that show higher rates of fake tracks, but using the tan(λ) variable can obscure exactly why these regions have higher fake rates. This highlights an underlying issue in having to use different variables in firmware; often the physical meaning of the variable is lost, making understanding bugs in firmware difficult.

quarks which are then hadronised leading to jets made up of light hadrons.

Electrons and muons are also produced from electroweak decays of the top quark and its decay products. These make up < 10% of the tracks created; fake tracks make up a similar proportion. It is also known that, in general, electron tracks are poorly reconstructed compared to muon tracks due to the effect of bremsstrahlung, where the electron radiates a photon as it accelerates in the CMS magnetic field, causing the electron to lose energy. At the GeV scale energies in the track reconstruction, bremsstrahlung dominates the electron energy losses [104] leading to a deviation from a perfect helix. Another issue with electron tracks is multiple scattering where the electron will change path as it deflects off material in the tracker layers. This is less of a problem with the higher mass muons and hadrons. These effects result in a worsened track resolution for electrons meaning that to a naive cut or poorly trained ML model they appear like fake tracks. A bias against electron tracks is undesirable, so instead of simply balancing between real and fake tracks, an equal number of electrons, muons, hadrons, and fakes are selected and used for training. This is achieved by up-sampling the muons, down-sampling the hadrons and using all the electrons and fakes in training.

Performance

The BDT performs well at separating real from fake tracks as is seen in figure 4.12. There is a clear red peak at low BDT output scores with a higher peak for genuine tracks at high BDT scores. There is a slight bias against PV tracks versus PU tracks shown in the small difference in the black and orange lines at low BDT scores. This is due to larger overlap between PV tracks and fake tracks in the chosen track parameters.

When comparing the BDT classifier to simple χ^2 cuts (those in table 4.4) as in figure 4.13, the BDT outperforms the simple cuts on χ^2 as it is taking into account more geometrical features of the tracks, as well as

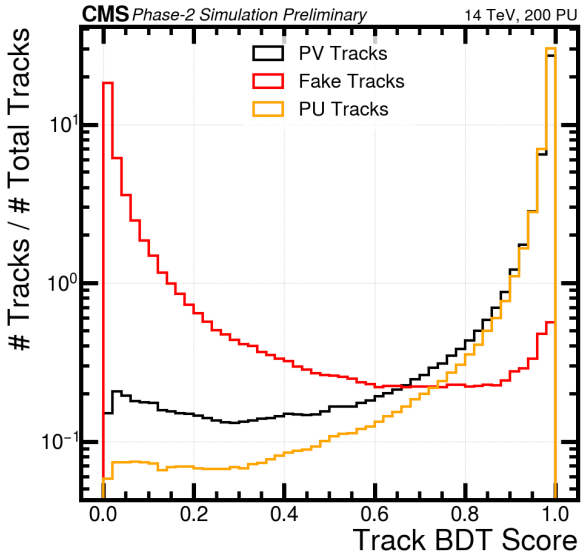


Figure 4.12: Track quality BDT output distribution split into tracks originating from a true particle associated to the primary vertex in the event (black), tracks originating from true pileup particles (orange) and those reconstructed without matching to a true particle (red). There is a clear differentiation between real and fake tracks in the BDT output meaning the BDT has learnt the task effectively. There are more PV tracks with low BDT scores than PU tracks which is due to overlap between PV and fake in the variables chosen for training, for example $\chi^2_{r\phi}$.

optimising the χ^2 cuts for different detector regions. With a drop in track finding efficiency of 1% the fake rate can be halved by the BDT with a threshold of 0.2. After this point the obvious fakes are removed and the track finding efficiency drops rapidly with a modest change to the overall fake rate, where obvious fakes are those that are clearly missing internal stubs or are at a very high χ^2. The BDT can achieve a 6% improvement on track finding efficiency at the same 1.5% fake rate as the χ^2 cuts. The other advantage of the BDT is the range of outputs it provides, different algorithms can choose working points across the threshold range rather than having to optimise their own χ^2 cuts for different use cases.

When comparing the performance of the BDT split into particle types in figure 4.14, it is clear that the BDT approach still outperforms the χ^2 cuts. In general, electron performance is poor; the maximum track finding efficiency achieved is 78% with the χ^2 cuts dropping this to 60%. The track finding efficiency after a cut on the output of the BDT drops quickly with threshold, as compared to the other particle types where the BDT is typically flat in efficiency across low thresholds. Muon performance is very good because of the clean tracks left by muons in the detector. The hadron performance is also good with a slow decline in track finding efficiency as the threshold increases up to 0.5 where the drop-off increases, always outperforming the χ^2 cut.

Figures 4.15a to 4.15c demonstrate that the BDT has not created artificial geometrical bias in its output, i.e. the same flat efficiency is observed across η and p_T, just lower at the different thresholds. However, as can be seen in the red χ^2 line in figure 4.15a, the basic cuts produce a bias in certain η regions where the efficiency drops off by a large amount, especially noticeable at $\eta = \pm 2$. This is a knock on effect of relatively lower quality tracks being produced in the endcap region, something that basic cuts cannot take into account unless the η region is split up and new cuts are optimised for each region. There is no geometrical dependence on ϕ with any of the quality cuts.

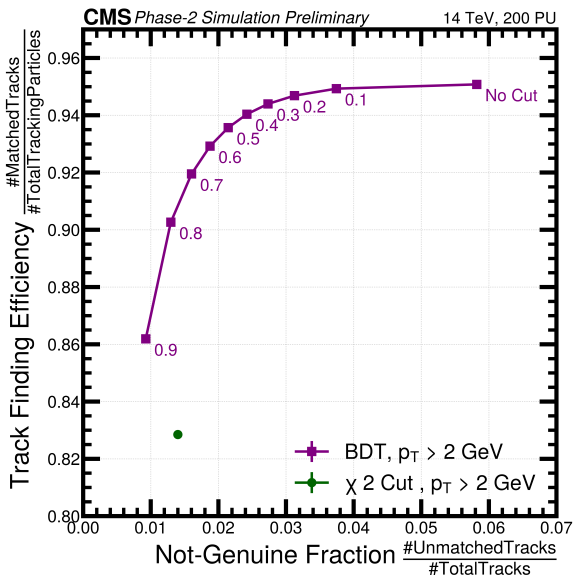


Figure 4.13: Track finding efficiency versus non-genuine fraction for the range of track quality BDT output scores. The points in purple show the effect of cutting on the BDT output score, and can be seen as moving a threshold across figure 4.12. No cut is the baseline track reconstruction with no change. As BDT threshold is increased the track finding efficiency barely changes, fewer than 0.5 % of real tracks have been removed at a threshold of 0.1. The fake rate drops as 11 % of fake tracks have been removed at this threshold. As threshold increases the number of real tracks being cut increases so the track finding efficiency drops but the fake rate drops more slowly as all the fakes have already been removed. The χ^2 cut of table 4.4 is shown in green with its low track finding efficiency.

The fake rate in 4.15c has clearly dropped and shows no strong p_T dependence. The BDT and χ^2 points are flatter than the no cut approach so no bias in p_T has been introduced by using these cuts, especially shown in figure 4.15b where the efficiency across p_T does not deviate from the no cut shape.

4.4.3 Firmware Implementation

The track quality BDT is implemented within the KF output firmware. Its place within the firmware chain after the routing step, shown in figure 4.6. This placement means that there are always only two replicas of the BDT per TF, no matter how many KF workers there are, meaning the resource usage is stable and small. The resource figures in table 4.3 will remain the same for the track quality for any TF subchain or full project. There is a small track transformation applied to the 96-bit track words for the BDT. Firstly, the number of stubs and number of internally missed layers is calculated directly from the hitpattern. Secondly, a scaling is performed on the z_0 and $\tan(\lambda)$ variables which move them to the arbitrary precision (AP) fixed-point format [106] of the BDT; a simple power of 2 division, implemented as a bit-shift. This is performed to match exactly the inputs the BDT was trained on and those it receives in firmware, essential for matching simulation, emulation, and firmware.

It is possible to vary the internal precision of the BDT. Larger precision ranges will be closer to the python 32-bit floating point precision but come at a resource cost. Using a C-simulation and varying the fixed precision width, an ap_fixed <11,6> [5] format was chosen. Figure 4.16a shows how the performance of the BDT changes with fixed-point precision selection. It is clear that the BDT needs at least 6 integer bits and at least 5 fractional bits for the ROC score not to drop. There is a corresponding drop in resources with this choice, LUT usage drops 30% when using <11,6> over <12,6> with only one fewer total bits, shown in figure 4.16b.

[5]This representation of fixed-point numbers is of the form <total bits, integer bits> where fractional bits = total bits − integer bits.

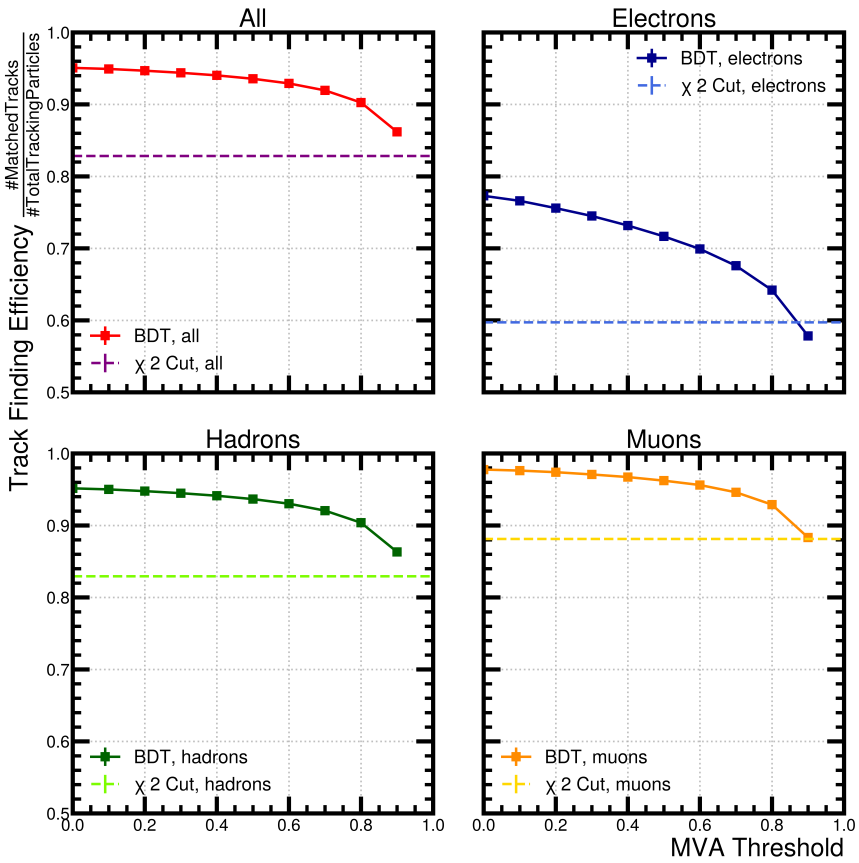


Figure 4.14: Track finding efficiency after quality cuts versus BDT cut threshold for different particle types for the BDT in the solid line and for the χ^2 cuts in the dashed line. A similar shape is seen across all four plots with a slow decline in track finding efficiency as threshold increases up to a threshold of 0.8 where there is a larger drop off. This decline in track finding efficiency is larger in electrons as electrons appear more fake than other particle types.

It is not possible to train a BDT in a quantised way with the fixed format already selected and used throughout the training. However, it was possible to provide it with the fixed-point variables for training, which prevents the BDT from learning thresholds outside its digitisation range and the only part that will be impacted by the final quantisation is the weighting of the individual trees. Another common way of reducing resource usage in NNs is to use pruning, iteratively removing close to zero weights to reduce overall network size. This is not possible in BDTs, but a different style of pruning can be used. The pruning takes into account the overall impact of single branches in a tree and if the impact on the loss function while training is below a certain threshold they are removed, allowing the BDT to ultimately be smaller after its training is complete. This pruning is controlled in training by the parameter γ and the impact on resource usage can clearly been seen in figures 4.16b and 4.16c with LUT and FF usage dropping between 1–2% for $\gamma = 100.0$, compared to $\gamma = 0.0$. However, this pruning can impact the performance of the BDT, with a clear drop off after $\gamma = 20.0$ —shown in figure 4.16a. After this point, the learning dynamics of the BDT are compromised by the high γ resulting in the variable performance between γs. While the pruning gives modest gains in resources compared to correctly selecting the fixed data format, this specific BDT is very small with a depth of three and 60 trees. For larger, deeper trees there will be more unnecessary splits, so the pruning could make a more significant impact.

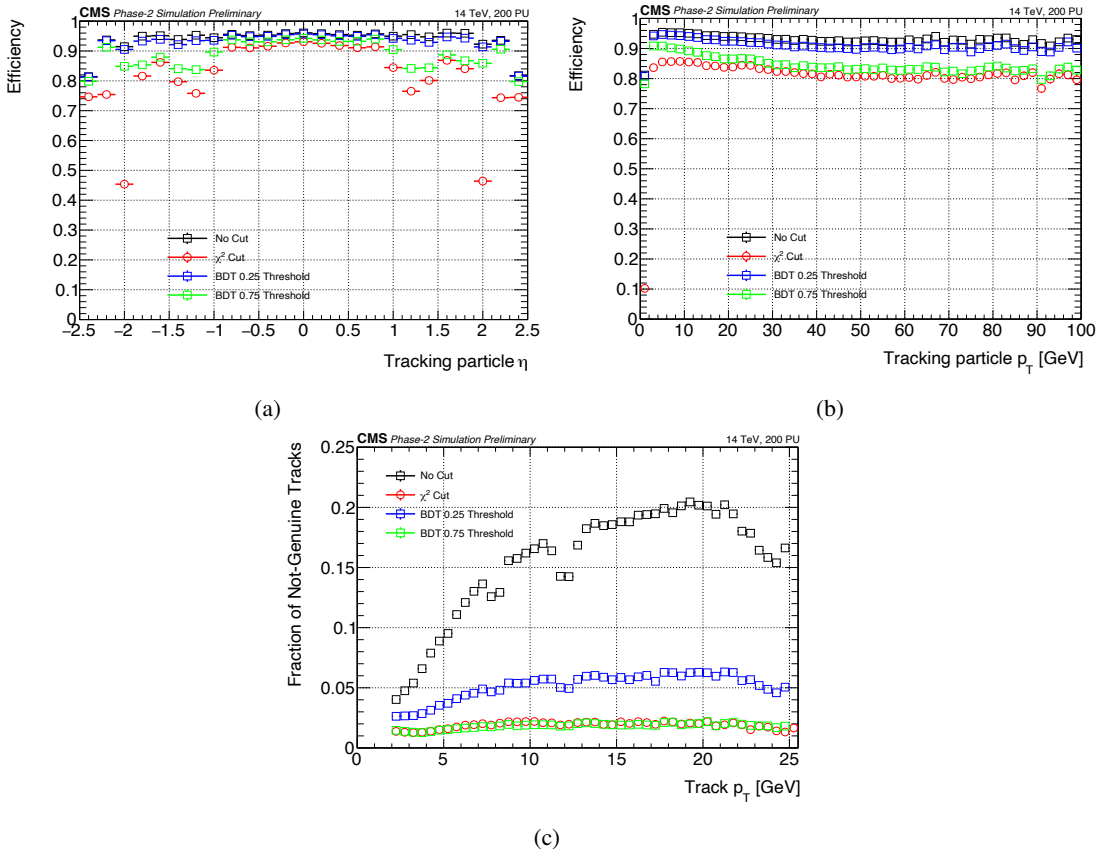


Figure 4.15: Level-1 track reconstruction parameters without cuts in black, after the χ^2 cuts in red and after two different BDT threshold cuts in green and blue. Figure (a) and (b) show track finding efficiency as a function of η and p_{T}. Figure (c) shows the rate of tracks not matched to generator particles as a function of p_{T}. From these plots it is clear that the BDT has less geometrical dependence on how fakes are removed than the χ^2 cuts highlighting the benefits of using more complex approaches to seemingly simple problems. The BDT offers a single threshold for downstream users making it easier to optimise cuts.

The BDTs themselves are implemented using the conifer package [98] in VHDL to simplify the interface with the rest of the KF output module. The firmware is designed to optimise latency. Every tree in the BDT is unrolled and every node comparison is performed in parallel. Each leaf of the tree is represented by a boolean which is used to create a set of leaf bits that accesses a LUT for the score. This means every possible true/false combination of the tree is stored, but as the trees are only three deep this is a small LUT. The score for each tree is then added to give a total score. The weighting each tree has in the total ensemble of trees can be rolled into the output score of each tree meaning that for any given set of inputs a weighted score for each tree is directly produced without having to perform the weighting multiplication in the firmware. This results in the entire BDT being implemented in FFs and LUTs, avoiding high latency DSP processes. The final score is binned according to the bins specified in table 4.1 and added as 3 bits to the track word. The latency and resource usage of the final BDT is shown in table 4.3.

In order to test this firmware, an emulation of the BDT was also written that takes a fixed-point version of the BDT created by the conifer package and gives it the same features as the firmware itself, with identical scaling carried out. This allowed the emulated and standard python versions of the BDT to be compared. From a comparison of emulation to simulation the correct scaling for each of the variables was tested along with selecting an ideal fixed-point representation for the model. Once the emulation and python were in agreement the emulation

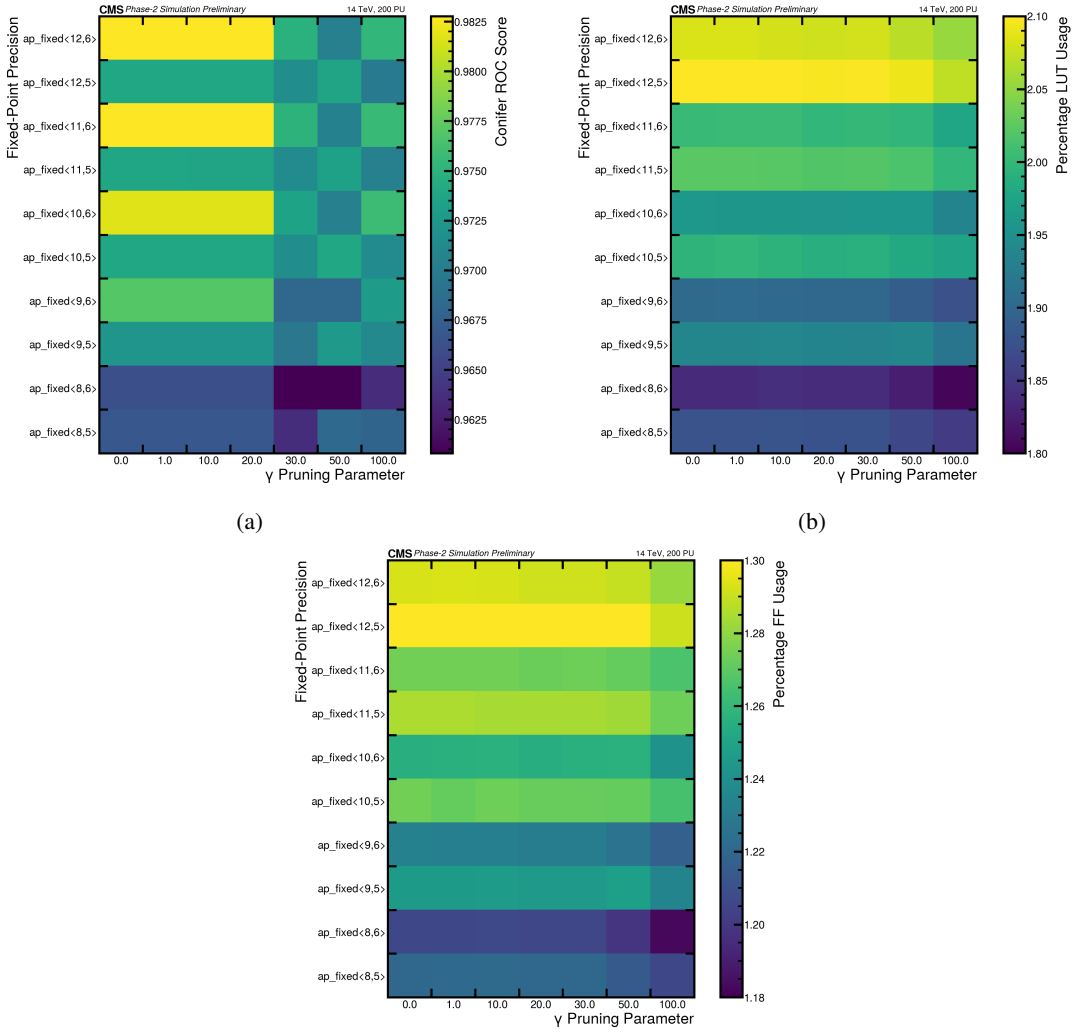


Figure 4.16: BDT performance and resource usage versus fixed-point precision and pruning parameter. Fixed-point precision is the format of numbers allowed for the parameters in the VHDL implementation of the BDT and the γ parameter is selected at training time. The ROC score in figure (a) is determined using a C-Simulation of the firmware and the resource usages are determined after a VHDL synthesis, with % LUTs of a VU7P chip in figure (b) and FFs in figure (c). As fixed-point prevision is decreased the ROC score drops suddenly showing that at least 6 bits are needed for the integer part and 5 bits are needed for the fractional part. The resource usage in figures (b) and (c) drop as smaller fixed-point numbers require less resources to be represented. As γ increases the resources decrease, shown in figures (b) and (c) as the trees have been pruned. This introduces poor learning in the GBDT shown in figure (a) with poor ROC scores after γ = 20.0.

was matched to firmware through a process of comparing the emulation to a C-simulation of the firmware. This process and then subsequent test in hardware of the BDT is described in chapter 6.

4.5 Conclusions

Track reconstruction is an essential part of the Phase-2 upgrade of the L1 trigger. Without it, trigger rates would be 40 times higher in the high pileup HL-LHC. Tracks will be used by the downstream vertex finding algorithm,

designed to find the primary vertex and associate tracks to it in order to veto pileup. They also play a major role in the correlator trigger which, with these tracks, can perform particle flow reconstruction of the event, a first for the CMS L1 trigger. Finally, the muon and e/γ subtriggers can use the tracks for isolation and e/γ separation, increasing their efficiency and improve their triggering performance.

The track finding algorithm is split into two stages; the first, a tracklet seeding stage, finds stubs in multiple seeding layers and projects tracks outwards from these seeds to create track candidates, matching other stubs to these tracks within a window. The second part, the Kalman filter, performs a track fit by iteratively adding stubs associated to the track to its KF state producing a final track with highly resolved parameters.

These tracks are transformed and routed to be sent to the rest of the L1 trigger by the KF output step that also runs a track quality BDT to provide a handle on fake tracks to the L1 trigger. This is the earliest usage of ML in the Level-1 trigger chain hence the simplicity of the model. These fake tracks are produced from collections of stubs that do not correspond to real particles in the detector and can negatively impact downstream users in a variety of ways. A BDT has been designed, trained, emulated and written in firmware to classify tracks as real and fake using a number of track parameters and track quality flags, selected because of their classification power. The track quality BDT outperforms basic cuts on the track χ^2 values when comparing track finding efficiency and overall fake rate, and provides a variable threshold for downstream users to select tracks based on their overall quality.

There are a number of improvements that could be made to the track finding, the most significant being displaced track finding which is under active development. Removing the beamspot constraint opens up many options to the Level-1 trigger such as Higgs decaying to long-lived light meson that can only be triggered on with displaced tracking due to the low energy of the meson. Displaced tracking could also improve electron performance, a major issue in the current track finding where the multiple scattering and bremsstrahlung cause the tracks to deviate from the expected helices. While more advanced approaches such as the Gaussian sum filter [107] that is currently used offline are impractical due to the requirement of having multiple parallel Kalman filters there are other possibilities for improving electron performance. One possibility is the extension of the track quality boosted decision tree into the Kalman filter updating step where a small BDT could identify electron tracks early and adapt the KF update accordingly. While this would increase the resource usage of the KF, it has the potential of dramatically improving the electron efficiency but also has the added advantage of flagging poor quality states early on in the KF processing and preventing unnecessary reconstruction of fake tracks.

As the Level-1 tracking is built around a flexible trigger architecture it could be possible in the future to update the track finding algorithm as improvements to firmware based algorithms improve. One possible area that is being explored is the use of graph neural networks for the initial track pattern recognition [108]. This has been shown to be a powerful technique for creating initial track candidates out of many hits in a detector but suffers from very high latency in FPGAs due to the graph building stage. If this could be improved, GNNs could be a useful tool for future track finding [92].

Chapter 5

The Global Track Trigger

The function of the global track trigger (GTT) is to perform global event-level reconstruction using tracks provided by the track finder [57]. The most important task, which is the primary vertex reconstruction, gives downstream algorithms the ability to veto pileup interactions through track and object to vertex association. This is especially important for the H_{T} and E_T^{miss} calculations performed in the correlator trigger, which are highly susceptible to background energy deposition. Additional GTT algorithms include track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ and jet finding, which can be used as cross-checks in the global trigger (GT), since they are robust against pileup. But they are not as accurate as their correlator trigger counterparts because of the neutral hadron and photon components to these calculations.

There is also opportunity in the GTT for novel light resonance reconstruction from tracks. One example of such a resonance is the $B_s^0 \rightarrow \phi\phi \rightarrow KKKK$ decay which searches for pairs of kaon tracks originating from the same vertex within a certain momentum distribution. These are used to reconstruct the ϕ mesons which in turn reconstruct the B_s^0. There are similar channels that can be targetted using only tracks giving CMS the opportunity to trigger on these channels in the L1 trigger for the first time.

5.1 System Design

The GTT receives track inputs from the track finder split into 18 input streams, two from each detector nonant split by positive and negative η. These tracks are packaged into 96-bit track words as specified in section 4.2.1. In a PU 200 scenario the expected number of tracks, $p_{\mathrm{T}} > 2$ GeV, is $O(100)$. To accommodate this number of tracks a total packet length of 108 clock cycles is used. This means each GTT instance sees a new event every 114 clock cycles[1]. The GTT will be implemented in a time-multiplexed system with a period of six with two FPGAs processing each event. One FPGA is dedicated to the core GTT algorithms; vertex finding, track $E_{\mathrm{T}}^{\mathrm{miss}}$, and track jets. This system is shown in figure 5.1 with each algorithm block shown along with the link decoding and encoding. The other FPGA will be dedicated to additional algorithms such as track-based resonances. As the track finder boards run with a time multiplexing of 18, each GTT FPGA will need to process three events, with three copies of the GTT firmware on each FPGA. Each FPGA will receive 54 input links from the track finders. The GTT has a conversion block that transforms the 64-bit track packets to track words, essentially the opposite of the last steps of the KF output firmware from section 4.3.3. There is also a p_{T} and η conversion that takes 1/R and $\tan(\lambda)$ to give more physical track parameters used in the rest of the system. This transformation is performed using a LUT of all possible p_{T} and η values within the digitisation ranges. This is the fastest, but not necessarily the most resource efficient method. An alternative method would be to directly perform the division of 1/R to give

[1]There is a 6 clock cycle buffer used to signal the start of an event

p_T directly, which could be implemented in far fewer LUTs at the expense of increased DSP usage and a far higher latency of at least six clock cycles versus the purely LUT implementation's one clock cycle.

The GTT FPGAs will output information in one link to the GT which will include track E_T^{miss} and jet information. Additional resonances will also be passed to the GT over one link. The vertex finding boards will also output the vertex to the correlator layers, implemented as a set of six boards per event divided in η, resulting in six additional output links per vertex finding FPGA to the correlator layers.

The physical GTT system was planned to be implemented on both Serenity and APx cards hosting two Xilinx Ultrascale+ VU9P FPGAs. These FPGAs have three SLRs giving an obvious splitting of three time multiplexing periods per FPGA. However, the larger Xilinx VU13P FPGAs has been selected for use in the final system. The VU13P is similar to a VU9P with an additional SLR which means single time slice resource usage is comparable. However, the input/output (I/O) to the FPGA is different, making the routing of tracks into the algorithms more complex. With four SLRs the splitting of the algorithms could instead see the core GTT algorithms in one SLR each and the final SLR for additional algorithms, such as resonant searches. This would reduce the total FPGA usage to just six for the entire GTT system which reduces the overall cost of the GTT requiring fewer processing boards and FPGAs but also reduced fibre optic routing in and out of the boards. However, it would require complex internal routing firmware inside the FPGA to send the tracks and outputs through this additional fourth SLR.

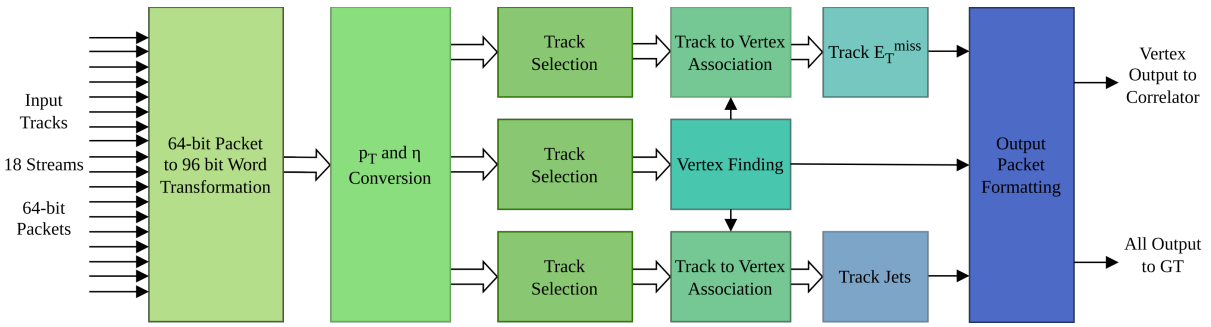


Figure 5.1: The GTT firmware structure for the core algorithms. This shows a single time-slice of the GTT with each individual firmware block shown, along with external and internal track streams and final output links to the Correlator and GT. There are 18 input track streams into the firmware block, these remain as 18 internal signals until after the p_T and η conversion. After this point the track streams are copied for the three parallel algorithms; track E_T^{miss}, track jets and vertex finding. The vertex finding provides the vertex for track-to-vertex association for the other two algorithms making it the critical path in the GTT.

5.2 Vertex Reconstruction and Track-to-Vertex Association

5.2.1 Motivation

One of the main factors impacting L1 trigger performance in the HL-LHC is pileup. The role of the vertex reconstruction and subsequent association of tracks to this vertex is to reduce the impact of this pileup by removing tracks reconstructed from pileup vertices. This results in up to a factor of 10 fewer tracks considered in the correlator trigger, thus allowing more complex algorithms to be implemented. It also prevents pileup energy depositions in the calorimeters being included in E_T^{miss} and jet-finding calculations.

In offline reconstruction the primary vertex is located by first selecting tracks that are consistent with vertices along the beamline with high enough quality. Then a clustering algorithm is used to find clusters of tracks along the z-axis. These candidate vertices are then fit using an adaptive vertex fitter [109] to determine the precise x-y and z coordinate. The primary vertex of an event is defined as the vertex with the highest $\sum p_{\mathrm{T}}^2$ and the typical resolution approaches 12 μm for vertices with at least 50 tracks [110]. This offline reconstruction is fundamentally different to the L1 requirements, in the L1 trigger only the highest p_{T} vertex is required and the lack of the pixel detector and relatively high track p_{T} cut from the track finder limits the resolution significantly giving a primary vertex resolution of 1 mm.

The L1 trigger vertex reconstruction needs to be able to output a vertex to the correlator trigger within 250 ns which is the time taken for the correlator to perform its own track parameter calculations. As the correlator calculations are the most time critical in the L1 trigger, any vertex bottleneck is to be avoided. The vertex finding algorithm must also fit within the overall GTT resource envelope of one SLR of a VU13P. These two requirements give strict resource and latency constraints on the algorithm. For the vertex reconstruction to be useful for track-to-vertex association the vertex resolution should also be within 1 mm as this is the resolution of the track z_0 parameter, as shown in figure 5.4. While there is no lower bound for vertex resolution it is not necessary for the algorithm to have significantly better performance than this 1 mm track resolution as when associating to the vertex the error will be dominated by the track resolution. Ultimately, the vertex finding performance is not as important as the track-to-vertex reconstruction and subsequent pileup veto.

Another key consideration of the vertex finding is that it is predominantly at use in high multiplicity events where $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_{T} are the key trigger paths and pileup makes a large difference to these results. For this reason key benchmarks of vertex finding are performed with an inclusive $t\bar{t}$ production sample with 200 PU events overlaid; this is the worst case scenario for the algorithm. All results shown below for the baseline, extended and neural network approaches are performed using $t\bar{t}$ PU 200 events.

5.2.2 Baseline Approach

The baseline approach to vertex finding uses an algorithm called FastHisto (FH). This algorithm is split into the following steps:

- Histogram tracks in z_0 weighted by their p_T.
- Run a three-bin window across the histogram summing all p_T in the window.
- Find the index of the maximum value of the resulting vector.
- Format this index to the required digitisation and return this and the $\sum p_{\mathrm{T}}$ of the vector bin.

This baseline approach uses a 256-bin histogram over a range in z_0 of -20.47 to 20.47 cm leaving a histogram width of 0.16 cm. A typical event is shown in figure 5.2. This figure demonstrates the number of clusters of tracks that are created as well as the large separation between primary vertex tracks that can occur with the poor track z_0 resolution at high η. This worsening of resolution is caused by the barrel geometry, demonstrated in figure 5.5, where the endcap layers are orientated to give better η and ϕ resolution rather than z_0. As is shown in figure 5.4, the bin width is similar to the minimum resolution of the tracks seen in the barrel region. Finer bins than this would offer little advantage to the vertex finding performance as the error would be dominated by the resolution of the tracks rather than the bin width and more bins would lead to higher resource usage in firmware, so the

compromise of 256 bins is used.

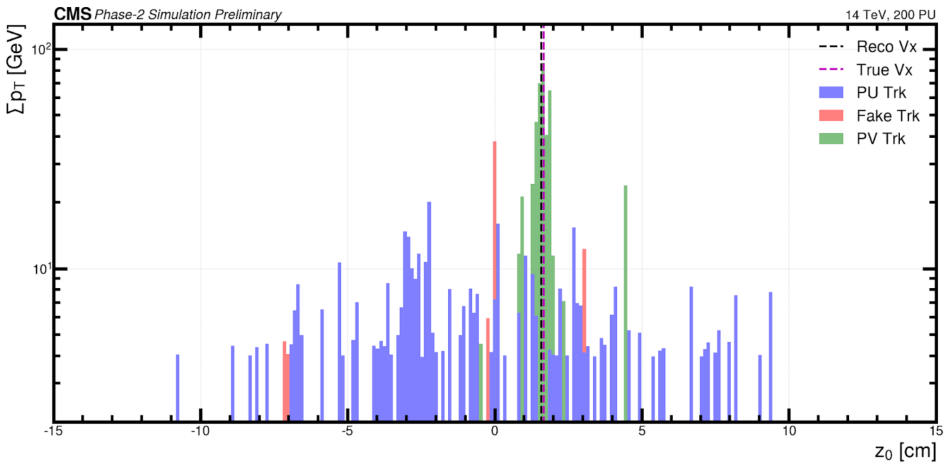


Figure 5.2: A single $t\bar{t}$ event with 200 PU demonstrating the FastHisto histogram. Each track has been histogrammed in z_0 with 256 bins, weighted by p_T. Colours denote track type with blue PU tracks spread evenly across the range. Fake tracks in red are randomly distributed. Green PV tracks are clustered around $z_0 = 2$ cm, the location of the true vertex (shown with the purple dashed line). The FH reconstructed vertex is shown in black; close to the true vertex indicating a well reconstructed vertex. The reason for this good reconstruction is the high p_T peak at the vertex, high p_T PV tracks are clearly seen making the vertex simple for FH to reconstruct.

The baseline approach to track-to-vertex association is based on the distance between the primary vertex and the z_0 parameter of the track. As is shown by figure 5.4 the resolution of this z_0 position is strongly dependent on η and as such a single threshold of distance between track and vertex is unsuitable. Instead, a set of five bins is used with a larger Δz_0 threshold allowed for higher η tracks. These are re-optimised as track reconstruction changes to ensure consistency for downstream users.

The implementation of the FH baseline in firmware uses several stages of routing logic to sort tracks into the histogram in z_0. Firstly, tracks enter the router in 18 track streams, their p_T and z_0 are extracted with the z_0 coordinate rescaled to the 256-bin histogram index. These tracks are sent through two stages of router and the p_T of each bin accumulated. The vertex is found by finding the bin with the highest p_T. The track-to-vertex association is a simple process of calculating the vertex distance and selecting the correct threshold based on η, which is stored in a LUT. This algorithm has been implemented in multiple ways; a VHDL only version, a HLS version, and a mixture that uses the VHDL histogramming step and the HLS vertex finding. The latencies and resource usages are summarised in table 5.1. The hybrid approach was designed to take advantage of the low resource usage of the VHDL histogramming, especially in the LUT usage, and the better features in the HLS vertex finding, for example, the ability to pass multiple vertices downstream. This algorithm has been tested in hardware as is detailed in chapter 6.

The baseline approach has some key advantages; the routing and creation of the z_0 histogram is fast (10 clock cycles) and handles high multiplicity PU 200 events without issue. This is particularly important in the high pileup scenario of the HL-LHC where the number of tracks being routed will be $O(100)$. The baseline approach is also effective with a vertex finding residual of 0.96 mm which is comparable to the z_0 resolution of the tracks. The track-to-vertex association is very simple, so is fast, with low resource utilisation, and performs well with a track-to-vertex association true positive rate (TPR) of 92% and corresponding false positive rate (FPR) of 10%.

Table 5.1: Resource usage and latencies of one SLR of a Xilinx VU13P running at 360 MHz for the FH implementations. These resource usages are estimates from a Vivado synthesis of the networks and the latencies are from a C-Simulation of the logic using ModelSim. The resource usages are for a single time-slice of the GTT. There is clear trade-off between latency and resource usage when comparing the HLS and HDL implementations, the far fewer resources of the HDL leads to double the latency. The mixed HDL + HLS approach takes the best from both and results in a fast, lightweight algorithm.

Implementation	Latency (clock cycles)	II (clock cycles)	LUTs%	DSPs%	BRAMs%	FFs%
HDL	63	1	7.2	0.0	5.7	4.2
HLS	28	1	28.8	0.0	0.0	4.2
HDL + HLS	38	1	7.8	0.0	4.8	3.3

One of the major issues with this baseline approach is the impact of fake tracks on the algorithm. As is seen in figure 5.2 at $z_0 = 0$ cm, the fake tracks in red can contribute to large spikes in p_T in the histogram. With these large spikes and relatively spread out primary vertex peaks, the vertex finding can select these spikes. This leads to the baseline approach producing a wide residual in z_0 because of completely erroneous vertex regression. Another issue is the vertex finding performance in event topologies with few high-p_T tracks, such as a $Z \rightarrow \mu\mu$ where the momentum in the event is shared between the high p_T muons. In events where these muons are forward (high η), the track z_0 can be poorly reconstructed leading to scenarios where the two muons do not appear in the same z_0 histogram bin (shown in figure 5.3). This then leads to the FH algorithm poorly reconstructing the primary vertex.

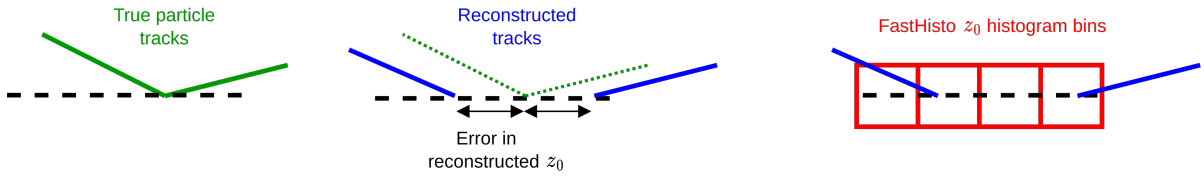


Figure 5.3: The effect of high η tracks on the FastHisto algorithm. The first pane shows the true primary vertex tracks in green; two high p_T tracks at high η. Pane 2 shows the reconstructed tracks in blue, with large uncertainty in the z_0 due to the high η of the tracks. Pane 3 shows the boundaries of the FastHisto binning. In this scenario the two high p_T tracks have been put in different bins and the FastHisto algorithm returns an incorrect primary vertex.

One solution to this would be to return multiple vertices, i.e. the top 10 highest p_T histogram bins and allow downstream algorithms to select all the tracks from all 10 vertices. While effective at ensuring all the true primary vertex tracks are passed downstream, the number of pileup tracks also passed downstream increases significantly. Another solution to this issue in these kinds of event topologies would be to rely on other trigger paths for their identification. For example, in high p_T muon scenarios, muon triggers will easily identify these events and the poorly reconstructed vertex would not impact trigger efficiency or purity.

The track-to-vertex association promotes high efficiency, accepting many true primary vertex tracks with a 91% TPR, but with low purity; 10% incorrectly matched tracks. A large proportion of these are due to the vertex being selected incorrectly. It is for this reason that the baseline GTT will output the top 10 vertices to be passed downstream which allows for more of the true primary vertex tracks to be used in downstream algorithms. Sending additional vertices means more processing downstream to chose which vertices to use but increases the

flexibility downstream if looser thresholds on primary vertex tracks were needed.

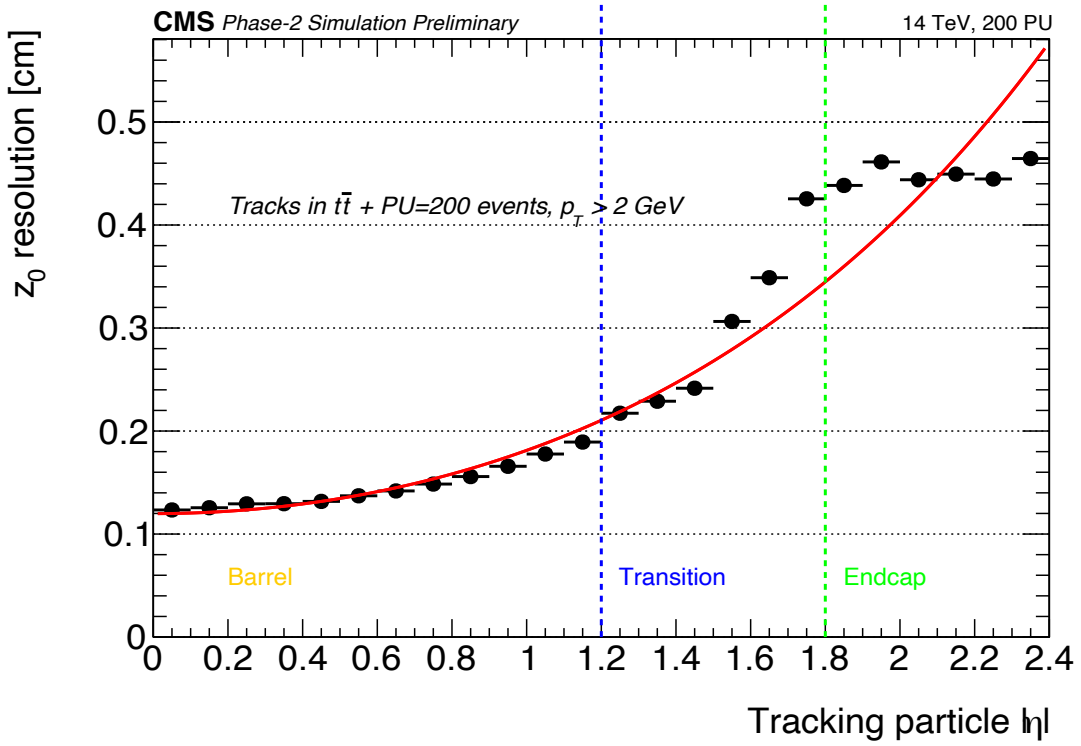


Figure 5.4: Reconstructed track z_0 resolution in $|\eta|$ bins, calculated by comparing reconstructed tracks to truth-matched tracking particles. The red line shows a $\cosh(|\eta|)$ fit to the data; an expected relationship with the barrel shape of the tracker and the resolution of p_{T} modules depending on their orientation (demonstrated in figure 5.5). The track z_0 resolution in the barrel region is very good at between 1 and 2 mm; the tracker modules are aligned to give high resolution in z_0. In the transition and endcap regions this becomes far worse as the endcap modules are aligned to precisely capture η and r information so the z_0 resolution is worse.

5.2.3 Extensions to the Baseline Approach

There are some key improvements that can be made to the FH approach. The first is to take into account the changing z_0 resolution as a function of η as illustrated by figure 5.4. In order to improve the vertex finding residual it is preferable to weight tracks with well reconstructed z_0 and to down weight those tracks with poorly reconstructed z_0. This was achieved with a weighting proportional to the $1/\eta$ of the track. While the resolution of the track follows a $\cosh(\eta)$ function this is difficult to implement in firmware, so a simpler version is used which approximates the relationship to:

$$\sigma_{z_0} = \frac{1}{1+\eta^2} \tag{5.1}$$

This, multiplied by the track p_{T}, is used as the weight in the histogram. As is shown by figure 5.6a the core of the residual is reduced by 10%, but the tails of the residual have increased by 32%. The reason for this is that fake tracks have a peaked η distribution i.e. fake tracks are more likely to be found at $|\eta| = 1.5$ (see figure 4.9b). Weighting tracks by η gives an overall higher weighting to fake tracks in certain regions leading to poor results.

To mitigate this, and the overall effect of fake tracks, an overall measure of track quality can be used. When using the E_T^{miss} cuts in table 5.3 to remove fake tracks, the tails of the residual are reduced. This reduction is

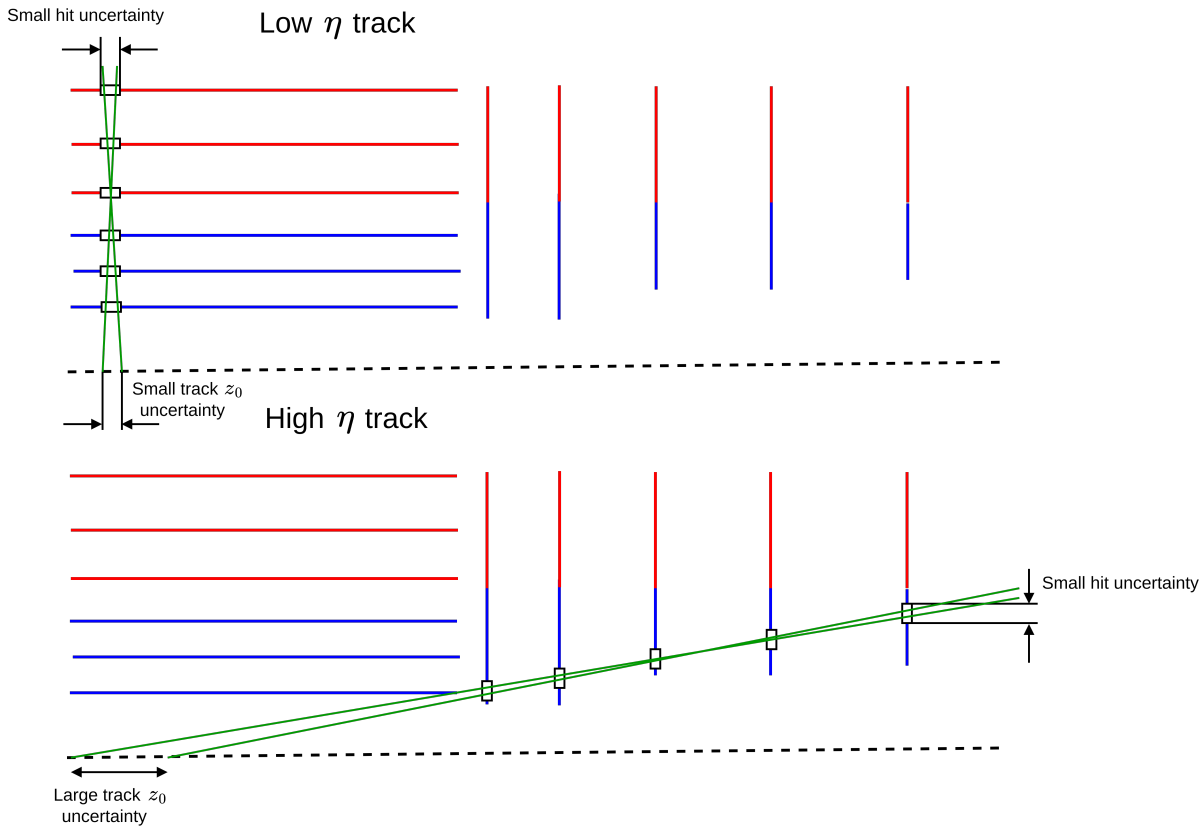


Figure 5.5: The effect of η on track z_0 resolution. In the barrel case, at the top, the uncertainty in the tracker modules run in the z_0 direction and leads to a small uncertainty in the reconstructed track z_0. A track with stubs in each of the layers can only be between the two green tracks shown in the figure. In the endcap case, on the bottom, the uncertainty in the tracker modules run in the radial direction meaning that with the same module uncertainty the compatible range of tracks shown in green has a far larger z_0 range. This effect leads to the deterioration in track z_0 resolution as η increases, shown in figure 5.4. The uncertainties in this diagram have been exaggerated to demonstrate the effect more clearly.

demonstrated in figure 5.6d. The core of the residual remains the same, shown in figure 5.6c. This follows from the fact that the tails are caused by apparent fake vertices caused by isolated fake tracks.

As has been described in section 4, a BDT is an effective tool for identifying fake tracks. The output of this BDT can be used in two different ways, firstly as a cut on tracks, removing those that appear fake at some threshold. This is an effective way of reducing the number of erroneously reconstructed vertices as the high p_{T} fake clusters are removed. As is shown in figure 5.7a and 5.7b there is a specific threshold at which the vertex finding is most effective. It outperforms the χ^2 cuts in both the tails and core of the residual. A combined weighting can then be created that uses both a cut on the BDT and a weighting from the η. This removes tracks with poor quality and is effective at reducing the tails of the histogram compared to the η weighting while maintaining the core performance of just the η weighting, when comparing figure 5.6a to 5.8a. The performance in the tails, however, is worse than the baseline FH meaning a more complex function using the BDT threshold and η is needed.

These additions to the baseline approach would have potentially large impacts to the resource usage and latency of the FH algorithm. The addition of a cut on the BDT output would be lightweight with a single clock

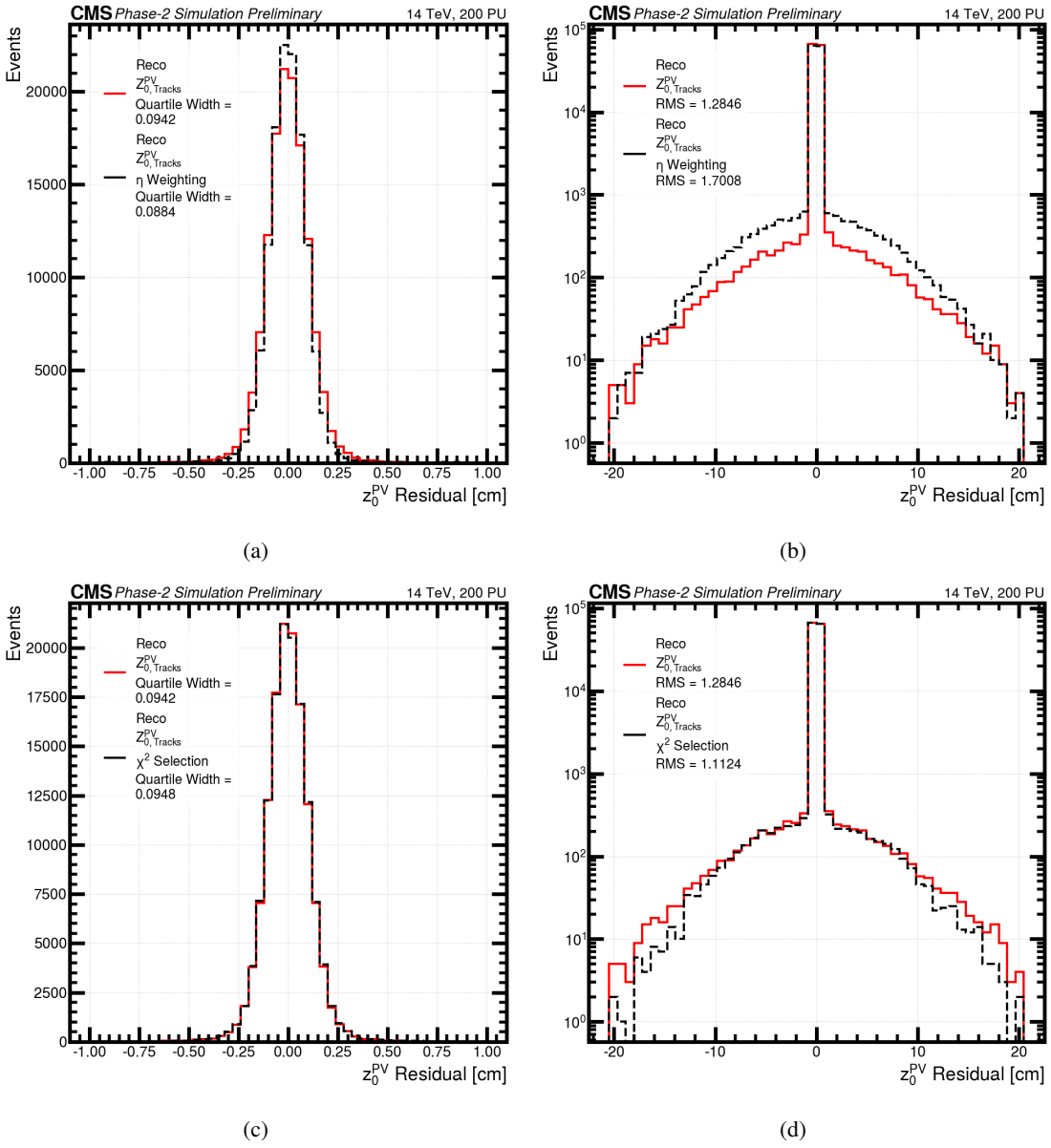
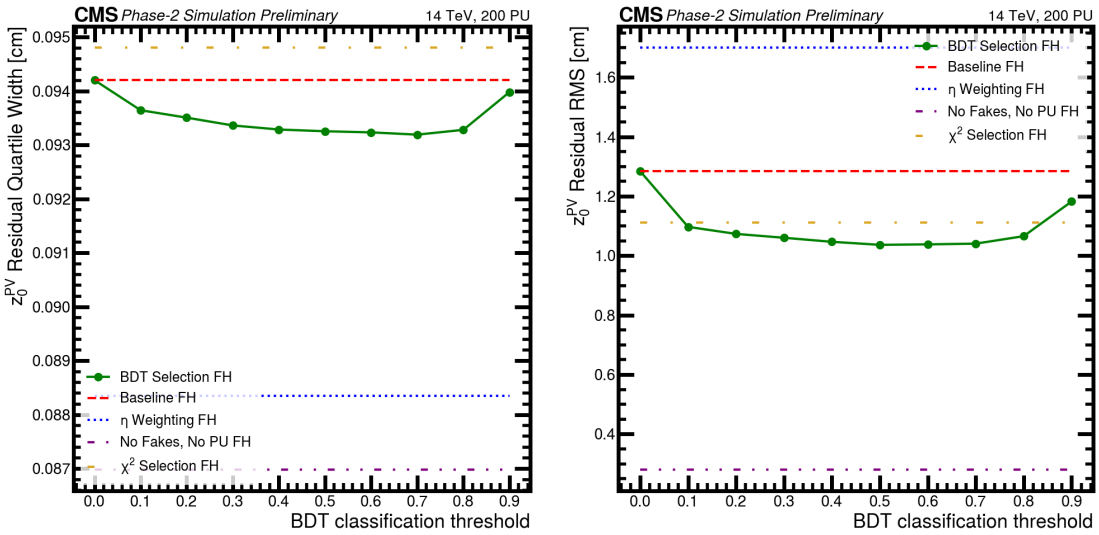
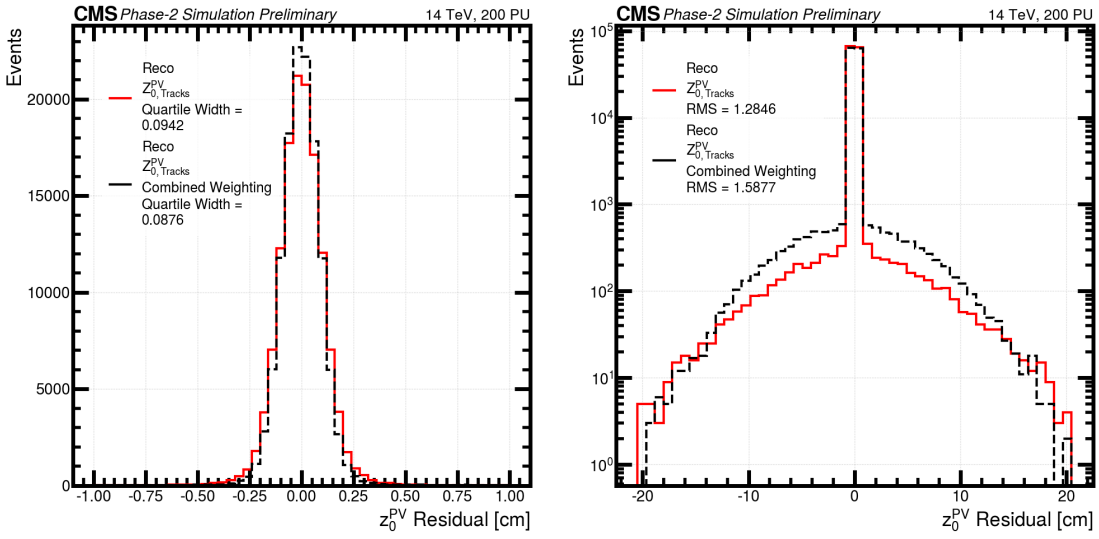


Figure 5.6: Vertex residual plots for different additional weighting scenarios shown in dashed black compared to the baseline FH in red. Plots on the left show the core of the residual whereas the logarithmic plots on the right highlight the tails. Residuals are defined as $z_0^{\text{truth}} - z_0^{\text{reco}}$. Quartile width is calculated by finding the distance between 32$^{\text{nd}}$ and 68$^{\text{th}}$ quartile. The root-mean-squared (RMS) of the residual is chosen as it is sensitive to large tails in the residual. Figures (a) and (b) show a weighting proportional to inverse η causing an improvement in the core of the residual but worse tails. Figures (c) and (d) show the residuals after a χ^2 quality cut that improves the residuals by removing some isolated fake tracks.



(a) (b)

Figure 5.7: Vertex finding performance for different extensions to FH. Shown is the baseline FH in red, η corrected, which includes the weight in equation 5.1, χ^2 corrected that uses the E_T^{miss} cuts in table 5.3 and a truth level removal of fakes and PU. In green is FH after a cut on the BDT track quality output at different thresholds. These plots show that only using the primary vertex tracks obviously gives the best performance and can be seen as the best-case scenario for vertex finding. The η weighting and χ^2 cut give different improvements in the two different metrics, η is better for the core of the residual in figure (a) and the χ^2 cuts are good for improving the tails. The BDT can improve both the core and the tails by removing poorly reconstructed tracks and fakes but the improvement in dependent on the threshold selected.



(a) (b)

Figure 5.8: Vertex finding performance for the combined extension of FH with an η weighting and cut on BDT score of 0.7. Shown is the baseline FH in red and the extended FH in dashed black. The improvement in the core of the residual is due to the η weighting and the BDT cut has improved the tails as compared to figure 5.1 but not enough to be better than the baseline approach.

cycle to evaluate each track. The inverse η^2 would be a complex operation, for full precision, a division would need to be implemented which would incur multiple DSPs per track input as well as over six clock cycles of additional latency. While these operations can be performed in parallel they are not necessarily simple to implement and the designing of the firmware to be optimised is not a straightforward task. The difficulties with implementing these algorithms in firmware can be avoided with if using a neural network approach where the firmware can be generically generated for a given network that can learn an ideal weighting in this scenario.

5.3 End-to-End Vertex Reconstruction

As has been shown previously, more complex weightings can lead to improved results for the FH algorithm. The additional weightings shown before were decided based on intuition about the tracks and changing parameters based on vertex finding performance. However, they were chosen based on $t\bar{t}$ performance and do not take into account the possibility of changing samples or even changing requirements from the vertex finding. A solution to this would be to use an NN-based approach to the task of vertex finding and track-to-vertex association. Given the interplay between vertex finding and the association both of these tasks were trained together in a single end-to-end network that optimises both tasks simultaneously. In order to take the previous FH and cut-based approach to this end-to-end network a number of steps were taken.

Firstly, a machine learning approach to generate a weighting function in the FH histogram is used. An NN with input features of track parameters would be able to form a weighting function of these tracks dependent on a changeable loss function, and on a variety of samples depending on the requirements. For example, if the core of the vertex residual is prioritised then a mean absolute error loss function on the vertex finding regression task could be chosen or if the tails of the residual were chosen to be more important than a mean squared error loss could be used. This approach to generating a weighting is not only more robust to changing requirements and event topologies therefore making the algorithm design simpler, but it is also simpler to implement in firmware. Instead of a custom written function that rescales track parameters and then mixes them in a polynomial, an NN can be easily exported to a firmware design language such as Vivado HLS and immediately deployed and redeployed if the network is retrained. This again simplifies the design process and as is shown in section 3.4 there are a number of techniques for optimising an NN for both resource usage and latency.

Secondly, the sliding window that is used to find the peak in the histogram was replaced. This three-bin window is in essence a single convolution with a linear activation function, i.e. a [1,1,1] convolution. When viewed in this way it is a very simple extension to remove the constraint that this has a linear activation and all three numbers in the window are equal and instead replace it with a one-dimensional CNN.

Finally, it has been shown that the track-to-vertex association is also improved by the use of an NN classifier [111] which can be understood by the classification task being more complex than just a Δz_0 cut. Many low p_T tracks will not be associated to the vertex and instead will come from an additional pileup interaction in a similar z_0 region to the primary interaction. There will also be low quality tracks which have a worsened z_0 resolution due to their poor reconstruction, which have the potential to overlap with the primary vertex. These tracks can be removed when incorporating both the p_T and quality of the track. These features can be used in conjunction with the Δz_0 and η to better classify tracks.

The two separate tasks of vertex reconstruction and track-to-vertex association are conceptually very different—while vertex reconstruction is an event-level regression task, the association is a track-level classification task. However, they both benefit from each other. The association is improved with the correct

selection of the vertex and a learning task in which the vertex finding is optimising a weight benefits from knowledge of whether a track originates from the primary vertex. For this reason, an end-to-end approach to the full reconstruction of both the vertex and vertex candidates is used. In this approach a single NN is used to learn both tasks. The first part of the network is a DNN which learns a weighting function for the histogram in z_0. This histogram is then given to a CNN in order to recognise the peak in the histogram, which is returned as the vertex. Within the network it is also possible to learn additional hidden features from the weight network and convolutional network, these latent features can be understood as an event-level feature which are not trained against but provide additional information to the rest of the network [112].

Finally, the vertex, along with internal latent features of the previous networks are passed onto a second DNN which classifies the tracks as originating from the vertex. The end-to-end NN learns a weighting function and peak finder that are not only optimised to find the correct vertex but also to classify tracks to the vertex [113]. Ultimately, the important part of this task is the vertex association as this is passed downstream for more complete event reconstruction. This end-to-end approach to vertex finding was originally used in [111], however, significant changes were made to the network architecture and training to allow the firmware implementation. These changes are detailed in section 5.3.2. The network described below is the final version that is implemented in firmware and differs from that described in [111].

This approach to the use of machine learning in firmware design not only allows for complex algorithms to be quickly realised both in testing and firmware, it also creates robust algorithms trained on large diverse datasets which can also be continually trained as changes to the dataset occur dynamically, as is discussed in chapter 7.

5.3.1 Architecture and Learning

The network architecture in figure 5.9 shows the three separate parts of the network; the weighting function feeding into the histogram, the pattern recognition convolutional layer which is passed to an ArgMax to give the vertex location, and the track-to-vertex association layers to classify tracks.

In order for this network to be end-to-end and train with both a vertex regression and association loss function every part needs to be differentiable for the back propagation. For this reason a custom TensorFlow [114] layer was written for the histogram. The histogram with bin h_i is filled with track z_0, weighted by weight w, which is a learnt function of track features; p_{T},η and BDT output. Each histogram bin is the weighted sum of all tracks in that z_0 bin:

$$h_i = \sum_{j=0}^{tracks} \delta(j \in \text{bin } i)\ w(p_{Tj}, \eta_j, BDT_j), \tag{5.2}$$

which is differentiated with respect to the weight resulting in the following gradients

$$\frac{\partial h_i}{\partial z_0} = 0 \quad \text{and} \quad \frac{\partial h_i}{\partial w} = \sum_{j=0}^{tracks} \delta(j \in \text{bin } i). \tag{5.3}$$

These equations allow the histogram to be used in both the forward and backwards passes of the NN training.

A standard ArgMax's differential is undefined so in order to use it within a back propagation setting a soft ArgMax is used. This ArgMax has an input of a vector x of N elements and is implemented in several standard

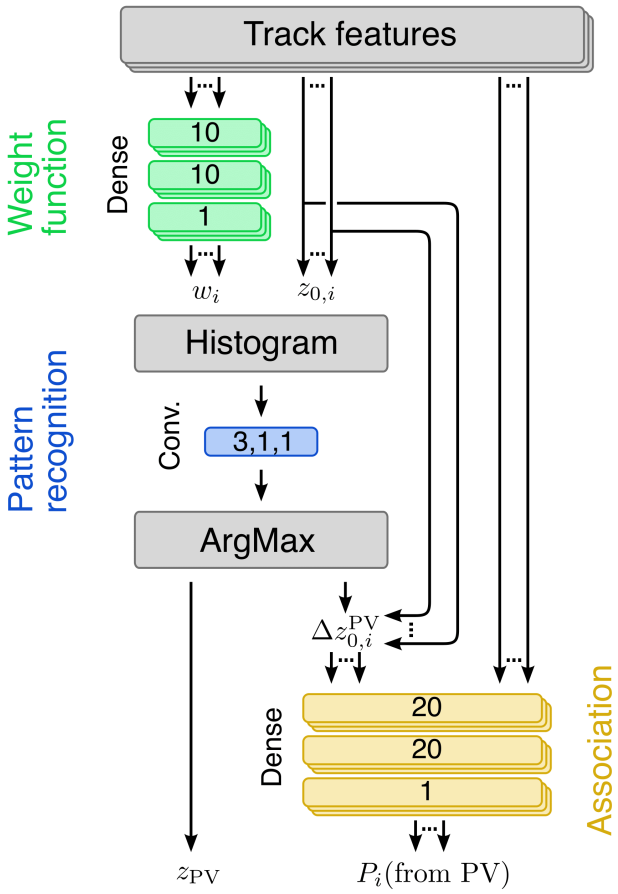


Figure 5.9: End-to-end neural network architecture, each subnetwork is shown in a different colour. The number of layer nodes is shown in the case of dense layers and kernel size for the convolutional layer. Track features are fed into the weighting function DNN, in green, that creates the histogram weight for each track. The histogram is passed through the peak-finding convolutional network, in blue, and the maximum bin is found with an ArgMax to return the primary vertex z_{PV}. With track i's distance from the primary vertex $\Delta z_{0,i}^{\mathrm{PV}}$ and track features a second DNN, in yellow, is used to learn the probability a track came from the regressed vertex. Both the event-level primary vertex regression and track-level track-to-vertex association are trained simultaneously.

TensorFlow layers and therefore differentiable, the steps are shown pictorially in 5.10.

Firstly, a SoftMax runs over the vector which returns a vector of values between 0 and 1 where 1 corresponds to the highest value in the vector with each value in the output vector found as:

$$\frac{e^{x_i}}{\sum_{j=0}^{N} e^{x_j}}. \tag{5.4}$$

The vector is rescaled by a temperature, T, which brings values in the vector closer together meaning the SoftMax is more effective at selecting a single highest value rather than multiple values, to give:

$$\frac{e^{x_i/T}}{\sum_{j=0}^{N} e^{x_j/T}}. \tag{5.5}$$

In order for this SoftMax to correspond to a bin index of the initial vector each bin is then multiplied by its own bin index, i.e. a single linear layer of weights between 0 and 255. Where the SoftMax returned 0 this multiplication leaves 0s and when a 1 is returned this multiplication results in the bin index where the 1 was located. Finally, the resulting vector is summed giving just the bin index where the SoftMax's 1 was located:

$$\sum_{i=0}^{N} i \frac{e^{x_i/T}}{\sum_{j=0}^{N} e^{x_j/T}}. \tag{5.6}$$

Using these equations implemented with TensorFlow operations the Argmax part of the network is usable in both forward and backwards passes.

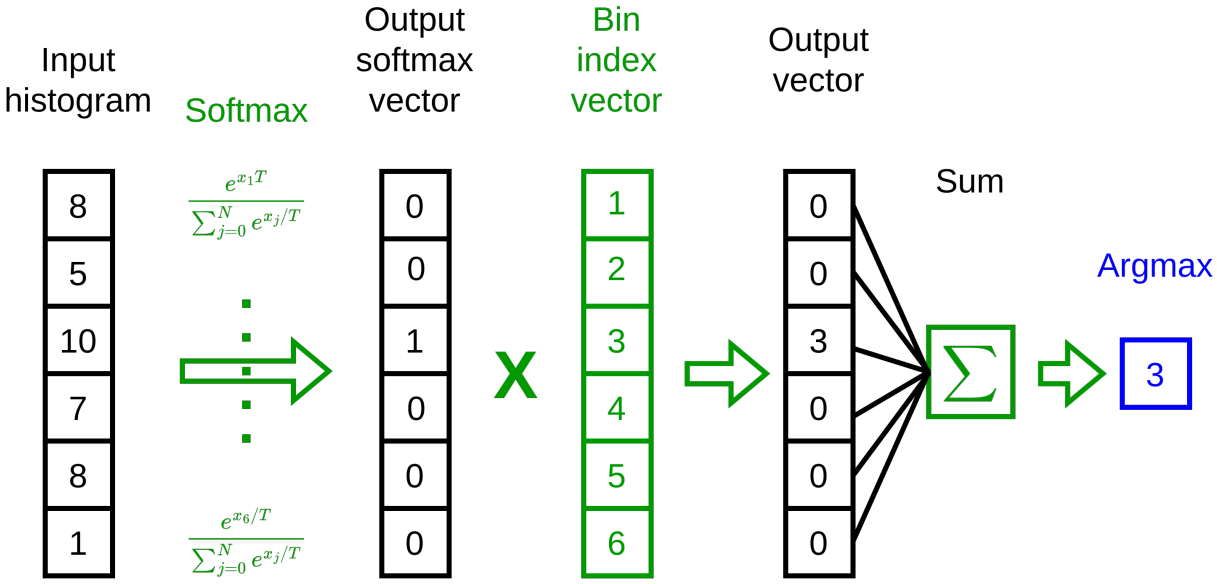


Figure 5.10: Representation of the differentiable Argmax processing steps. The first step is the temperature scaled softmax in equation 5.5 which isolates the highest value in the input vector. The second step multiples the output vector by its bin indices which leaves a vector where the only value is the index of the bin that the value is in. This final vector is summed over to leave just the index that the maximum value was in. All stages of this calculation are differentiable and can be used in backpropagation.

The training of the network uses a two part, equally weighted loss function. The vertex regression is controlled by a Huber loss function [115] which has a loss, L_δ with a tunable Huber δ. For one event with predicted vertex V_p and true vertex V_t the loss is:

$$L_\delta(V_t - V_p) = \begin{cases} \frac{1}{2}(V_t - V_p)^2 & \text{if } |(V_t - V_p)| \geq \delta, \\ \delta(|(V_t - V_p)| - \frac{1}{2}\delta), & \text{otherwise,} \end{cases} \tag{5.7}$$

which is quadratic for small $(V_t - V_p)$ and linear at larger $(V_t - V_p)$ which gives control over prioritising the core of the vertex residual and the tails. Higher Huber δ would lead to a quadratic loss function for more values of $(V_t - V_p)$ which would reduce the tails of the residual.

The track-to-vertex association uses a binary cross entropy loss function with predicted association probability of p_i and actual class of y_i for track i over an event with N tracks:

$$L(\mathbf{y}) = \frac{1}{N} \sum_{i=0}^{N} -(y_i \log(p_i) + (1 - y_i) \log(1 - p_i)). \tag{5.8}$$

The cross-entropy-loss is a single value for each event.

In addition to these loss functions there are a number of auxiliary losses which are used to improve the network's learning. The first two are simple ridge and lasso regularisation terms. The ridge regularisation has the effect of drawing weights closer to zero, which makes subsequent pruning more effective [78]. This effect is due to the ridge regularisation imposing the condition that the sum of all weights should be small and so prioritising overall fewer weights in the network. The lasso regularisation causes the network weights to be smaller in magnitude with a penalty on the squared magnitude of the weights which makes network quantisation more effective. [79]

The final loss function that is added controls the magnitude of the track weights which are used in the histogram. If let unconstrained, the track weights will be arbitrarily large meaning that two separate training cycles will have different sized track weights being learnt. This not only reduces reproducibility but introduces difficulty when selecting an appropriate quantisation when compressing the network. An additional loss function is applied to the track weights that are used in the histogram. This loss calculates the top 10% of the learnt track weights and returns a loss proportional to the mean square of this top 10% thus penalising high magnitude learnt track weights.

The NN uses three track features as input to the weight network, p_{T}, $|\eta|$ and the BDT score, these were selected following on from the arguments presented in section 5.2.3. These values are given as their digitised forms in order to simplify feature conversion logic in firmware. The combination of these track features combine in a non-trivial way, but there are correlations between the track features and learnt weight as shown in figures. 5.11a, 5.11b and 5.11c. From figure 5.11a it can be seen the correlation with p_{T} is linear, therefore similar to the standard FH. However, the linear relationship weakens for high p_{T}, which could be due to the network filtering away high p_{T} fake tracks. Real primary vertices have multiple mid to high p_{T} tracks whereas incorrectly regressed vertices have single very-high p_{T} fakes. A down-weighting of these very-high p_{T} tracks can reduce the impact of these isolated spikes. Figure 5.11b shows a complex relationship in η with no clear correlation being made, figures 5.12a–5.13d demonstrate that central tracks have a higher weighting when their p_{T} is higher, most evident when observing the high weighting in the bottom right and section of each plot. Figure 5.11c demonstrates a complex filtering performed by the network, i.e. tracks with higher BDT scores have higher weights, but this relationship is more clearly seen in figures 5.12a–5.13d where each MVA score has a progressively higher weight across all values of p_{T} and η.

These features of the weighting function and its correlations with track features demonstrates the utility of using NNs as function approximations and as guides when writing firmware. The network has the advantage of automatically updating this function as different data is given to the network, whereas as purely analytical approach would require re-analysing the data and creating a function which would need re-implementing in firmware; the network weights can simply be updated in the firmware within the same footprint.

5.3.2 Firmware Implementation

When implementing NNs in firmware there are many constraints and design choices; latency, resource budget, available input information etc, described in section 3.4. There were many steps from taking the original end-to-end NNs described in [111] to ones that could be used in firmware, and they were adapted in many ways.

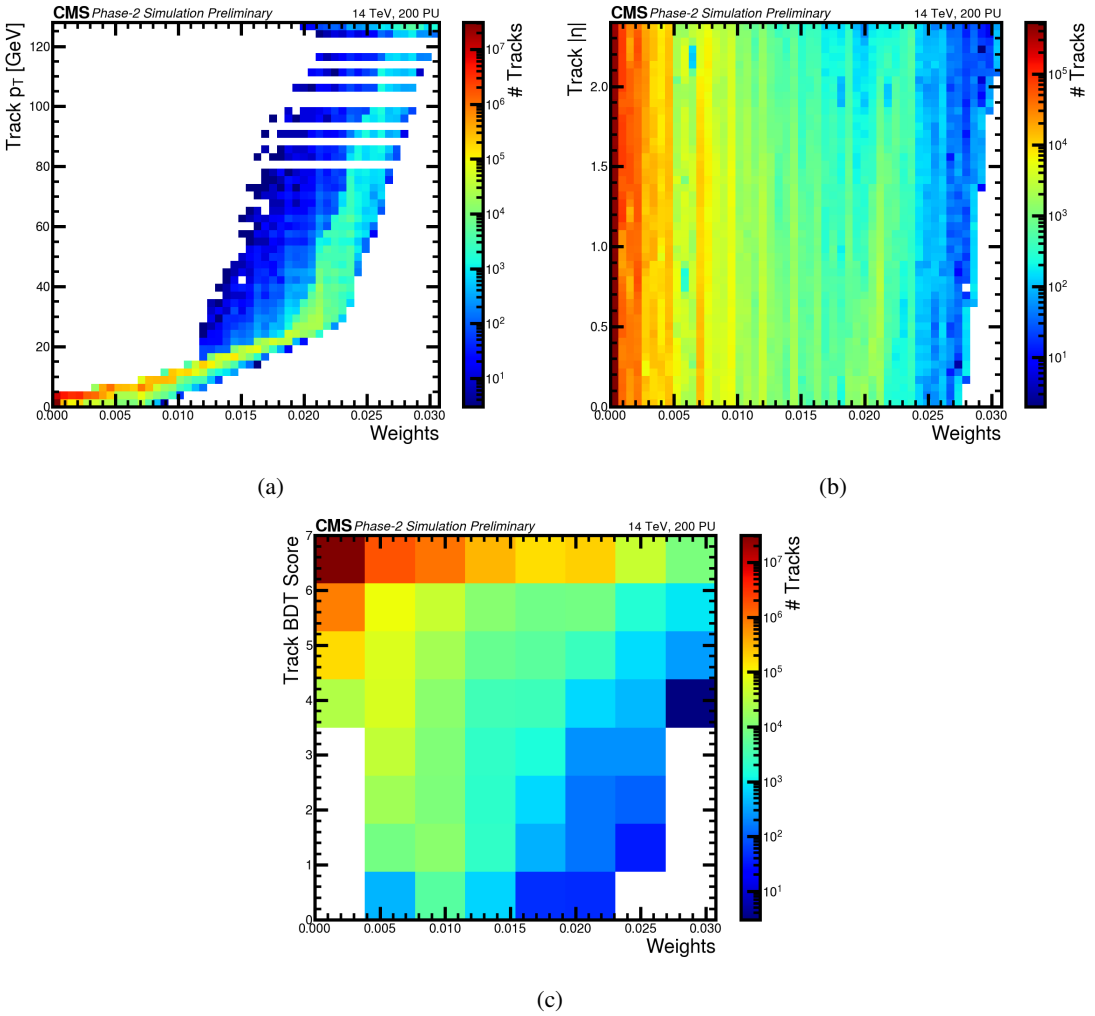
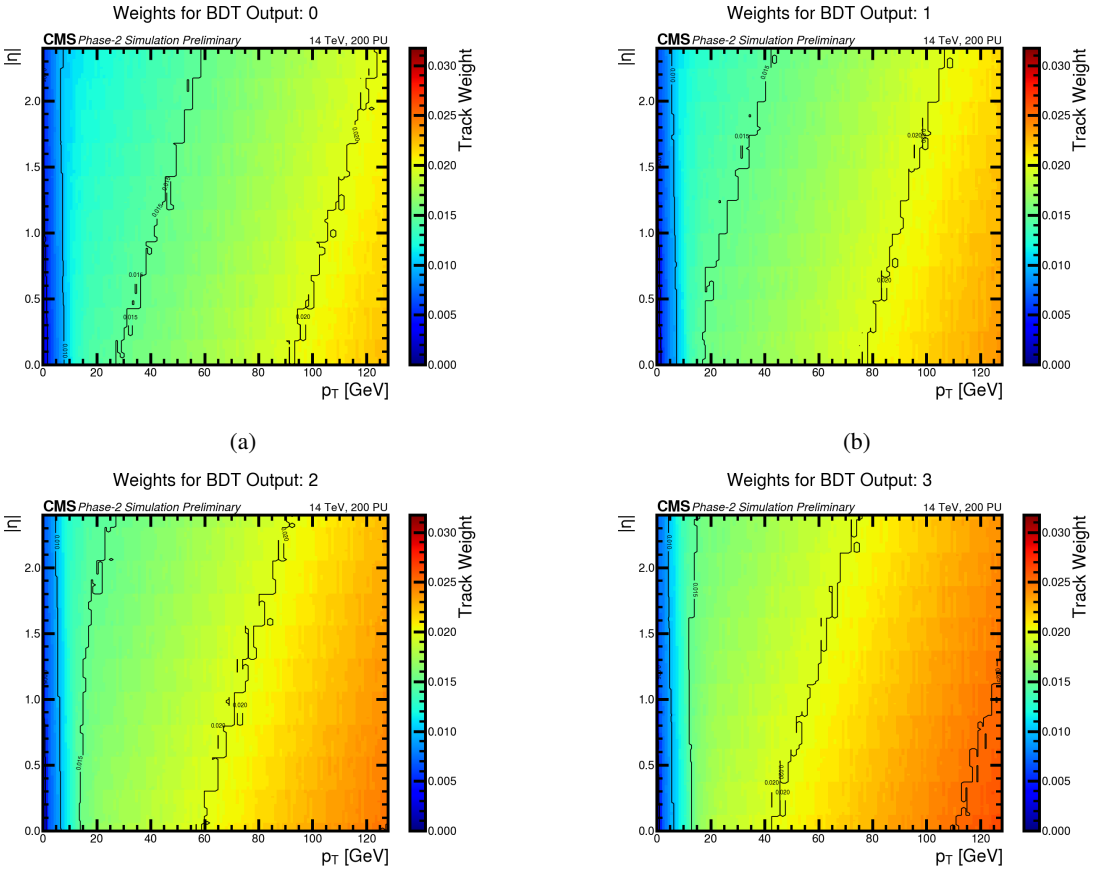


Figure 5.11: Learnt track weights as a function of input track features. Shown are two-dimensional histograms over the entire testing dataset. Figure (a) shows a broadly linear relationship with track p_T that weakens at high p_T reducing the impact of high p_T fake tracks. Figure (b) shows the η correlation with no clear relationship but when splitting the output by the BDT output a clearer relationship is seen where central tracks are more highly weighted (figures 5.12a–5.13d). Figure (c) demonstrates that in general higher BDT scores lead to higher weighting, more obviously seen when comparing figures 5.12a and 5.13d. The horizontal bars at high p_T in figure (a) are due to the quantisation of p_T where only certain values of p_T are allowed; the quantisation is non-linear meaning there are fewer bins at higher p_T hence the obvious splitting.

Architecture

The original NN used a similar weight generation network architecture and histogramming, which was kept the same because of the relatively low number of network parameters. The original CNN was a two-part, eight-layer CNN that aimed to find the peak pattern and then regress the vertex directly with multiple convolutional layers, max pooling layers and dense layers. Due to the high latency cost of CNNs this was not usable in firmware and was replaced by the single one dimensional CNN and differentiable ArgMax layer. This has the advantage of using existing FH firmware for the final vertex finding. With this new architecture of simple weight network feeding into histogram then to a one-dimensional CNN and differentiable ArgMax, it was possible to recreate FH in TensorFlow by limiting the weight layers to just p_T, and the CNN to just a [1,1,1] convolution. This was used to verify both the histogram and differentiable ArgMax were performing as expected.



(a) (b) (c) (d)

Figure 5.12: End-to-end network weight as a function of p_{T} and η for the full ranges possible for p_{T} and η as dictated by firmware constraints, split by BDT output values 0–3. These plots were created by evaluating the network weight at every point in the available feature space. These plots demonstrate that higher p_{T} tracks have higher weighting and that as p_{T} increases, lower η tracks have a higher weighting. This follows from lower η tracks having better z_0 resolution. Higher values of the BDT score lead to higher weightings, especially for higher p_{T} tracks.

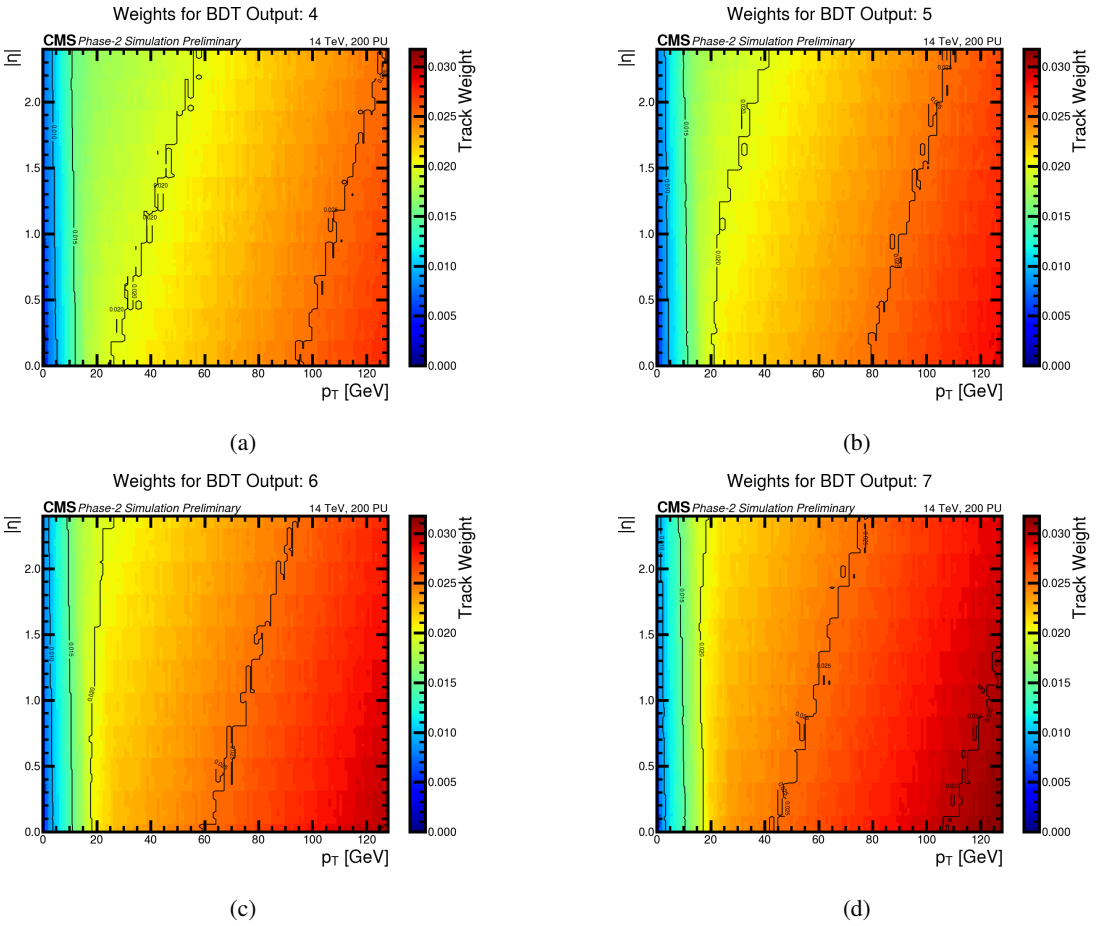


Figure 5.13: End-to-end network weight as a function of p_{T} and η for the full ranges possible for p_{T} and η as dictated by firmware constraints, split by BDT output values 4–7. These plots are a continuation of those shown in figures 5.12a–5.12d and continue to demonstrate that for higher BDT scores the weighting increases, especially for high p_{T}, low η tracks.

The dense association layers were kept the same, again on account of their simplicity, though various improvements were made to the weight initialisation and regularisation of the networks to achieve more stable training.

Input Features

When originally designed, the end-to-end networks used multiple track features, p_{T}, η, χ^2s and information from the hitpattern. The χ^2 variables and hitpattern are measures of quality of the track and so the track quality BDT was used instead as it is a single variable giving the same information. This reduces the number of input features and simplifies the networks. These input variables were also translated to their quantised equivalents as these are what is available in the firmware. p_{T} and η are calculated in the GTT input conversion firmware from the $\frac{1}{R}$ and $\tan(\lambda)$ of the track word and have specific bit widths. The BDT output is a three-bit integer taken directly from the track word. These variables were used throughout training to simplify conversion firmware, saving latency and resources. Track z_0 was also quantised in training, again to simplify the interface in firmware and to give realistic performance. In training, a bin index to cm conversion was used to allow the regression loss function to operate, but this conversion is not used in firmware.

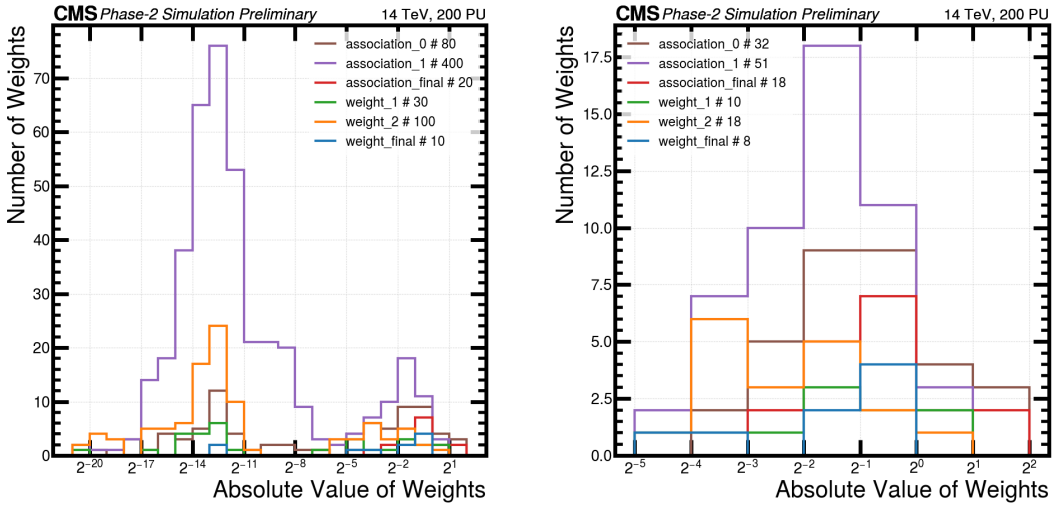
Quantisation

A hybrid of quantised-aware training and post-training quantisation was selected because of the performance advantage versus just a post-training quantisation [116]. The floating point network was first trained and then quantised to avoid the instability of training the quantised network directly. This process of transferring learnt parameters of one complex network to another simpler network before retraining is similar to the idea of transfer learning that is used to compress networks [116]. This quantisation was performed layer-by-layer using the QKeras [93] package and inspecting the spread of weights in every layer. A process which was necessary over an automatic quantisation because of the custom TensorFlow layers included in the network. Layer-by-layer quantisation versus whole model quantisation, while giving the same performance, allowed small gains in resource usage as unnecessary bits were removed. As can be shown in table 5.2 there is a significant reduction in resources when quantisation is used (comparing floating point to quantised) and it is essential for meeting the resource requirements. Post-training quantisation was performed on the network (once pruning rounds had been complete) to ensure that the quantisation selected was exactly the size of the network parameters which allows small gains to be made to the overall resource usage.

Pruning

A custom pruning cycle implementation was needed for this network as it has non-standard TensorFlow layers. A trained and quantised network was evaluated and all the weights close to zero were chosen. These filled a mask that set these weights to zero. This mask was used along with the previously trained network in a new training cycle where the network was trained with the pruned weights set to zero. Quantisation was performed before the pruning cycles began as the quantisation itself has the effect of pruning some weights, those weights already too small to be represented by the restricted bitwidths are lost, so the first pruning round just removes these weights. This method of incrementally pruning weights is far more stable than removing all the weights at the end of the training cycle and allows for more weights to be removed while maintaining network accuracy. For the end-to-end NN eight pruning cycles were used targetting only the track weight and association layers—the CNN is too small for effective pruning. Figure 5.14a shows the floating point network range of values spanning 2^2 down to 2^{-22} with $O(1000)$ total parameters. Figure 5.14b show the quantised and pruned network with a range of values from 2^2 down to 2^{-5} with $O(100)$ total parameters. This range is a quarter that of the floating point network with a factor 10 fewer parameters. The effect of this is clear when comparing floating point to quantised and pruned in table 5.2.

The quantisation seems to have a larger effect than the pruning which is due to the quantisation effectively being a first pruning step.



(a) Floating-point, unpruned NN.

(b) Quantised and pruned over eight iterations NN.

Figure 5.14: Parameter distributions for the specified NNs. Each layer of the network is shown and labelled weight and association respectively. The number in each legend entry indicates the total number of weights remaining in each layer. There is a clear reduction in the overall range in the absolute value of weights, the floating-point network in (a) uses a range between 2^2 and 2^{-22} whereas the quantised and pruned network only used between 2^2 and 2^{-5}. The number of weights has also dropped significantly. As is shown by the purple line, the hidden association layer has the most parameters and the largest reduction with nearly eight times fewer parameters after pruning.

hls4ml

The hls4ml [73] tool is used to transpile the NNs to firmware. As it is designed for standard NN layers, the end-to-end NN was split into three separate subnetworks; the dense weight network, the convolutional pattern network, and the dense association network. These are individually transpiled by hls4ml and inserted into the existing HDL FH which contains the histogramming and ArgMax as well as the functionality for extracting track features from the track word and outputting a correctly formatted vertex. The standard method of implementing one-dimensional CNNs in hls4ml will aim to optimise the resources being used by implementing a single convolutional block in firmware. This firmware is given the input histogram in chunks, and then outputs these chunks after processing in a pipelined manner. This has the issue of increasing the overall latency of the convolutional step significantly, i.e. if the histogram is processed in 256 chunks it introduces an additional 256 clock cycles of latency to the convolutional processing, which is outside the required latency to return the vertex to the correlator layers. In order to avoid this, the CNN layer is implemented in a fully unrolled manner in a custom version of hls4ml where the entire histogram is processed in parallel. This reduces the latency down to the total processing time of the histogram only, as outlined in table 5.2b.

It was found, when synthesising and subsequently implementing the firmware for deployment on the target FPGA, that there was very large fanout of control logic for the individual networks. This causes congestion in the firmware placement on the FPGA and subsequently causes it to not meet timing. This was a particular problem in the fully parallel one-dimensional CNN which requires many signals to be replicated and fanned out to 256 copies of the processing logic. In order to reduce this behaviour, flushable pipelines are implemented which are designed

to reduce blocking signals that cause the high fanouts. The more modern high level synthesis tool Vitis HLS is used in preference to the older Vivado HLS for its implementation of flushable pipelines. All networks, once hls4ml has produced their firmware, are reprocessed using Vitis HLS and meet timing when implemented on the FPGA. This required rewriting parts of hls4ml to allow its use with Vitis as this is not currently supported by hls4ml.

The firmware implementation of the end-to-end approach inserts the individual networks within the existing baseline firmware written in VHDL. Instead of single p_{T} weighting the weighting network takes the initial track as input, returning the weight which is used to fill the histogram. This network is replicated for each of the 18 input track streams allowing for parallel processing of the tracks, reducing total latency. The resulting 256-bin histogram is inputted with each bin in parallel into the convolutional network that returns a resulting 256-bin convolved histogram. The peak of this histogram is then found and sent over output links to the correlator and GT as well as being used internally by other GTT algorithms, as is shown in figure 5.1.

The association network is inserted in the GTT preceding the track E_T^{miss} calculation, again in 18 copies to match the internal track parallelisation. This firmware block waits for the vertex from the previous step before running over the tracks which are stored in FIFOs. The association network will also be inserted into the correlator firmware after the initial track decoding. This allows a reduced set of tracks can be passed downstream to the particle flow layers, this will be replicated multiple times to match the time multiplexing within the correlator.

Emulation

In order to verify the correct operation of each part of the NN in firmware an emulation was needed. This was written in the CMS Software CMSSW package (implemented in C++) and follows a similar design layout to the firmware. Individual TensorFlow networks are saved and loaded as tensor graphs in C++ which are inserted within the FH emulation. This was verified by individually confirming track-by-track the correct weighting was seen, and the correct vertex was returned versus the python model. The final release of this emulation and comparison using the correlator emulation is still ongoing. The vertexing emulation is also used to compare to the firmware implementation, again track-by-track comparisons are made one clock cycle at a time through a firmware simulation to verify the firmware. Again, firmware verification and testing over multiple full events is ongoing. The hardware implementation of this firmware is detailed in chapter 6.

5.3.3 Comparison to Baseline

The end-to-end approach outperforms the baseline approach in key metrics. Figure 5.15a shows the core of the vertex finding residual, which shows equivalent performance between the baseline approach, in red, and neural networks; quantised (QNN) in blue, quantised and pruned (QPNN) in orange, and floating point trained (NN) in green. This is to be expected as the main target of the NN approach was in reducing the tails of the residual, demonstrated in figure 5.15b which shows a 34% reduction in the RMS error. The reduction in tails can be more easily seen by comparing figures 5.17a and 5.17b which show a large reduction in the events surrounding the core residual. The baseline approach can be seen to have a slight bias towards returning a PV closer to 0 shown by the oval shape of the distribution whereas the NN approach does not show this shape. This is an advantage of the NN approach, with less biased predictions. The reason for this improved performance in the tails is due to the effective down weighting of fake tracks in the end-to-end approach; with information on track quality it is able to use this information to avoid situations where isolated high p_{T} fakes appear as the vertex.

Table 5.2: Resource usage and latencies of one SLR of a Xilinx VU13P running at 360 MHz for the split NNs. Total resource usage for all 18 replications of the weight and association networks, corresponding to the number of parallel track streams used in the Level-1 trigger (the pattern network is only implemented once as it is an event-level operation) are given. The NN approaches are additional to the HDL implementation resources and latencies detailed in 5.1 as they use existing parts of the baseline firmware. These resource usages are estimates from a Vivado synthesis of the networks and the latencies are from a C-Simulation of the logic using ModelSim.

(a) Resource usage and latencies for the weight neural network.

Weight Network	Latency (clocks)	II (clocks)	LUTs%	DSPs%	BRAMs%	FFs%
Floating Point	8	3	21.1	62.1	0.0	5.4
Quantised	10	4	13.5	9.7	0.0	5.4
Quantised & Pruned	8	1	4.9	0.0	0.0	1.6
Vitis HLS	8	1	4.3	0.0	0.0	2.2

(b) Resource usage and latencies for the pattern neural network. This network is not pruned owing to the already small kernel size; so no pruned figures are shown.

Pattern Network	Latency (clocks)	II (clocks)	LUTs%	DSPs%	BRAMs%	FFs%
Floating Point	8	8	5.9	5.6	0.00	2.4
Quantised	8	8	6.1	0.0	0.00	2.6
Vitis HLS	8	8	6.1	2.8	0.00	3.0

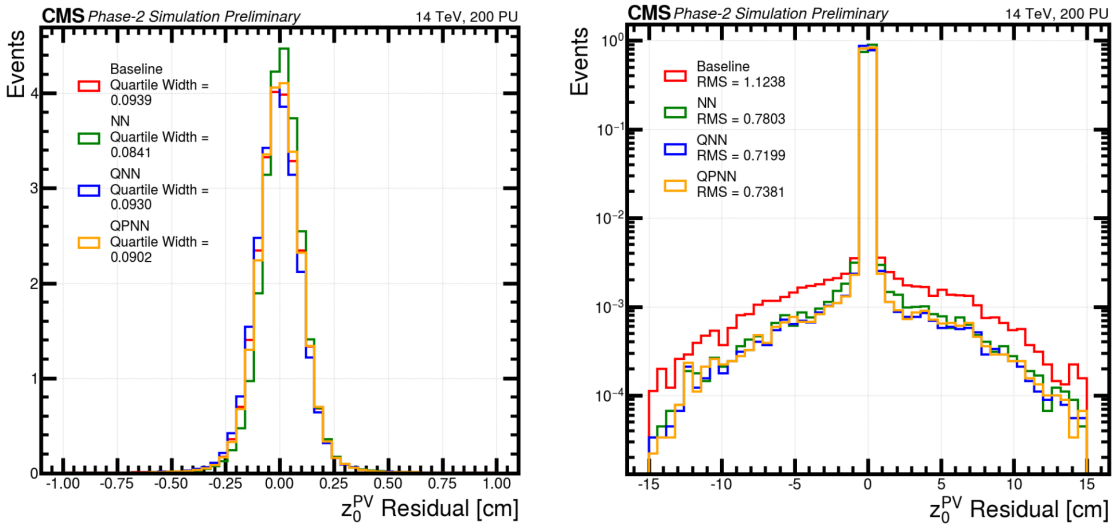
(c) Resource usage and latencies for the association neural network.

Association Network	Latency (clocks)	II (clocks)	LUTs%	DSPs%	BRAMs%	FFs%
Floating Point	13	1	45.9	256.5	0.0	10.8
Quantised	11	1	42.1	22.7	0.0	10.8
Quantised & Pruned	7	1	21.6	0.0	0.0	5.9
Vitis HLS	9	1	21.1	0.0	0.0	7.0

The NN approach to track-to-vertex association also outperforms the baseline, shown in figure 5.16. For the same 93% efficiency as the baseline, the NN approach achieves a 5% FPR, alternatively for the same 10% FPR the NN achieves a 96% TPR. As is shown in figures 5.15a, 5.15b and 5.16 the quantisation and pruning of the networks makes a small difference to the overall performance in their respective physics metrics but as was shown in table 5.2 there is a large reduction in resources. The performance of the end-to-end approach is a mixture of a knock-on effect of the improved vertex finding; fewer incorrect vertices so track matching is overall better across the entire dataset, and a more optimised relationship between track Δz_0,track η and the other track features.

This improvement in performance does impact the resource usage of the GTT as a whole. The resource and latency figures shown in table 5.2 are additional to those shown in table 5.1. The correlator layers expect to receive a vertex within 250 ns. This was easily achieved by the previous baseline approach which returns a vertex within 100 ns when running at 360 MHz. The additional NNs for vertex finding introduce an additional 30 ns when taking into account the required feature transformation and input pipelining. This is still well below the 250 ns threshold required for the vertex. Overall, the resource usage of the baseline vertex finding is low, it fits easily within the

VU13P floorplan at only 8% of the LUTs used for all three GTT copies. When adding the NNs this increases to 40% of the LUTs and 6% of the DSPs which is still far below the total resources on the VU13P, even with the additional core GTT algorithms.



(a) (b)

Figure 5.15: Neural network z_0 residuals for both the floating point (NN), quantised (QNN) and pruned (QPNN) neural networks versus the baseline FH approach. Figure (a) on the left shows the core of the residual, performance here is driven by the z_0 resolution of the tracks. Figure (b) on the right shows the tails of the residual, performance is impacted by the presence of fake tracks. The NN approaches are slightly better than the baseline approach in the core of the residual, shown in figure (a) and significantly better in the tails of the residual in (b). The differences between the NNs is down to the subsequent training rounds they undergo, the QNN has double the training that the floating point network has and the QPNN has eight extra rounds of training while it is pruning.

5.4 Track-Based Missing Transverse Energy

The track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation is a core GTT algorithm and is a simple vector sum of track p_{T}. The calculation looks for observable missing energy in the transverse plane. The transverse plane is used as opposed to the total energy in the entire event for two reasons. Firstly, the CMS detector is hermetic in ϕ but not in η and large amounts of unreconstructable energy is lost down the beampipe. Secondly, due to the colliding proton bunches travelling along the beampipe their final momentum after the collision is unknown so conservation of momentum in the longitudinal direction cannot be used. The missing transverse energy sum is expected to be zero because of the conservation of momentum in the r–ϕ plane, however there are a number of reasons why this is not the case. The most obvious being with unobservable particles in the detector, i.e. neutrinos that do not deposit energy in the CMS detector and so appear as missing energy. These can be inferred through other decay products that are detected, such as those signalling electroweak physics—W and Z bosons. Another reason for missing E_{T} could be new physics; any large missing E_{T} without the reconstruction of the electroweak process of a neutrino being produced ($\mathrm{Z} \rightarrow \nu^{+}\nu^{-}$ or $\mathrm{W} \rightarrow \ell\nu$) could signal an undetected particle. These are of interest as they could be the signal of dark matter production in a hadron collider.

While track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ is an interesting quantity as it can describe the overall charged momentum in an event it is not used as a direct trigger for a number of reasons. The main one being that there are other particles that do not leave tracks, i.e. uncharged particles that deposit energy in the calorimeters; photons and neutral hadrons.

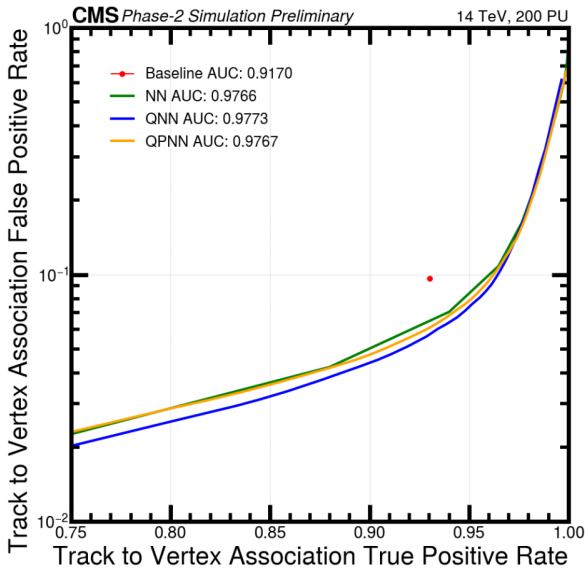
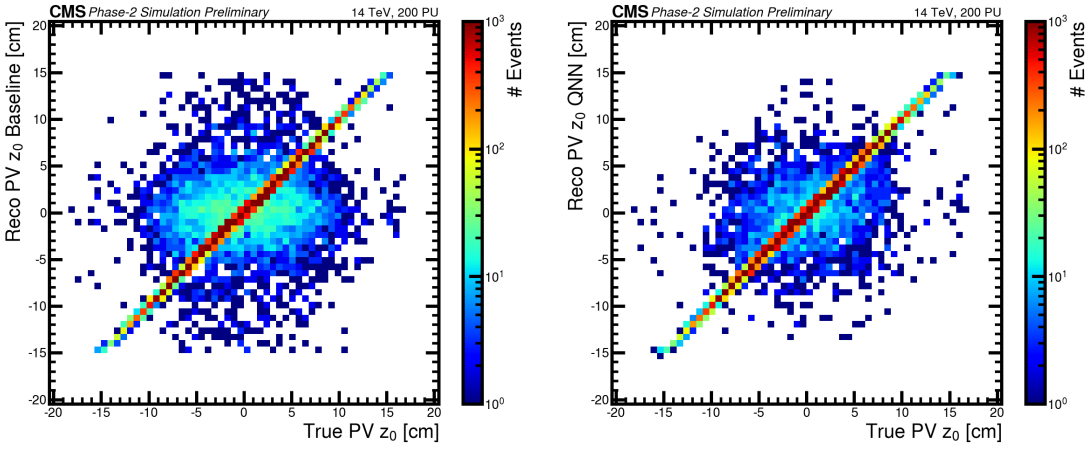


Figure 5.16: Receiver operating characteristic (ROC) curve for the neural network approaches. Shown in red is the baseline cut-based approach. Given in the legend is the area underneath the curve, commonly used as a metric for classification performance, values closer to 1 are considered better classifiers. The baseline approach area is calculated as a triangle from the origin, to the point and then to (1,1). As the baseline approach is a single cut, it is a single point in the ROC space. The NNs in green, blue, and orange have flexible decision thresholds and have different FPR and TPR values based on what threshold is cut on and perform roughly equivalent. All NN approaches outperform the baseline approach with better TPRs for the same FPR and vice versa.



(a) (b)

Figure 5.17: Two-dimensional histogram plots showing the spread of the reconstructed versus true primary vertices. The diagonal strip shows the core of the residual where the reconstruction is accurate. The area around the strip shows the tail distribution. Shown on the left is the baseline approach, the right the quantised and pruned NN. When comparing baseline to NN, it is clear the spread of incorrectly regressed vertices falls. It can also be seen in the cyan oval in the baseline figure that the baseline approach returns 0 for the vertex more than is expected. A bias that is not present in the NN approach, which is erroneous in a more symmetric way.

Other reasons for not using track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ would be tracks below the 2 GeV p_{T} threshold, though it is unlikely they will cause significant missing E_{T} as it is expected the low p_{T} pileup tracks are homogenous in ϕ.

PUPPI-based $E_{\mathrm{T}}^{\mathrm{miss}}$ will be used as a trigger path with a threshold of 200 GeV giving a rate of 18 kHz, taken

from the simplified L1 trigger menu in [57]. In extreme high pileup scenarios, PU > 300, correlator $E_{\mathrm{T}}^{\mathrm{miss}}$ becomes significantly worse because of the additional energy deposition in the calorimeters, which is difficult to isolate from the PV energy in an event. In these scenarios, track $E_{\mathrm{T}}^{\mathrm{miss}}$ can be used as a tool to compare to the correlator version though the effectiveness of tracking and track-based quantities is not fully understood in these extreme pileup scenarios. For general operation the Track $E_{\mathrm{T}}^{\mathrm{miss}}$ is simply used as a cross-check for the GT and could potentially be used in more complex algorithms in the future.

There are various ways to improve tracker $E_{\mathrm{T}}^{\mathrm{miss}}$, the first being the vetoing of fake tracks. As previously discussed in chapter 4, these tracks can be high p_{T} and the inclusion of such tracks in a missing energy calculation can change the result entirely, especially when thresholds for $E_{\mathrm{T}}^{\mathrm{miss}}$ are of the same order of the p_{T} of these tracks. Effective fake track removal is vital for improving $E_{\mathrm{T}}^{\mathrm{miss}}$. Another important aspect of the $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation is the vetoing of pileup, as discussed in the previous sections this requires a vertex and the association of tracks to this vertex. While a cut-based approach is an effective baseline, the improved vertex finding and track-to-vertex association of the end-to-end approach improves $E_{\mathrm{T}}^{\mathrm{miss}}$ even further.

5.4.1 Track Quality

The baseline approach to track quality is simple cuts on a variety of track parameters as detailed in table 5.3.

Table 5.3: Track quality parameter thresholds for track-based $E_{\mathrm{T}}^{\mathrm{miss}}$. For a detailed discussion of each of these parameters, see section 4.4.1.

Parameter	Threshold
p_{T}	≤ 2 GeV
$\lvert\eta\rvert$	≤ 2.4
N_{stubs}	≥ 4
χ^2_{bend}	≤ 4
χ^2_{rz}	≤ 9
$\chi^2_{r\phi}$	≤ 12

The inclusion of both track quality cuts and track-to-vertex association improves the track $E_{\mathrm{T}}^{\mathrm{miss}}$ rate for minimum bias samples, dropping the L1 threshold at a rate of 35 kHz from 250 GeV to 70 GeV. Just the track-to-vertex association sees a drop from 250 GeV to 200 GeV. This is shown in figure 5.18 when comparing χ^2 cuts in blue to the no selection in pink and z_0 cuts only in red. When using the BDT for track quality estimation it can be seen a similar trigger rate can be achieved.

Again, when considering the track $E_{\mathrm{T}}^{\mathrm{miss}}$ turn-on curve it can be seen in figure 5.19 that the inclusion of the z_0 cuts can slightly improve the turn on when triggering on SUSY events but the inclusion of the χ^2 cuts can greatly improve performance. The use of the BDT slightly improves the trigger turn-on but not by a large margin. However, this plot also highlights the issue with the track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation that even with highly effective cuts on fake tracks the 95% efficiency level cannot be reached. As this is compared to an offline $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation taken from the true primary vertex tracks the reason for this poor performance is the vertex reconstruction and track-to-vertex association performance not being effective, especially in large missing E_{T} events.

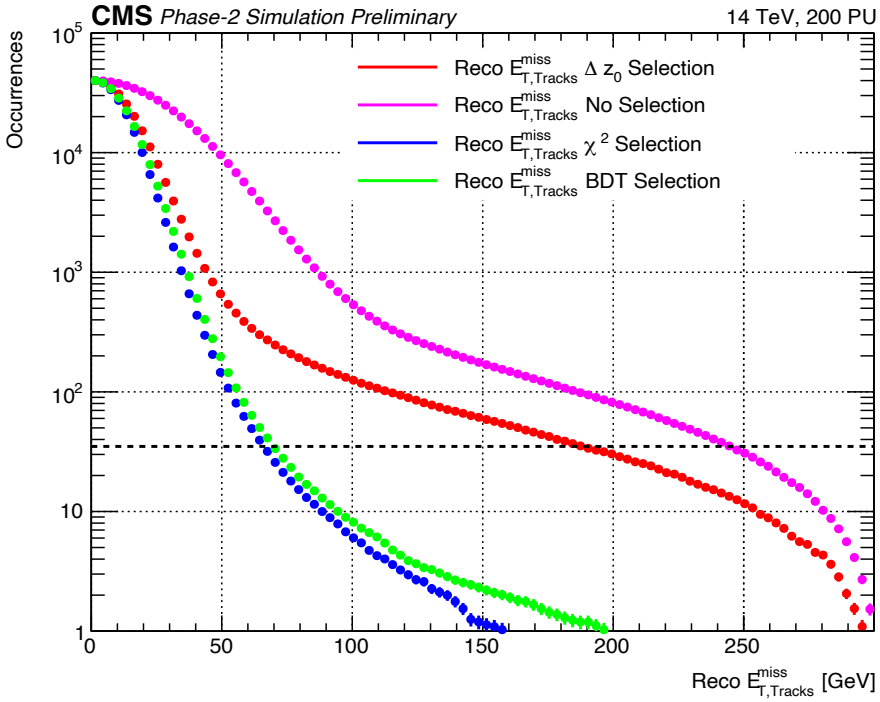


Figure 5.18: Track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ trigger rate in a minimum-bias sample with different track selections performed before reconstruction. In pink, no selection is performed. In red, selection based only on the track distance to the reconstructed PV is performed. In blue, the cuts in table 5.3 are performed. In green, a cut on the BDT output is performed with a threshold of 0.85. Associating tracks to the vertex has a small effect as in general the pileup is isotropic in ϕ and with low p_{T}, so sums to close to zero. A large difference is seen when applying quality cuts as fake tracks are not isotropic and can have high p_{T}. The black line shows a 35 kHz rate showing that at a fixed rate the quality cuts have a far lower threshold for triggering meaning a better selection can be performed.

5.4.2 End-to-End Performance

The use of the end-to-end approach to vertex finding and track-to-vertex association has been shown to perform better at finding the correct vertex and finding the tracks associated to the vertex. This translates to ultimately better performance in the track $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation. Figure 5.20a shows that the NN and QNN approaches can reduce the large tail in the resolution; the tail is due to low $E_{\mathrm{T}}^{\mathrm{miss}}$ events where the error between reconstructed and true $E_{\mathrm{T}}^{\mathrm{miss}}$ is very large in comparison to the actual $E_{\mathrm{T}}^{\mathrm{miss}}$. This is improved by more effective fake track removal and better track-to-vertex association in these events where single fakes or low numbers of incorrectly matched tracks makes a large difference. The peak of the resolution is also improved by the NN approaches. These events are typically higher $E_{\mathrm{T}}^{\mathrm{miss}}$ events where the error is relatively low compared to the true $E_{\mathrm{T}}^{\mathrm{miss}}$. This is improved by the NN approaches because of generally better performance at matching tracks to the vertex correctly. While the vertex was probably correctly located by the baseline approach and the $E_{\mathrm{T}}^{\mathrm{miss}}$ is generally well reconstructed, slight improvements of the NN approaches in matching every track to the vertex see this improvement in the 0.0 resolution bin.

5.5 Conclusions

The GTT is a core part of the Phase-2 CMS L1 trigger with its function in finding the primary vertex. This calculation allows tracks and object association, and the subsequent vetoing of pileup in downstream calculations.

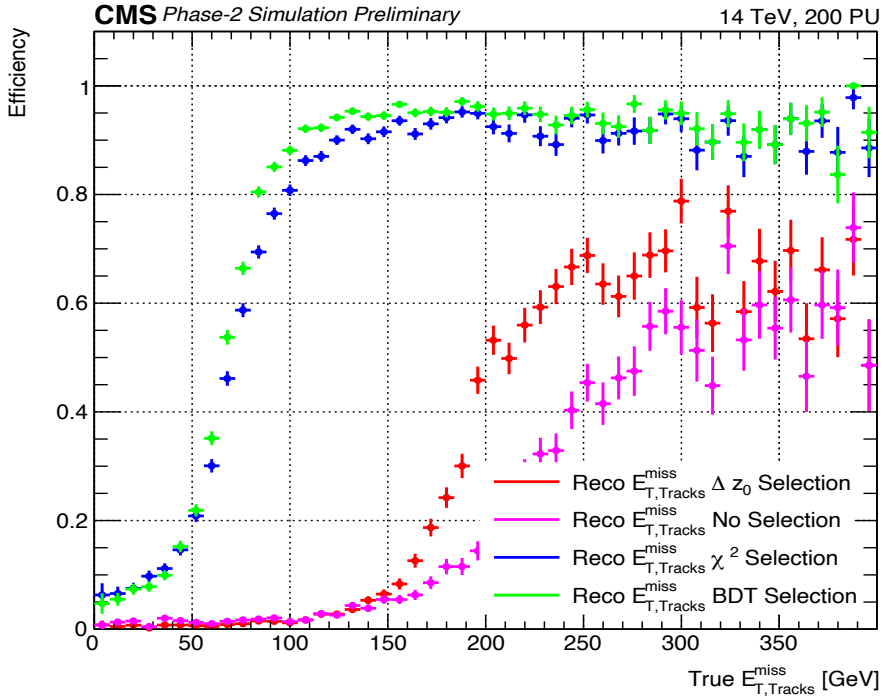
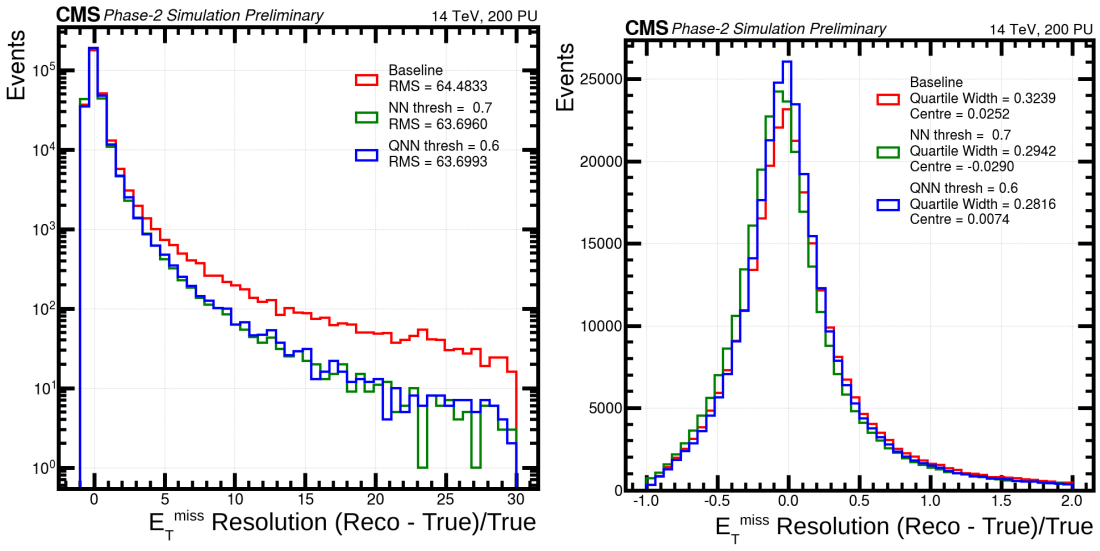


Figure 5.19: Track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ trigger turn-on curve in a SUSY sample with a fixed rate of 35 kHz. Efficiency for the reconstructed $E_{\mathrm{T}}^{\mathrm{miss}}$ is calculated using the $E_{\mathrm{T}}^{\mathrm{miss}}$ for truth matched primary vertex tracks in order to directly compare the reasonable best case for track-based $E_{\mathrm{T}}^{\mathrm{miss}}$. Different cuts are performed before reconstruction, as described in figure 5.18. The pink line demonstrates that simply summing all the tracks in an event can only accurately reconstruct the $E_{\mathrm{T}}^{\mathrm{miss}}$ half the time if it is greater than 300 GeV, the other half a high p_{T} fake causes the $E_{\mathrm{T}}^{\mathrm{miss}}$ calculation to be poor. This is similar for selecting tracks based on their Δz_0 in red, but the turn-on happens earlier at 200 GeV meaning at least some of those fakes can be cut away by only selecting some tracks. The quality cuts have a large effect showing 95 % efficiency for $E_{\mathrm{T}}^{\mathrm{miss}} > 100$ GeV because they can effectively remove fake tracks from the calculations.

Not only does this make complex algorithms, such as particle flow, possible because of a reduced set of inputs but also allows background to various trigger calculations to be reduced. There is a strict set of latency and resource requirements for the primary vertex finding which dictate the design of primary vertex finding algorithms.

The baseline FH approach was designed to meet these requirements and performs well with a ∼1 mm vertex resolution, ∼100 ns latency, and a sub 8% resource usage of the target of one SLR of a VU13P FPGA. The track-to-vertex association uses simple η-based Δz_0 cuts to account for the changing track z_0 resolution and achieves a 91% efficiency for finding true primary vertex tracks but with a 10% false positive rate. There are key areas where this baseline approach can be improved. The first is using quality cuts to reduce the impact of fake tracks on the calculation, using either χ^2 cuts or BDT-based quality cuts the vertex finding performance in the tails of the residual improves by 22%. Another area for improvement is in the core of the residual, when taking into account the relationship of track resolution on η and re-weighting tracks with this the core of the residual reduces by 10% but with the additional affect of widening the tails of the residual. Combining both track quality and this η relationship into one weighting function does not improve FH, which indicates a more complex relationship of track parameters is needed. Also, the firmware implementation of a complex polynomial is not trivial and the polynomial decided on may not be optimal for a variety of event topologies.

To have the improved performance of a more complex vertex reconstruction without the firmware issue an NN to guide the generation of an ideal function was used. This not only allows an ideal weighting function to be



(a) (b)

Figure 5.20: Track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ Resolution plots comparing the baseline approach using the FH vertex finding, cut-based track-to-vertex association and χ^2 quality cuts and the NN approaches in green and blue. On the left the logarithmic plot highlights the tails whereas the non-logarithmic plot on the right shows the core peak of the residual. The long tail in figure (a) is due to events where the reconstructed $E_{\mathrm{T}}^{\mathrm{miss}}$ is 5-30 times higher than the true $E_{\mathrm{T}}^{\mathrm{miss}}$. This is due to a low true $E_{\mathrm{T}}^{\mathrm{miss}}$ and a large p_{T} discrepancies such as from fake tracks or from a few poorly associated tracks. Figure (b) shows the core of the resolution where the predicted and true $E_{\mathrm{T}}^{\mathrm{miss}}$ are high and similar, any small improvement to vertex location and track-to-vertex association will improve the performance.

optimised for vertex finding performance over a range of datasets, but also simplifies the firmware design process using tools such as hls4ml that allow complex NNs to be transpiled without needing advanced firmware design. The idea of using NNs was expanded to include a CNN for vertex peak finding and a DNN for track-to-vertex association. These networks, while training on separate tasks, benefit from the knowledge gained by the other networks and so an end-to-end training approach with two loss functions was used. This outperforms the baseline approach, reducing the tails of the residual by 35% and dropping the track-to-vertex false positive rate to 6%. The network, on account of its size and complexity, is pruned and quantised in a number of steps to reduce its firmware footprint and significant savings can be made without impacting vertex finding and association performance. The firmware implementation uses hls4ml with custom histogramming firmware. The full chain of weight and pattern networks take 150 ns to make an inference and increase the resource usage to less than 50% of the target FPGA SLR.

Another algorithm in the GTT is track-based $E_{\mathrm{T}}^{\mathrm{miss}}$, a calculation that has been shown to be vulnerable to fake tracks and so quality cuts are imposed. When using χ^2 and BDT quality cuts the trigger rate reduces and efficiency increases, however the two methods have similar performance and overall have poor performance from a trigger perspective. This is due to the poor performance of the baseline vertex finding and track-to-vertex association. Using the NN approach to find trigger rate and efficiency is yet to be completed, however preliminary studies of $E_{\mathrm{T}}^{\mathrm{miss}}$ resolution indicate the NN approach can outperform the baseline approach. More studies are needed to understand the usefulness of the improved track-to-vertex performance downstream; other users such as PUPPI or track isolation triggers may see better performance, which directly impacts trigger thresholds.

Chapter 6

Demonstrators

6.1 Introduction

A major part of the Phase-2 upgrade of the CMS Level-1 trigger is the testing of algorithms on hardware with bit-accurate inputs. Not only will this ensure that the algorithm is performing as expected without losing data or incurring additional latency, it also allows prototype hardware to be tested and refined.
There are key tests that need to be performed for all algorithms in the L1 trigger:

1. Does the firmware simulation match emulation?
2. Does the firmware meet timing requirements and fit within the target FPGA?
3. Does the firmware, when implemented in the FPGA, match simulation and emulation? Does it have the expected latency?
4. Does the firmware match emulation when part of a wider systems integration test?

In general there is no one solution to debugging the firmware so that it meets all these requirements. However, there are some common techniques, and the process with some key strategies are listed below. All firmware builds for all parts of the tests detailed in the rest of the chapter went through this process.

Test 1

In order to solve issues with firmware emulation mismatches (test 1) the main strategy is to inspect the firmware at a clock cycle level. This is done with a C-Simulation of the firmware, in this case ModelSim. It gives signal traces of every internal firmware signal, which can be matched to individual tracks or objects in the emulation. A number of issues can be found when observing these traces. One being incorrect synchronisation of signals, where some calculation in the firmware has a different latency to another but require synchronised outputs. This can be solved by adding internal pipelines to the firmware to slow signals by a number of clock cycles. Another possible issue would be if there is mismatch in the values of parameters in the firmware compared to emulation. The most obvious reason for this is simply the calculations being incorrect, which is typically simple to solve. More complex issues can occur due to rounding or truncation occurring in the firmware with restricted bitwidth of variables, which is not being fully emulated. This can be solved by changing the internal bitwidths in firmware or accurately implementing the effect in emulation. The process of firmware to emulation matching normally starts with every track being incorrect and obvious large issues being solved such as synchronisation errors and then edge cases being discovered when 1000s of tracks are passed through the firmware. These edge cases tend to be

truncation and rounding errors and require careful scrutiny of every part of a chain of calculations to find the errors.

Test 2

Issues of a firmware build not meeting timing (test 2) occur during the implementation phase of the build where a design tool aims to physically map the logic generated by the synthesis step onto the gates on an FPGA. To solve these issues the problem is diagnosed using the Vivado design tool which highlights which paths within the firmware have the Worst Negative Slack (WNS). The WNS is the path that has the highest latency for a signal to pass through it and is expressed as how far from reaching the desired clock frequency a signal is. For example, with a 360 MHz clock frequency, a single clock cycle takes 2.7 ns. If a signal within the firmware takes 3 ns to get from where it is at the start of the clock cycle to where it needs to be at the end of the clock cycle the WNS is -0.3 ns and the design has not met timing.

Many cases of not meeting timing are seen when a calculation has been written to occur too quickly for the logic to run. For example, a common issue that occurs in DSPs is where the firmware design expects a calculation to be performed in one clock cycle but because of the logic implementation in hardware the DSP can only run in three clock cycles[1]. This issue of asking firmware to run too quickly does not give issues when running in simulation and so cannot be noticed when debugging in the previous step. It is only when the firmware is implemented that these issues arise and once a timing issue has been solved, normally by adding additional pipelining, the firmware needs to be rematched to emulation due to the change in signal synchronisation.

Another common timing issue is in the placement of firmware blocks within an FPGA. The physical path length that a signal needs to pass through in logic incurs a latency due to the speed of the electrical connection. If this is too long or passes through too many layers of logic the design will not meet timing. This can be particularly troublesome when a design crosses between SLRs. Due to the placement of logic gates in the FPGA silicon there are specific pathways for signals to pass between SLRs which are slower than signals passing within the SLR. To solve this, placement of firmware blocks on the physical FPGA is needed and observing floorplans, such as those shown in figure 6.7, is important for understanding how the design tool has placed the firmware blocks. Specific constraints can be designed for a firmware build such as restricting firmware to one SLR.

A final technique for meeting timing requirements is to include additional objectives for the design tool. The translation of firmware to logic gates and placement of a design on the FPGA itself can optimise different things, for example the Vivado design tool may optimise overall resource usage rather than timing when placing firmware and there are specific objectives such as additional re-timing and re-placement which can be used to optimise the design for timing. Once a design has met timing it can be loaded onto an FPGA and run in hardware.

Test 3

Once firmware has been loaded onto the FPGA and tested it can occasionally differ from a firmware simulation (test 3). In theory, a firmware simulation should be exactly the same as the physical FPGA as the simulation is modelling every logic gate. However, one of the key differences is that a simulation begins the signals in its simulated FPGA at clock cycle one of the simulation. In a real FPGA, which has been running for up to days at a time, there are internal signals that have been changing since the FPGA was turned on, so there can be a mismatch

[1]DSP blocks are implemented in three separate operations, a pre-adder, a multiplier and a final add/subtract accumulator [48].

between internal signals that are used to trigger dataflow. This is solvable by using reset signals and known signals that occur in the input data. For example, all input pattern files have start of orbit (a flag that the LHC will provide to the CMS trigger) and start of event flags within their link data. These can be used to trigger the start of data running in a firmware design and ensure that the internal state of the firmware is the same at every clock cycle for both simulation and hardware.

There can be other issues with firmware not having a stable starting state; in simulation non-initialised signals will automatically be zero but in hardware they may not be, and this introduces issues especially at the start of running firmware. This can be difficult to diagnose and typically would need additional writing of internal signals out to files for in-depth debugging as without specific debugging tools it is not possible to observe the internal signals once the firmware is in the FPGA. Solutions to this can be to define an initial state for the firmware that is reproducible, though with complex designs this can be difficult, or to allow the firmware to reach a steady state by introducing a multi clock cycle latency at the beginning of running. These types of issues in hardware to simulation mismatch can be time-consuming to debug because of the multiple hour process of synthesising, implementing, and running the firmware on hardware.

Test 4

The final step of ensuring correct function of the firmware is to use a wider systems integration test (test 4). In these cases it is not typically the firmware that is providing the source of the problem owing to the extensive testing that has been carried out previously. The only significant issue can be in the correct setting up of input and output buffers for the firmware, which is not always straightforward because the buffers in firmware do not necessarily match the optical transceivers in hardware. For this, specific link mapping needs to be created by testing every link and confirming where the data is being transferred from and to within the firmware. As the L1 trigger project is in a relatively early stage, the setting up of hardware for link testing has not been fully streamlined and while large progress has been made on the development of a common online software for board and testing control, the process of link mapping was still performed manually. In the future it is expected that this will be automatic, and the firmware engineer will not need to worry about how the hardware is set up to allow for maximum flexibility. There can be other issues when setting up hardware for link testing such as synchronisation between the boards not being set up correctly and the shared clock source not working as intended. Again, this is not an issue with the firmware itself but requires specific debugging and setting up of the hardware; in the future this will be streamlined and automated.

The specific hardware that was used for both single board testing (test 3) and multi-board testing (test 4) is listed below. The board agnostic extensible, modular, data processor (EMP) Framework used to simplify the design and testing process is also described.

6.2 Hardware

6.2.1 Apollo

The Apollo platform [67] is a two-part advanced telecommunications computing architecture (ATCA) blade, which will be used for the TFP (and the IT backend electronics), consisting of a service module (SM) and a command module (CM). The SM handles the ATCA communications, clock distribution and power delivery to the other module; the CM. A revision 1 Apollo board with both the SM and CM is shown in figure 6.1a.

The CM is an interchangeable module that slots into the SM that hosts the FPGA for algorithms and optical transceivers for data flow. In this design the FPGA is directly soldered onto the CM PCB giving a higher stability for data being transferred to the optical receivers, also on the CM. For the track finder project it is envisaged that a CM hosting two Xilinx Virtex Ultrascale+ VU13Ps will be used with the algorithm split between the two FPGAs. The TFPs will use 162 Apollo boards to cover the entire outer tracker. For the track finder tests, detailed below, a previous generation (revision 1) Apollo was used hosting a single VU7P FPGA with eight multi gigabit transceivers (MGTs) providing up to 32 links to other boards.

The final TFP will be implemented on the revision 2 Apollo which is able to host the larger VU13P FPGAs as well as having more links between the two FPGA sites. Information learnt from the tests described below helped to refine the revision 2 Apollo; another key part of performing single board and multi-board integration tests.

6.2.2 Serenity

The Serenity platform [68] comprises; an ATCA blade that handles the power distribution, clocking, optical interfaces, and an onboard CPU to control and manage the board, and FPGA daughter boards. Many different varieties of FPGA can be mounted onto these daughterboards which can be swapped on the main ATCA blade giving a high degree of flexibility for the platform. A two socket Serenity is shown in figure 6.1b

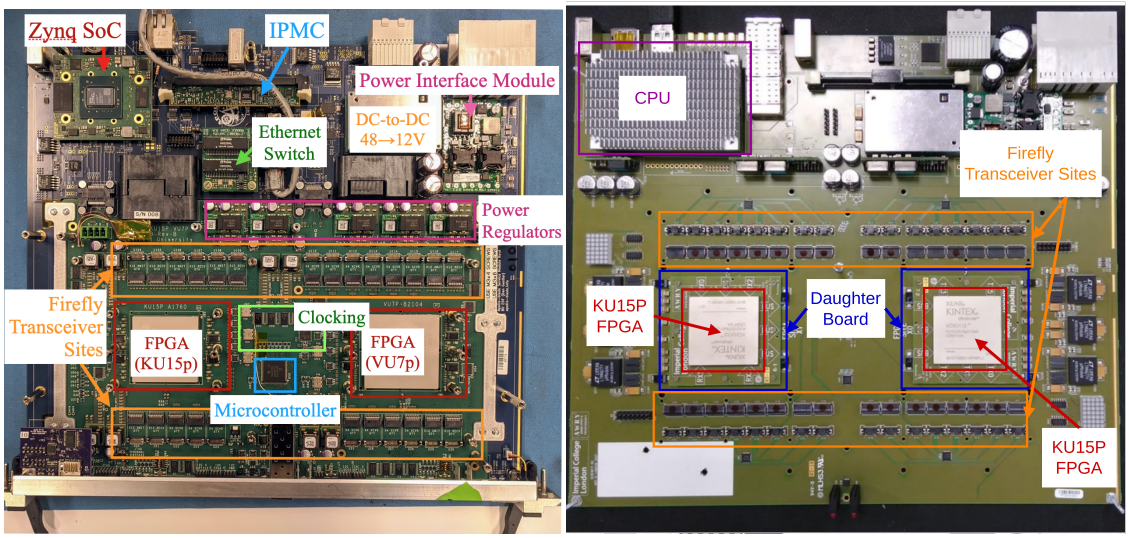
There are multiple revisions of the Serenity board with between one and two FPGAs. It is envisaged that half of the GTT project will use Serenities mounted with two VU13P FPGAs. However, there are further revisions to the specification of the GTT which could see the system using a single VU13P fixed to a Serenity board in order to simplify the system. As these boards are currently not available, the tests below used older specification Serenities with a variety of different Xilinx FPGAs mounted.

6.2.3 Extensible, Modular, Data Processor Framework

The EMP Framework is an infrastructure for FPGAs that separates board specific firmware such as clock infrastructure and I/O buffers from the algorithm. It is designed to simplify the design process by having generic interfaces for the central payload algorithm and specific firmware for individual boards. This allows firmware to be developed that works on multiple boards without the central payload firmware changing.

The framework provides input and output buffers which data can be loaded into and played through the algorithm as well as interfaces for MGTs meaning data can also be provided by fibre optic links allowing for board-to-board testing. The EMP framework has an accompanying software package that allows these buffers to be manipulated and pattern files to be loaded into the FPGA. These files have clock-by-clock data (64-bit hexadecimal packets) for every input buffer on the FPGA and provide a generic interface for comparing input and output to emulation.

There is also work ongoing on developing an online software that controls the boards while in operation, allowing firmware to be quickly loaded onto the boards and control data being passed into a board. This again will simplify large scale integration tests of algorithms.



(a) (b)

Figure 6.1: Figure (a) shows the Apollo revision 1 board with both the CM and SM. The CM hosts the FPGAs and optical transceivers (Fireflys [70]), it is the large green PCB shown in the photograph. The SM controls the power distribution, clocking and communications for the board, it is the blue PCB surrounding the CM. The CM is interchangeable depending on the use case of the board [117]. Figure (b) shows a two socket Serenity with the two separate daughterboards hosting the two KU15P FPGAs. The Firefly optics, power distribution, clocking and communications are handled by the rest of the ATCA blade with only the daughterboards being swappable, which allows different FPGAs to be mounted to the board [68]. The two boards show some similarity in their form factor as dictated by the ATCA specification but the differing design principles of changeable modules for the FPGAs sets them apart.

6.3 Testing Setup

The first two tests of the firmware described above were performed on a standard PC (but with additional RAM to aid firmware development), these tests were performed manually for each firmware build as the firmware is in an early stage of development. The firmware testing and writing was performed alone with minimal guidance, but any issues were resolved with other project members.

It is envisaged that as the firmware becomes more mature, automatic tests and builds of the firmware should be developed making use of dedicated build clusters so that a version of the firmware for any given system is readily available to deploy. This should also include standard emulation–firmware tests to ensure no performance is being lost in each firmware iteration. This kind of infrastructure is essential for the final commissioning of the trigger system and will see rapid development in the coming years.

The other hardware tests, described below, were performed at the CMS tracker integration facility which contains the power and cooling infrastructure for the boards. This facility is used by many members of the CMS upgrade project in order to develop both the hardware and firmware for the Phase-2 project. The hardware tests themselves made use of existing board and firmware infrastructures. The Serenity and Apollo boards have board control software used to load firmware and play data as well as setting up physical fibre optic links. In firmware the EMP framework described above was extensively used throughout testing as it simplifies the data transfer within the firmware.

The hardware tests below were again performed alone with guidance from experts when specific issues arose;

all the setting up of boards and fibre optics were performed alone. This allowed fast progress through tests as different configurations could be trailed without issue. The single board and multi-board tests were performed manually with commands being sent to each board to set up buffers, clocks, MGTs etc.

In the future more standard testing workflows are needed and are currently under development. One of the major projects in development is the online software for controlling boards as part of the EMP framework software. This will allow all tests to be run through a single command with configuration files to indicate what firmware and which buffers etc. This utility will greatly simplify the board testing process and allow multiple files to be passed through each algorithm for full scale testing of firmware.

6.4 Track Finder Single Board Testing

As is discussed in chapter 4 there are many stages in the track finding algorithm, all of which are in active software and firmware development. This makes the testing of individual algorithm blocks and different subchains necessary to verify the firmware is operating as expected. The track finding algorithm is split into two parts; the tracklet track seeding stage written in both VHDL and Vivado HLS[2], and the Kalman Filter written in VHDL. Both of these parts have different testing performed. The tracklet firmware uses test vectors for each substage of the algorithm generated from the emulation which are automatically compared to the C-Simulation of the firmware as developments are made to both emulation and firmware. This simplifies the testing process and ensures continual firmware to emulation agreement. It has the disadvantage of needing test vectors creating which is a time-consuming process. The testing in hardware is also complicated by this approach as the generated HLS firmware does not contain structural firmware to handle input and output of data on a physical FPGA.

The Kalman Filter is written in VHDL and uses the EMP framework to handle data input and output of the firmware. In its current state the use of the EMP framework makes it difficult to run multiple pattern files through the buffers and MGTs; while many events can be loaded into the buffers by increasing their size this can only work up to the resources available on the FPGA and for large designs this is a limited approach. However, the use of the EMP framework simplifies the testing and allows for pattern files to be generated quickly for any event.

Figure 6.2 shows the general setup for single board algorithm testing for the track finder. An emulator is used to create input pattern files for the desired firmware and then the output pattern files can be compared to the emulated output. A C-simulation of the firmware is used to first verify the unsynthesised firmware. Once synthesised and implemented on the desired FPGA (after meeting timing) this firmware can then be loaded onto the Apollo board and tested.

6.4.1 Subchain Testing

Various track finder subchains were first tested to ensure they gave the correct output, each is listed in table 6.1.

The single input setups in subchains $1, 3$ and 4 in table 6.1 were designed to test individual modules in the skinny chain setup as detailed in chapter 4. This allowed a simplified version to be first tested before more complex scenarios. For these tests the KF output step does no routing of tracks from the KF as only positive η tracks are created. The test in 2 saw a full set of KF workers as would be expected for the full chain setup leading into the KF output which then routes the tracks based on η. The final test in subchain 5 was designed to create a

[2]VHDL is used in top level control logic and in some memory implementation due to the low-level control available

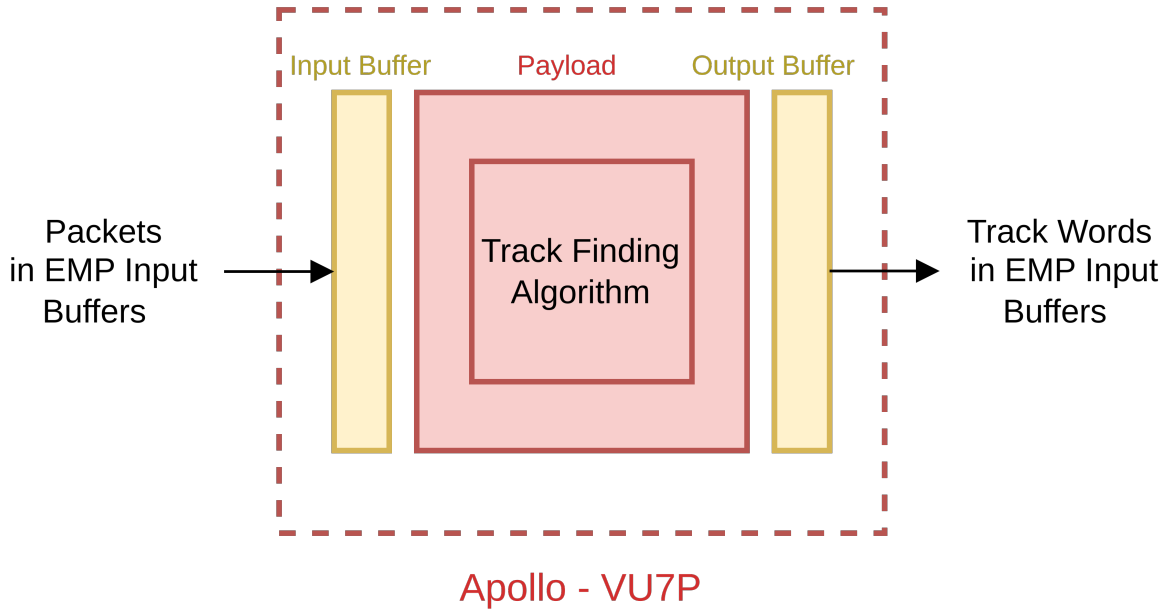


Figure 6.2: The general setup for single board algorithm testing for track finder algorithms. Input files are loaded onto the Apollo board using the online software which are then loaded into the EMP input buffers. These input files varied in size depending on the algorithm and their structure for each TF sub-algorithm is listed in table 6.1. The EMP framework pushes data from the input buffers through the algorithm which outputs into the output buffers for the online software to read out to files. The algorithm under test was changed with the list of algorithms and expected outputs shown in table 6.1.

full event's worth of tracks, all detector nonants of the KF output running on the same FPGA. While this is an unrealistic scenario for the real tracker system, it serves the purpose of generating all nonants of tracks to be output into buffers or across fibre optics. This allows GTT algorithms to be tested in board-to-board integration tests, which need all nonants of tracks.

All tests followed the same overall structure. Firstly, the firmware was for each algorithm was implemented within the payload block of the EMP framework. This required ensuring the correct input and output buffers were being used. Once complete the firmware was simulated using ModelSim and compared to the C++ emulation. Both the firmware simulation and emulation used EMP pattern files as input and output, with output files compared to find differences between the two. At this point corrections were made to each algorithm if they did not match emulation, typically this was preformed at a single clock cycle level ensuring every internal signal within the firmware was correct. Edge cases were found after running the simulation for many events. Once matching emulation, the firmware was synthesised and implemented for the desired board; in this step any timing issues were found and corrected. The built firmware was loaded into the Apollo board using the EMP software and pattern files played through the boards by loading pattern files into the EMP buffers. The output pattern files were read out of the EMP output buffers and compared to emulation and firmware simulation.

Figures 6.3a–6.3h show the result of testing ~1000 events in emulation and hardware (after verifying C-simulation matching) of the full KFin–KF–KF output chain (subchain 4 in table 6.1). As can be seen in these figures there is perfect agreement in all the different track parameters when comparing output trackwords. This demonstrates the firmware is operating as expected when in a physical board. For this reason the results of the other tests in table 6.1 are not shown as they are all shorter chain tests and have the same level of agreement.

Table 6.1: Different track finder subchains that were tested including their input and output formats. The KF refers to the Kalman Filter where each track is formatted as a single 64-bit packet and each stub word (seven in total) is a 64-bit packet. The KFin track is the track that is created by the KF input module, which is used as input to the Kalman Filter. The TBout refers to the Track Builder output which is the final step of the tracklet firmware and is formatted by the KF input module. The track words used as output are the tracks 96-bit tracks specified in table 4.1 and split into 64-bit packets as per table 4.2.

Subchain	Input	No. Input Buffers	Algorithm	Output	Latency (cycles)
1	$1\times$ KF Tracks + Stubs	8	KF output	$2\times$ Track Words	19
2	$8\times$ KF Tracks + Stubs	64	KF output	$2\times$ Track Words	19
3	$1\times$ KFin Tracks + Stubs	8	KF + KF output	$2\times$ Track Words	287
4	$1\times$ TBout Tracks + Stubs	8	KFin + KF + KF output	$2\times$ Track Words	308
5	$9\times$ KF Tracks + Stubs	72	$9\times$ KF output	$18\times$ Track Words	19

6.4.2 Skinny Chain Testing

The full skinny chain testing saw an implementation of the tracklet firmware within the EMP framework. The floorplan of this implementation is shown in figure 6.4 where the relative sizes of the firmware blocks can be seen in comparison to the total VU7P on the Apollo revision 1. The majority of the algorithm is the tracklet firmware owing to its complexity.

The implementation of this firmware and subsequent loading onto an Apollo board gave the opportunity to play pattern files through the firmware in a simple way and compare to emulation. However, as can be seen in figure 6.5 there were significant issues with this tracklet implementation due to additional stubs and tracks being created by the tracklet firmware. This issue was mainly seen when transferring from firmware C-simulation to hardware testing, where the hardware disagrees with the firmware simulation. Figure 6.5 shows the effect of this discrepancy where a clear difference between hardware and emulation is seen. This can be isolated to the tracklet implementation when comparing figure 6.5 to 6.3c where perfect agreement was seen with just the KF chain. Work is ongoing to produce an implementation of the skinny chain firmware that does not see this discrepancy using more up-to-date versions of the tracklet firmware implemented using HLS over VHDL. While these tests have been successful and the discrepancy seen in the skinny chain output has been resolved, the exact issue leading to the previous problems seen in hardware running are not fully understood.

6.4.3 Track Quality Testing

The final addition to the KF output step is the track quality BDT described in chapter 4. This was tested in a similar way to the tests listed in section 6.1 but with the additional check of the BDT 3-bit output. The latency of the testing increased by 14 clock cycles in each test, as is expected by the latency of the BDT. Figure 6.3d shows the performance of the track quality BDT in hardware testing where it matches emulation 100% over ~1000 events in the full KF chain setup. The relatively small size of the BDT can be seen in figure 6.4, highlighted in orange, demonstrating the small impact this BDT has on the overall firmware size. This BDT size will not change with the full implementation of both the tracklet and Kalman Filter for the full outer tracker.

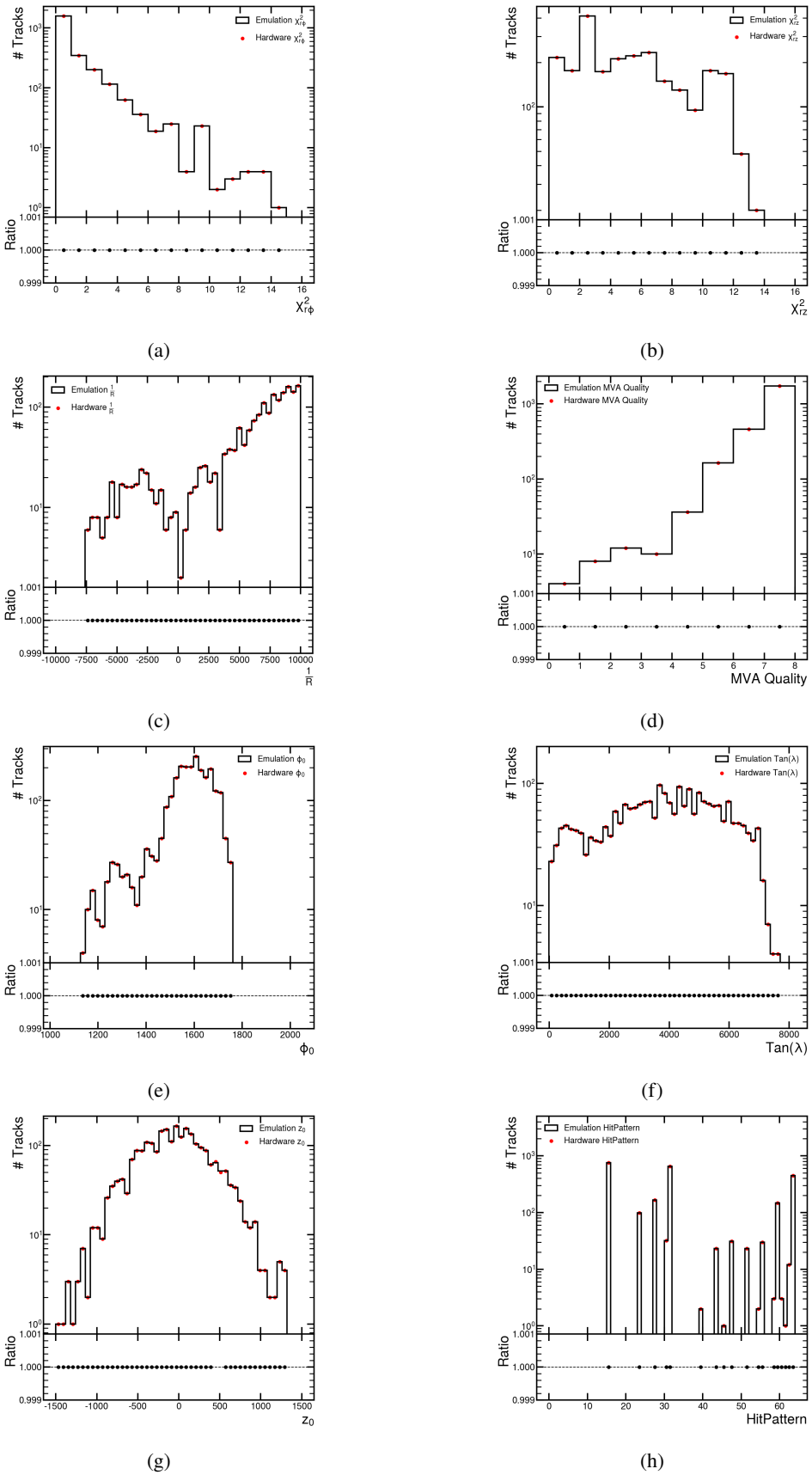


Figure 6.3: Hardware–emulation agreement for all track parameters using the KF chain setup, run over 1000 events on an Apollo revision 1 hosting a VU7P FPGA, showing perfect agreement between the hardware result and emulation result. As the firmware was tested in its skinny-chain set up not all variables have their full parameter ranges, for example the skinny chain runs in a single η region meaning the $\tan(\lambda)$ values are restricted.

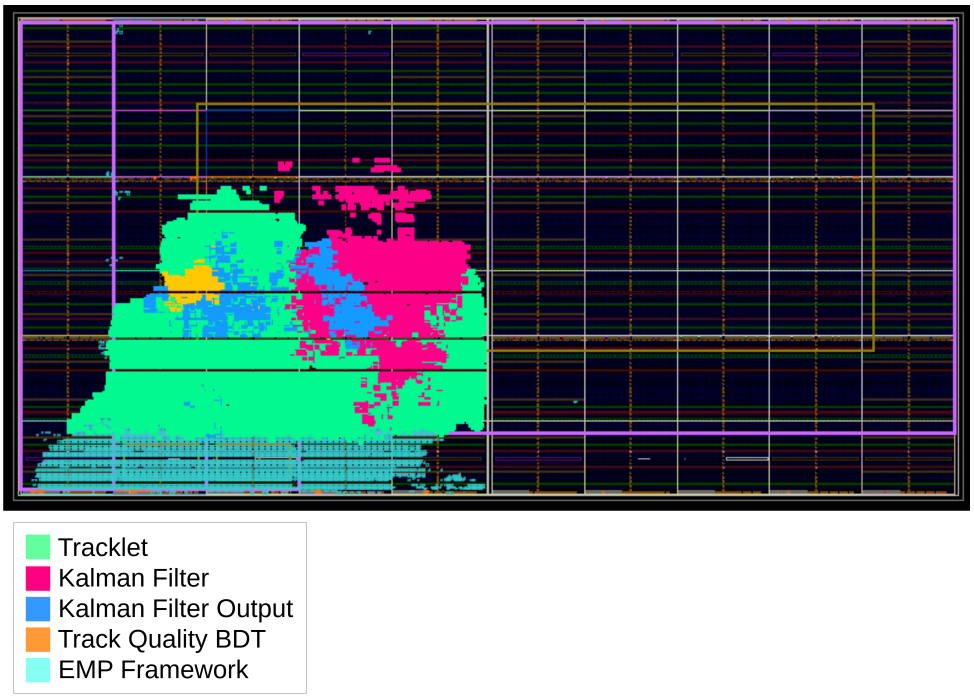


Figure 6.4: Floorplan of the Skinny Chain implemented on a VU7P. Highlighted are the individual parts of the hybrid algorithm as well as the EMP framework. The placement of the firmware block demonstrates how the implementation tool operates. The EMP framework firmware is close to the edge where the physical connections from MGTs to the FPGA are placed. The tracklet firmware is spread outwards from this as it needs to be connected to the input buffers. The KF sits close to the tracklet firmware with part of the KF output module within it; the track transformation step. The rest of the KF output module has the BDT within it and is running the routing. The final part of the KF output module links back to the EMP buffers but is hidden behind the Tracklet firmware, close inspection reveals some blue below the green.

6.5 GTT Single Board Testing

The GTT was also tested in single board setups. Due to the emulators for the GTT algorithms not being complete, multiple event testing was not possible. However, single events were tested, comparing individual parts of the firmware to emulated versions. This methodology is not sustainable for wider testing of the firmware but allowed for in-depth debugging of individual firmware blocks. Figure 6.6 shows the setup for single board testing of vertexing algorithms. The "GTT input" represents firmware used to translate the 64-bit packets provided by the track finder back to 96-bit track words. Part of this firmware block is the translation from $\frac{1}{R}$ to p_{T} and $\tan(\lambda)$ to η using LUTs.

6.5.1 End-to-End Vertexing Testing

The end-to-end vertexing neural network, described in chapter 5, was first tested using a ModelSim simulation and individual tracks compared between emulation and firmware ensuring the correct weighting was being generated by the weight network, and the correct convolved histogram was being outputted. Once this process was verified the network was implemented for a VU9P. The floorplan of the resulting implementation is shown in figure 6.7, where the networks easily fits within the envelope of one SLR. This firmware was used to create a bitfile that was loaded onto a Serenity with a VU9P daughter card and the firmware tested using the EMP software by loading pattern files into the input buffers of the FPGA. While a single test of one event gave the correct vertex being outputted, multiple events have yet to be tested due to the functionality of generating large sets of input pattern files and remaining issues with the CMSSW emulation of the vertex finding, which must be resolved before large scale testing can be performed.

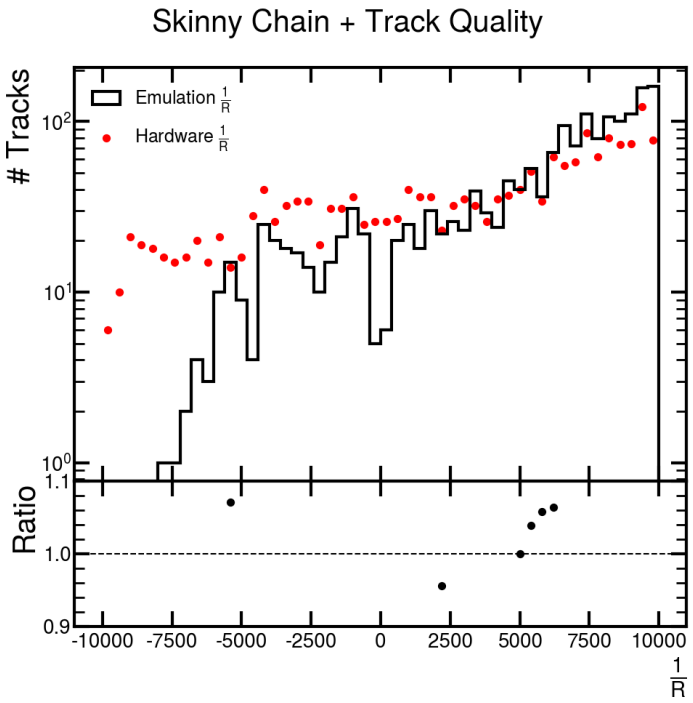


Figure 6.5: Hardware–emulator agreement for the Skinny chain in the $\frac{1}{R}$ track parameter run over 1000 events on an Apollo revision 1 hosting a VU7P FPGA, showing significant discrepancy between the hardware and emulation results. The hardware version of the skinny chain has additional tracks and stubs from unknown origin, this is leading to the increased number of tracks seen, especially at low $\frac{1}{R}$.

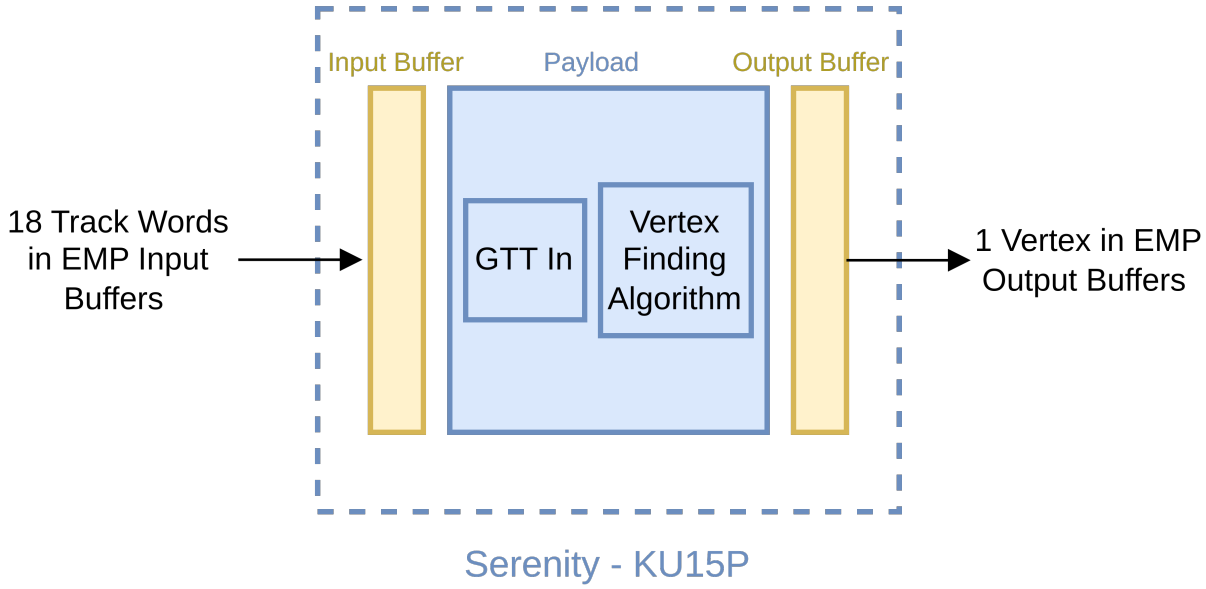


Figure 6.6: The general setup for single board algorithm testing for GTT algorithms. Pattern files with 18 input streams per clock cycle were loaded into the input buffers. The splitting of track words into these streams is described in Chapter 4. The algorithm under test was split into a GTT input step and vertex finding with an output of one vertex per event (every 108 clock cycles) expected in the output buffers.

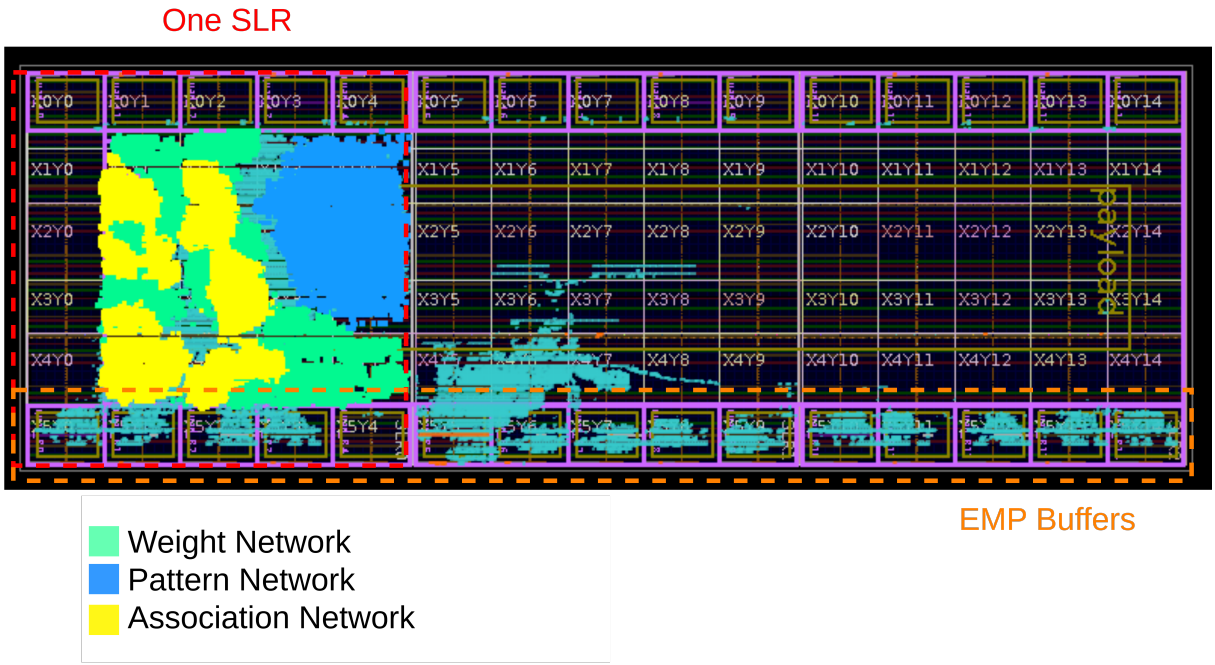


Figure 6.7: The floorplan of the end-to-end neural network as implemented on a VU9P, the weight network in green, pattern network in dark blue and association network in yellow. The weight and association networks are replicated 18 times in this firmware build for the 18 internal track streams. The pattern network has only one replication. The pattern network is relatively large because of the unrolled nature of the firmware implementation, the network has to process a 256-bin histogram in parallel resulting in a large spread of resources.

6.6 Integration Testing

Part of the commissioning of the L1 trigger is the testing of multiple algorithms from different subsystems with data being transferred between processing boards via fibre optic links.

The interface between the track finder and GTT was tested in a variety of ways. These tests were performed in the order listen below with the complexity of the test increasing each time.

1. Single board individual algorithm testing, as detailed above. (Figures 6.2 and 6.6)
2. Single board loopback testing; KF output and GTT input in the same board, data transferred by loopback fibre. (Figure 6.8)
3. Multiple board testing; single fibre optic connecting KF output to GTT input. (Figure 6.9)
4. Multiple board testing; 9 KF output instances to vertex finding. (Figure 6.10)

Single board tests (1) were performed to validate firmware performance in a standalone manner, these are described above. Single board loopback testing (2) was performed in order to remove the complication of synchronisation of timing between two boards, shown diagrammatically in figure 6.8. This test relies on internal clock generation in a single board but still includes a fibre optic connection. To the algorithms involved, the input and output are identical to a board-to-board test; full MGT datapaths are used between the algorithms. In the EMP framework this addition of MGTs is a simple flag in a top level file. In physical hardware testing the fibre optic is inserted into the board where each fibre optic is a 4-channel bidirectional cable. There are several extra steps involved in setting up a board for using fibre optics. Firstly, various flags are needed in the EMP software to set up the board for communication through the fibre optic, these ensure the synchronisation is correct and that the fibre optic cable is seated correctly. Secondly, when using physical fibre optics, a link mapping between the Apollo or

Serenity fibre optic plugs and the internal EMP MGT signals is needed. This can be performed by using a null algorithm in the payload and observing exactly where data is routed to within the input and output buffers. Once the null algorithm performance has been verified, a real algorithm can be used. The loopback test in (2) used an EMP input pattern file in the standard EMP buffers; data was played through the KF output step and outputted into an EMP MGT. This MGT sent data through the fibre optic which was inserted back into the same board. The second half of the algorithm, the GTT input step, used the EMP MGT as input with the data coming from the fibre optic being passed through this part of the algorithm which outputted into the EMP output buffers.

This test proved successful, but additional latencies were needed for the algorithms due to the path length of the fibre optic cable. The GTT input step looked for "start of event" flags in its input data stream, these flags were placed in the output datastream by the KF output and signalled when the GTT input should start processing. These kinds of flags are essential in fibre optic testing and the wider trigger system as the actual latency from the fibre optic is unknown so using counters could lead to desynchronisation between algorithms.

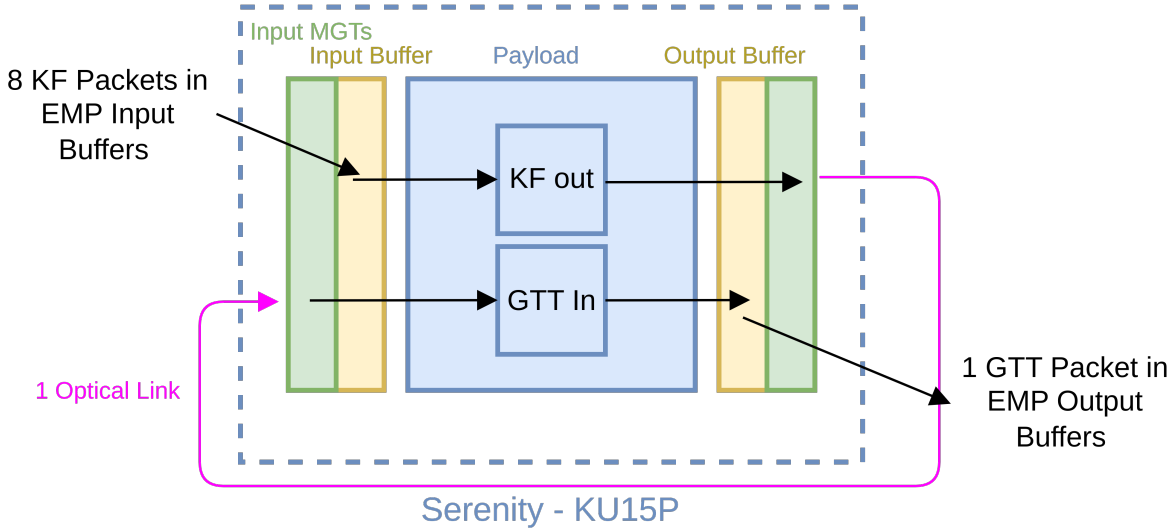


Figure 6.8: The setup for single board loopback testing between the KF output and GTT input. Firstly, input data is inserted into the EMP input buffers which are pushed through the KF out part of the algorithm. This outputs track words into the output MGTs. A fibre optic cable inserted into the MGT transfers the data back into the same board into a different MGT site. This is read into the EMP input buffer and subsequently the GTT input part of the algorithm. Finally, the algorithm outputs converted track words into the EMP output buffers and into a file.

When transferring to full board-to-board tests it is essential that both boards have the same reference clock source. This is provided in the setup by a DAQ and TCDS hub (DTH) [118] which distributes the trigger control and distribution system (TCDS) [119] signal to both an Apollo for running the track finder and Serenity for running the GTT, these boards are labelled in figure 6.13. In order to synchronise the boards, both boards were manually set to use the TCDS clock source using a flag within the board control software. However, this lead to many issues especially with the older Apollo. Trial and error, and reloading firmware into the boards and DTH allowed the clock source to be locked, and this debugging has lead to improvements in the revised Apollo as well as overall board control software. This low-level debugging of the boards and control systems is essential for the trigger project and a large part of the reason to perform integration testing of individual algorithms and subchains. Once the boards were synchronised a link-mapping was performed so that it was understood which output buffers the Apollo was using and connected to which Serenity input buffers. Null algorithms were used for this to

manually create the mapping, however, newer versions of the online control software allow this to be performed automatically. Automatic link mapping is performed using link metadata where flags in header data packets are used to identify which link the data is on; these developments form part of a standardised protocol for the CMS trigger system. For the actual testing, the boards were first loaded with their respective firmware and then commands set the two boards up for link communication which verifies the links are actually working and ready to send and receive data.

Multiple board tests began with just a single fibre between the boards (3), enabling two output links of tracks to be transferred between the boards. This is shown diagrammatically in figure 6.9. This was used to verify the link setup worked correctly, and that the tracks being sent by the KF output could be decoded by the GTT input step.

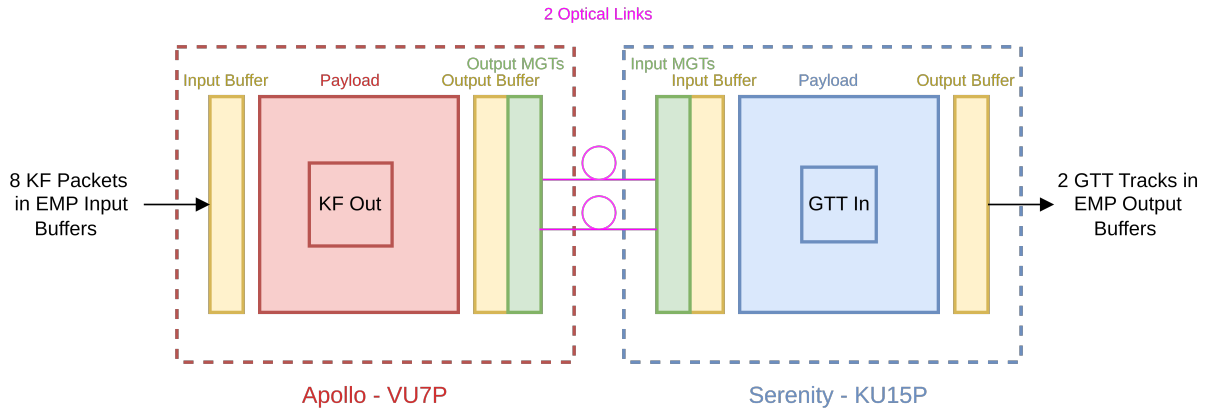


Figure 6.9: The setup for board-to-board single fibre testing between the KF output and GTT input. This setup is similar in concept to the loopback test in figure 6.8 but with a different board running the GTT input algorithm. Instead of the fibre optic cable being reinserted into the same Apollo board it is inserted into a separate Serenity. To the algorithms the input and output data is identical, the difference is in the use of EMP buffers and different MGT sites.

Once this test was successful a more complete test was performed, which set out to test a full set of tracks being sent to the GTT board which could then run a full vertex finding algorithm. The Apollo payload consisted of nine KF output steps; one for each detector nonant. This results in 18 output packets per clock cycle, so 18 96-bit track words every three clock cycles. These packets are organised into 4-channel fibre optic links between the boards for a total of five fibre optic cables. Pattern files were loaded into the Apollo input buffers, played through the algorithm and then outputs sent into the Apollo output MGTs. The Serenity running the GTT algorithm received these track words into its input MGTs which were then transferred into the payload. The payload outputted into the Serenity output buffers, which was then read as a text file. The payload of the Serenity was the GTT input step that decoded the 64-bit track packets into 96-bit track words and a copy of the FastHisto vertex finding which outputted its vertex into the output buffer. A diagrammatic setup of the board-to-board test is shown in figure 6.10. Figures 6.11 and 6.12 show the firmware floorplans after implementation is complete. Figure 6.13 shows a photograph of the setup.

This test successfully generated a vertex for single events, and demonstrated the readiness of the GTT firmware to accept all 18 tracks from fibre optic links. While the track finder test was less realistic for the track finder subsystem it still demonstrates how tracks can be outputted across fibre optics successfully. One of the key steps going forward for the overall trigger project is continuing these integration tests and adding complexity either in the form of stub inputs from a physical tracker module into the track finding or to add additional parts of the

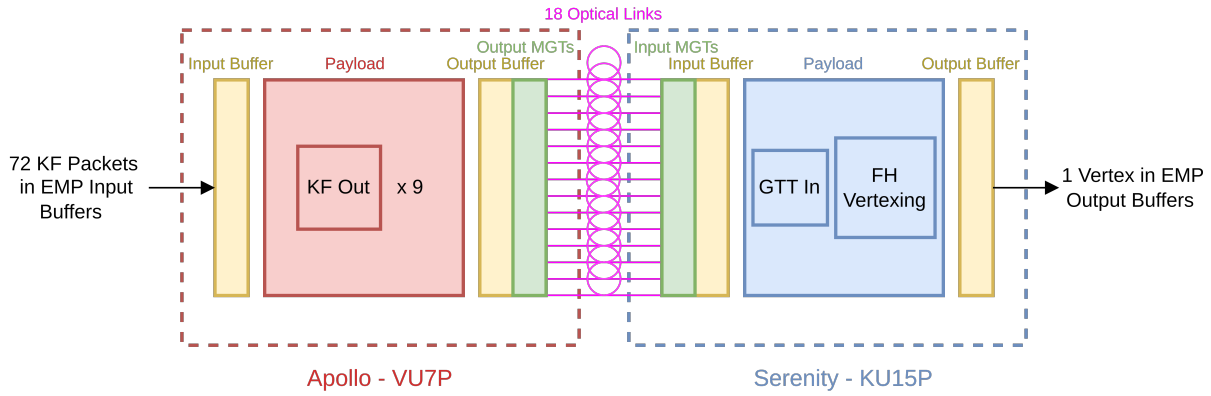


Figure 6.10: The setup for board-to-board testing between 9 × KF output and vertexing. Larger input pattern files are pushed through a payload containing nine KF outputs which all output two streams of track words into the output MGTs of the Apollo. Five fibre optics cables of 4 links each transfer the 18 track words into the Serenity input buffers. An implementation of the GTT input and full FastHisto vertex finding use all 18 track words to perform vertex finding and output one vertex per event into the Serenity output buffers.

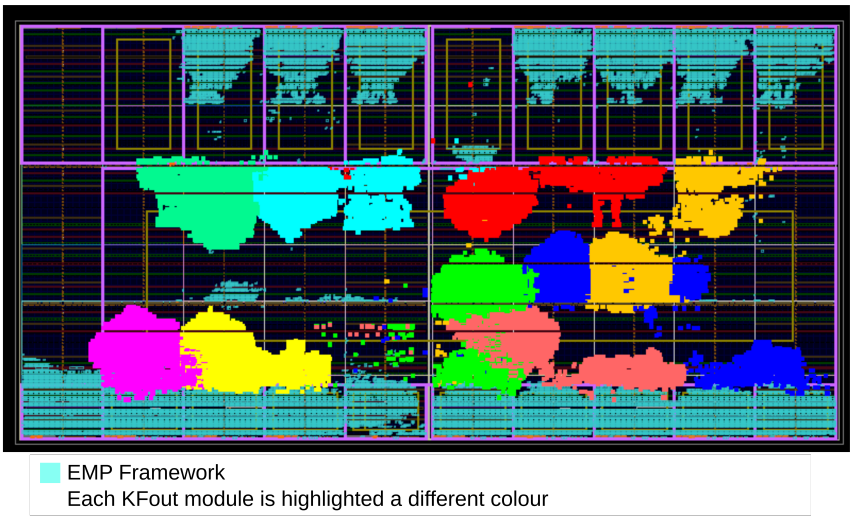


Figure 6.11: Floorplan of the VU7P on the Apollo with nine individual KF output modules, highlighted in different colours. Used to generate all 18 links of tracks for TF to GTT integration testing. The KF output modules are evenly distributed around the chip, close to the EMP buffers. The modules will have been placed close to the specific buffers they read and write from.

trigger chain after the GTT. The key takeaway from these tests is the readiness of the systems to perform complete integration tests. A large part of integration tests is not the actual running of algorithms for many events, but the setting up of boards and EMP buffers for the tests. These tests described in this chapter are the cumulation of significant firmware and control software debugging, showing the feasibility of longer tests, which can now be performed with only small changes to the internal payload firmware.

6.7 Conclusions

The testing of firmware is essential when developing algorithms for the upgrade of the Level-1 trigger. It ensures that algorithms do work in the target FPGA and that the custom hardware boards developed for this project also

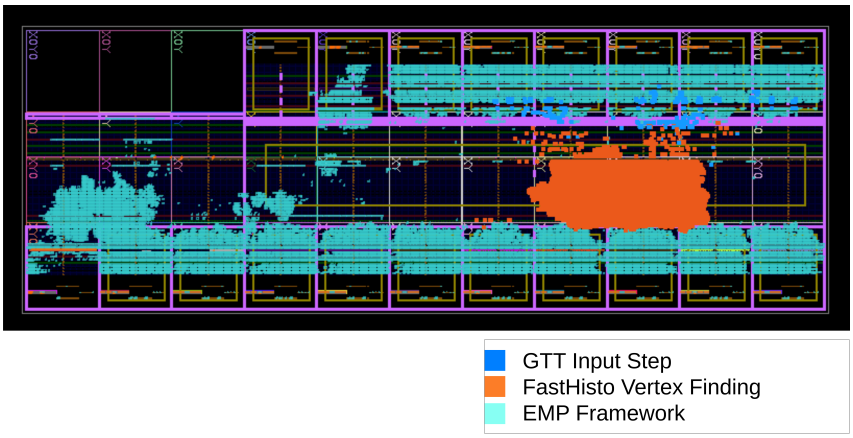


Figure 6.12: Floorplan of the KU15P GTT firmware. Used as part of the TF to GTT integration testing. Shown is the relative sizes of the GTT input versus FastHisto, where FastHisto is far more resource intensive. In this design all EMP buffers are enabled leading to the light blue around the entire chip, in a real production design this would not be necessary as the exact buffers for the I/O would be known beforehand.

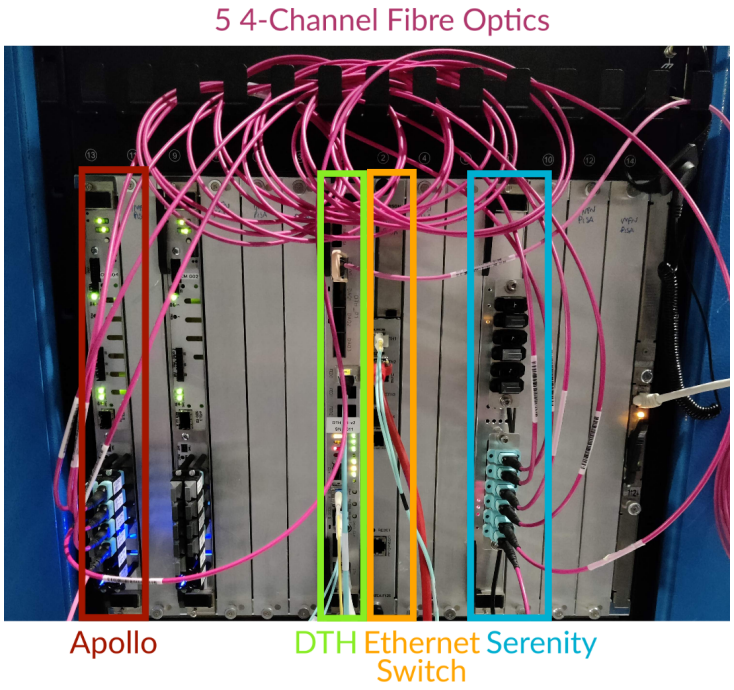


Figure 6.13: Photograph of the TF to GTT integration testing setup with boards labelled. On the far left is shown the Apollo board highlighted in red. On the far right is the Serenity board, highlighted in blue. These are connected by five pink fibre optic cables each with four optical links each. The DTH in green distributes the common clock and TCDS signals for all the boards via the ATCA backplane, required for synchronising the boards for the board-to-board tests. The Ethernet switch in yellow is used to communicate with the boards and DTH externally.

work as expected.

Multiple tests of different subchains and subsystems were performed in order to verify that the track finder and GTT firmware was operating as expected, each test builds on another and saw multiple rounds of simulating and matching to emulation. These tests will continue as the firmware for each subsystem develops. A major next step for the track finder is the demonstration of one nonant of the full algorithm, which will require the algorithm to be split over two FPGAs. The GTT firmware developments are also ongoing, with major next steps including a full

version of the GTT including all sub-algorithms on one board and a complete test of the end-to-end vertex finding.

A final set of tests that were performed was a multiple board integration test where a full set of tracks were sent from an Apollo board to a Serenity board hosting the GTT vertex finding. This test is useful in verifying firmware in a scenario similar to actual running, and larger scale integration tests are a priority for the Level-1 trigger project going forward. Future integration tests will include sending tracks to the correlator trigger boards to verify their interface with the track finder as well as exploring multiple track finder boards being used to create tracks and send them downstream; a key test of the synchronisation across the track finder subsystem. Finally, more complex tests involving multiple subsystems will test the full trigger system with track finder boards sending tracks to the GTT that sends its vertex to the correlator being a key next step.

Chapter 7

The Deployment of ML in Changing Environments

Many high energy physics experiments operate in very low latency environments where data is being streamed or triggered from a detector with data rates of up to 100 TB/s. In the coming years machine learning will be taking a large role in processing this data as it is streamed or triggered in order to reduce these data rates, and can offer reconstruction or decisions close to the level of full offline analysis at very low latencies. This use of ML for real-time inference in highly complex experiments opens possible issues of out of distribution (OOD) inference due to a changing experimental environment or because of the common practice of training a model on perfect Monte Carlo (MC) simulation and deploying it in real world scenarios. This chapter discusses these issues and offers some solutions that are currently being used or tested. The second half of this chapter discusses this issue specifically in the low latency environment of the CMS L1 trigger where incorrect inference leads to loss of data and degraded physics performance. An example scenario for maintaining ML performance in a changing environment is also introduced.

7.1 Changing Environments and the Implications for ML algorithms

Much of modern machine learning follows a very standard workflow: generate a training dataset, train the model, and deploy the model to make predictions on new data. This is the same whether using simple gradient boosting on a set of decision trees, or training a complex multilayered CNN. To achieve good performance the training dataset will have been carefully cleaned and shuffled before being used and will be batched and replayed to the ML model multiple times during training. Typically, these training datasets are considered to be independent and identically distributed (IID); "independent" meaning that sample one cannot predict what sample two is, and "identically distributed" meaning that all the samples of a dataset belong to the same probability distributions. Through these techniques, ML models have achieved better-than-human performance in a wide range of tasks and are able to quickly make predictions across a range of domains. But these scenarios are specifically stationary; the models are being deployed in the same environment in which they were trained. This can be very useful if it is certain that the inference environment is stationary and follows the same distribution as the training data, but in real-world examples it is unlikely that this is the case, for example, a dynamic physics experiment with imperfect detectors where the whole inference environment cannot be predicted.

There are many scenarios where the environment a model is being deployed in is not only different to the

perfect training dataset, but it will also be non-stationary and evolve over time. These changes could be gradual domain shifts that lead to a variation in a model's output; for example a sensor that is used as input to the model which degrades over time. Or these could be significant changes to the environment where, for example, the user might want the model to learn about a new class for a classification problem.

The most basic approach to dealing with changing environments would be to do nothing—leave the old model in the changing environment and allow it to make predictions. This could be suitable if there is an understanding of exactly what the model is predicting and how this relates to its input features, so it can be quantified how the model might change with the environment. However, in "black-box" algorithms such as deep neural networks, the internals of the decision-making process of a model are hidden, which can obscure the predictions being made in these changing environments and make them difficult to understand and correct for [120]. Some correlations between data and prediction can be observed and this can be enough for users to be confident in the model's output. Another situation where doing nothing is useful is if it is known that the changing environment does not affect the model as the features are orthogonal to the changing environmental factors, or the model has been trained in a way to be robust to certain changes, for example rotational invariant models that still make the same prediction if an image is upside down.

Sometimes this is the only option; ML models in embedded devices, where it is not possible to change the model, will make predictions no matter how the environment changes. Another situation where leaving a model without changing is the case where collecting new data for training is not possible. If the ML model is being used to make predictions on a stream of data, which is then discarded after inference, there is no data storage. This poses a particular problem for ML in a changing environments as there is no feedback to the user about what the information being supplied to the network actually is. The loss of inference data can be very common. Simple lack of storage or privacy concerns can mean the environment changes are unknown, particularly a problem where the model inference is making an important decision, such as in a high speed trigger.

In high energy physics, it is typical to train and benchmark a model on MC-generated datasets and then deploy these on real data. In offline analysis this process is highly effective and can, for example, boost signal-to-background classification compared to simple cut-based approaches [75]. For offline analysis, the many steps of previous reconstruction create a stable inference environment for ML models and typically high-level features of an event are used, such as number of jets or missing E_{T}, which makes these ML models robust to a shifting experimental environment. However, as ML is pushed further into the original reconstruction chain and into the online reconstruction and triggering itself, the inference environment will shift with a changing detector and there will be differences between training datasets and inference data. In these scenarios, models that are robust to their environment are essential as in many cases once their inference has been made the original data is lost. Understanding how this happens is also essential for correctly estimating systematic uncertainties in an experiment.

7.2 Solutions

There are a number of more resilient solutions to the issue of changing environments in ML models. One could be to design the model in a robust manner so that its output does not change with the environment. Other solutions can be as simple as retraining the model to account for the new environment on a regular basis, or adding in a prediction of uncertainty from the model so that downstream users can use the output in an informed way. Finally, there is another area of research in training the model in the inference environment non-stop, called continual learning, never-ending learning, or lifelong learning [121]. This paradigm of learning sees a model continually updated as new data is experienced with various techniques employed to maintain and improve model

performance.

7.2.1 Robust Machine Learning

Building-in model robustness is an important part of any ML training where the model is deployed on real-world data; it allows a model to produce reasonable outputs for OOD inference data and the user can be confident that this is a reasonable answer. For example, a typical methodology in training CNNs is to add additional adversarial samples to training data which can be as simple as flipping or offsetting images, or introducing noise to the sample. This ensures that a model is not learning from small features in an image but instead overall concepts that are invariant to image topology or noise [122].

A method of developing robust NNs used in the LHCb trigger system is to constrain the Lipschitz constant of the network [123]. The Lipschitz constant for a function describes how variable it is; by constraining this in an NN, the overall variability of a NN is reduced [124]. In practice this was performed by bounding the gradient of the model in each of the weights, acting as a weight normalisation for each weight in a network. This is very generalisable to different network architectures, and is a powerful method for increasing the robustness of an NN. For this to be successful, there are a number of things to consider: thoughtful weight initialisation is needed when normalising them through training which is not always straightforward. Also, because of this Lipschitz constraint on the network, other commonly used constraints and techniques for reducing the size of a network may not be successful. While the Lipschitz normalisation draws the weights one way, weight regularisation to keep them small might draw them in another leading to poor performance in both metrics. Finally, if an environmental change is large enough, even the most robust network will be unable to fully describe this environment. In these cases some retraining will need to be performed, while small detector changes might not lead to this extreme example, sudden loss of detector subsystems could result in very different inputs to a network.

7.2.2 Retraining and Redeployment

One of the most basic methods to deal with a changed environment for an ML model is to retrain it with the new data and redeploy it. Figures 7.1a to 7.1d outline these techniques. Figure 7.1a shows basic training of a model on a large dataset. Figure 7.1b shows the simplest approach to new data, which would be to train a brand-new model on the new data and redeploy it. This approach would be suitable if a large enough data sample had been collected to retrain the model, and it did not matter what the performance of the model in the previous data was, as this new training would lose all knowledge of the previous training. This method of updating ML in a high energy physics environment is already seen in the X-ray diffraction experiments at the Stanford Linear Accelerator Center [125], where a collected dataset is used to update and redeploy models, in this situation there is enough time to create a large enough dataset and entirely retrain a fresh model.

Figure 7.1c shows how a new model could be trained on a mixture of the initial training and collected new data. This would be suitable if data storage was not an issue and the domain shift in the changing environment was not large. Otherwise, there would be “tug-of-war” dynamics in the training where one part of the data pulls the loss function in one way and the other part of the data pulls in the other resulting in a model that has weak performance in all the data [126].

Figure 7.1d shows how a pre-trained model can be trained on new data in a top-up training style where the old model is trained on new data in a new training, typically with a lower learning rate so that the model parameters

do not move a large amount. This would be suitable if compute availability is an issue and only small trainings could be performed. In some situations large combined datasets might be an issue to store or collect, for example where a model is deployed in remote locations with no connection back to a central server, but additional training is needed to maintain performance in the changing environment.

There are a number of advantages to these general set of methods:

- Simplicity. No complex training is needed, the model can be trained with new data with standard gradient descent techniques.
- No need to fully understand the changed environment. If it is known the environment has changed, but the user does not want to quantify it, new data can be collected and used to retrain or top-up train the network. However, if the changing environment is well understood, the top-up training or retraining can be tuned to give maximum performance.
- Reproducibility. If the model state is stored after each training, then it can always be rolled back to a previous state.

However, there are some significant disadvantages:

- Computational cost. Every time a network needs retraining this could take up significant resources, especially if the model is large. For top-up trainings there is still an overhead but typically less compared to a full training of a model.
- Data availability. This method relies on a stream of labelled data being stored for the model, not only could this be difficult to create if the specific application does not store its data, but it also introduces another computational overhead of storing large amounts of data and also labelling this data.
- Obscures the model's response in the environment. If the model is just retrained regularly there is no indication of how the model might be changing its response as a function of the changing environment; if there are large gaps between retrains there could be significant periods where the model performance is compromised without a clear way of knowing.
- Catastrophic forgetting. When a model is introduced to a new set of data, different from its previously trained task or scenario, it forgets its prior knowledge of that previous scenario. This is an issue if the changing environment reverts to its previous state, or good performance is required over a range of scenarios not just the one that was most recently trained.
- The retraining, or top-up training could alter the model's architecture. For example, if a different pruning or quantisation scheme is needed for the retrained or topped-up network to maintain performance. This could be an issue for an embedded application where tight constraints are placed on the model's architecture.

This approach of retraining or top-up training a model is a default across a wide range of scenarios where data is freely available and previously learnt scenarios do not matter. However, as models get larger, and their training energy costs increase, starting from scratch retraining a model is computationally expensive so methods for avoiding large data collection and/or long training cycles should be investigated.

7.2.3 Uncertainty Quantification

A useful way of understanding how a model is impacted by a changing environment is with some method of uncertainty quantification. If it can be understood how certain a model is of its prediction, trends can be observed.

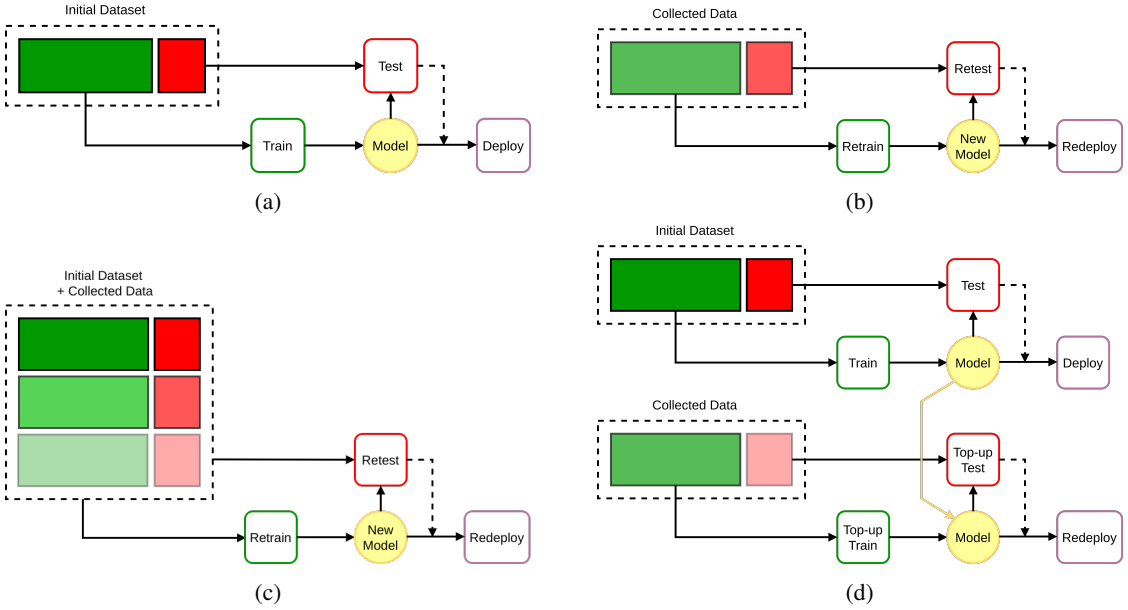


Figure 7.1: Diagrams for different model retraining scenarios, each show a training dataset in green used for training the model and a testing dataset in red for testing the model. Pending a successful test the trained model is deployed. Figure (a) shows the standard training procedure on an initial dataset, where it is first trained, then tested, then deployed. Figure (b) shows retraining on a newly collected dataset with a new model in the same steps as figure (a). Figure (c) shows a combined retraining of a new model with all data initial and collected, then a new model trained in the same methodology as figure (a). Figure (d) shows a top-up training scheme where a pre-trained model is given new data to train on. The initial training of the model follows the same scheme as figure (a), the trained model is then trained again on newly collected data in a top-up training with its performance evaluated in a top-up testing.

For example, if the model's certainty is steadily decreasing the environment might be changing as the inference environment becomes further away from the training environment. The use of uncertainty quantification also allows downstream users of a prediction to make a choice over how to use that prediction. For example, if a model predicts an image is of a cancerous tumour but only with a 60% certainty, a trained doctor can decide to review that image [127]. Sometimes, an estimate of certainty of a model is built into its training, for example using a softmax layer as an output activation for a classification model will return a number between 0 and 1 representing the "probability" that the classification is of the class 1. This simple way of understanding model certainty can be used as a dynamic threshold for downstream users depending on how they want to use the models output. Both the ML methods described previously, the track quality BDT and track-to-vertex association part of the end-to-end vertexing NN, use a dynamic threshold as their outputs for their respective classification tasks as it allows downstream users to select tracks based on their own requirements. However, this is not a true certainty of a model and is typically overconfident [128].

In general, techniques in uncertainty quantification are ultimately simplifications of a wider concept in ML, Bayesian inference [129]. At a very basic level, ML seeks to answer a question of given the data x, what was y? This is learnt through training a model which takes a loss function with the model as the parameters and finds the optimal model to minimise the loss. This provides a single model that gives the best answer to the question. However, it is not a complete description of the model in that loss function space. It is more informative to ask, given the data x what is the probability of y? This can be found by describing the whole distribution of models that map x to y, while it is possible to estimate a full distribution of models in very simple cases, it is very computationally expensive or simply impossible most of the time, especially given the many dimensions

associated with the kind of problems being solved by ML. Instead, simplifications can be made that look for subsets of models in this space and use the combination of those to make predictions with an associated uncertainty.

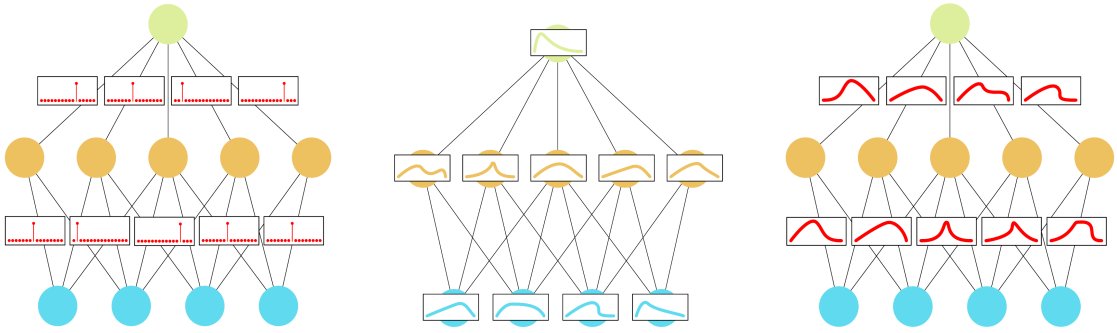
One very common method for simplifying the Bayesian inference problem is with ensemble techniques, where multiple models are evaluated on the same data and an average of their responses is used. This technique is well-established; the method of bagging or boosting in decision trees creates many weak learners that are either used together to predict (bagging) or used to improve the next tree as the learning process continues (boosting). In the case of decision trees these techniques are essential for good performance over large ranges of inputs where the decision trees are very simple [130]. For NNs, techniques are different as the base learner NN is more complex and can learn more complex relationships between the inputs and targets. Ensemble techniques need multiple models with slightly different outputs which could be created from training multiple models on different slices of a larger dataset with differences to their initial weights [128]. Another method in NN uncertainty awareness is using Monte Carlo drop out, where a single trained network is evaluated multiple times with different nodes randomly removed [131]. In both these methods the combined output of the multiple models or multiple inferences are combined, and their spread used as the uncertainty on the mean of their predictions. The issue with these kinds of methods is that the understanding of the cause for uncertainty is being removed; randomly splitting the data and models or randomly removing parts of the network and evaluating it a number of times is obscuring the network evaluation process.

Ensemble techniques are used often where other techniques are not possible or not understood, but they come at a computational cost. Training multiple models on different data and then evaluating multiple models takes time and large datasets, while dropout techniques still need a network to be evaluated multiple times to give an averaged answer. There are methods for making the computational cost lower, for example subnetworks within a wider network learning different subtasks which all give an answer on a given input [132], which are then combined to give an uncertainty on the answer. Other solutions include adding additional outputs to the output layer of an NN that gives an estimation of uncertainty such as one that predicts the distance between inference inputs and training inputs [133].

A more rigorous approach to uncertainty in NNs is with a Bayesian neural network (BNN) [134], shown in figures 7.2b and 7.2c. These NNs are stochastic networks trained with Bayesian inference meaning the weights and biases (figure 7.2c) or the activations (figure 7.2b) of the network are distributions over ranges of parameters. A BNN aims to capture a model's distribution of weights in the loss function space rather than optimising them, as is the case with a standard NN. The BNN cannot be trained using differentiation and back propagation as is the case with standard NNs; instead probability distributions over the inputs need to be defined to parametrise the network weights and biases. Ultimately, BNN techniques are complex and require in-depth knowledge of feature distributions and the probabilities involved in the predictions of classes making them fundamentally difficult to set up and train, and also understand, but have the significant advantage of giving a robust estimation in OOD samples and a true estimate of uncertainty. There are currently no methods for deploying BNNs on an embedded device because of the overheads associated with having distributions of parameters in the networks, making them unsuitable for many real-time high energy physics environments.

Overall there are advantages to uncertainty quantification:

- The ML model does not just blindly predict. Instead, it gives an uncertainty in a prediction which can be used by downstream users to decide how to use the prediction.



(a) (b) (c)

Figure 7.2: Standard point estimate NN (a). Stochastic NN with distributions over the network activations (b). Stochastic NN with distributions over the network weights (c) [134].

- ML models are more robust to OOD samples and will indicate when this OOD occurs, useful in understanding how changing environments are impacting model performance.

There are some key disadvantages:

- There is no improvement in the prediction if the environment changes, only an indication that is has, so the network or combination of networks still need to be retrained or top-up trained.
- These methods require more computation. Any ensemble technique requires multiple networks to be trained and multiple inferences, while BNNs pose a large computational cost in their training to properly parametrise the weight distribution in the network.
- The complexity of the models increases. While it is understood that there is uncertainty involved in a prediction, knowing why this uncertainty is high is not always obvious and introducing more randomness into NNs with dropout and ensemble techniques further obscures what is being learnt from the data.

7.2.4 Continual Learning

A more fundamental approach to the issue of a changing inference environment is to allow a model to learn while in the changing environment. The idea of continual learning (CL), lifelong learning or incremental learning is a general technique in ML with multiple different areas of research, all with the goal of allowing a model to change as new data is introduced to it [121]. This could be in the form of introducing new classes for a model to also classify ("class-incremental learning") or new data where the domain is shifting from the original training ("domain incremental learning"). The second scenario is the focus of the following discussion because of its application in the changing environments found in the CMS L1 trigger.

The actual process of CL can be achieved in a variety of ways. They all attempt to maintain a model's performance across its full range of learning, so avoiding the catastrophic forgetting involved in retraining models but also allow the model to be sufficiently generalisable as new data is seen. Figure 7.3 shows the setup of the Avalanche library [135] used for CL in the study below. The ML model undergoing retraining is in purple and its CL training is controlled by a separate training module, in red. Some commonly used CL training techniques are listed below, but this is not an exhaustive list in this rapidly changing field. New data is organised in experiences, $e_{1..n}$, and introduced to the model training incrementally.

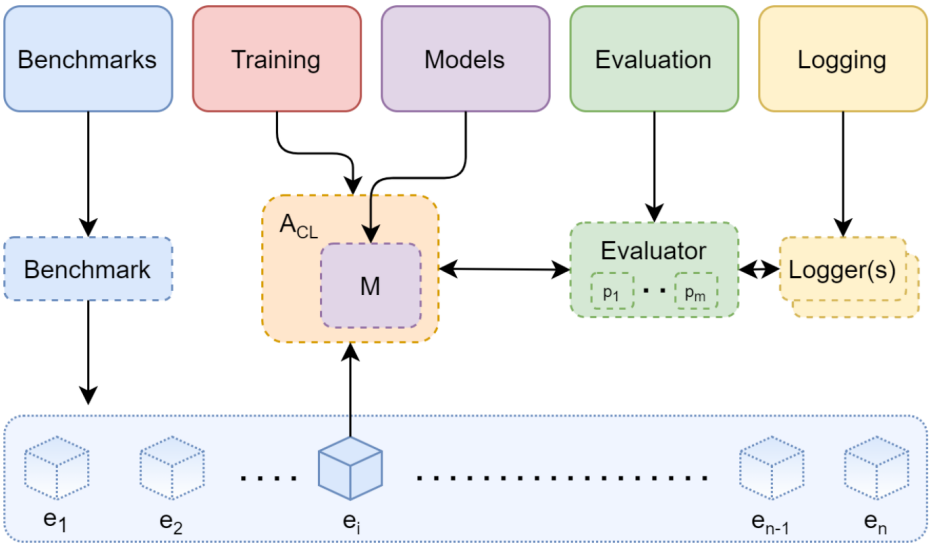


Figure 7.3: The setup of the Avalanche library [135] for continual learning. Modules within the library control different aspects of model training and dataset generation. The benchmark module creates a stream of experiences with which the model incrementally learning on data. The training module controls the CL training loop and is flexible to a variety of different CL training paradigms listed in section 7.2.4. The model module interfaces with the PyTorch ML library [136] for the underlying ML model being used. The evaluation module evaluates the model's performance using different metrics on a valuation dataset. The logging module monitors training.

Replay

Replay is a CL technique where knowledge of previous data or tasks is kept and mixed in with new data or tasks as training progresses [137]. This maintains performance in these previous data while also learning on new data. The simplest form of replay is to store old data in a buffer and mix it in with new data as it is trained, which is very simple but requires old data to be stored in some way. Replay can also introduce tug-of-war dynamics if the old data is significantly different to the new. A more advanced technique is to use a second generative model which generates data for the first model to train on [138]. This generative model is trained on the previous experience so is distilling the old knowledge back to the first model. This removes the need for any stored data but can be computationally costly. A method of avoiding this would be to use an internal feature level generative model that is then replayed to the target model internally. Overall, replay is relatively computationally cheap and non-complex way of improving model learning, however it is not the most powerful method and still needs some samples or information to be stored for later replay.

Regularisation

Regularisation covers two broad strategies: weight regularisation [139] and function regularisation [140], which both add additional terms to the learning that aim to balance old and new data. Weight regularisation evaluates the importance of model parameters after an initial training and then introduces a loss function, penalising changes to the important model parameters as new data is introduced. This can be seen as creating a core model with changeable smaller models updating as the data changes. This is useful in situations where the fundamental distributions of feature to target mapping does not change, but small fluctuations in the data need to be calibrated for. In function regularisation, a teacher model trained on old data distils knowledge to the currently training model—the student. As new samples are introduced, the student learns on the new data but is also penalised as its predictions move from the teacher models. Regularisation techniques do not need any previous data storage, only

information about the important weights. This can introduce complex learning schemes to take into account the regularisation and can face issues when the environment changes significantly from the original training.

Architecture

Another methodology in continual learning is based on the model's architecture. There are another two broad methods. The first is parameter isolation where different parts of a network are used to learn from the data [141]. A large model is trained on an initial dataset and the unimportant parameters are removed, through processes such as pruning. When new data is introduced, the remaining part of the network is fixed and pruned parts are then released to learn from the new data. This is a similar process to the weight regularisation but with a set mask over the parameters dedicated to certain tasks or data. Another method is to allow the model to dynamically grow as new data or new tasks are added [142]. While this method is useful for new task situations where distinct new parts of a model are needed to learn the new tasks, the dynamic approach cannot be used in an embedded application where the size of a network is very limited. Again, no external data is needed for these techniques but parameter isolation is not suitable for large domain shift and while dynamic networks can learn even in very changeable environments, they are difficult to deploy in an embedded scenario.

Online Continual Learning

Online continual learning focuses on the problem of updating an ML model without any previously stored data, updating on single data points at a time [143] [144] [145]. This is necessary in embedded situations where data is passing through a model constantly, but it has no external storage or no time to train in a standard epoch–batch style workflow. Again, there are many possibilities for the methodology, depending on the specific situation, but a common technique is to maintain a running class mean as new data is predicted on. At inference the label is outputted based on the closest class mean to the data. While this does not update the model itself this acts as a method for allowing the output to change as the input data changes [146]. Other solutions include performing in-batch knowledge distillation, i.e. computing gradients within a training batch to ensure a network does not forget each batch as it trains. This also opens up a possibility of doing in-batch data augmentation to improve robustness. This method allows an NN to prevent forgetting across previous batches without having to store the full training dataset and on-the-fly augmentation also reduces memory cost [147].

There are some clear advantages to the continual learning approach:

- Avoids catastrophic forgetting. Some information from the initial training is kept by the model so that it does not lose performance in previously seen data
- Adapts to a changing data stream. There is no need to quantify the changing environment; the model in the environment will adapt to it. However, this can obscure the environment itself and the response of the model to it.
- Robustness. If a learning environment has an underlying feature to target mapping with noise introduced by small domain shift the CL methods will learn this underlying target and ignore the noise so will become more robust over time.
- No stored data. CL strategies do not necessarily need previous data to be stored, instead information about previous tasks can be encoded in the model or in a different model.

However, there are disadvantages:

- Dynamic models. In an embedded setting this creates issues for deployment where set model sizes are needed.
- Needs a continual stream of labelled data. Sometimes this is not accessible, the labelling of data is not necessarily possible, though there are techniques to update models without needing labelled data.

7.3 Application of Continual Learning to the Level-1 Trigger

The Phase-2 CMS upgrade plans to include many ML algorithms in a variety of places from the hardware L1 trigger, through to the HLT and offline analyses. Specifically in the L1 trigger, as has been discussed in previous chapters, ML plays a major role such as: BDTs for evaluating track quality, NNs for vertex regression, and global trigger event topology NNs for identifying VBF events [57]. The ML used in the L1 trigger has an important role and confidence in their responses is needed, as if an event is not triggered on it will be lost, which makes understanding their outputs and being confident in this output vital before deploying them. ML in the trigger also shares a variety of other features:

- They are all lightweight models; pruning and quantisation are needed so that these models fit on the resource constrained FPGAs. This makes their overall robustness to a wide range of possible inputs limited; the process of making a network small sacrifices its overall generalisability and encoding possible environmental changes at a network architecture level is impossible.
- They are all focused; they have a specific task using specific detector data, for example the vertexing NN only uses high level track information to find the vertex and associate tracks. There are no general purpose event reconstruction NNs that use all parts of the detector at high granularity. This makes them effective at these tasks, but not generalisable and susceptible to their inputs changing in unexpected ways.
- They are trained on MC data. As they are for a future system that will not take data until 2029, the detector that they are trained on does not exist. Assumptions can be made about how the future detector will operate and detector simulations are a vital part of all algorithm development. However, it is not understood how these ML algorithms might change their output given the real detector. Part of this issue is that this real detector will change over time and the inference environment for these algorithms is a dynamic one that could be different to the perfect IID training environment that underpins all of these ML algorithms.

Issues remain with the deployment of ML in the CMS L1 trigger, the following section hopes to address some of these operational issues and how to build robust ML system for real-time inference in a changing environment.

7.3.1 CMS as a Changing Environment

The CMS experiment is a non-stationary environment; many factors cause the experimental conditions to change, and the online systems need to be able to react to these conditions. The changing environment of CMS varies at a number of different timescales which have implications across the DAQ chain and in the offline reconstruction of events. These changes can be broadly split into within a fill (from seconds to hours), inter-fill (between fills of the LHC), and at a multi-month or year level where technical stops and detector re-commissioning occurs.

During a Fill

At the fastest level, the order of seconds to hours, the conditions within a fill of the LHC change. At this timescale the LHC is filled with protons, the beams ramped to collision energy, stabilised and then stable collisions are run

for several hours before being dumped. In this stable beam period, luminosity levelling is used to maintain a high, stable luminosity for the LHC experiments. This is needed as over time the proton bunches diffuse and the luminosity decreases, so a variety of techniques are used to squeeze and change the angle of the beams so that the instantaneous luminosity in the interaction points is high and stable as possible [45]. These small changes will make little difference to the detector response and will have a negligible impact on the ML in the L1 trigger as some level of preprocessing is performed before reaching the ML, i.e. tracks are reconstructed, or calorimeter clusters are determined, so the ML is protected from the lowest level changing conditions. The global trigger will receive some information from the beam optics and radiation monitoring systems specifically related to the beam timing [148], while this allows triggers such as the zero-bias trigger to perform, there is no feedback of beam conditions back into the L1 trigger as there is not enough time to send and receive the data.

At this fast level variation, there would be scope for an online CL scheme that sees ML algorithms react to the changing conditions. There are, however, a number of issues with this approach; the first being a lack of reproducibility for the full CMS analysis. To fully understand the detector systematically and generate accurate backgrounds for physics analyses, full detector emulation studies, including the trigger system, are used. If a trigger algorithm internally updates, it would be impossible to have an accurate emulation of the full system. Another issue would be in the computational complexity; most of the trigger system is at the limit of the computational capabilities of the FPGAs being used. They are resource and latency limited and the addition of complex CL algorithms that might draw on external buffers for weights and biases would be difficult without reworking large parts of the algorithms. Within the CMS trigger there is not enough variation to justify this approach, but in more dynamic experiments that do not follow the LHC fill structure (short periods of data taking and not taking data) and instead continually take data without stopping, an online learning approach might be suitable to avoid detector downtime.

A more useful strategy would be to monitor the output of the ML to understand the robustness of the algorithms in stable conditions. Uncertainty quantification techniques would be ideal here, where the certainty of an output can be monitored to see if the ML is seeing input data it expects and providing confident answers to downstream algorithms. However, for reasons discussed above, uncertainty quantification results in an increased computational complexity due to multiple network evaluations so is unlikely to be used in the CMS L1 trigger without further research into possible fast uncertainty quantification.

Between Fills

At the inter-fill level there could be more significant changes to the environment and opportunities to update algorithms. Detector effects start to become noticeable at this level, for example radiation damage in the detector can be monitored and corrected for. The ECAL is an example of a subsystem that performs inter-fill calibration because of the optical degradation of the scintillating crystals over the course of an LHC run. During the fill specific signals such as the π_0 mass or $Z \rightarrow \mu\mu$ resonance are reconstructed and monitored, and a laser system is used to assess the optical characteristics of the crystals [149]. These methods are combined to update scale factors used by the DAQ system in the ECAL for the trigger and offline analysis. The tracker has automated alignment during the inter-fill period [150], which is needed to ensure the hit resolution of $O(0.01)$ mm in the presence of small temperature effects or magnetic field variations. These methods demonstrate that the changing CMS environment does impact the reconstruction algorithms and while in Phase-1 the granularity of the L1 trigger was lower and such changes were less noticeable, the higher granularity and more complete event reconstruction performed by the Phase-2 L1 trigger could be impacted by these detector changes.

Another potential issue at this between fill level is specific failure for a detector system, for example part of the cooling or high voltage systems breaking causing a part of the detector to fail. This provides difficulty for any trigger algorithm that might depend on the entire subdetector as missing information would appear anomalous. The standard approach to dealing with this issue is to mask the detector region, i.e. no data would come from it, but the remaining system would continue as usual. This is problematic in ML algorithms that might rely on geometric features of the data. In these scenarios some updating would be needed of the algorithms to ensure performance is not lost.

It is feasible for an ML algorithm to be retrained in this timescale using a dataset collected during the fill. This scheme would require a stream of labelled "truth" data to be created to train the algorithm which could come from a number of sources. The most obvious is a full offline event reconstruction which performs as close to truth-level reconstruction as possible with full granularity detector data. However, this method has the issue that offline data that is collected has to have been triggered on to have been collected, so there is some level of bias associated with this data. Typically, the triggered events will have followed a strict trigger path based on some physics models that are of interest to analysis, obscuring potentially interesting signals that are not triggered on meaning the ML is biased against them. Using minimum-bias or zero-bias triggered data would avoid this but limit the dataset. This issue of bias is not significant because the ML in the trigger does not perform full event reconstruction and instead reconstructs specific objects which are seen across a wide range of events. An alternative would be using the 40 MHz scouting data [74], which collects unbiased trigger primitive data to attempt to find physics that standard trigger paths cannot. This potentially would not have the same level of truth reconstruction as a full offline reconstruction as all that is collected is the trigger primitives not the full granularity detector data, but would not suffer from any bias from the trigger pathways.

Currently, firmware implementations of ML require full rebuilds of firmware to update the algorithms which poses a problem for updating ML between fills of the LHC as there would not be enough time to rebuild and redeploy the algorithms. However, in the next five years progress could be made towards having ML algorithms that use updatable RAM that can be externally updated without having to rebuild firmware.

The way in which these ML algorithms could be updated at this timescale is open, the datasets collected could potentially be very large, over several hours with a 7.5 kHz event rate $O(100$ million) events could be collected, which compared to the $O(100{,}000)$ simulated events currently used for training, is significantly higher. However, the computational cost of training on this level of data for many ML algorithms simultaneously and evaluating their performance is not practical, so a smaller subset could be used. The example described in section 7.3.2 compares a non-retrained model to one top-up trained and one top-up trained using a CL algorithm to show the potential for CL methods in updating L1 trigger ML which would be suitable for this between fill timescale.

Technical Stops

During technical stops of the LHC, CMS is re-commissioned using a variety of commissioning runs. These are designed to evaluate the detector and perform alignment and corrections needed for the upcoming run of the LHC. All detector subsystems are separately commissioned and any significant detector damage evaluated and corrected for. Over the course of several years of running, radiation damage to the detector results in worsening performance. This is particularly noticeable in the calorimeter systems where the ECAL crystals optically darken with radiation dose or with the tracker where radiation damage causes increased leakage current in the silicon

sensors necessitating running at lower temperatures to maintain performance. These issues will naturally need accounting for in trigger algorithms and a variety of updates are made to thresholds in the trigger algorithms before running begins, once the current detector performance is understood. Also, extensive MC simulations of the detector systems are used which allow the detector ageing to be simulated and accounted for to ensure performance is not lost. At this timescale it is possible to create large MC simulation campaigns for the detector commissioning and large samples of real data can be collected to commission algorithms. It could be envisaged that an entire ML model could be retrained or improved in the large timescales involved and fully re-implemented for the L1 trigger system. The mechanism of how this would be achieved has not yet been decided.

7.3.2 Level-1 Trigger ML in a Changing CMS

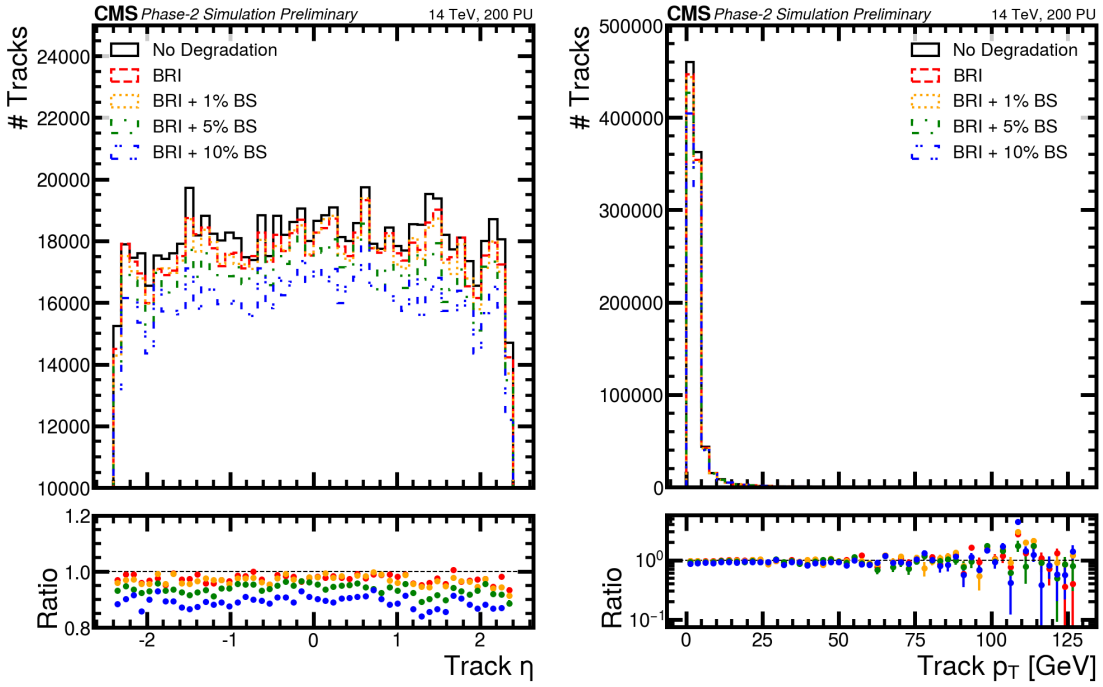
To explore the implications of using ML in a changing CMS environment and explore different approaches to training this ML, an example CNN that identifies fake vertices was used. While not an algorithm that is present in the CMS Phase-2 L1 trigger, it provides an example algorithm with which changing detector environments can be studied. A set of datasets which emulate different tracker degradations over time were used to train and evaluate the CNN. The CNN was trained in three different ways; the first being simply training on a large perfect detector sample and evaluated on degraded samples, the second being trained on the large perfect detector sample, then top-up trained on smaller degraded samples before being evaluated on the degraded samples. Finally, a CL learning approach to these top-up trainings was explored to compare its performance.

Tracker Degradation

To emulate tracker degradation, five different samples were used. The first was a large unmodified perfect detector sample created from a top quark pair production events with 200 soft hadronic PU interactions overlaid on top. This was used as a benchmark, no degradation, sample that represents the sample used to train an L1 trigger algorithm using perfect simulated data. It is large in comparison to the degraded samples, 100,000 versus 5,000 events for the degraded samples. The second sample introduces a bias rail inefficiency (BRI). This inefficiency is an intrinsic issue with the CMS tracker modules where the bias rail that crosses the silicon and provides the high voltage for the depletion region cuts across the cathode and anode areas in certain PS modules. This means the silicon detector is inefficient for particles that pass at high angles of incidence close to this bias rail as the electrons and holes created by its ionisations have no anode or cathode to register the ionisation. The BRI sample represents running the ML in a detector with an intrinsic issue that was not properly accounted for in the original training. The other three samples emulate radiation damage with different levels of broken strip detectors, 1, 5 and 10 % of the strips are randomly removed from the detector layers. This broken strip emulation is not necessarily a fully realistic demonstration of how the tracker might fail as it would be expected that the higher radiation regions closer to the interaction point would break first, so future work will investigate a more realistic degradation of the tracker. Another area of further study would be the emulation of high voltage or cooling failures which would lead to detector degradation in specific areas. ML models performing event-level reconstruction using geometric features could see significant performance differences in these scenarios.

Figures 7.4a and 7.4b show the L1 trigger reconstructed track η and p_{T} for the different degraded samples. It is clear that there is no geometric dependence on the degradation, the η distribution is still flat, however, the number of tracks being produced has dropped. This demonstrates that the degraded regions are causing the track finding algorithm to find fewer tracks, which is due to the constraint of a minimum of four stubs being allowed for a reconstructed track. The p_{T} distribution shows a similar trend. In general, the shape of the distribution is

constant, but the number of tracks is lower, especially the case in low p_T bins where each degradation sees fewer tracks being reconstructed.

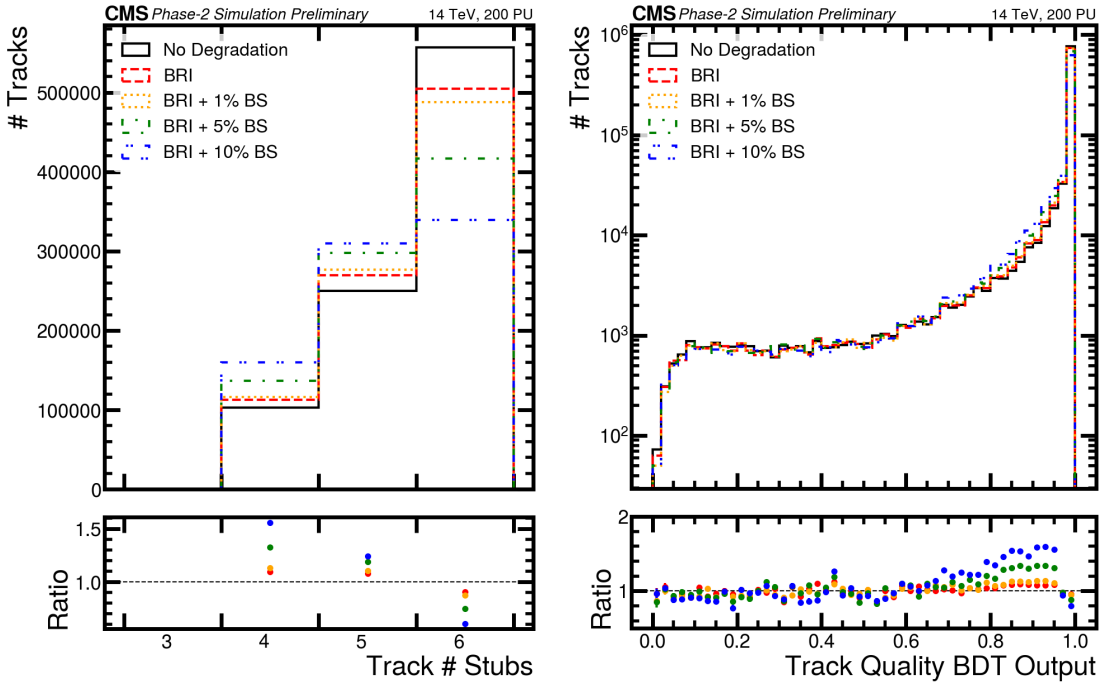


(a) (b)

Figure 7.4: L1 trigger reconstructed track parameter distributions for different levels of tracker degradation. The leftmost plot shows the η distribution across the different samples with BRI referring to the bias rail inefficiency and X% bad strips referring to a random removal of random strips from the outer tracker. The rightmost plot shows the p_T distribution for the same set of samples. Both figures demonstrate a drop in the number of tracks being produced as the inefficiencies are added with 10 % fewer tracks seen with the worse degradation. The lost tracks tend to be lower p_T with a larger drop in the smallest p_T bin in figure (b).

When analysing the track quality for different degradations, as is shown in figures 7.5a and 7.5b, it is clear that the quality of the tracks is more complex than the simple drop in numbers that was seen in the track helix parameters. Figure 7.5a shows that the number of stubs used to create the tracks drops, which is due to the random removal of sensors and intrinsic loss in efficiency from the BRI meaning tracks have to be reconstructed with fewer stubs. There is a drop in tracks with six stubs and increase in tracks with four and five stubs with the size of the drop increasing with the level of degradation. This causes complex results for the track quality BDT that relies on stub information to evaluate track quality, as is shown in figure 7.5b. The number of the highest quality tracks has dropped as the track degrades but the number of mid quality tracks, BDT score of 0.7–0.95 has increased. This is likely due to many tracks with apparently good quality, high χ^2 scores, but with fewer than expected stubs, so these tracks score relatively highly but not as high as possible. There is little change to the lowest quality tracks which will already have four stubs and also low χ^2 scores. Not only does this demonstrate that detector degradation can cause significant changes to track parameters, but it creates non-trivial effects in ML algorithms trained on perfect detector conditions, careful consideration of how to use the BDT output or retraining of the BDT to allow for fewer stubs for high quality tracks is needed to maintain performance.

Due to the overall drop in the number of tracks across the detector and subtle changes to track quality, an



(a) (b)

Figure 7.5: L1 trigger reconstructed track quality parameter distributions for different levels of tracker degradation. The leftmost plot shows the number of stubs used to reconstruct the track distribution across the different samples with BRI referring to the bias rail inefficiency and X% bad strips referring to a random removal of random strips from the outer tracker. The rightmost plot shows the BDT distribution for the same set of samples. Figure (a) shows fewer stubs are being used to create tracks in the degraded samples, simply because there are fewer stubs in the detector. The BDT, that uses the number of stubs as input, has a complex result with more mid-quality tracks being produced but fewer very high quality and low quality tracks in the degraded samples.

event-level ML task was chosen as the example for using ML in this degraded detector environment.

Fake Vertex Identification

As has been discussed in chapter 5, the regression of the primary vertex is of high importance for the L1 trigger. The baseline approach to vertex finding, FastHisto, bins tracks in z_0 weighted by their p_T and then uses a 3-bin window to find a peak in the histogram. This proves effective when there is a high p_T cluster of many tracks in one region of z_0. An example of this kind of event is shown in figure 7.6 where the band of tracks at $z_0 = 1$ cm are tracks originating from the primary vertex and FastHisto has found this high p_T peak shown in the first row. Other variables such as the BDT score and number of stubs show that these tracks are of a high quality and more likely to be real tracks rather than incorrectly reconstructed fake tracks.

However, there are specific cases where the baseline FastHisto vertex finding algorithm incorrectly finds the vertex causing large tails in its truth to reconstructed vertex residual. The key reason for this is the presence of fake tracks; some high p_T fakes appear as isolated tracks in the z_0 histogram but with a high p_T such that the peak finding identifies these isolated clusters as the vertex. This kind of event can be seen in figure 7.7 where a high p_T fake is shown at $z_0 = 10$ cm, the low BDT score demonstrates this is a poor quality track more likely to be fake. The peak at $z_0 = -5$ cm is the true primary vertex but the peak in p_T is not large enough compared to the fake track p_T.

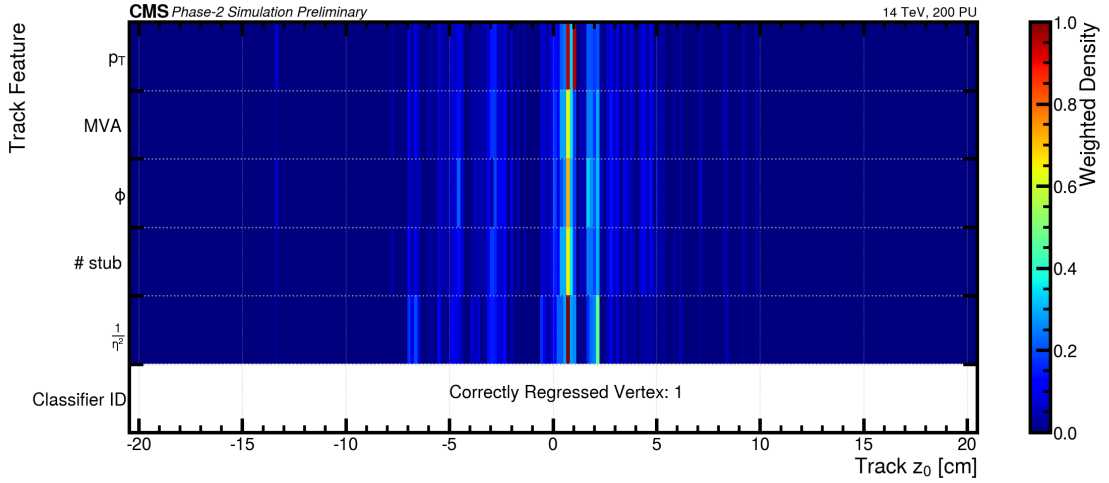


Figure 7.6: L1 trigger reconstructed track parameter two-dimensional histogram generated from histogramming all tracks in z_0 weighted by different track parameters and stacked. This specific two-dimensional histogram, taken from a top quark pair production sample shows an event where the baseline FastHisto vertex finding algorithm has correctly identified the vertex. The large band of tracks at z_0 = 1 cm is the true primary vertex with many tracks with high p_T and high BDT scores (labelled MVA), because of the high p_T peak, the vertex is well reconstructed by FastHisto.

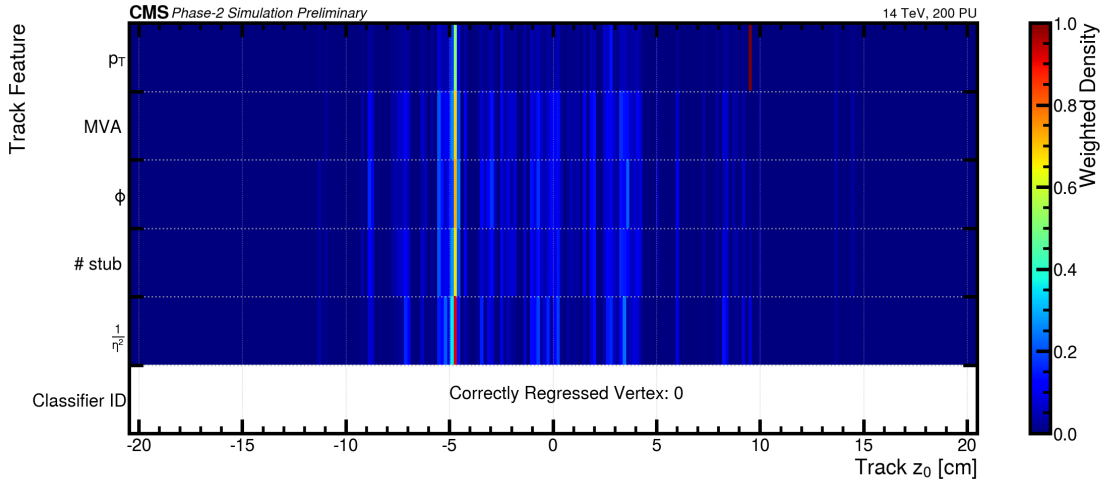


Figure 7.7: L1 trigger reconstructed track parameter two-dimensional histogram generated from histogramming all tracks in z_0 weighted by different track parameters and stacked. This specific two-dimensional histogram, taken from a top quark pair production sample shows an event where the baseline FastHisto vertex finding algorithm has incorrectly identified the vertex. The high, many track peak at z_0 = -5 cm is the true primary vertex. However, the high p_T, low BDT score isolated track, which is likely fake, at z_0 = 10 cm has higher p_T so is incorrectly selected as the primary vertex.

A classifier that aims to differentiate these kinds of events was used as the example algorithm for an L1 trigger with degraded samples. This is an event-level classification task with a clear dependence on number of tracks in a cluster and overall p_T of the cluster so is suitable with the datasets with tracker degradation. Two-dimensional histograms shown in figures 7.6 and 7.7 were created with different track parameters histogrammed over z_0. The track helix parameters selected; p_T, ϕ, and $\frac{1}{\eta^2}$ describe the track helix, with $\frac{1}{\eta^2}$ used as it is proportional to the track z_0 resolution therefore important in the quality of vertex regression. The track quality BDT score and number

of stubs were used to describe the quality of the track with the number of stubs being particularly important in these tracker degradation samples, as demonstrated by figure 7.5a. These two-dimensional histograms were used as input to a CNN classifier.

No Retraining Model

A CNN was designed to identify the fake vertices using the set of histograms shown in figures 7.6 and 7.7. The CNN is a small, five layer model with two convolutional layers and three dense layers and a final sigmoid activation function. A simple network was chosen in order to be similar to the types of networks used in the L1 trigger where size is very important. The CNN was trained using a large perfect detector simulation, the "no degradation" sample shown in the section above, of 200,000 events. The same model, without any retraining, was then evaluated on the test datasets for the no degradation sample and the four degraded samples. Figure 7.8 demonstrates with the black curve that the performance of the model in the domain it was trained in is very high, a 98% true positive rate with close to 0% false positive rate for the vertex identification task.

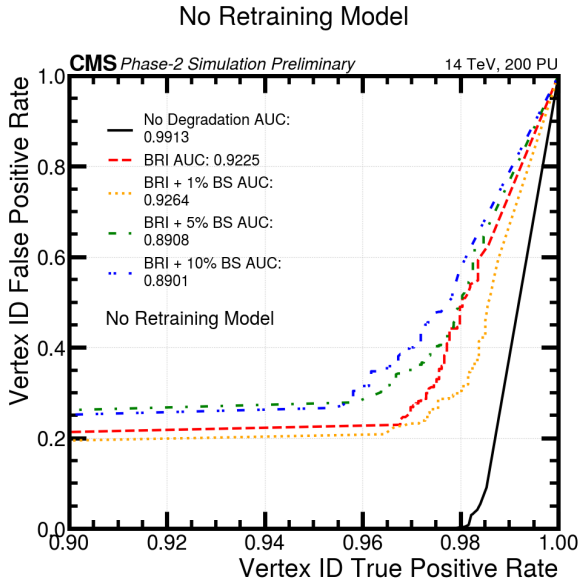


Figure 7.8: Receiver operating characteristic curve for the CNN trained only on the no degradation sample. Performance in the corresponding test dataset of the no degradation sample is shown in black, the other degraded samples are shown in the other colours. The model's performance in its training domain is very high, shown by the black curve. The drop in performance on the other samples is due to the degradation; as the degradation increases the performance drops. There is a small difference when comparing the BRI and BRI + 1% BS samples due to the relatively low statistics of the testing samples.

The performance in the other samples is significantly worse, as shown by the other coloured lines. The introduction of the BRI in red causes the area under the curve to drop 7% with a significantly higher false positive rate of 20% at very high true positive rates. The addition of the 1% bad strips makes little difference compared to just the BRI sample, with higher loss in strips dropping the performance even more. Overall it is clear that the model is not robust to OOD samples and is ill-prepared for a degraded detector. These results clearly demonstrate the issue with ML with the degraded tracker; if this model had been left without any re-training the performance for downstream algorithms would have been worse.

Top-Up Training

In order to improve the performance of the model in these changed environments, the CNN was top-up trained in a scheme shown in figure 7.1d where new, smaller datasets (5000 events) of the degraded samples were introduced to the model one sample at a time. For training, a lower learning rate was used so as not cause the model to vary far from its previous working point, since it is expected that the loss function minima will have moved slightly, and a low learning rate will move the model gradually to this. After each training, with the new data samples, the model was evaluated on the corresponding test data set. Figure 7.9 shows that performance across the degraded samples has improved, the ROC AUC has improved by around 2% across the samples with the false positive rate dropping to 10% for high true positive rates.

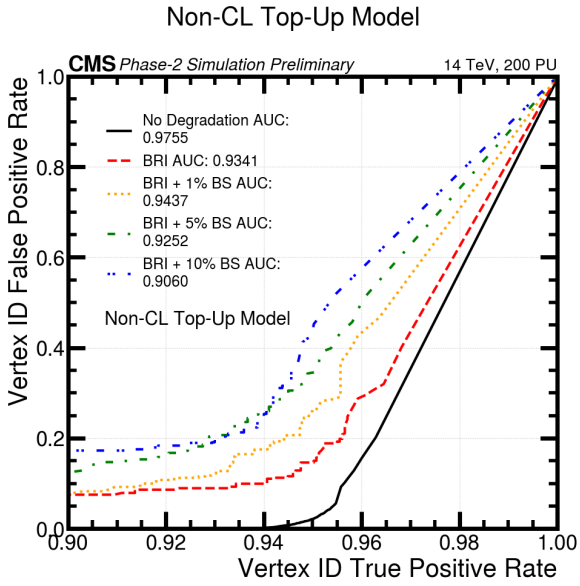


Figure 7.9: Receiver operating characteristic curve for the CNN trained on the no degradation sample and then top-up trained on the degraded samples. Performance in the corresponding test dataset of the no degradation sample is shown in black, the other degraded samples are shown in the other colours. The top-up trained model has improved its performance in the degraded samples as it has been trained to ignore the noise that the degradation introduces, it is clear that larger degradations are still harder to learn from. The model has lost its performance in the original domain because of the top-up training which is an example of a model forgetting its previous training.

This is encouraging, showing that doing some level of top-up training to the model helps its performance, as the detector degrades. However, the performance in the original no degradation sample as dropped, for the same close to 0% false positive rate the true positive rate has dropped to 94%. This is not a significant decrease but demonstrates some level of forgetting in the model that shows the retraining has in some way perturbed the original learning. While in this specific example the degradation might not be transitory, there are examples where forgetting previous training is problematic. For example, in the ECAL the crystals that are optically damaged by radiation recover over time, meaning a model trained on the non-damaged crystals and then retrained on the damaged ones will have worse performance as the crystals recover.

Continual Learning Top-Up Training

As has been discussed, there are better ways of treating NN top-up training with the inclusion of CL algorithms. In this example case a CL algorithm was chosen for the top-up trainings where each new set of degraded samples

were introduced to the model as new experiences to learn from, shown by figure 7.10. A replay buffer strategy was used that kept a running buffer of events as the model trains so that a small number of previous events in previous samples are kept and replayed to the model as it learns. This means the model is able to maintain its performance across all the samples and the retraining from any one sample does not impact the training from the others. The Avalanche end-to-end library for CL was used for the splitting of the data into experiences and incremental learning [135].

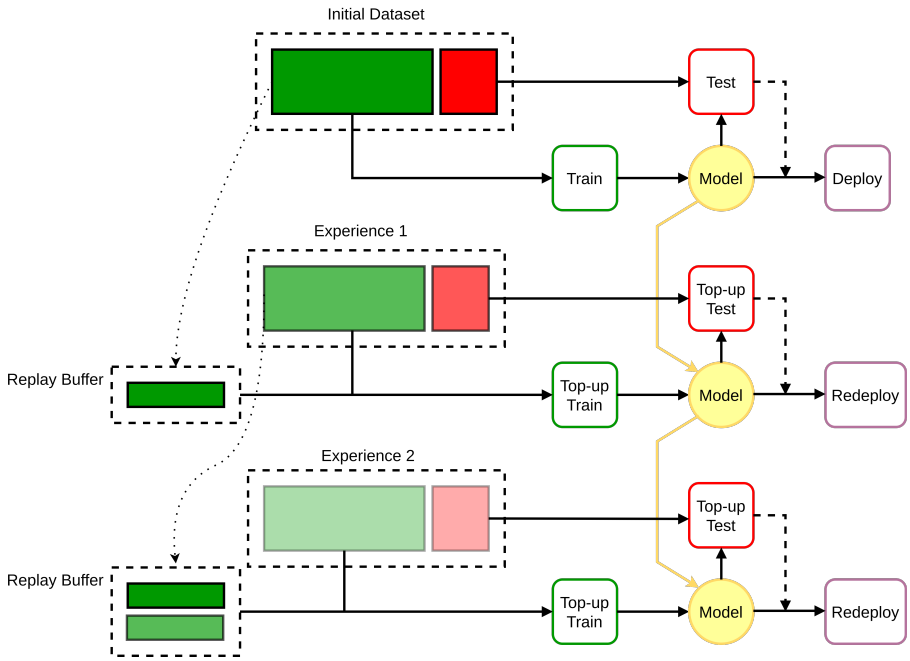


Figure 7.10: Diagram for the CL replay retraining scenario, shown is a training dataset in green used for training the model and a testing dataset in red for testing the model. Pending a successful test, the trained model is deployed. After each experience a small part of the previous experience is added to the replay buffer and also use to top-up train the model.

Figure 7.11 shows that this has a strong impact on the performance of the model. The performance across all the degraded samples has significantly improved; the false positive rate has dropped to close to 0 % for all the samples at high true positive rates. The overall performance across all the different samples is similar, demonstrating that the CL trained model is more robust to the changing detector conditions as it has learnt and maintained its knowledge across the range of the degraded detector. This means it has learnt features from the data that are independent of the detector degradation, something the other models have been unable to do. This is a desirable feature for ML models in the L1 trigger which need to have robust performance as the detector changes. There is worse performance in the original, no degradation, dataset as compared to the model with no top-up training; some information that was learnt has been forgotten by the network but less than when no CL was performed. This is a result of using the replay buffer strategy where the model is constantly re-learning from the previous data it has seen. Other techniques such as weight regularisation have yet to be explored, which could prevent any forgetting.

Overall it is clear that a CL strategy to updating this example L1 trigger model outperforms a basic top-up training. The model is more robust to the changed environment. There is a small computational overhead for storing a replay buffer of events for the model to retrain on, but there are other CL methods that avoid this that can be explored. CL is a promising solution to the issue of changing environments for ML in the L1 trigger. More

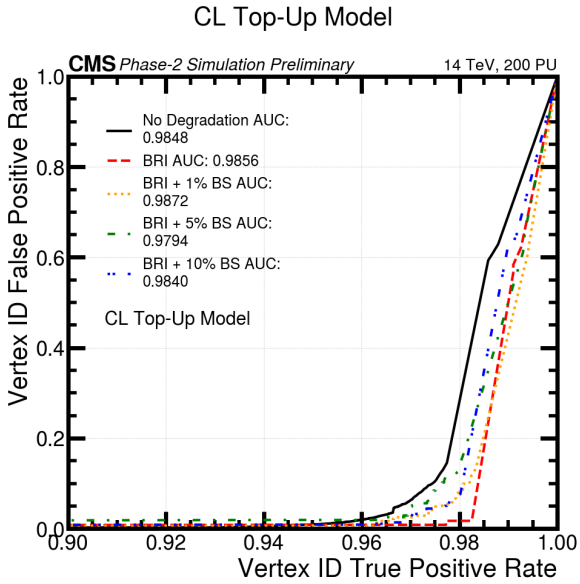


Figure 7.11: Receiver operating characteristic curve for the CNN trained on the no degradation sample and then top-up trained on the degraded samples using a replay buffer continual learning algorithm. Performance in the corresponding test dataset of the no degradation sample is shown in black, the other degraded samples are shown in the other colours. The performance across all degraded samples has improved, when comparing figures 7.8 and 7.9, the CL model is more robust to the changing environment. The performance in the original domain, while better than the top-up trained model in figure 7.9, is not as good as the model solely trained in this domain in figure 7.8. The CL model is able to forget less about its original training but still suffers from some level of forgetting.

research is needed with a variety of different models across the L1 trigger to understand the possible impact of the CMS environment and the methods for maintaining high performing ML algorithms for triggering.

7.4 Conclusions

The use of ML in the changing environment of the CMS L1 trigger poses a problem for these algorithms. Many make important trigger decisions, so their inferences need to be robust and reliable. However, they are ill-prepared for a changing environment. The exact nature of the changing environment is unknown and the model's response to it even more so, therefore solutions are needed to maintain ML performance and not lose physics sensitivity. There are many options for improving the performance of an ML algorithm in a changing inference environment.

Techniques improving the robustness of models to OOD predictions are common, with LHCb already using Lipschitz constrained models for their trigger systems. This is a relatively simple solution that can be adopted by many ML models planning to be used in the CMS L1 trigger, but the learning dynamics of this approach coupled with the aggressive size and latency reduction methods already in use may not yield good performance. Some consideration of model robustness should always be part of the training of L1 trigger ML.

Retraining and redeploying the ML should be a baseline approach for the ML in the trigger. Regularly updating and calibrating trigger algorithms is common practice in CMS and ML should be no exception. Collecting datasets and using them to retrain the models can easily be done at the commissioning steps of the L1 trigger and even smaller updates at the fill level are feasible if the ability to update ML models without rebuilding complex firmware becomes a reality.

Uncertainty quantification is a powerful technique that has the significant advantage of passing on a certainty of a model's prediction downstream, something that allows informed decisions to be made with an ML inference. However, common uncertainty quantification techniques require a large computational overhead such as multiple evaluations of a model which makes them unsuitable for a fast triggering environment.

Finally, a continual learning approach to ML in the L1 trigger has potential to improve the performance of these models as the environment changes and can result in a model that is more robust to the changing environment. An example algorithm trained to identify fake vertices and deployed in a degraded tracker environment has its performance significantly improved by using a CL workflow. This is the first use of CL in CMS or any of the LHC experiments and further research is needed to understand how this can be applied to existing ML algorithms being used in online systems across these experiments.

Chapter 8

Conclusion

The high luminosity upgrade of the Large Hadron Collider (LHC) will produce unprecedented amounts of data for its associated experiments. This increase in data will allow precise characterisation of the Higgs boson and will facilitate the search for possible new physics. In order to maximise its performance in this new era, the CMS experiment is to be upgraded. These upgrades include a replacement of the endcap calorimetry and tracker in order to deal with the increased radiation and improve their performance in the high pileup environment. The outer tracker modules are to be replaced with specialised two-layer modules which can perform an on-detector p_{T} cut on charged particle tracks, needed to provide tracks to the Level-1 trigger. This hardware-based Level-1 trigger will also be upgraded with a longer latency, higher rate and input from the outer tracker for the first time. These upgrades are essential to maintain the physics sensitivity with the high pileup environment. Techniques, such as vertex finding and particle flow, will be possible for the first time and extensive use of machine learning (ML) will be seen in the trigger, something that is only possible with the all-FPGA architecture using large, resource abundant FPGAs.

Track reconstruction at the Level-1 trigger is under development. A hybrid algorithm using a tracklet seeding step and a Kalman filter for track fitting will see charged particle tracks of $p_{\mathrm{T}} > 2$ GeV being reconstructed in less than 4 μs at the full 40 MHz bunch crossing rate. Significant work is ongoing in the implementation of this algorithm in firmware with various subchains of the full system already being tested. The final step of the algorithm, designed to route and format fitted tracks has been implemented. This Kalman filter output step calculates final track parameters to be transferred to the rest of the Level-1 trigger system and routes the tracks in η ready for transmission over fibre optics. The firmware, written in VHDL, has been tested on an Apollo board with a total latency of 81 ns and sub 2% resource usage. The main target for the track reconstruction in the coming years is the realisation of the full algorithm in hardware with a split over two large FPGAs on a single board. The way in which the algorithm is split and the transfer of data between FPGAs will need to be taken into account to deliver an algorithm with high track finding efficiency and a sub 4 μs latency.

Part of the track reconstruction is the creation of tracks not originating from true charged particle tracks. These "fake" tracks are a result of the high combinatorics of detector hits and can pose issues for downstream algorithms. The presence of high p_{T} fake tracks can lead to incorrect vertex finding and can also lead to mis-identification of particles in downstream Level-1 trigger algorithms. A lightweight boosted decision tree (BDT) has been trained to identify fake tracks based on some geometric information as well as the overall quality of the track fit. Which detector layers are hit and overall number of hits also provide useful discrimination between real and fake tracks. The BDT can maintain the same track finding efficiency and halve the overall fake track rate, without introducing geometrical biases that cuts on just the track χ^2 create. The BDT has been implemented in firmware using the

conifer package with a 36 ns latency and < 0.5% resource usage. Incremental improvements to this method of fake track rejection are expected as the track finding algorithm evolves. Other possibilities to improve its performance include using stub residual information directly from the Kalman filter either to remove fake states in the Kalman filter itself or to give a more detailed information about track quality to downstream algorithms.

A key subsystem of the Level-1 trigger that uses the reconstructed tracks is the global track trigger (GTT). It is designed to perform primary vertex finding as well as calculating track-only quantities such as track jet finding and track-based $E_{\mathrm{T}}^{\mathrm{miss}}$. The reconstruction of the primary vertex in an event is essential for discriminating between pileup events and the interesting physics signature of the hard scatter. This information is used by the pileup per particle identification algorithm downstream to weight candidates and reduce the impact of pileup on full detector jet finding and energy sums. The baseline method for primary vertex finding, FastHisto (FH), is an algorithm that bins tracks in z_0 weighted by their p_{T} and finds the peak after a three-bin window is run along the histogram. This method is effective and fast but lacks discrimination for fake tracks or consideration of geometric features that change the reconstructed track z_0 resolution. Different weightings for the FH algorithm were explored that introduce track χ^2, track quality BDT output and track η. Individually these weightings improved the performance of the vertex finding in different ways, but their combination did not yield overall improvements.

An alternative approach to finding an ideal weighting for FH is in the form of a dense neural network. This method, when combined with a convolutional neural network for peak identification and a dense neural network for track-to-vertex association formed the end-to-end approach to vertex finding and association. This three-part neural network trained with a two-part loss function finds an ideal weighting for the histogram based on vertex location and overall track-to-vertex performance giving a better overall performance across both metrics; a 34% improvement in the root-mean-square of the vertex finding residual and half the rate of tracks being incorrectly assigned to a vertex. The network was designed to be compatible with the strict resource and latency requirements of the Level-1 trigger with extensive use of quantisation and pruning to reduce the network size further. The network was implemented in firmware using the hls4ml tool but with a variety of modifications due to the custom network structure. Overall, the network can find the vertex in 100 ns with 30% of the resources of the target chip for all replications of the network parts. While this approach is shown to improve the track-based $E_{\mathrm{T}}^{\mathrm{miss}}$ performance, the overall impact for downstream algorithms is yet to be fully understood, and current work focusses on this. Progress on fully integrating this network within the existing Level-1 trigger firmware continues and new strategies for reducing the network size further are being explored.

Part of the testing of the Level-1 trigger is testing algorithms on their target FPGA and hardware processing boards. These tests involve loading algorithm firmware onto FPGAs and comparing the output of the firmware to emulation in a variety of stringent tests. Other tests include fibre optic communications to verify overall systems integration as well as fibre optic integrity and board functionality. Various subchains of the track finding firmware were tested as well as the track quality BDT with high agreement for 1000 events run through the algorithms. Single GTT algorithms were also tested as well as the communication between the track finding and GTT algorithms. A version of the FH vertex finding was tested using the input via fibre optics from a track finding board with 18 separate links of tracks being fed into the vertex algorithm. These tests will be continually repeated to verify updates to firmware as the Level-1 trigger project progresses and automatic testing protocols are being developed to streamline this process.

Extensive use of ML is planned for the Level-1 trigger, however there are open questions over the performance of this ML in the dynamic environment of the CMS detector. To deal with changing environments there are a number of solutions. Training models in robust ways can lead to better performance under domain shift. These

methods typically introduce additional regularisation to models in order to find the most important parts of a feature set that are constant under changing domain. While these methods can be effective, study into the interplay between these regularisations and the techniques used to create light-weight fast ML for a trigger scenario has yet to be seen. Other methods include uncertainty quantification that can highlight domain shift as it happens. While not a solution to domain shift, it provides more information to downstream algorithms to take into account the possible uncertainty in a model's output. Typical methods include simple ensemble methods with multiple passes though similar models or more robust Bayesian neural networks with stochastic weights and biases. The suitability of these methods in a low latency trigger environment is questionable because of the additional overhead involved, but research into fast uncertainty quantification is ongoing. A final method to deal with a changing environment is with continual learning (CL), where an ML model is updated as the environment changes so that performance remains constant in the changing inference environment. Various methods, including regularisation techniques or replaying old data to the model can improve the overall robustness of models. The use of CL in the context of a Level-1 algorithm was explored with a simulation of the degraded CMS tracker. An algorithm, designed to identify fake vertices, performed poorly on degraded samples when no retraining or consideration of this degradation was performed. With simple top-up training the model improved but with a CL replay approach the model maintained its performance across the degraded samples demonstrating the feasibility of this method in the Level-1 context. This is the first time CL has been applied to an online ML workflow at the LHC, but much more research is needed to understand how this method could be used and what are the practical implications for the wide range of ML that will be used in the Level-1 trigger.

References

[1] M. Gell-Mann, *The Eightfold Way: A Theory of Strong Interaction Symmetry*, (1961) (cit. on p. 21).

[2] E. Fermi, *Quantum Theory of Radiation*, Rev. Mod. Phys. **4** (1 1932) 87–132 (cit. on p. 21).

[3] S. L. Glashow, *Partial-Symmetries of Weak Interactions*, Nuclear Physics **22** (1961) 579–588 (cit. on p. 21).

[4] S. Weinberg, *A Model of Leptons*, Phys. Rev. Lett. **19** (21 1967) 1264–1266 (cit. on p. 21).

[5] P. W. Higgs, *Broken Symmetries, Massless Particles and Gauge Fields*, Phys. Lett. **12** (1964) 132–133 (cit. on p. 21).

[6] F. Englert and R. Brout, *Broken Symmetry and the Mass of Gauge Vector Mesons*, Phys. Rev. Lett. **13** (9 1964) 321–323 (cit. on p. 21).

[7] S. Bilenky, *Neutrino oscillations: From a historical perspective to the present status*, Nuclear Physics B **908** (2016) 2–13 (cit. on p. 21).

[8] J. Alcaraz et al., *Precision Electroweak Measurements and Constraints on the Standard Model*, (2010), arXiv: 1012.2367 [hep-ex] (cit. on p. 22).

[9] N. Arkani-Hamed et al., *The Littlest Higgs*, Journal of High Energy Physics **2002** (2002) 034 (cit. on p. 22).

[10] The LEP Working Group for Higgs Boson Searches, *Search for the Standard Model Higgs boson at LEP*, Physics Letters B **565** (2003) 61–75 (cit. on p. 22).

[11] The CDF Collaboration and The D0 Collaboration and the Tevatron New Physics and Higgs Working Group, *Combined CDF and D0 Upper Limits on Standard Model Higgs-Boson Production with up to 6.7 fb^{-1} of Data*, 2010, arXiv: 1007.4587 [hep-ex] (cit. on p. 22).

[12] J. H. Oort, *The Force Exerted by the Stellar System in the Direction Perpendicular to the Galactic Plane and some Related Problems*, Bull. Astron. Inst. Netherlands **6** (1932) 249–287, URL: http://cds.cern.ch/record/436532 (cit. on p. 22).

[13] S. P. Martin, *A Supersymmetry Primer*, Advanced Series on Directions in High Energy Physics (1998) (cit. on p. 22).

[14] L. R. Evans and P. Bryant, *LHC Machine*, JINST **3** (2008) S08001. 164 p (cit. on p. 23).

[15] The ATLAS Collaboration, *ATLAS: Detector and Physics Performance Technical Design Report. Volume 2*, (1999), URL: https://cds.cern.ch/record/391177 (cit. on p. 23).

[16] The CMS Collaboration, *The CMS Experiment at the CERN LHC. The Compact Muon Solenoid Experiment*, JINST **3** (2008) S08004 (cit. on p. 23).

[17] The LHCb Collaboration, *The LHCb Detector at the LHC*, Journal of Instrumentation **3** (2008) S08005 (cit. on p. 23).

[18] The ALICE Collaboration, *The ALICE experiment at the CERN LHC*, Journal of Instrumentation **3** (2008) S08002 (cit. on p. 23).

[19] *Updated LHC Schedule*, Accessed: 2023-06-02, URL: https://hilumilhc.web.cern.ch/article/ls3-schedule-change (cit. on p. 24).

[20] S. R. Davis, *Interactive Slice of the CMS Detector*, https://cds.cern.ch/record/2205172, 2016 (cit. on p. 25).

[21] The CMS Collaboration, *CMS Physics: Technical Design Report Volume 1: Detector Performance and Software*, CERN, 2006, URL: https://cds.cern.ch/record/922757 (cit. on p. 25).

[22] A. Denner et al., *Standard Model Higgs-boson Branching Ratios with Uncertainties*, The European Physical Journal C **71** (2011) 1753 (cit. on p. 27).

[23] The CMS Collaboration, *The CMS Tracker System Project: Technical Design Report*, Technical design report. CMS, CERN, 1997, URL: https://cds.cern.ch/record/368412 (cit. on p. 28).

[24] S. Baffioni et al., *Electron reconstruction in CMS*, tech. rep., 2006, URL: http://cds.cern.ch/record/934070 (cit. on p. 28).

[25] W. Erdmann, *Vertex reconstruction at the CMS experiment*, Journal of Physics: Conference Series **110** (2008) 092009 (cit. on p. 29).

[26] The CMS Collaboration, *The CMS Phase-1 Pixel Detector Upgrade*, Journal of Instrumentation **16** (2021) P02027 (cit. on p. 29).

[27] The CMS Collaboration, *The CMS Electromagnetic Calorimeter Project: Technical Design Report*, Technical design report. CMS, CERN, 1997, URL: https://cds.cern.ch/record/349375 (cit. on p. 29).

[28] The CMS Collaboration, *Time reconstruction and performance of the CMS electromagnetic calorimeter*, Journal of Instrumentation **5** (2010) T03011 (cit. on p. 30).

[29] C. Biino, *The CMS Electromagnetic Calorimeter: overview, lessons learned during Run 1 and future projections*, Journal of Physics: Conference Series **587** (2015) 012001 (cit. on p. 30).

[30] The CMS Collaboration, *The CMS Hadron Calorimeter Project: Technical Design Report*, (1997), URL: https://cds.cern.ch/record/357153 (cit. on p. 30).

[31] The CMS Collaboration, *Particle-Flow Event Reconstruction in CMS and Performance for Jets, Taus, and MET*, tech. rep., 2009, URL: https://cds.cern.ch/record/1194487 (cit. on p. 30).

[32] The CMS Collaboration, *The CMS Muon Project: Technical Design Report*, Technical design report, 1997, URL: https://cds.cern.ch/record/343814 (cit. on p. 30).

[33] The CMS Collaboration, *Performance of the CMS drift-tube chamber local trigger with cosmic rays*, Journal of Instrumentation **5** (2010) T03003 (cit. on p. 30).

[34] ". C. Collaboration", *Large CMS cathode strip chambers: design and performance*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **453** (2000) 182–187, Proc. 7th Int. Conf on Instrumentation for colliding Beam Physics (cit. on p. 30).

[35] The CMS Collaboration, *The CMS RPC detector performance and stability during LHC RUN-2*, Journal of Instrumentation **14** (2019) C11012 (cit. on p. 31).

[36] The CMS Collaboration, *CMS TriDAS project: Technical Design Report, Volume 1: The Trigger Systems*, Technical design report, URL: https://cds.cern.ch/record/706847 (cit. on p. 31).

[37] The CMS Collaboration, *The TriDAS Project: Technical Design Report, Volume 2: Data Acquisition and High-Level Trigger*, (2002), URL: http://cds.cern.ch/record/578006 (cit. on p. 32).

[38] The CMS Collaboration, *CMS Technical Design Report for the Level-1 Trigger Upgrade*, tech. rep., 2013, URL: https://cds.cern.ch/record/1556311 (cit. on p. 32).

[39] The CMS Collaboration, *Performance of the CMS Level-1 trigger in proton-proton collisions at $\sqrt{s} = 13$ TeV*, Journal of Instrumentation **15** (2020) P10017 (cit. on p. 32).

[40] The CMS Collaboration, *Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC*, Physics Letters B **716** (2012) 30–61 (cit. on p. 33).

[41] The CMS Collaboration, *A Portrait of the Higgs Boson by the CMS Experiment Ten Years After the Discovery*, Nature **607** (2022) 60–68 (cit. on pp. 33, 34).

[42] *CMS Public Supersymmetry Publications*, Accessed: 2023-05-01, URL: https://cms-results.web.cern.ch/cms-results/public-results/publications/SUS/SUS.html (cit. on p. 33).

[43] The CMS Collaboration, *Measurement of the Top Quark Mass Using a Profile Likelihood Approach with the Lepton+Jets Final States in Proton-Proton Collisions at $\sqrt{s} = 13$ TeV*, 2023, arXiv: 2302.01967 [hep-ex] (cit. on p. 34).

[44] The CDF Collaboration, *High-Precision Measurement of the W Boson Mass with the CDF II Detector*, Science **376** (2022) 170–176 (cit. on p. 34).

[45] O. Aberle et al., *High-Luminosity Large Hadron Collider (HL-LHC): Technical design report*, CERN Yellow Reports: Monographs, CERN, 2020, URL: https://cds.cern.ch/record/2749422 (cit. on pp. 35, 151).

[46] A. Dainese et al., *Report on the Physics at the HL-LHC, and Perspectives for the HE-LHC*, tech. rep., 2019, URL: https://cds.cern.ch/record/2703572 (cit. on pp. 35, 39).

[47] F. Bruno, *FPGA programming for beginners : bring your ideas to life by creating hardware designs and electronic circuits with SystemVerilog*, Packt Publishing, Limited, 2021, ISBN: 1-5231-4212-X (cit. on p. 35).

[48] AMD Xilinx, *UltraScale^{TM} Architecture and Product Data Sheet: Overview*, URL: https://docs.xilinx.com/v/u/en-US/ds890-ultrascale-overview (cit. on pp. 36, 49, 124).

[49] D. Contardo et al., *Technical Proposal for the Phase-II Upgrade of the CMS Detector*, tech. rep., 2015, URL: https://cds.cern.ch/record/2020886 (cit. on p. 39).

[50] The CMS Collaboration, "Collisions recorded by the CMS detector on 14 Oct 2016 during the high pile-up fill", 2016, URL: https://cds.cern.ch/record/2231915 (cit. on p. 40).

[51] ATLAS Collaboration, *Observation of Electroweak Production of a Same-Sign W Boson Pair in Association with Two Jets in pp Collisions at $\sqrt{s} = 13$ TeV with the ATLAS Detector*, Phys. Rev. Lett. **123** (16 2019) 161801 (cit. on p. 39).

[52] The CMS Collaboration, *Measurement of Vector Boson Scattering and Constraints on Anomalous Quartic Couplings from Events with Four Leptons and Two Jets in Proton-Proton Collisions at $\sqrt{s} = 13$ TeV*, Physics Letters B **774** (2017) 682–705 (cit. on p. 39).

[53] F. Veloso, *Searches of top FCNC interactions with the ATLAS detector*, (2022), URL: https://cds.cern.ch/record/2804177 (cit. on p. 40).

[54] The CMS Collaboration, *Prospects for HH measurements at the HL-LHC*, tech. rep., CERN, 2018, URL: http://cds.cern.ch/record/2652549 (cit. on pp. 41, 42).

[55] A. A. Savin and on behalf of the CMS Collaboration, *Future Physics with the CMS Experiment at the High-Luminosity LHC*, Physica Scripta **96** (2021) 124003 (cit. on p. 43).

[56] G. Hernández-Tomé, G. López Castro, and P. Roig, *Flavor violating leptonic decays of τ and μ leptons in the Standard Model with massive neutrinos*, The European Physical Journal C **79** (2019) 84 (cit. on p. 42).

[57] The CMS Collaboration, *The Phase-2 Upgrade of the CMS Level-1 Trigger*, tech. rep., 2020, URL: https://cds.cern.ch/record/2714892 (cit. on pp. 44, 50, 53, 54, 70, 95, 118, 150).

[58] The CMS Collaboration, *The Phase-2 Upgrade of the CMS Tracker Technical Design Report*, tech. rep., 2017, URL: https://cds.cern.ch/record/2272264 (cit. on pp. 45, 47, 69).

[59] G. Hall, M. Raymond, and A. Rose, *2-D PT module concept for the SLHC CMS tracker*, Journal of Instrumentation **5** (2010) C07012 (cit. on p. 45).

[60] The CMS Collaboration, *The Phase-2 Upgrade of the CMS Endcap Calorimeter*, tech. rep., 2017, URL: https://cds.cern.ch/record/2293646 (cit. on pp. 46, 49).

[61] R. Yohay, *The CMS High Granularity Calorimeter for High Luminosity LHC*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **958** (2020) 162151, Proceedings of the Vienna Conference on Instrumentation 2019 (cit. on p. 48).

[62] The CMS Collaboration, *The Phase-2 Upgrade of the CMS Barrel Calorimeters*, tech. rep., 2017, URL: https://cds.cern.ch/record/2283187 (cit. on p. 48).

[63] F. Ferri, *The CMS ECAL Phase-2 upgrade for high precision energy and timing measurements*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **958** (2020) 162159, Proceedings of the Vienna Conference on Instrumentation 2019 (cit. on p. 48).

[64] The CMS Collaboration, *The Phase-2 Upgrade of the CMS Muon Detectors*, tech. rep., 2017, URL: https://cds.cern.ch/record/2283189 (cit. on p. 48).

[65] D. Abbaneo et al., *CMS muon system phase 2 upgrade with triple-GEM detectors*, in proceedings of the *2015 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, 2015 pp. 1–4 (cit. on p. 48).

[66] The CMS Collaboration, *The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger*, tech. rep., 2021, URL: https://cds.cern.ch/record/2759072 (cit. on p. 49).

[67] E. S. Hazen et al., *The APOLLO ATCA Platform*, in proceedings of the *Topical Workshop on Electronics for Particle Physics — PoS(TWEPP2019)*, vol. 370, 2020 pp. 120 (cit. on pp. 49, 125).

[68] A. W. Rose et al., *Serenity: An ATCA prototyping platform for CMS Phase-2*, in proceedings of the *Proceedings of Topical Workshop on Electronics for Particle Physics — PoS(TWEPP2018)*, vol. 343, 2019 pp. 115 (cit. on pp. 49, 126, 127).

[69] Samtec, *Firfely^{TM} Application Design Guide*, URL: http://suddendocs.samtec.com/ebrochures/firefly-brochure.pdf (cit. on p. 49).

[70] G. Fedi et al., *Lessons Learnt from the First Vertical Slice of the CMS Outer Tracker*, Journal of Instrumentation **18** (2023) C01041 (cit. on pp. 49, 127).

[71] A. Zabi, *Design of the CMS calorimeter trigger upgrade from Phase I to Phase II of the LHC*, Journal of Physics: Conference Series **1162** (Jan. 2019) 012040 (cit. on p. 50).

[72] G. Petrucciani, *Particle Flow Reconstruction in the CMS Level-1 Trigger for the HL-LHC*, tech. rep., 2019, URL: https://cds.cern.ch/record/2650974 (cit. on p. 51).

[73] J. Duarte et al., *Fast Inference of Deep Neural Networks in FPGAs for Particle Physics*, Journal of Instrumentation **13** (2018) P07027–P07027 (cit. on pp. 54, 67, 113).

[74] T. O. James, *The Level 1 Scouting System of the CMS Experiment*, tech. rep., 2023, URL: https://cds.cern.ch/record/2852916 (cit. on pp. 55, 152).

[75] K. Albertsson et al., *Machine Learning in High Energy Physics Community White Paper*, Journal of Physics: Conference Series **1085** (2018) 022008 (cit. on pp. 57, 142).

[76] X. Huang, L. Wu, and Y. Ye, *A Review on Dimensionality Reduction Techniques*, International Journal of Pattern Recognition and Artificial Intelligence **33** (2019) 1950017 (cit. on p. 58).

[77] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag, 2006 (cit. on p. 58).

[78] R. Tibshirani, *Regression Shrinkage and Selection via the Lasso*, Journal of the Royal Statistical Society. Series B (Methodological) **58** (1996) 267–288 (cit. on pp. 58, 108).

[79] A. E. Hoerl and R. W. Kennard, *Ridge Regression: Biased Estimation for Nonorthogonal Problems*, Technometrics **12** (1970) 55–67 (cit. on pp. 58, 108).

[80] H. Robbins and S. Monro, *A Stochastic Approximation Method*, The Annals of Mathematical Statistics **22** (1951) 400–407 (cit. on p. 59).

[81] J. Goldberger et al., *Neighbourhood Components Analysis*, in proceedings of the *Advances in Neural Information Processing Systems*, vol. 17, MIT Press, 2004 (cit. on p. 59).

[82] M. Kramer, *Autoassociative Neural Networks*, Computers and Chemical Engineering **16** (1992) 313–328 (cit. on p. 59).

[83] J. R. Quinlan, *Induction of Decision Trees*, Machine Learning **1** (1986) 81–106 (cit. on p. 60).

[84] L. Breiman, *Bagging Predictors*, Machine Learning **24** (1996) 123–140 (cit. on p. 61).

[85] B. P. Roe et al., *Boosted Decision Trees as an Alternative to Artificial Neural Networks for Particle Identification*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **543** (2005) 577–584 (cit. on p. 61).

[86] T. Chen and C. Guestrin, *XGBoost: A Scalable Tree Boosting System*, in proceedings of the *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 2016 pp. 785–794 (cit. on p. 61).

[87] H.-D. Block, *The Perceptron: A Model for Brain Functioning.*, Reviews of Modern Physics **34** (1962) 123 (cit. on p. 61).

[88] S. Linnainmaa, *Taylor Expansion of the Accumulated Rounding Error*, BIT Numerical Mathematics **16** (1976) 146–160 (cit. on p. 62).

[89] K. Fukushima, *Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements*, IEEE Transactions on Systems Science and Cybernetics **5** (1969) 322–333 (cit. on p. 62).

[90] K. Fukushima, *Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position*, Biological Cybernetics **36** (1980) 193–202 (cit. on p. 63).

[91] F. Scarselli et al., *The Graph Neural Network Model*, IEEE Transactions on Neural Networks **20** (2009) 61–80 (cit. on p. 64).

[92] A. Elabd et al., *Graph Neural Networks for Charged Particle Tracking on FPGAs*, Frontiers in Big Data **5** (2022) (cit. on pp. 66, 93).

[93] C. N. Coelho et al., *Automatic Heterogeneous Quantization of Deep Neural Networks for Low-Latency Inference on the Edge for Particle Detectors*, Nature Machine Intelligence **3** (2021) 675–686 (cit. on pp. 67, 112).

[94] J. Ngadiuba et al., *Compressing Deep Neural Networks on FPGAs to Binary and Ternary Precision with HLS4ML*, Mach. Learn. Sci. Tech. **2** (2021) 015001 (cit. on p. 67).

[95] S. A. Janowsky, *Pruning Versus Clipping in Neural Networks*, Phys. Rev. A **39** (12 1989) 6600–6603 (cit. on p. 67).

[96] M. C. Mozer and P. Smolensky, *Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment*, in proceedings of the *Advances in Neural Information Processing Systems*, vol. 1, 1989 pp. 107–115 (cit. on p. 67).

[97] FastML Team, *hls4ml*, 2023, URL: https://github.com/fastmachinelearning/hls4ml (cit. on p. 67).

[98] S. Summers et al., *Fast Inference of Boosted Decision Trees in FPGAs for Particle Physics*, Journal of Instrumentation **15** (2020) (cit. on pp. 67, 91).

[99] T. Aarrestad et al., *Fast Convolutional Neural Networks on FPGAs with hls4ml*, Mach. Learn. Sci. Tech. **2** (2021) (cit. on p. 67).

[100] C. E. Brown, *CTD2022: Track Finding and Neural Network-Based Primary Vertex Reconstruction with FPGAs for the Upgrade of the CMS Level-1 Trigger System*, May 2022, URL: https://doi.org/10.5281/zenodo.8119492 (cit. on p. 69).

[101] E. Bartz et al., *FPGA-Based Tracklet Approach to Level-1 Track Finding at CMS for the HL-LHC*, EPJ Web of Conferences **150** (2017) 00016, ed. by C. Germain et al. (cit. on p. 70).

[102] G. Hall et al., *A Time-Multiplexed Track-Trigger for the CMS HL-LHC Upgrade*, Nuclear Instruments and Methods in Physics Research A **824** (2015) (cit. on p. 70).

[103] R. Frühwirth, *Application of Kalman filtering to track and vertex fitting*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **262** (1987) 444–450 (cit. on p. 71).

[104] S. Eidelman et al., *Passage of Particles Through Matter*, Physics Letters **B592** (2004) 1 (cit. on pp. 73, 87).

[105] S. Summers, "Application of FPGAs to Triggering in High Energy Physics", Ph.D. thesis Imperial College, London, 2018, URL: https://cds.cern.ch/record/2647951 (cit. on p. 86).

[106] AMD Xilinx, *Overview of Arbitrary Precision Fixed-Point Data Types*, URL: https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Overview-of-Arbitrary-Precision-Fixed-Point-Data-Types (cit. on p. 89).

[107] W. Adam et al., *Reconstruction of electrons with the Gaussian-sum filter in the CMS tracker at the LHC*, Journal of Physics G: Nuclear and Particle Physics **31** (2005) N9 (cit. on p. 93).

[108] *CTD2022: Graph Neural Networks for Pattern Recognition & Fast Track Finding*, Zenodo, June 2023, URL: https://doi.org/10.5281/zenodo.8119773 (cit. on p. 93).

[109] W. Waltenberger, R. Frühwirth, and P. Vanlaer, *Adaptive Vertex Fitting*, Journal of Physics G: Nuclear and Particle Physics **34** (2007) (cit. on p. 97).

[110] The CMS Collaboration, *Description and Performance of Track and Primary-Vertex Reconstruction with the CMS Tracker*, Journal of Instrumentation **9** (2014) 10009 (cit. on p. 97).

[111] A. Shtipliyski, "Systems and Algorithms for Low-Latency Event Reconstruction for Upgrades of the Level-1 Trigger of the CMS Experiment at CERN", Ph.D. thesis Imperial College, London, 2019, URL: https://cds.cern.ch/record/2708430 (cit. on pp. 104, 105, 108).

[112] A. Bordes et al., *Learning Structured Embeddings of Knowledge Bases*, Proceedings of the AAAI Conference on Artificial Intelligence **25** (2011) 301–306 (cit. on p. 105).

[113] C. Brown et al., *Neural Network-Based Primary Vertex Reconstruction with FPGAs for the Upgrade of the CMS Level-1 Trigger System*, Journal of Physics: Conference Series **2438** (2023) 012106 (cit. on p. 105).

[114] M. Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015, URL: http://tensorflow.org/ (cit. on p. 105).

[115] P. J. Huber, *Robust Estimation of a Location Parameter*, The Annals of Mathematical Statistics **35** (1964) 73–101 (cit. on p. 107).

[116] Y. Cheng et al., *Model Compression and Acceleration for Deep Neural Networks: The Principles, Progress, and Challenges*, IEEE Signal Processing Magazine **35** (2018) 126–136 (cit. on p. 112).

[117] A. Albert et al., *The Apollo ATCA design for the CMS track finder and the pixel readout at the HL-LHC*, Journal of Instrumentation **17** (2022) C04033 (cit. on p. 127).

[118] J. Hegeman et al., *First Measurements with the CMS DAQ and Timing Hub Prototype-1*, PoS **TWEPP2019** (2020) 111 (cit. on p. 135).

[119] J. Hegeman et al., *The CMS Timing and Control Distribution System*, in proceedings of the *2015 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, 2015 pp. 1–3 (cit. on p. 135).

[120] Y. Zhang et al., *A Survey on Neural Network Interpretability*, IEEE Transactions on Emerging Topics in Computational Intelligence **5** (2021) 726–742 (cit. on p. 142).

[121] L. Wang et al., *A Comprehensive Survey of Continual Learning: Theory, Method and Application*, 2023, arXiv: 2302.00487 (cit. on pp. 142, 147).

[122] L. Schmidt et al., *Adversarially Robust Generalization Requires More Data*, in proceedings of the *Advances in Neural Information Processing Systems*, vol. 31, 2018 (cit. on p. 143).

[123] O. Kitouni, N. Nolte, and M. Williams, *Robust and Provably Monotonic Networks*, 2023, arXiv: 2112.00038 (cit. on p. 143).

[124] H. Gouk et al., *Regularisation of Neural Networks by Enforcing Lipschitz Continuity*, 2020, arXiv: 1804.04368 (cit. on p. 143).

[125] Z. Liu et al., *Bridging Data Center AI Systems with Edge Computing for Actionable Information Retrieval*, in proceedings of the *2021 3rd Annual Workshop on Extreme-scale Experiment-in-the-Loop Computing (XLOOP)*, 2021 pp. 15–23 (cit. on p. 143).

[126] Hadsell, Raia and Rao, Dushyant and Rusu, Andrei A. and Pascanu, Razvan, *Embracing Change: Continual Learning in Deep Neural Networks*, Trends in Cognitive Sciences **24** (2020) 1028–1040 (cit. on p. 143).

[127] A. Kurz et al., *Uncertainty Estimation in Medical Image Classification: Systematic Review*, JMIR Med Inform **10** () e36427 (cit. on p. 145).

[128] Y. Ovadia et al., *Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift*, 2019, arXiv: 1906.02530 (cit. on pp. 145, 146).

[129] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press, 2022, URL: probml.ai (cit. on p. 145).

[130] T. G. Dietterich, *An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization*, Machine Learning **40** (2000) 139–157 (cit. on p. 146).

[131] Y. Gal and Z. Ghahramani, *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*, in proceedings of the *33rd International Conference on Machine Learning*, vol. 48, 2016 pp. 1050–1059 (cit. on p. 146).

[132] M. Havasi et al., *Training Independent Subnetworks for Robust Prediction*, 2021, arXiv: 2010.06610 (cit. on p. 146).

[133] J. Z. Liu et al., *Simple and Principled Uncertainty Estimation with Deterministic Deep Learning via Distance Awareness*, 2020, arXiv: 2006.10108 (cit. on p. 146).

[134] L. V. Jospin et al., *Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users*, IEEE Computational Intelligence Magazine **17** (2022) 29–48 (cit. on pp. 146, 147).

[135] V. Lomonaco et al., *Avalanche: an End-to-End Library for Continual Learning*, in proceedings of the *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021 pp. 3595–3605 (cit. on pp. 147, 148, 159).

[136] A. Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep LFarning Library*, in proceedings of the *Advances in Neural Information Processing Systems*, vol. 32, 2019 pp. 8024–8035 (cit. on p. 148).

[137] D. Rolnick et al., *Experience Replay for Continual Learning*, in proceedings of the *Advances in Neural Information Processing Systems*, vol. 32, 2019 (cit. on p. 148).

[138] G. M. van de Ven, H. T. Siegelmann, and A. S. Tolias, *Brain-Inspired Replay for Continual Learning with Artificial Neural Networks*, Nature Communications **11** (2020) 4069 (cit. on p. 148).

[139] J. Kirkpatrick et al., *Overcoming Catastrophic Forgetting in Neural Networks*, Proceedings of the National Academy of Sciences **114** (2017) 3521–3526 (cit. on p. 148).

[140] P. Pan et al., *Continual Deep Learning by Functional Regularisation of Memorable Past*, in proceedings of the *34th International Conference on Neural Information Processing Systems*, NIPS'20, 2020 (cit. on p. 148).

[141] H.-J. Chen et al., *Mitigating Forgetting in Online Continual Learning via Instance-Aware Parameterization*, in proceedings of the *Advances in Neural Information Processing Systems*, vol. 33, 2020 pp. 17466–17477 (cit. on p. 149).

[142] A. A. Rusu et al., *Progressive Neural Networks*, 2022, arXiv: 1606.04671 (cit. on p. 149).

[143] T. Zhu et al., *"Parameter-efficient Continual Learning Framework in Industrial Real-time Text Classification System"*, in proceedings of the *2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, 2022 pp. 315–323 (cit. on p. 149).

[144] Z. Mai et al., *Online Continual Learning in Image Classification: An Empirical Survey*, 2021, arXiv: 2101.10423 (cit. on p. 149).

[145] T. L. Hayes and C. Kanan, *Online Continual Learning for Embedded Devices*, 2022, arXiv: 2203.10681 (cit. on p. 149).

[146] S.-A. Rebuffi et al., *iCaRL: Incremental Classifier and Representation Learning*, in proceedings of the *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017 pp. 5533–5542 (cit. on p. 149).

[147] E. Fini et al., *Online Continual Learning Under Extreme Memory Constraints*, in proceedings of the *Computer Vision – ECCV 2020*, 2020 pp. 720–735 (cit. on p. 149).

[148] The CMS Collaboration, *The Phase-2 Upgrade of the CMS Beam Radiation, Instrumentation, and Luminosity Detectors: Conceptual Design*, tech. rep., 2020, URL: https://cds.cern.ch/record/2706512 (cit. on p. 151).

[149] M. A. Wadud, *The CMS ECAL Calibration and Monitoring*, tech. rep., 2019, URL: https://cds.cern.ch/record/2797776 (cit. on p. 151).

[150] The CMS Collaboration, *Strategies and performance of the CMS silicon tracker alignment during LHC Run 2*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **1037** (2022) 166795 (cit. on p. 151).

List of Acronyms

AP arbitrary precision.

ASIC application specific integrated circuits.

ATCA advanced telecommunications computing architecture.

BDT boosted decision trees.

BNN Bayesian neural network.

BRAM Block random access memory.

BRI bias rail inefficiency.

BX bunch crossing.

CL continual learning.

CM command module.

CMS Compact Muon Solenoid.

CNN convolutional neural network.

CoM centre-of-mass.

CSC cathode strip chamber.

DAQ data acquisition.

DNN dense neural network.

DSP Digital signal processor.

DT drift tube.

DT decision tree.

DTC data, trigger and control.

DTH DAQ and TCDS hub.

ECAL electromagnetic calorimeter.

EM electromagnetic.

EMP extensible, modular, data processor.

FF Flip flop.

FH FastHisto.

FIFO first in first out.

FPGA field programmable gate arrays.

FPR false positive rate.

GBDT gradient boosted decision trees.

GCT global calorimeter trigger.

GEM gas electron multipliers.

GMT global-muon trigger.

GT global trigger.

GTT global track trigger.

HCAL hadronic calorimeter.

HDL hardware description language.

HEP high energy physics.

HF hadronic forward.

HGCAL high granularity calorimeter.

HL-LHC High Luminosity LHC.

HLS high level synthesis.

HLT high level trigger.

I/O input/output.

II initiation interval.

IID independent and identically distributed.

IT inner tracker.

KF Kalman filter.

L1 Level-1.

LEP Large Electron Positron.

LHC Large Hadron Collider.

LUT Look-up table.

MC Monte Carlo.

MGTs multi gigabit transceivers.

ML machine learning.

MVA multivariate analysis.

NN neural network.

OOD out of distribution.

OT outer tracker.

PF Particle Flow.

PU pileup.

PUPPI pileup per particle identification.

PV primary vertex.

QCD quantum chromodynamics.

QED quantum electrodynamics.

RMS root-mean-squared.

ROC Receiver operating characteristic.

RPC resistive plate chamber.

SLRs super logic regions.

SM Standard Model.

SM service module.

SUSY Supersymmetry.

TCDS trigger control and distribution system.

TF track finder.

TFP track finder processor.

TPR true positive rate.

TPs trigger primitives.

VBF vector boson fusion.

VBS vector boson scattering.