UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Graph neural networks for event reconstruction at the Large Hadron Collider

DOI: 10.1002/0470841559.ch1

Volume n of m

by

Jacan Chaplais

MPhys Theoretical Physics

ORCiD: 0009-0002-4092-1878

A thesis for the degree of
Doctor of Philosophy

July 2025

University of Southampton

Abstract

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Doctor of Philosophy

Graph neural networks for event reconstruction at the Large Hadron Collider

by Jacan Chaplais

The Large Hadron Collider (LHC) serves as a powerful experimental platform for investigating the fundamental constituents of matter under extreme conditions. However, the reconstruction of boosted objects – highly energetic particles whose decay products manifest as dense sprays of hadrons or jets – presents enduring challenges. Accurate reconstruction of these objects is essential not only for validating the Standard Model but also for probing potential new physics phenomena such as heavy resonances and exotic particles. To address these challenges, this thesis explores the application of Graph Neural Networks (GNNs) for event reconstruction at the LHC. GNNs offer a transformative approach by leveraging the relational and geometric structure inherent in collider data. They are permutation invariant and generalise well to point clouds of variable size, which are ideal properties for collider physics. Additionally, they can be designed to respect physical properties such as Lorentz equivariance, rotational symmetry, and infrared and collinear safety, making them suitable for the complexities of jet clustering and boosted object reconstruction. This research explores GNN-based frameworks, incorporating novel methodologies to enhance reconstruction fidelity, and establishing new approaches to node prediction in message passing architectures.
This thesis presents our exploratory work as a kind of A - Z prototype for conducting Graph Machine Learning (ML) studies on simulated collider data. Three major contributions to the field are provided. First, we introduce a robust software ecosystem tailored for collider data analysis, enabling seamless data manipulation and model integration. Second, we propose an innovative clustering algorithm that dispenses with traditional jet definitions, instead incorporating simulation-based information, such as particle ancestry and momentum, to achieve superior clustering granularity. Finally, we demonstrate the application of IRC safe Interaction Networks with the novel Bright Edge Classification for effective node classification, and Cluster Double Sifting for the reconstruction of boosted Higgs bosons and top quarks, achieving state-of-the-art performance in providing detector-level constituents for the reconstruction of the top quark.

Contents

List of Figures **ix**

List of Tables **xiii**

Listings **xv**

List of Acronyms **xv**

Declaration of Authorship **xvii**

Acknowledgements **xix**

I Theory and methodology **1**

1 Introduction **3**

2 Particle detectors **7**
2.1 Cloud chambers 7
2.2 Modern experiments at the LHC 10
2.3 Producing tracks and towers 11
2.3.1 Temporal resolution and pileup 12
2.4 Detector subsystems 13
2.4.1 The inner tracking system 13
2.4.2 The electromagnetic calorimeters 14
2.4.3 The hadronic calorimeters 15
2.4.4 The solenoidal electromagnet 15
2.4.5 The Outer Hadronic Calorimeter 16
2.5 Detector geometry 17

3 QCD and parton showers **23**
3.1 The parton model and proton collisions 23
3.1.1 Radiation from outside the hard momentum transfer 24
3.2 Introducing jets 25
3.2.1 Parton Showering 25
3.2.2 Hadronisation 26

4 Jet Physics **29**
4.1 Jet clustering algorithms 29

4.1.1 The Snowmass accord . . . 29
4.1.2 Sterman-Weinberg jets . . . 30
4.1.3 Sequential recombination algorithms . . . 31
4.1.3.1 JADE algorithm . . . 32
4.1.3.2 Generalised k_T . . . 33
4.2 Infrared and collinear safety . . . 34
4.3 Jet grooming . . . 35
4.3.1 Filtering . . . 36
4.3.2 Trimming . . . 36
4.3.3 Pruning . . . 37
4.3.4 How does this relate to our work? . . . 38

5 Machine learning on graphs 39
5.1 Choosing models for collider data . . . 39
5.1.1 Multilayer perceptron . . . 39
5.1.2 Nonlinear activation functions . . . 40
5.1.3 Jet images . . . 42
5.1.4 Recurrent neural networks . . . 42
5.1.5 Transformers . . . 43
5.1.6 Generative models . . . 44
5.2 Graphs as flexible data structures . . . 45
5.2.1 Aside: distinguishing the generation DAG *vs.* the input graph . . . 46
5.2.2 Neighbourhood representations . . . 47
5.2.2.1 Edge List . . . 47
5.2.2.2 Adjacency list . . . 48
5.2.2.3 Adjacency matrix . . . 48
5.2.2.4 Laplacian matrix . . . 49
5.2.2.5 Incidence matrix . . . 49
5.2.3 Graph connectivity . . . 50
5.3 Message passing . . . 50
5.3.1 The graph network formalism . . . 51
5.3.2 Computational graphs . . . 54
5.3.3 Expressiveness is in the aggregation function . . . 55
5.4 Interaction networks . . . 57
5.4.1 Edge update block . . . 57
5.4.2 Vertex update block . . . 58
5.4.3 Global update block . . . 59
5.4.4 GN formalism and flavour considerations . . . 59
5.5 Energy weighted message passing networks . . . 60
5.5.1 Graph construction . . . 60
5.5.2 IRC safe message passing . . . 62

II Free and open source software contributions 65

6 Performance considerations for Python in HEP 67
6.1 Is Python fast enough? . . . 67

6.2 But what is a compiler? . . . 70

7 Semantic heterogeneous data structures with graphicle 75
7.1 Limitations in the current tools . . . 75
7.2 Graphicle for semantic and relational databases . . . 77
7.2.1 Data structures: components and composites . . . 78
7.2.2 Masking case study: momentum and PDG codes . . . 79
7.2.3 The Composite pattern for `MaskGroups` . . . 82
7.2.4 Programming paradigms and their role in `graphicle` . . . 85
7.2.4.1 Object-oriented programming . . . 85
7.2.4.2 Functional programming . . . 86
7.2.4.3 Polymorphism and reusability . . . 86
7.2.4.4 Blending paradigms for optimal design . . . 87
7.2.5 Topological information with AdjacencyList . . . 87
7.2.6 Heterogeneous data composites . . . 88
7.3 Beyond data structures: graphicle's modules . . . 93

8 Data generation with `showerpipe` 95
8.1 Using `pythia8` directly . . . 95
8.1.1 Generating data . . . 95
8.1.2 Difficulties in Python . . . 97
8.2 Wrapping `pythia8` to become Pythonic with `showerpipe` . . . 98
8.2.1 The `PythiaGenerator` interface . . . 98
8.2.2 The `PythiaEvent` interface . . . 99
8.3 Generation DAGs as a view on showering and hadronisation . . . 100

9 Preprocessing techniques for generating supervision targets 103
9.1 Clustering via graph traversal . . . 103
9.2 Improved clusters with momentum matching . . . 104
9.2.1 In praise of anti-k_T . . . 109
9.3 Process dependence . . . 110

III Experimental results 113

10 Higgs boson reconstruction using EWMP networks 115
10.1 Link prediction as clustering . . . 115
10.1.1 Formulation . . . 115
10.1.2 Challenges . . . 116
10.2 Architecture . . . 118
10.2.1 Input graph structure . . . 118
10.2.2 Bright edge activation . . . 119
10.2.3 The feed-forward mechanism and hyperparameter tuning . . . 122
10.3 Dataset generation . . . 124
10.4 Results . . . 125

11 Top quark reconstruction with cluster double sifting 129
11.1 Dataset generation . . . 130

11.2 Applying the model without modification . . . 131
11.3 Upgrading performance using a double sift . . . 132
11.3.1 First sift: considering initial results as noise removal . . . 132
11.3.2 Second sift: shifting and fully connecting the input . . . 133
11.3.3 Classifier correlations motivating the double sift approach . . . 134
11.4 Results . . . 135
11.5 Further work . . . 138

12 Conclusions **141**
12.1 Summary of key findings . . . 141
12.2 Contributions to the Field . . . 142
12.3 Future Directions . . . 142
12.4 Final Thoughts . . . 143

Appendix A Converting `motherList` to COO edges **145**

Appendix B Proof: removing cluster constituents lowers mass **149**

Appendix C Performance benchmarks for graphicle **151**

Appendix D Top quark reconstruction with ancestry information using jet images **159**
Appendix D.1 Disclaimer . . . 160

Appendix References **161**

List of Figures

2.1 Photograph of the cloud chamber track observed by Anderson, proving the existence of the positron. . . . 9
2.2 Showing the detector subsystems of CMS building outwards as the radius increases (Sirunyan et al. (2017)). . . . 13
2.3 Energy resolution for pions as a function of beam energy measured with ECAL + HB, and with ECAL + HB + HO for the beam being shot at **(a)** $\eta = 0.22$ and **(b)** $\eta = 0.56$ (Abdullin et al.). . . . 18
2.4 Describing how detectors develop tracks and towers for various kinds of particles, highlighting specifically which subsystems are active for which particles (Rizzi (2018)). . . . 19
2.5 Final state particles for a $p\ p \to t\bar{t}$ process, in which t quarks decay into b quarks and $W^{\pm}$ bosons. For the t quark the hadronic decay $W^{+} \to c\ \bar{s}$ occurs, whereas the $\bar{t}$ results in the leptonic decay $W^{+} \to e^{-}\ \bar{\nu}_{e}$. Marker size indicates hardness, given as $\log(1 + p_T)$. The marker styles shows which final state parton from the hard process the displayed particles reproduce (background is UE). Details on how these clusters were formed is given in chapter 9. Cuts of $p_T > 0.5$ and $|\eta| < 2.5$ are applied to approximate tracking and calorimeter sensitivity in CMS's detector barrel. 20
2.6 Cylindrical geometry of the inner detector wall, displayed both as a cylindrical wireframe, and the unrolled form which we hereon refer to as the $\eta - \phi$ plane. . . . 21

3.1 Depicting the interaction of two partons from colliding protons, with a centre-of-mass energy of 13 TeV. The energy transfer goes on to produce a pair of top quarks, which are longitudinally boosted in the centre-of-mass / lab frame. . . . 24

5.1 The generic Graph Network architecture, from Shlomi et al. (2021). . . . 52
5.2 Demonstrating the node update function as a rooted subtree centred at the receiving node, with grey boxes representing the shared weights at each layer. As layers are stacked, the computational graph grows, representing additional hops in the input graph. . . . 55
5.3 Visualising data flow for the three flavours of GNN, based on message normalisation (Bronstein et al., 2021). . . . 56

5.4 A k-nearest neighbour graph in the (η, ϕ)-plane will have a different structure when any particle q splits to r and s. The set $\mathcal{S}$ denote the particles in the jet when there is no splitting, while $\mathcal{S}'$ denotes the particles with q splitting. We show the directed edge connection to i from its three nearest neighbours with red on either side. The neighbourhood set $\mathcal{N}(i)$ has b in it, however when q splits, $\mathcal{N}'(i)$ does not contain b. Therefore, the graph's structure prevents a smooth extrapolation between the two scenarios in the infra-red and collinear limit. This is not the case for a radius graph with radius R_0 in the (η, ϕ) plane, which is shown with black connections. We also include the self-loop of i, by using the closed neighbourhood sets $\mathcal{N}(i)$ and $\mathcal{N}'(i)$, since the node i could also split into two particles (Konar et al. (2022)). 62

9.1 Comparison of mass reconstructed with `graphicle.select.hierarchy()` *vs.* Monte-Carlo truth mass for partons in the hard event. `hierarchy()` approximates the momentum of the parton in the hard event by summing the momenta of the topological descendants of the parton in the final state, found via BFS. Masses were computed over the 100k events provided in the test datasets for the respective partons. 105
9.2 Demonstrating the overly-inclusive definition of clusters reconstructing a hard parton by simply forming them from the hard parton's descendants. Topological descendants from the t quark in the generation DAG can be seen to collimate in several distinct regions, where only one is centred on the relevant b descendant prior to hadronisation. We see that hadronisation mixes ancestry with the UE. 108
9.3 Single top quark mass reconstruction improved by matching cluster constituent directions to the top quark descendants before hadronisation, as per equation 9.2. Bin width is 100 MeV. 110
9.4 Feynman diagrams depicting the processes during the proton-proton collision, which we study in this thesis using our MCEGs. Time runs from left to right, such that incoming particles are on the left, virtual mediating particles are horizontally aligned in the centre, and outgoing particles are on the right. 111
9.5 Improved single top quark mass reconstruction for $pp \to t\bar{t}$. Bin width is 300 MeV. 112

10.1 This graph shows an event which is structured as a graph input structure. Here our model's output predictions are superimposed, where the node illumination / confidence score is given by the colour. The shape of the markers indicates the supervision targets, where crosses represent UE and circles represent Higgs descendants. If the markers are filled, the model correctly predicted their class, and if the marker is empty the model incorrectly predicted the class, *eg.* an empty circle represents a false negative. 120
10.2 Cartoon depiction of the node illumination from incoming bright edges. If just one incoming edge is bright, then the node itself is illuminated / bright. Otherwise, the node is dark. A node may self-illuminate via a self loop. 122

10.3 Comparison of mass reconstructions of boosted Higgs bosons. The graphicle trace represents the target based on ancestry information. The Predicted trace represents the model. Anti-k_T is shown for comparison with standard techniques. 126
10.4 Demonstrating the deviation of Higgs boson mass reconstructions produced by our model when the hardest constituent of the Higgs cluster undergoes a split with a range of energy fractions and angles to the original particle. 128

11.1 Depicting the predictions after a single application (sift) of our GNN model, superimposed with the predictions with the false positive predictions removed, and the ground truth (Graphicle). 135
11.2 2D heatmaps indicating the distribution of predicted top quark clusters filtered by classifier categories of true positive (TP) labels, false positve (FP) labels, and target supervision labels. The origins are shifted to their respective centres. The TP and FP heatmaps represent the same underlying cluster, so share an origin. The target cluster uses its own origin. These are averaged over to 100k events of the test dataset. 136
11.3 Heatmap of the difference between true positive and false negatives. Negative regions indicate locations in the $\eta - \phi$ plane where FNs are more common than TPs. 138
11.4 We compute histograms to perform a comparison of the top-quark mass reconstruction, over the 100,000 events of the validation dataset. Graphicle's clusters are formed using the complete event record, and then kinematic cuts of $|\eta| < 2.5$ and $p_T > 0.5$ are applied. The cutoff radius for wide angle radiation used by graphicle is 1.0. Anti-k_T is applied over the final state particles with the kinematic cuts already applied. We pass $R = 0.35$, and use a Monte-Carlo truth based tagger to identify the b and $q\ \bar{q}$ jets. Predicted shows the performance of our GNN model with a double-sifting approach, trained on the graphicle clusters. The resolution of the graphicle mass histogram shows a striking improvement to resolution, when compared with the anti-k_T approach. Our model shows that it is able to learn patterns in these improved clusters, resulting in a significant improvement in resolution to the anti-k_T clusters, as well. In particular, the double-sifting approach reduces the tail of high mass reconstructions observed both from single-sift and anti-k_T approaches. . . 139
11.5 Heatmap representing the model's confidence at various levels of error, where a perfect reconstruction is at $\delta m = 50\%$. Each row are the distribution of confidence scores for the nodes in all events within that mass reconstruction error range. 140

Appendix C.1 Depicting the bit layout of a 32-bit floating point number (Wikipedia contributors (2025a)). 155

List of Tables

2.1 Typical temporal resolutions and deadtimes of common charged particle detectors (Particle Data Group et al., 2020). 12

7.1 Table of contents for `graphicle`'s `data` module, providing all of the data structures. 78

8.1 Displaying the getter methods for particle properties provided by `pythia8.Particle`. 96
8.2 Attributes exposed by `PythiaEvent` to access data for all particles generated in an event at once, exposed as `ndarrays`. 99

10.1 Hyperparameters configuring our Higgs reconstruction GNN. Hyperparameters marked with * indicate these were tuned using random search. 124
10.2 Model performance metrics, evaluated over the test dataset after training has completed in the final epoch. 125

11.1 Results table showing GNN performance metrics of the cluster reconstruction of the top quark from detector-level data. 132
11.2 Hyperparameters configuring our two sifts of the top reconstruction GNN. Hyperparameters marked with * indicate these were tuned using random search. Other hyperparameters were hand tuned to exploit different resource requirements. 136
11.3 Results of the model following double sift classification. 137

List of Acronyms

UE Underlying Event

ML Machine Learning

NN Neural Network

GAN Generative Adversarial Network

VAE Variational Autoencoder

GNN Graph Neural Network

GCN Graph Convolutional Network

IN Interaction Network

CNN Convolutional Neural Network

RNNs Recurrent Neural Networks

NLP Natural Language Processing

GN Graph Network

MLP Multilayer perceptron

MPNN Message Passing Neural Network

EWMPN Energy Weighted Message Passing Network

ReLU Rectified Linear Unit

PReLU Parametrised ReLU

AUC Area Under the Curve

HEP High Energy Physics

QCD Quantum Chromodynamics

QED Quantum Electrodynamics

IRC Infrared and Collinear

SM Standard Model

LHC Large Hadron Collider

CMS Compact Muon Solenoid

ECAL Electromagnetic Calorimeter

HCAL Hadronic Calorimeter

HB HCAL Subsystem Within the Barrel

HO Outer Hadronic Calorimeter

DTs Drift Tube Chambers

CSCs Cathode Strip Chambers

RPCs Resistive Plate Chambers

MIPs Minimum Ionising Particles

ISR Initial State Radiation

DGLAP Dokshitzer-Gribov-Lipatov-Altarelli-Parisi

MCEGs Monte-Carlo Event Generators

CA Cambridge / Aachen

GPS Global Positioning System

DAG Directed Acyclic Graph

COO Coordinate List Representation

kNN k-Nearest Neighbours

JIT Just-in-Time

AOT Ahead-of-Time

LLVM Low Level Virtual Machine

IR Intermediate Representation

PDG Particle Data Group

OOP Object Oriented Programming

FP Function Programming

ABC Abstract Base Class

API Application Programming Interface

LHE Les Houches Event

BFS Breadth-First Search

PV Photovoltaic

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as:

Signed:..... Date:.....

Acknowledgements

Researching at the intersection of two highly competitive fields was a daunting task. I'd like to thank the wonderful people who supported me through the highs and lows of this project.

First and foremost, to my Autism Study Skills Mentor, Jamie Williams. Jamie has been providing me with support for 10 years on the dot at the time of publishing this thesis. During my journey from an undergraduate to a doctoral candidate, he has been incredibly generous with his time and support. His generosity, encouragement, and wise advice has provided me the insight and stability I've needed to achieve all that I have this past decade.

Next, to my best friends Mike Stuart, Ewan Richardson, and Kate Griffin, who were always willing listen to me enthusiastically share my research ideas, despite working in completely different fields, and supporting me through the smooth and rough times.

Jessie Taylor, who was always up for a game of virtual Crazy Golf in the evenings after work when things were getting intense, thank you.

Within my office, the other PhD candidates in the Southampton SHEP group have been tremendous sources of stimulating conversations and support. My desk is littered with random nick nacks, including rubber ducks, slide whistles, and pirate dolls made of string, given as unexpected gifts which reminded me that I was part of a community. Mauricio Diaz, Giorgio Cerro, Dalius Stulga, Rajnandini Mukherjee, Ben Kitching-Morley, Ahmed Elgaziari, Joe Mckeon: thank you for wasting my time in the most restorative ways, so that I was always smiling when I went back to work. Particular thanks to Giorgio and Mauricio for their stimulating conversations and support of my specific research objectives. Working alongside them has been a great pleasure. Also, to Henry Day-Hall, for your generous support as I began my work, thank you!

To my supervisors, Stefano Moretti and Srinandan Dasmahapatra, thank you for taking a chance on me, and giving me creative freedom. I am proud of what we have achieved, and grateful for the opportunity to have done it.

Finally, to my Godmother Judith. Thank you for being such an inspiration to pursue my curiosity in Science and Nature.

To Bo, a very good dog.

Part I

Theory and methodology

Chapter 1

Introduction

The pursuit of understanding the fundamental structure of the universe lies at the heart of particle physics, where experiments and theoretical frameworks converge to probe the most basic constituents of matter and the forces governing their interactions. Over the last century, this endeavour has progressed through a succession of groundbreaking discoveries, from the identification of the electron to the monumental detection of the Higgs boson in 2012 at CERN's Large Hadron Collider (LHC). These milestones have culminated in the development of the Standard Model (SM), a quantum field theory encapsulating our current best understanding of electromagnetic, weak, and strong interactions.

Despite its extraordinary predictive power, the SM has paradoxically created challenges for further progress in fundamental physics. Its unmatched accuracy in describing known phenomena has left a scarcity of anomalous experimental data to guide theoretical breakthroughs. This lack of clear deviations from the SM limits the ability to identify new physics directions, creating a sense of stagnation in exploratory frameworks. However, this challenge opens the door to novel data-centric methodologies to uncover subtle patterns and extract deeper insights, thereby reinvigorating the search for transformative discoveries in particle physics.

At the experimental frontier, high-energy collisions at the LHC serve as a test-bed for generating and studying particles under extreme conditions. However, the immense complexity of the collision environment, characterised by overlapping interactions, soft emissions, and the non-perturbative nature of hadronisation, presents significant challenges. Boosted objects – highly energetic particles whose decay products are collimated into dense sprays of hadrons called jets – represent one such challenge. Accurately reconstructing these objects is critical for probing both SM processes and potential new physics, such as heavy resonances or exotic particles decaying in boosted configurations.

The advent of machine learning (ML) has transformed the landscape of data analysis in high-energy physics (HEP). Graph Neural Networks (GNNs), in particular, offer a promising paradigm for leveraging the inherent relational and geometric structure of particle interactions. By representing particle momenta and detector outputs as graph-structured data, GNNs enable the formulation of reconstruction tasks in ways that preserve physical symmetries, such as permutation invariance and collinear safety. This work explores the potential of GNNs for reconstructing boosted objects, advancing both the theoretical understanding of these methods and their practical application to collider data. The research is inherently exploratory, aiming to investigate novel approaches and present proofs of concept rather than fully polished solutions or established methodologies. By stepping away from well-trodden paths, this thesis seeks to open new avenues for understanding and leveraging simulation-based data in high-energy physics.

This thesis addresses three interconnected objectives: first, to establish a comprehensive software ecosystem for analysing simulated particle physics data, streamlining data manipulation and analysis tasks; second, to define a novel form of clustering that supersedes traditional jet definitions by leveraging simulation-based information, including particle ancestry and momentum, addressing the specific challenges posed by colour flow and its impact on defining ancestry; and third, to apply GNNs to this task to achieve detector-level reconstruction of boosted particle constituents, culminating in a novel approach to top quark reconstruction, which has been historically underexplored due to the complexities of colour flow in simulation data.

The thesis is structured as follows. Chapter 2 explores the design and operation of particle detectors, focusing on the types of data they collect and their relevance to collider experiments. Chapter 3 delves into quantum chromodynamics (QCD) and parton showers, describing how jets are formed through divergences, hadronisation, and related phenomena. Chapter 4 discusses jet physics, detailing jet definitions, clustering algorithms, and jet grooming techniques. Chapter 5 provides an overview of machine learning in high-energy physics, introduces graph representations, presents a taxonomy of graph neural networks, and describes the Interaction Network and Energy Weighted Message Passing Network architectures. Chapter 6 outlines the creation of a software ecosystem for simulating and analysing particle physics data, emphasising the formation of heterogeneous data structures from simulation outputs. Chapter 7 introduces a novel clustering method that supersedes traditional jet definitions by incorporating simulation-based information, such as particle ancestry and momentum. Chapter 8 combines the Energy Weighted Message Passing Network with the Interaction Network and a novel activation layer to achieve superior Higgs boson reconstruction. Chapter 9 applies the developed model to top quark reconstruction, addressing challenges posed by colour flow and utilising the

innovative cluster double sifting technique to achieve state-of-the-art performance. Finally, Chapter 10 presents the conclusions, summarising the key findings of the thesis and discussing potential avenues for future research in high-energy physics and machine learning methodologies.

Chapter 2

Particle detectors

> We can almost see protons and electrons in a Wilson chamber; we can almost see mass being conserved. We do not actually see these things; but what we do see has a very close relation to them.
>
> – Eddington (1939)

> Who ordered that?
>
> – I. I. Rabi's reaction to the discovery of the muon (1936)

This chapter starts with an overview of the early techniques used to detect and measure the properties of particles. As we gain familiarity for the important features of particle physics data, we will set context for subsections, which explore the growth in scale and resolution detector technology. This will put us on firm ground to understand the physical processes and resulting simulated data which is at the heart of our study.

2.1 Cloud chambers

The crucial testing of the theoretical framework defined by the SM relies on particle detector experiments. Detectors began with humble setups. Cloud chambers were tabletop particle detectors, whose construction may be demonstrated within minutes by any secondary school lab technician[1]. The result is a super-saturated chamber of isopropyl alcohol, which forms clouds that ionise easily when particles pass through. This leads to trails of ions along the trajectories of these particles. Due to the

[1] Simply line the bottom of a fish tank with felt, or any absorbant material. Squirt in isopropyl alcohol until the material is moist. Place dry ice into a deep tray, and then cover with a metal lid. Finally, flip the fish tank and place it over the tray. Switch the lights off, shine a torch in the tank, and watch the streaks of subatomic particles.

electrostatic attraction between the ions and the polar alcohol molecules, the ions act as a nucleation site for condensation, making the trails visible to the naked eye. Early particle physicists would watch for these tracks, recording the trajectories to identify and study the particles observed.

Simple geometric analysis of these condensation tracks can yield surprisingly detailed information about the particles producing them. For instance, some tracks appear thinner than others. This represents particles with lower ionising powers, such as β particles; more highly ionising radiation, *eg.* α particles, result in much thicker tracks. Track length, too, offers insights into either the kinetic energy or the mass of particles. Generally, high energy particles produce longer tracks, whereas short tracks tend to be produced by particles with either low kinetic energy or high mass. The link between geometry and physical properties was pushed further by Skobelzyn (1927), who applied a magnetic field to deflect the particle trajectories.

Charged particles are deflected when they are in motion relative to magnetic fields, as described by the Lorentz equation:

$$\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}). \tag{2.1}$$

Using this law, and combining it with equations for helical motion and particle kinematics, it is possible to determine the polarity of the particle's charge, and further calculate the particle's momentum.

By applying a magnetic field (say, upwards), we can examine the curvature of the path to determine if the particle is positively (clockwise motion) or negatively (anticlockwise motion) charged. Determining momentum, energy, and mass involves more detailed cross-referencing between the radius of curvature, magnetic field strength, and the geometric analyses discussed above. The interested reader is directed to Gupta and Ghosh (1946) for both experimental and theoretical details.

This setup, in fact, provided the first experimental confirmation for the existence of antimatter, as predicted by Dirac (1928). Anderson (1933) observed a particle whose behaviour was in every detail identical to an electron, but with an opposite direction of curvature, implying a positive charge, see figure 2.1. Thus the positron was discovered.

Astonishingly, for a brief period of time in the 1930s, Anderson's experiments with cloud chambers resulted in the discovery of half of the known particles in existence at the time! Prior to Anderson's work, only electrons, protons, and neutrons were known. Following his discovery of the positron, he subsequently discovered the muon and anti-muon. Muons are a form of charged lepton, of which there are three *generations*: electrons, muons, and tauons, in order of increasing mass. Muon detection has developed significantly over the years, and we will pay particular attention to

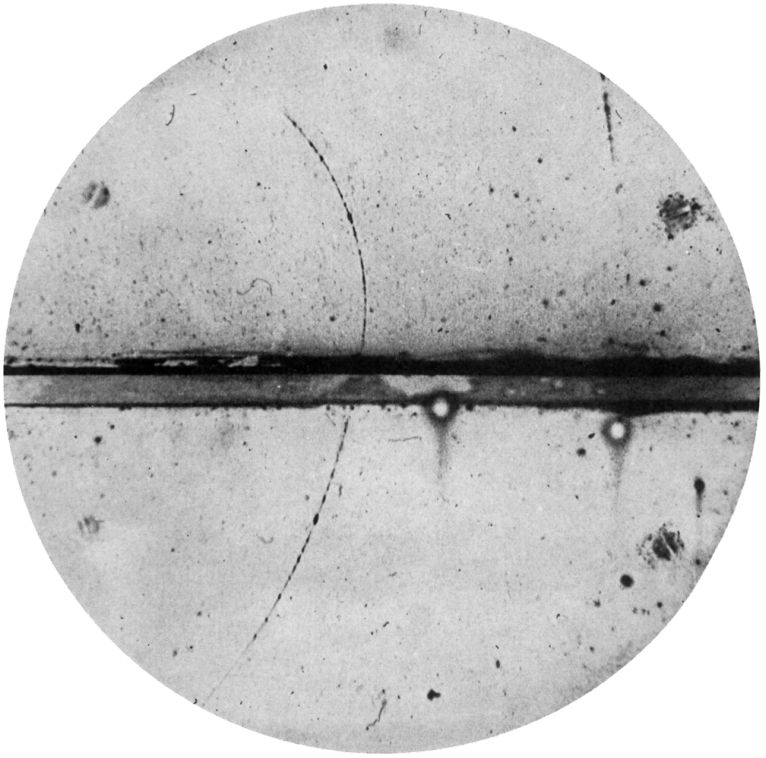


Figure 2.1: Photograph of the cloud chamber track observed by Anderson, proving the existence of the positron.

this. In fact, the detector we will be using for reference throughout this work, the Compact Muon Solenoid, as its name suggests has specialised components for direct muon detection.

2.2 Modern experiments at the LHC

Shrek: [Detectors] are like onions–
Donkey: They stink?
Shrek: No–
Donkey: They make you cry?
Shrek: No!
Donkey: Oh, you leave them out in the Sun, they get all brown, start sprouting little white hairs?
Shrek: No – *layers*! Onions have layers; [detectors] have layers… Onions have layers… You get it, [they] both have layers.

– Shrek (2001)

While exciting in their simplicity of construction, and clear correspondence of observations with underlying physics, cloud chambers are much too limited for modern studies. In the halcyon days of tabletop experiments, comparatively low energy particles were the primary focus of study. It is natural that particle physicists first structured their theories using low energy interactions. A great deal was not understood, and many particles appearing in early experiments were entirely unexpected. So numerous were these particles, that Oppenheimer is said to have described it as a "subnuclear zoo" in a 1956 public lecture at the Rochester VI conference (Johnson, 1999), and the term *particle zoo* stuck. Ultimately attempts to organise the particle zoo were successful. The solution was introduced independently by both Gell-Mann (1964) and Zweig (1964). The underlying structure is known as the Eightfold-Way, which describes the particle zoo as combinatoric expressions of ways to organise 6 fundamental particles (the quarks of the SM) in groups of 2 or 3 (mesons and baryons), forming a vast array of composite particles. As descriptions of the low energy domain became more detailed, it was necessary to probe higher energy scales in order to encounter new phenomena and verify theoretical ideas. A century on from these initial experiments, we have introduced colliders capable of not just observing, but creating particles at ever higher energy scales, and have densely packed the surrounding space with detectors whose sensitivity and throughput has exploded, to study their behaviours in incredible detail.

The LHC is the largest particle physics experiment in the world by several metrics, boasting a circular collider with a 27 km circumference. There are four large general purpose detectors attached to this loop. These are ATLAS, CMS, ALICE, and LHCb. They are referred to as "general purpose" because they are designed to study many physics processes, so enabling greater flexibility in experiment design in response to developments in theory. CMS is so named for its high number of detector components within a **Compact** volume, its ability to directly detect **Muon**s, and its

superconducting **Solenoid**al magnet, the largest ever manufactured at the hand-in date for this thesis. In this work, when connecting our approach and results to experimental details, we will use the operating parameters of CMS as a guide. To do this, we consider the types of data these detectors output, and how this reflects the theoretical understanding of the phenomena providing the input.

2.3 Producing tracks and towers

Modern detectors continue to study particles by tracing their paths in the presence of magnetic fields. Such analysis is referred to as *track finding and reconstruction*. The innermost region is wrapped in a cylindrical shell of *silicon pixel detector* components. In fact, several layers of silicon pixel and strip detectors build outward, so successive detections at higher radii form a trajectory, hence the name "track reconstruction". Additionally, components known as *calorimeters* measure the energy of particles by stopping them. Stopping high energy particles causes them to shower, and the cascade of captured particle energies is known as a *tower*. So, we expand our vocabulary to include towers, as well as tracks, to represent detector data.

The tracks and towers are collected by a series of detector subsystems. As eluded to above, these subsystems form cylindrical layered shells, built outwards around a central beamline. The central beamline is a highly focused circular path, created with strong magnetic fields, along which the high velocity charged particles are restricted to move, via equation 2.1. At the LHC, the particles accelerated are protons, one of the two baryons forming nuclear matter, containing quarks uud. These protons are obtained by ionising monatomic hydrogen gas. They are then accelerated in dense clusters, known as bunches, each of which containing 100 billion protons. These bunches are spaced 7 m apart, such that successive bunches pass an observer in the detector's reference frame every 25 ns.

However, merely accelerating particles offers us very little. Instead, bunches are crossed with each other, by accelerating them in opposite directions around the LHC beamline. Thus, every 25 ns, two groups of 100 billion high speed protons come into range and facilitate approximately 20 head-on collisions, with an aim to produce high mass particles in the interactions. Note that – while this may not seem like many collisions per crossing – one key benefit of a circular collider *vs.* a linear collider is that the accelerated particles which do not collide have additional opportunities to do so on subsequent revolutions. The centre-of-mass collision energy between any two protons is 13 TeV which, for context, produces similar conditions to that of the Universe one picosecond after the Big Bang[2].

[2]Or, more relatably, the kinetic energy equivalent of two mosquitos flying head-on at each other. This may seem underwhelming, but the fact this energy is concentrated on *two subatomic particles*, rather than macroscopic animals, yields more extreme results.

The detection and analytical work begins here, to determine what interactions occurred, and crucially, if any particles we wish to study were created.

2.3.1 Temporal resolution and pileup

It is worth noting that the ~ 20 overlapping proton-proton collisions will need to be individually resolved. This puts a demanding requirement to maximise the temporal resolution and minimise the dead time of detectors. Dead time refers to the minimum time separation required between two detectable hits on the same detector channel. It is the period during which a detector is unable to register a new event after detecting a previous one (Particle Data Group et al., 2020). Table 2.1 displays modern timing performances of detector components.

We can tell from these data that it is generally not possible with current technologies to resolve 20 collision events per 25 ns, leading to multiple proton collisions being recorded as the same event. This is a form of contamination called *pileup*, and strategies for mitigation are included in chapter 4.3. Pileup mitigation in the data analysis phase is an active area of research, and obtaining effective strategies will prove even more important once the High-Luminosity upgrade of the LHC is completed, as this is expected to result in an order of magnitude increase in the number of concurrent collisions (Cassese, 2022). However, this is not the research focus of this thesis.

Table 2.1: Typical temporal resolutions and deadtimes of common charged particle detectors (Particle Data Group et al., 2020).

Detector Type	**Time Resolution**	**Dead Time**
Resistive plate chamber	1 ns	—
Streamer chamber	2 µs	100 ms
Liquid argon drift [7]	~200 ns	~2 µs
Scintillation tracker	100 ps/n	10 ns
Bubble chamber	1 ms	50 ms
Proportional chamber	2 ns	20–200 ns
Drift chamber	2 ns	20–100 ns
Micro-pattern gas detect.	<10 ns	10–100 ns
Silicon strip	few ns	~50 ns
Silicon pixel	few ns	~50 ns

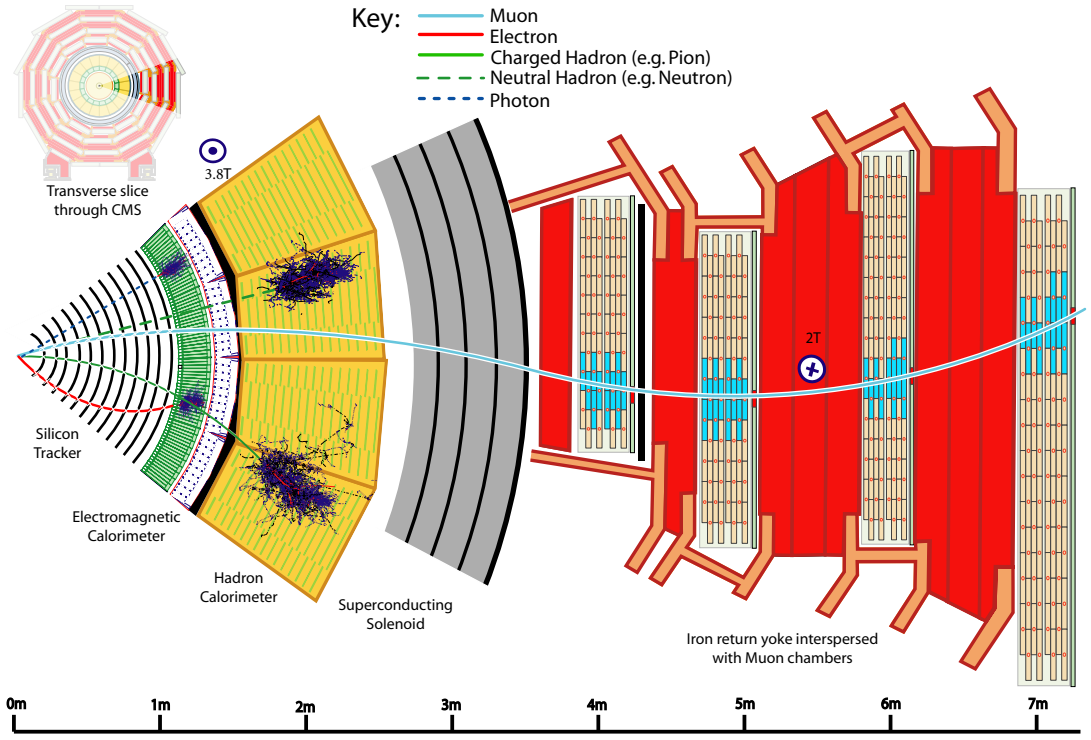


Figure 2.2: Showing the detector subsystems of CMS building outwards as the radius increases (Sirunyan et al. (2017)).

2.4 Detector subsystems

We now take a moment to break down the structure of the CMS detector, building up from the innermost subsystems outwards in radius. Figure 2.2 shows this structure. We will start from the silicon tracker, pass through the electromagnetic and hadronic calorimeters, discuss the solenoidal magnet, and finally the Muon chambers contained within the iron yolk. This is a far cry from the tabletop experiments of cloud chambers, stretching from a radius of 3.3 cm to over 7 m.

2.4.1 The inner tracking system

At a radius of 3.3 cm, the innermost shells are formed of silicon[3] pixel detectors (as mentioned above), building out in 3 layers to a 10.2 cm radius. These are the highest resolution detector subsystems in CMS, with an area of $100 \times 150\ \mu\text{m}^2$ per pixel (The CMS Collaboration et al., 2008).

[3]Other semiconductor detectors exist. Silicon may be fabricated into $\sim 100\ \mu$m thick layers, so the temporal response is competitive, as well as the spatial resolution. However, germanium is another popular choice (Particle Data Group et al., 2020).

Techniques such as charge sharing may boost this resolution yet higher (Kotliński, 2001; Boronat et al., 2015). It is essential that the inner subsystems are high resolution, as the particle flux is high at these radii, with ~ 1 million particles flowing per square millimetre each second. As the particle flux tapers off due to the increasing radius, 10 layers of *silicon microstrip detectors* are used, since the lower resolution of detector strips is sufficient to reconstruct the less densely packed particle trajectories. The flux ranges from 60,000 to just 3,000 particles mm^{-2} s^{-1} between radii 22 cm - 1.15 m. This approach provides us with high resolution track data for charged particles, but we are unable to detect uncharged particles in this way.

2.4.2 The electromagnetic calorimeters

Electrically neutral particles present a challenge for detection. In order to be detected, they must be stopped by a nuclear collision. Naturally, such collisions result in a significant momentum transfer, and thus produce particle showers of their own. These cascades are modelled as cones emanating from the point of interaction. It is important to wait until the charged particles have been captured before triggering the cascades from these neutral particles; detecting neutral particles prior to charged particles would lead to even higher fluxes on the already heavily burdened silicon pixel and strip subsystems. Since these detectors capture the towers of energy produced by the original particle, they are known as calorimeters. When calorimeters produce cones with a limited lateral spread of this energy, we describe the showers as having a smaller *Molière radius*, which results in higher resolution particle detection. The Molière radius is defined as the radius of a cylinder centred along the direction of a particle, such that 90% of its energy is deposited within the surrounding material (Molière, 1948). The Molière radius for a particle detection depends on the type of particle being detected, and the material from which the calorimeter is fabricated. CMS has two calorimeter subsystems for which the material choice is matched to two respective classes of particles.

Photons are captured first in an electromagnetic calorimeter (ECAL). This subsystem measures the energies of particles via the electromagnetic interaction, primarily via photons and electrons / positrons. Lead tungstate ($PbWO_4$) crystals are used in CMS's ECAL subsystem, selected for its high electromagnetic cross section, and small Molière radius of 21.9 mm for photons (ECA, 1997). This leads to a transverse granularity of $\Delta\eta \times \Delta\phi = 0.0175 \times 0.0175$ (or 22×22 mm^2), see chapter 2.5. Additionally, its low density of 8.28 g cm^{-3} results in a low hadronic cross section. This means that the effective cross-sectional area between incoming photons and the nuclei within $PbWO_4$ is high, leading to more frequent interactions, and the converse is true for incident hadrons. The 67.4 tonne collection of 61,200 crystals are packed tight, to enhance accuracy of the missing energy measurement from the detected

interactions. This effectively lets hadrons past, which are subsequently captured by the hadronic calorimeter (HCAL) subsystem.

2.4.3 The hadronic calorimeters

HCAL is formed of 16 alternating layers of brass (10% Zn, 90% Cu) absorber and plastic scintillator tiles. Plastic scintillator tiles contain wavelength-shifting (WLS) fibres, which capture the scintillating photons produced by particle interactions, and pass them on to photodetectors. The materials from which this subsystem is constructed are selected for several desirable properties.

Copper boasts a short interaction length λ_{int}, meaning the HCAL subsystem within the barrel (HB) can trigger many showers even with limited thickness. This is blended with Zinc to form a brass alloy, which improves the machinability of the material. From a structural perspective, both brass and plastic also offer support to the detector subsystem.

The non-magnetic nature of these materials is crucial, for two reasons (HCA, 1997). On one hand, it prevents the detectors from interfering with the uniform central field in CMS. On the other, it prevents the subsystem from experiencing magnetic forces which could put it under stress. This is crucial in this region of the detector, as the strength of the magnetic field in this region is extremely powerful, at 4 T.

2.4.4 The solenoidal electromagnet

This magnetic field is produced by the subsequent layer of the CMS apparatus: the solenoidal electromagnet. With a length of 12 m, and inner diameter of 6 m, the 12 tonne solenoid is the largest ever constructed. By enclosing the inner tracking system, the precision of measurements, such as particle impact parameters and secondary vertex locations, are enhanced. This is due to its strong bending power of 12 Tm, which is necessary to deflect the high inertia particles travelling near light speed, providing kinematic information via equation 2.1.

Additionally, despite the HB plastic scintillator's non-magnetic fabrication, the magnetic field does affect HB's detector response. This leads to an intrinsic brightening of the scintillator response of $\sim 5 - 8\%$. This occurs due to the magnetic field stimulating polymer excitation in the plastic, which increases the energy of ultraviolet light emitted. The brightness is further improved due to the strong bending power of the magnetic field, since low energy electrons are deflected, producing longer path-lengths over which to interact through the subsystem.

The particles which continue to propagate beyond the inner tracking and calorimetry subsystems pass through this solenoid, so their energy is not captured by barrel detectors. Passage through the solenoid causes it to act as an additional absorber, adding $1.4\lambda_{\text{int}}/\sin\theta$ interaction lengths, where θ is the angle of inclination of incident particles, in spherical polar coordinates (HCA, 1997; Abramov et al., 2001; The CMS Collaboration et al., 2008). We make use of this effect in the penultimate detector subsystem.

2.4.5 The Outer Hadronic Calorimeter

The barrel HCAL (HB) is constrained in width by the available space between the outer radius of the ECAL (1.77 m) and the inner wall of the solenoid (2.95 m). While this compact design effectively captures hadron showers within its boundaries, it struggles to contain high-energy hadrons fully, particularly at pseudorapidity $\eta = 0$. As a result, some hadronic showers are only partially absorbed, leading to energy leakage that degrades the calorimeter's hermeticity. This energy loss is particularly detrimental to physics studies that rely on precise missing transverse energy (E_T^{miss}) measurements, such as searches for new physics or the production of neutrinos, where accurately accounting for undetected energy is critical (Abdullin et al.).

To address this, the outer hadronic calorimeter (HO) subsystem was implemented (HCA, 1997; The CMS Collaboration et al., 2008; Abramov et al., 2001). Simulations have demonstrated that the addition of HO significantly reduces energy leakage, leading to improved energy resolution across a range of particle energies. Figure 2.3 illustrates this improvement (Abdullin et al.).

The HO leverages the solenoid and its cryostat as additional absorbers, which help capture late-developing showers. This is particularly important for higher-energy hadrons, as their interactions may start deeper within the detector volume. HO's design incorporates alternating active and absorber layers to facilitate the development and detection of these hadron showers, mirroring the layered structure used in HB.

In addition to hadron detection, the HO subsystem plays a key role in tagging muons. Minimum ionizing particles (MIPs), such as muons, can traverse the calorimeters with minimal energy loss and are efficiently detected by HO at a 90% efficiency rate, with noise levels below 20%. This high efficiency ensures reliable muon tagging, which complements the standalone muon system in CMS. While resistive plate chambers (RPCs) in the barrel muon system can trigger muon detection, their standalone efficiency is approximately 72%, making the contribution of HO critical for robust muon detection (Abdullin et al.).

Muons are a unique and vital particle for physics studies at CMS. Unlike electrons, which are prone to significant radiative losses as they traverse the detector material, muons are much less affected by such losses. This characteristic enables the reconstruction of muon 4-momentum with exceptional accuracy, particularly when combining information from the muon system and the inner tracker.

Muon detection is central to high-resolution measurements, such as reconstructing the Higgs boson's mass in decay channels involving muons. For example, the four-muon decay channel provides the best mass resolution among Higgs decay modes, making precise muon tracking essential for these studies. Combining the measurements from the inner tracker and muon system yields an order-of-magnitude improvement in momentum resolution at low momenta (The CMS Collaboration et al., 2008).

The CMS muon system consists of three primary detector components:

1. **Drift Tube Chambers (DTs)** in the barrel region, which provide precise position and timing information for muons at lower pseudorapidity values.
2. **Cathode Strip Chambers (CSCs)** in the endcaps, designed to handle the higher particle fluxes and radiation levels at high pseudorapidity.
3. **Resistive Plate Chambers (RPCs)**, which serve as a complementary system for fast triggering across both barrel and endcap regions.

Together, these systems allow CMS to achieve highly accurate muon identification and momentum reconstruction, earning the detector its namesake.

Figure 2.4 summarises our description, showing which particles are targeted by each subsystem.

2.5 Detector geometry

The LHC is a 27 km toroidal particle accelerator. We can consider CMS to be a linear section along its circumference, which forms the site of our simulated collision experiments. This forms a cylindrical coordinate system, see figure 2.6.

The azimuthal direction, ϕ, has periodic boundaries. The beam axis, or z-axis, is the axis of rotational symmetry of the cylinder, and the direction along which particle beams are accelerated. Motion perpendicular to this beam axis is said to be *transverse*. Events in which a high proportion of the collision energy is used in the hard momentum transfer produce particles which have high transverse momentum, p_T, thus evolving with trajectories directed at the central region of the detector. Transverse momentum is calculated as

$$p_T = \sqrt{p_x^2 + p_y^2}, \tag{2.2}$$

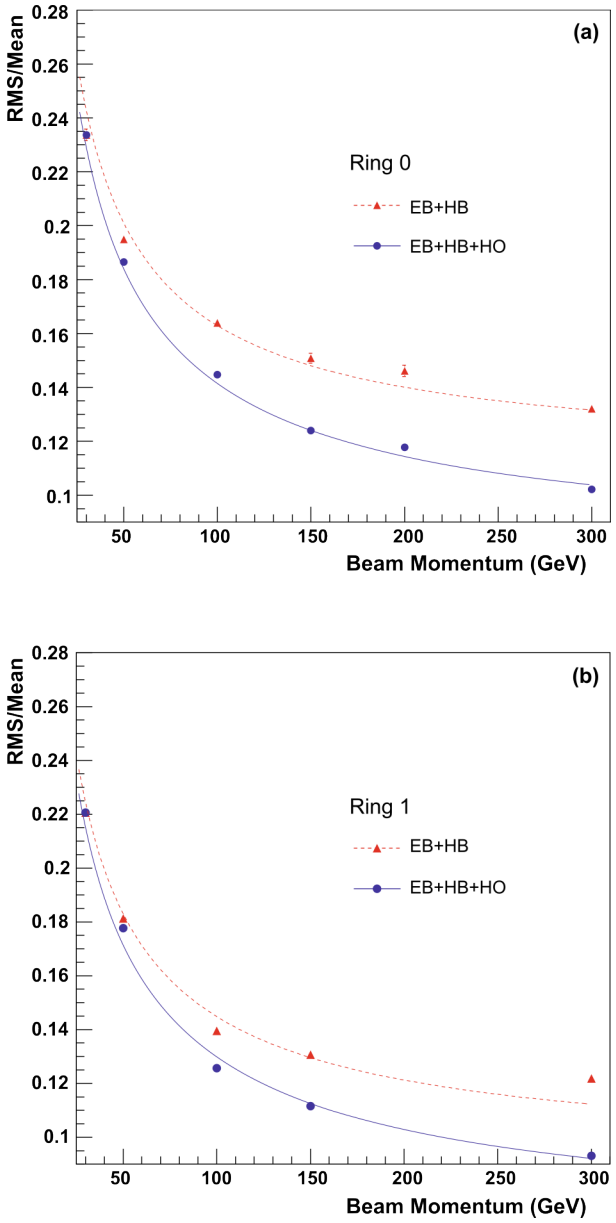


Figure 2.3: Energy resolution for pions as a function of beam energy measured with ECAL + HB, and with ECAL + HB + HO for the beam being shot at **(a)** $\eta = 0.22$ and **(b)** $\eta = 0.56$ (Abdullin et al.).

where p_x and p_y are the Euclidean components of momentum in the plane perpendicular to the beam axis, z.

As we shall see in chapter 3.1, when two composite particles collide, such as protons, the products of the interaction may be left with residual momentum. This can be quite sizeable, leading to distortions in the particle distributions due to space-time effects from special relativity. As a result, the obvious choice of coordinates to complete our cylindrical geometry for the collider, distance along the beam axis is not fit for purpose.

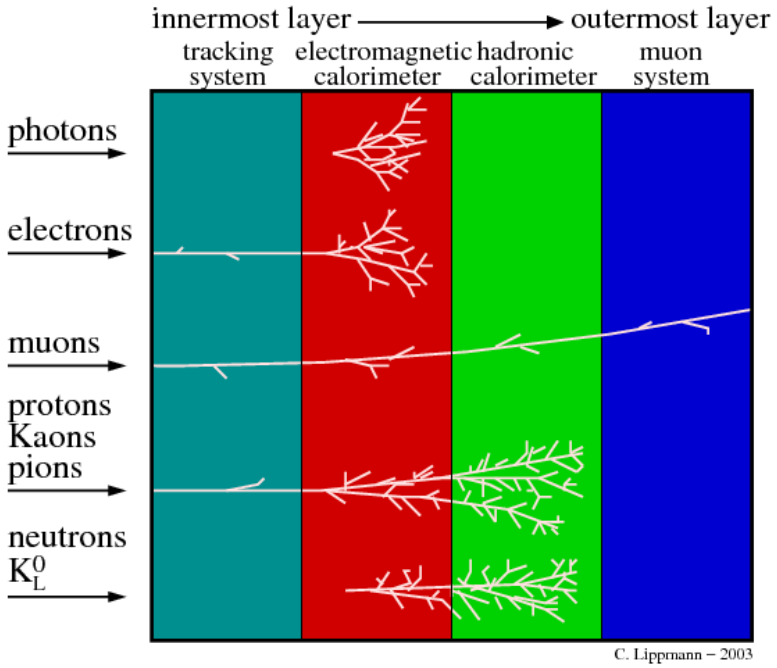


Figure 2.4: Describing how detectors develop tracks and towers for various kinds of particles, highlighting specifically which subsystems are active for which particles (Rizzi (2018)).

Rapidity is the hyperbolic angle which describes the separation of two inertial frames of reference:

$$y = \frac{1}{2} \log \left(\frac{E + p_z}{E - p_z} \right). \tag{2.3}$$

It can thus characterise the displacement from the central region of the detector, and has the desirable properties that it may be shifted with a simple addition, and the difference between the rapidity of two particles is Lorentz invariant to boosts along the beam axis.

However, pseudorapidity η is more often taken as the practical coordinate, rather than rapidity y. It's widely adopted in high-energy physics experiments due to its direct relationship with the polar angle (θ) of a particle's trajectory relative to the beam axis . Defined as

$$\eta = -\ln[\tan(\theta/2)], \tag{2.4}$$

pseudorapidity relies only on θ, which is straightforward to measure using the detector's geometry (Ellis et al., 1996). Pseudorapidity and rapidity for a given particle are the same in the limit $m \to 0$; this is a reasonable approximation as long as the particle has a high ratio of spatial momentum with rest energy. For massive particles or composite objects like jets, the difference between η and y can be non-negligible, making rapidity the more appropriate measure in such contexts (Ellis et al., 1989; Marzani et al., 2019). However, in the high-energy collisions we shall be

studying, where particles often have very large momenta relative to their masses, the approximation $E \approx p$ holds, and η closely matches y. This makes pseudorapidity a reliable and convenient coordinate for most experimental analyses, allowing for efficient descriptions of particle trajectories while retaining reasonable accuracy in the high-energy limit. Therefore, from here on, we shall take η as our longitudinal detector coordinate.

The simplicity of psuedorapidity's definition in terms of θ contrasts with rapidity y, which requires knowledge of a particle's energy E and longitudinal momentum p_z for its calculation. By using η, experimentalists can describe particle trajectories without needing to measure energy, which may be challenging for certain particles like neutral hadrons. This makes pseudorapidity particularly convenient for visualising particle distributions and analysing detector data.

Detector geometries are often designed to have uniform coverage in pseudorapidity, further motivating its use. Cylindrical detectors like those in the CMS and ATLAS experiments segment their components, such as tracking systems and calorimeters, based on $|\eta|$ (Huth et al., 1990). For instance, the tracker typically covers $|\eta| < 2.5$, while calorimeters extend to $|\eta| < 5$ (Bayatian et al., 2006). This segmentation ensures that detector acceptance is naturally expressed in terms of pseudorapidity, simplifying the analysis and comparison of data across different regions. Additionally, pseudorapidity's direct relationship to θ allows experimentalists to map the spatial coverage of the detector straightforwardly, enhancing the practicality of experimental setups.

We can view the final state particles as scattered across a pseudorapidity-azimuth ($\eta - \phi$) plane (Tkachov, 2002), shown in figure 2.5. As mentioned in chapter 3.1, in this study we are interested in processes which have high p_T, and thus are mostly concentrated at the central region of the detector barrel. For these purposes, it is common to remove radiation that would not be picked up by these detectors, by applying *cuts* to the data, usually $|y| < 2.5$ and $p_T > 0.5$. For experiments which are concerned with detecting high rapidity emissions, hemispherical *end caps* are filled with detectors at either end of the barrel, but these are not considered here.

The Euclidean distance between particles within this plane is invariant to boosts,

$$\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}. \tag{2.5}$$

ΔR is a quantity frequently used to cluster final state particles, and provide a geometric quantity to act as a window defining the nearby particles.[4]

[4]We will use this later to define locality in our graph data structures.

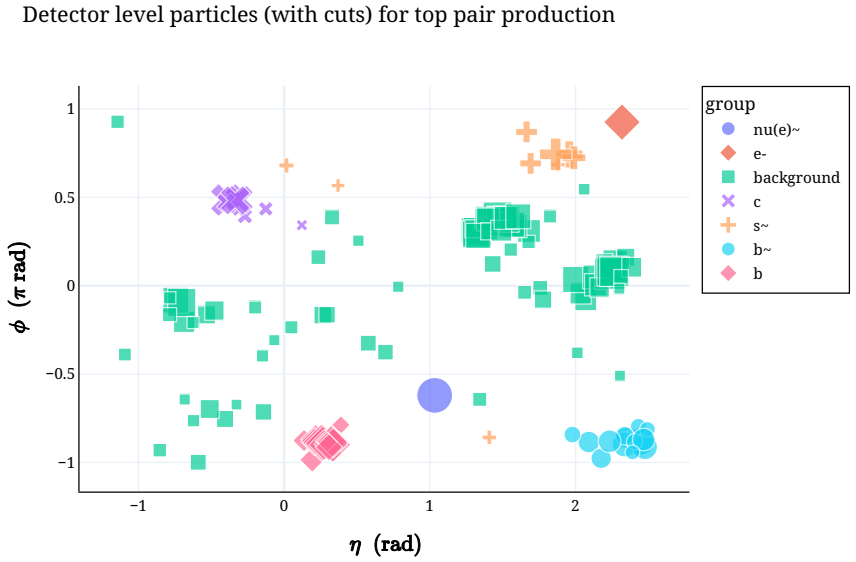
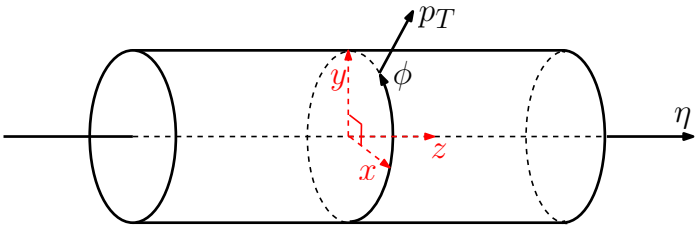
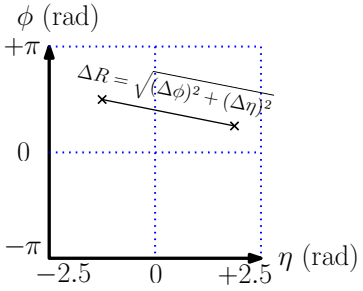


Figure 2.5: Final state particles for a $p\ p \to t\bar{t}$ process, in which t quarks decay into b quarks and $W^{\pm}$ bosons. For the t quark the hadronic decay $W^{+} \to c\ \bar{s}$ occurs, whereas the $\bar{t}$ results in the leptonic decay $W^{+} \to e^{-}\ \bar{\nu}_{e}$. Marker size indicates hardness, given as $\log(1 + p_T)$. The marker styles shows which final state parton from the hard process the displayed particles reproduce (background is UE). Details on how these clusters were formed is given in chapter 9. Cuts of $p_T > 0.5$ and $|\eta| < 2.5$ are applied to approximate tracking and calorimeter sensitivity in CMS's detector barrel.



(a) Detector barrel coordinates. Cartesian coordinates are shown in red, and the detector geometry coordinates we employ in collider physics are shown in black.



(b) Demonstrating the "unrolled" view of the $\eta - \phi$ plane. The angular displacement ΔR is defined as a Pythagorean distance measure.

Figure 2.6: Cylindrical geometry of the inner detector wall, displayed both as a cylindrical wireframe, and the unrolled form which we hereon refer to as the $\eta - \phi$ plane.

Chapter 3

QCD and parton showers

In the previous chapter, we were cavalier in the statement that particles "showered". Why should particles interact at all? How do interactions lead to the creation of new particles? Why are most of these particles unstable, decaying on their own via fundamental interactions spontaneously? We will address some of these questions at a high level throughout this chapter.

To do this, we will heuristically explore the quantum field formulation of the forces driving the phenomena we study. It must be stressed, though, that this work is primarily focused on the analysis of simulated data. While illuminating, the precise low level details of the physical interactions which produce this data were largely taken for granted, and did not substantially drive our program of research. With that disclaimer out of the way, let's dive in.

3.1 The parton model and proton collisions

As discussed in chapter 2, at the LHC proton-proton collision experiments are performed, with a centre-of-mass energy of 13 TeV[1]. We understand the structure of protons via the *parton model*. First proposed by Feynman in the 1960s, the parton model is essentially an approximation to a more fundamental theory where quarks and gluons are the quanta of a field theory of strong interactions, similar to how electrons and photons function in quantum electrodynamics (QED) (Close, 1979). A parton is a point-like constituent of a hadron, which we understand to be quarks and gluons.

During a high energy proton-proton collision, we can use this model to describe how the structure affects the momentum transfer. We assume that, at these energy scales,

[1] This is not the only kind of particle collision that occurs at the LHC. Heavy ion collision experiments are also conducted, but these are not relevant for our research.

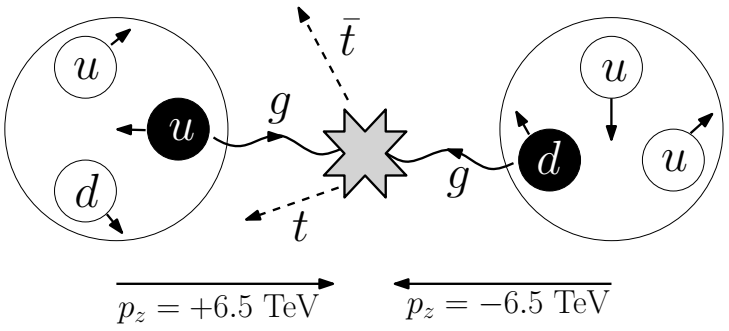


Figure 3.1: Depicting the interaction of two partons from colliding protons, with a centre-of-mass energy of 13 TeV. The energy transfer goes on to produce a pair of top quarks, which are longitudinally boosted in the centre-of-mass / lab frame.

the partons are effectively independent of one another. Therefore, the momenta of individual partons may vary wildly, despite the overall momentum of the proton being zero in its rest frame. This model suggests that the strongest interaction is likely to happen between just one *active* parton from each proton, giving rise to the highest momentum transfer. The energy of this collision may be transferred to produce a new particle-antiparticle pair, such as a $t\bar{t}$ pair, see figure 3.1.

However, as the interacting active partons may have any proportion of their respective proton's collision momentum, it is not possible to guarantee that the overall momentum of the hard (high momentum) collision will be zero. Therefore, it is very likely that the particles produced in the hard collision will have a significant relativistic boost of their momenta in the longitudinal / beam axis (typically taken as the z-axis). This is why the use of (pseudo)rapidity introduced in chapter 2.5 is important, since we need coordinates which are invariant under relativistic boosts.

3.1.1 Radiation from outside the hard momentum transfer

If we consider the momentum transfer between the active partons in the hard interaction, we notice that not all of the collision energy is accounted for. The momentum transferred during the hard collision is used to produce high energy particles, *eg.* $q\bar{q} \to t\bar{t}$. However, these have finite masses, which often fall far below the collision energy. The energy gap between the collision energy and the masses of the partons produced is filled by Bremsstrahlung radiation, emitted from the active partons, called initial state radiation (ISR) (Ellis and Soper, 1993). Due to the potentially wide gap between momentum transfer and collision energy, this ISR is not necessarily soft (Marzani et al., 2019).

Thinking beyond the hard momentum transfer also begs the question: what do the remaining partons in the protons do during the collision? These typically undergo much softer interactions with each other, resulting in low p_T emissions. Since

spectator partons are not considered part of the "initial state" of the hard interaction, this is called the Underlying Event (UE), rather than ISR. It is typically softer than the ISR, and while earlier analyses have treated it as uncorrelated with the hard process (Ellis and Soper, 1993), later studies explore the more nuanced connection between the two (Larkoski et al., 2020). Typically, we can expect a diffused radiation pattern over the detector, though in general its contribution can vary from highly diffused to point-like (Salam, 2010). In any case, the phase space taken up by the UE develops differently to that of the hard partons or the ISR.

Both the hard partons and ISR possess colour charge, and high energy, making hard parton emission inevitable. These produce cascades of splits and radiative decays, reducing the energy scales of parton interactions. Ultimately, the particles entering the detector have energies orders of magnitude lower than those in the hard process. Until the energies fall below a certain threshold (Λ_{QCD}), this can be described using perturbative QCD. After this point, different non-perturbative models must be used. As the energy scale of the UE is already low, it is entirely governed by non-perturbative QCD. We shall discuss the shift from the perturbative to non-perturbative regimes more in the next subsections.

It is our intention to study the properties of the particles produced in the hard process. We do so by analysing the final products of the particle cascades initiated from them. When clustered, these form a proxy of the particles we wish to study, which we call *jets*.

3.2 Introducing jets

Jets are sprays of collimated hadrons originating from the fragmentation of partons. These arise due to QCD colour confinement, which prevents quarks and gluons from existing as free particles. The process of jet formation encompasses multiple stages, including parton showering, hadronisation, and jet clustering. We shall present showering and hadronisation in this section, but dedicate chapter 4 in its entirety for jet clustering, in which we will address the need to mitigate the effects of ISR and the UE.

3.2.1 Parton Showering

The evolution of a parton begins with parton showering: a series of small-angle splittings governed by QCD dynamics. The probability of a parton emitting a gluon or splitting into other partons is described by the Dokshitzer-Gribov-Lipatov-Altarelli-Parisi (DGLAP) equations (Larkoski et al.; Ellis

et al., 1996). These equations incorporate splitting functions, such as $P_{qq}(z)$, $P_{qg}(z)$, $P_{gq}(z)$, and $P_{gg}(z)$, which encode the likelihood of specific branching events between quarks and gluons. The theoretical details of the DGLAP equations are beyond the scope of this study, however we can gain the key insights which will form the QCD backdrop of our study from equation 3.1 and equation 3.2.

The probability of a parton (e.g., X) emitting a gluon or splitting into another parton is given by:

$$P(X \to Xg) \sim \alpha_s \int \frac{dE}{E} \frac{d\theta}{\theta}, \tag{3.1}$$

where α_s is the QCD coupling constant, E is the energy of the emitted parton, and θ is the angle of emission. Notice two features of this relationship (Salam, 2010; Marzani et al., 2019):

1. The divergence at small θ shows that small-angle emissions dominate, leading to collimated sprays of hard particles.
2. The divergence at low E results in low energy emissions dominating, leading to broadening and wide angle emissions of soft particles.

The first of these observations signals the need for collinear safety. That is, observables used to reconstruct properties of X must be invariant if a particle is replaced by two particles whose momenta have the same / very similar direction, and the sum of which is equal to the original. The second observation gives rise to the need for infrared safety. In other words, observables must not be affected if the final state constituents have many near-zero energy particles superimposed in random locations. At the perturbative level, this is achieved through cancellations between real and virtual corrections. Infrared and collinear (IRC) safety will be explored further in chapter 4.2.

3.2.2 Hadronisation

As the energy scale approaches a fixed limit, which we call Λ_{QCD}, perturbative QCD breaks down. Partons then transform into colour-neutral hadrons in a non-perturbative process known as hadronisation. While theoretical frameworks like Quantum Chromodynamics (QCD) describe parton dynamics at high energies, the transition to hadrons necessitates phenomenological models due to the breakdown of perturbative techniques at low energy scales. In this section, we take a high level overview of the theoretical underpinnings, phenomenological models, and practical implications of hadronisation, with an emphasis on its relevance to jet analysis.

In QCD, the strong coupling constant α_s gains a dependence on the energy scale of interactions during renormalisation. This is described with the Renormalisation

Group (RG) equation

$$\alpha_s \to \alpha_s(\mu) = \frac{1}{\beta_0 \ln\left(\mu^2/\Lambda^2_{\text{QCD}}\right)}, \tag{3.2}$$

where β_0 is a constant term which depends on the number of colours (QCD charges) and active quark flavours in the interaction. Taking limits to two extremes of this equation tells us two important details about QCD for our work:

1. At the limit of infinite energy, $\mu \to \infty$, the strong coupling vanishes, $\alpha_s \to 0$
2. At the lower limit of energy, $\mu \to \Lambda_{\text{QCD}}$, the strong coupling diverges, $\alpha_s \to \infty$

How is this relevant for our showers? Well, in the high energy limit, we are allowed to consider colour-charged partons as moving effectively freely, experiencing no confinement due to the small coupling strength. However, as the energy decreases, the binding potential becomes insurmountably large. Consider bound $q\bar{q}$ states, which are called mesons. Below Λ_{QCD}, one would have to invest so much energy into separating the two bound quarks, that prior to breaking their bond, another $q\bar{q}$ would be created in a pair production. These would pair off with any bare quarks before they are able to be observed. Colour confinement is hence unavoidable at low energy scales.

As such, we must have a description of how the colour reconnects when we dip below the Λ_{QCD} threshold. Since the strong coupling constant diverges at these scales, it is not possible to use perturbative techniques to calculate the behaviour of colour. So, it is with a heavy heart that we must abandon QCD in favour of phenomenological models: physics-motivated descriptions, based more on heuristic ideas of the physics than rigorous theory.

Physicists employ phenomenological models implemented in Monte-Carlo event generators (MCEGs) to simulate the complex transition between QCD and phenomenology. These models play a crucial role in connecting theoretical predictions at the parton level to experimental observables based on hadrons. In our work, we rely on Pythia to perform our showering and hadronisation. The sole method of hadronisation used in the current version (v8.3) is the Lund String model, and is therefore the most important model for our purposes (Sjöstrand et al., 2015).

In the string model, partons are connected by a string-like potential, which fragments into hadrons as the energy increases. This is most easily described for e^+e^- annihilation (Ellis et al., 1996). The colour field between the produced quark and antiquark collapses into a string-like configuration as they move in opposite directions. The string has uniform energy per unit length, corresponding to a linear quark confining potential. The string breaks up into hadron-sized pieces through spontaneous $q\bar{q}$ pair production in its colour field.

If a gluon splits perturbatively into a $q\bar{q}$ pair during parton shower evolution, an additional string segment is produced. Gluons that remain at the end of the shower lead to kinks in the string segment that connect them. Each string segment then breaks up into hadrons. The string model has undergone significant refinements to form the basis of the JETSET simulation program, which gives a good description of hadronic final states in e^+e^- annihilation (Ellis et al., 1996).

While the string model is the relevant hadronisation technique for our work, we note that other models exist. Herwig (Bähr et al., 2008; Bewick et al., 2024) and Sherpa make use of the cluster model, whereby colour-connected partons form low-mass clusters, which decay into hadrons (Höche, 2015). However, as we merely wish to give a high level overview, we shall leave the details of these techniques to be explored by the interested reader.

We now have a working model of the most relevant theory and approaches which power our particle collision simulations to our work. MCEGs are extremely sophisticated programs, developed over decades with extensive testing and improvements. In this work, we respect that these models have been well verified against experimental data, and concern ourselves primarily with the data record produced by the combined simulations of MadGraph5 (Alwall et al., 2011) and Pythia (Bierlich et al., 2022). In chapter 7.2, we make significant contributions to the creation of data structures to manipulate this record. The ideas of showering and hadronisation are explored further in chapter 8.3, where the process is represented topologically using directed acyclic graphs.

Chapter 4

Jet Physics

4.1 Jet clustering algorithms

As introduced in chapter 3.2, jets are collimated sprays of hadrons, formed due to the process of parton showering and hadronisation. A precise and systematic approach is necessary to map the observed hadronic sprays back to the original hard interactions. This is achieved using jet clustering algorithms. These algorithms group particles based on kinematic and geometric criteria, ultimately providing insight into the nature of the underlying physics processes, such as the production and decay of a Higgs boson into a $b\bar{b}$ pair.

All algorithms we shall present here are IRC safe. We shall explore specifically what this means for jets, and how it applies in a detector context in chapter 4.2.

4.1.1 The Snowmass accord

The Snowmass accord (Huth et al., 1990), established in 1990, outlines several important criteria that a good jet clustering algorithm should satisfy. These criteria are aimed at ensuring that the algorithms are both practical for experimental analysis and theoretically sound for comparison with calculations (Salam, 2010; Marzani et al., 2019). The key criteria are:

1. **Simple implementation in experimental analysis:**
   The algorithm should be easy to implement in experimental settings. This means the algorithm should not be too computationally intensive and should be applicable to the type of data collected in experiments, such as tracks and calorimeter towers.

2. **Simple implementation in theoretical calculations:**
   The algorithm must also be straightforward to implement in theoretical calculations. This allows theorists to make predictions that can be directly compared with experimental results. The algorithm should be well defined mathematically to facilitate theoretical computations.
3. **Defined at any order of perturbation theory:**
   The algorithm should be well-defined at any order of perturbation theory. This ensures that calculations can be performed to any desired level of precision. This also relates to the concept of IRC safety, see chapter 4.2.
4. **Yields finite cross-sections at any order of perturbation theory:**
   The algorithm must yield finite cross-sections at any order of perturbation theory. This is also a key feature of an IRC safe algorithm. This is because, in quantum field theory, calculations can result in infinite results if not handled carefully. If the jet algorithm is IRC safe, those infinities are avoided or controlled, allowing for sensible comparisons with experimental data.
5. **Yields a cross section that is relatively insensitive to hadronisation:**
   The algorithm should yield a cross-section that is relatively insensitive to the hadronisation process. As discussed in chapter 3.2.2, hadronisation refers to the process where quarks and gluons turn into observable hadrons, and is a non-perturbative process that is difficult to model accurately. Therefore, jet algorithms should aim to minimise this sensitivity to allow more reliable comparison between parton-level calculations and hadron-level observations. This also means that observables built from jet quantities should be as little sensitive as possible to non-perturbative effects like hadronisation and the UE.

4.1.2 Sterman-Weinberg jets

The Sterman-Weinberg algorithm (Sterman and Weinberg, 1977), introduced in 1977, was among the first methods to define jet cross sections within perturbative QCD (Ellis et al., 1996). This algorithm was specifically designed for e^+e^- collisions and classified an event as containing two jets if at least a fraction $1 - \epsilon$ of the total event energy was captured within two cones with a half-opening angle δ (Banfi et al., 2006; Salam, 2010). For this reason, it is referred to as a "cone" algorithm.

The parameters δ and ϵ provide flexibility in distinguishing between two-jet and multi-jet events. To avoid incorrect classification, extreme values such as $\epsilon \to 0$ or $\epsilon \to 1$, and excessively small δ, are generally avoided. Furthermore, the specific values of δ and ϵ must be tuned according to the requirements of the physics analysis being performed. This use of angular and energy thresholds to define jets is a hallmark of cone-based algorithms. The Sterman-Weinberg method enabled a consistent

perturbative QCD calculation for determining the probability of observing two jets in an event (Salam, 2010).

Despite its historical significance, the Sterman-Weinberg jet definition has limitations in both theoretical and experimental analyses of multi-jet final states. One significant drawback is that cones with fixed half-angles δ do not efficiently tile the solid angle phase space. As a result, more advanced jet definitions have been developed to address these shortcomings (Ellis et al., 1996).

4.1.3 Sequential recombination algorithms

To overcome the limitations of cone-based approaches, sequential recombination algorithms were developed. These methods progressively combine particles into jets by calculating a pairwise distance measure d_{ij}, which is defined based on the particles' energies and their angular separation. The purpose of the clustering process is to simulate the behavior of QCD parton branching, grouping together particles that are likely to have originated from the same hard parton (Banfi et al., 2006; Moretti et al., 1998).

In these algorithms, the process begins by treating each particle as an independent object, often referred to as a "pseudojet". The algorithm calculates all pairwise distances d_{ij} between the pseudojets, as well as the beam distance d_{iB}, which quantifies the proximity of each pseudojet to the beam line. The next step is to identify the smallest distance among the computed values. If the smallest distance corresponds to a pairwise distance d_{ij}, the two associated pseudojets are merged into a single new pseudojet. On the other hand, if the smallest value corresponds to a beam distance d_{iB}, the associated pseudojet is promoted to a jet and is no longer considered in further clustering. This procedure is repeated, recalculating the distances at each step, until all particles have been grouped into jets (Chakraborty et al., 2022; Cacciari et al., 2008; Krohn et al., 2009).

The definition of the distance measure d_{ij} is a fundamental feature that distinguishes between different sequential recombination algorithms and determines their clustering behaviour (Banfi et al., 2006). Ideally, this distance measure must capture the divergences in QCD matrix elements, ensuring that particles are clustered together when they are soft or collinear, as such configurations correspond to singularities in QCD calculations.

At the same time, a well-designed distance measure must avoid introducing any artificial closeness due to momentum variations that do not correspond to physical divergences. This property ensures that particles are clustered only when they share a genuine physical relationship, reflecting the underlying parton dynamics (Banfi et al., 2006).

The jet clusters serving as a proxy for particle reconstructions are represented by binary trees. These trees are loosely considered to represent the possible history of decays and fragmentations from the high energy particles to the low energy constituents. They are formed by repeatedly merging "nearby" pairs of detector-level particles, until no more nearby particles are left, determined by the analyst of the event choosing a suitable distance threshold. In our work, we have access to more complete data via simulations resulting in directed acyclic graphs, rather than trees, see chapter 5.2 and chapter 8.3.

Prominent examples of sequential recombination algorithms include the JADE algorithm, the k_T algorithm, and its modern extensions, such as the Cambridge/Aachen and anti-k_T algorithms (Cerro et al., 2022; Salam, 2010). Each of these methods employs a unique distance measure, leading to distinct clustering patterns and characteristics in the resulting jets.

4.1.3.1 JADE algorithm

The JADE algorithm uses a distance measure, d_{ij}, to determine which particles should be clustered together into jets (Salam, 2010; Moretti et al., 1998). The distance between particles i and j is defined as

$$d_{ij} = \frac{2E_iE_j(1-\cos\theta_{ij})}{E_{\text{tot}}^2}, \tag{4.1}$$

where E_i and E_j are the energies of the particles, θ_{ij} is the angle between them, and E_{tot} is the total energy of the event. The algorithm iteratively combines the pair of particles with the smallest d_{ij} until all remaining pairs have a d_{ij} greater than some threshold value d_0.

This distance measure, d_{ij}, is directly related to the invariant mass of the particle pair, m_{ij} (Moretti et al., 1998). For massless particles, the invariant mass squared is given by $m_{ij}^2 = 2E_iE_j(1-\cos\theta_{ij})$ (Dokshitzer et al., 1997; Bartel et al.). Thus, the JADE algorithm's distance measure is essentially the squared invariant mass of the particle pair, normalised by the total energy of the event squared.

While the JADE algorithm was a widely used and effective approach to jet clustering for many years, the use of invariant mass in its distance measure can lead to limitations. Specifically, the JADE algorithm tends to cluster soft, widely separated particles together early in the clustering process (Moretti et al., 1998; Salam, 2010; Ellis et al., 1996). This is because the invariant mass of two soft particles moving in opposite directions can be smaller than the invariant mass of a soft particle and a nearby hard one, even if the soft particles are not physically related. This behaviour is

undesirable because it can result in the formation of spurious jets that do not correspond to a single hard scattering. These spurious jets, sometimes referred to as "phantom jets", do not align with the underlying physics of the event (Dokshitzer et al., 1997; Ellis et al., 1996).

JADE's preference for clustering soft particles also leads to larger hadronisation corrections, complicating the comparison of parton-level calculations with hadron-level observations (Moretti et al., 1998; Catani et al., 1991). This makes the JADE algorithm less robust in terms of accurately reflecting the underlying hard scattering process.

These issues with the JADE algorithm's distance measure led to the development of alternative algorithms like the k_T algorithm. We introduce the generalised form of this next, but specifically the use of relative transverse momentum instead of invariant mass, addresses some of the shortcomings of JADE. This is because this approach is more sensitive to collinear emissions than widely separated soft particles.

4.1.3.2 Generalised k_T

The generalised k_T algorithm refines JADE[1] by introducing a new form of distance measure with a tunable parameter p, offering greater flexibility and control over the clustering process (Krohn et al., 2009; Dokshitzer et al., 1997; Salam, 2010; Marzani et al., 2019):

$$d_{ij} = \min(p_{Ti}^{2p}, p_{Tj}^{2p})\Delta R_{ij}^2, \tag{4.2}$$

where ΔR_{ij}^2 follows the same definition as equation 2.5, measuring the angular separation in the rapidity-azimuth plane.

As in JADE, we define a characteristic distance such that, if d_{ij} exceeds it, we promote our pseudojet to become a jet and remove it from the set of constituents to cluster. We call this the beam distance d_{iB}, and it is defined as

$$d_{iB} = p_{Ti}^{2p} R^2, \tag{4.3}$$

where we introduce R as a tunable jet radius. This is a descriptive name, if a little misleading. While increasing the value of R does indeed lead to larger jets (and vice versa), it is not exactly equal to the circular radius of a jet. Indeed, jets are rarely conical / have a circular cross section on the detector wall. The shape of the jet is determined by the choice of tunable parameters.

[1] The precursor to generalised k_T was k_T, which was formulated with a similar distance measure similar to JADE's. This is reformulated in generalised k_T.

Setting hyperparameters R and p must be done with care to achieve the data analyst's goals. A choice of R corresponds to a bias in expectations for how collimated the sprays of particles forming the jets is. The selection of p has a more nuanced effect on the composition and geometry of the produced jets.

If passed as a positive number, typically $p \geq 1$, softer (smaller p_T) particles result in smaller d_{ij}, and thus are prioritised in the construction of jets, *ie.* they are clustered first. This makes jets very sensitive to soft radiation, and can result in significant pollution with low energy particles that did not originate from the particle we wish to reconstruct (Marzani et al., 2019). Algorithms using this value of p are referred to as k_T (Catani et al., 1991).

The Cambridge / Aachen (CA) algorithm (Dokshitzer et al., 1997) does away with this sensitivity by simply relying on detector coordinates and geometry, rather than transverse momentum . This is done by setting $p = 0$.

Anti-k_T instead prioritises the hard (high p_T) particles in reconstructing jets, by setting $p = -1$. This results in hard jet cores iteratively merging soft particles moving outwards, forming generally circular jets. This makes anti-k_T robust to pollution (Krohn et al., 2009). Additionally, the conical jet structure centred on hard particles, which are inherently easier to detect, improves the quality of calibration in detectors.

From a theoretical perspective, the binary tree produced by the k_T algorithm corresponds more closely to the expected ancestral structure of parton decays and fragmentations. This is due to the fact that k_T's distance measure mimics the enhanced frequency of soft emissions, predicted by QCD. That is, k_T's small distance measure between soft particles ought to reconstruct their parent partons more faithfully, with a history that looks similar to the parton showering process. However, the practical benefits in experimental contexts has led to anti-k_T becoming the standard method of clustering in collider experiments[2] (Cacciari et al., 2008; Gallicchio and Schwartz, 2010).

4.2 Infrared and collinear safety

IRC safety was introduced when we discussed parton showering in chapter 3.2.1. As it stems from the underlying QCD that governs parton evolution at the high energy scale, it is crucial for both theoretical predictions and experimental analyses. We have already discussed the divergences which arise in QCD when particles undergo collinear splitting or soft emissions. In this section we will explore the practical considerations we will need to make in analysing our jets.

[2]Many analyses will first cluster using anti-k_T to obtain convenient jet shapes and filter noise, then apply CA clustering to obtain a binary tree whose branchings more closely resemble the expectations of QCD.

To recap, IRC safety arises due to the diverging probability of the following two ways a parton can decay:

- collinear splitting: a single particle splits into multiple particles traveling in nearly identical directions.
- soft emission: A particle radiates low-energy particles in various directions.

When trying to form clusters at the detector-level to capture the phase space of hard partons we wish to reconstruct, this places the following conditions on our algorithms to ensure IRC safety. Any observable constructed from our clusters must remain unchanged if:

- a particle is replaced with a set of collinear particles
- particles with negligible energy are superimposed all over the detector wall

This can be formalised as follows with the following two equations. For collinear splitting:

$$V_{m+1}(\ldots, k_i, k_j, \ldots) \longrightarrow V_m(\ldots, k_i + k_j, \ldots) \quad \text{if } k_i \parallel k_j. \tag{4.4}$$

For soft emission:

$$V_{m+1}(\ldots, k_i, \ldots) \longrightarrow V_m(\ldots, k_{i-1}, k_{i+1}, \ldots) \quad \text{if } k_i \to 0. \tag{4.5}$$

Where V_m is an observable based on a cluster of m particles with momenta k_i. V_{m+1} corresponds to a particle splitting. In equation 4.4 a particle with momentum $k_i + k_j$ splits into two collinear particles, with momenta k_i and k_j respectively. In equation 4.5 a particle with vanishing momentum k_i is inserted into the cluster. In both cases, the observable V_{m+1} is required to tend to V_m.

In practice, the finite resolution of detectors limits the precision with which jet algorithms can resolve particle trajectories and energies. In the case of IRC safety, this can work in our favour, regularising our measurements. When particles are too soft or their angular separation falls below the detector's granularity, their signals are either merged or discarded. However, even if detectors may experience a mitigated effect of IRC splittings, the high probability of these processes requires that observables describing them are invariant to their effects to garner any respect in the HEP phenomenology community. If IRC safety can't be guaranteed, divergences may propagate through perturbative calculations, leading to potentially infinite cross sections. By enforcing IRC safety, we guarantee proper cancellation of divergences in perturbative calculations.

4.3 Jet grooming

So far we have discussed methods for forming clusters from particle momenta scattered over the $\eta - \phi$ plane. These methods locate jets as proxies for the particles in the hard process. But as we discussed in chapter 3.1.1, there are other sources of radiation entering the detectors than the final state products of the hard collision. These include ISR and the UE. Additionally, the phenomenon of pileup (PU) was introduced in chapter 2.3.1, and this contributes to a substantial increase in jet contamination.

In order to gain high resolution when reconstructing hard partons we wish to study, experimentalists employ strategies for subtracting contaminating radiation, which is broadly called *jet grooming* (Larkoski et al.; Marzani et al., 2019). We discuss a few specific techniques which come under this umbrella term for the remainder of the chapter.

4.3.1 Filtering

Jet filtering (Butterworth et al., 2008) is a jet grooming technique designed to reduce contamination from soft radiation by focusing on the hardest substructure of a jet. It achieves this by re-clustering the jet's constituents with a smaller radius, R_{filt}, and selecting only the n_{filt} subjets with the largest transverse momentum (p_T) (Salam, 2010; Marzani et al., 2019). This process helps isolate the relevant hard substructure of the jet, improving its suitability for further analysis.

The choice of R_{filt} is determined by the analysts. However, two prominent works identify a similar approach. Krohn et al. opted to use a radius of half the original jet, $R_{\text{filt}} = R_0/2$, and Salam (2010) refined this in a $H^0 \to b\bar{b}$ study, using the same measure, but applying a minimum radius of 0.3, *ie.* $R_{\text{filt}} = \min(0.3, R_{b\bar{b}}/2)$. Ultimately, the choice of R_{filt} is made based on the suitability for the specific study being carried out.

Once the re-clustering is complete, the algorithm identifies smaller subjets within the re-clustered jet. It then selects the n_{filt} subjets with the largest p_T. The parameter n_{filt} is typically set to match the expected number of hard prongs (distinct hard substructures) within the jet, plus one additional subjet to account for a hard gluon emission. The selected subjets are combined to form the filtered jet, which is then used for further analysis. By retaining only the hardest subjets, filtering effectively removes soft, wide-angle radiation that is unlikely to originate from the primary hard scattering process.

Filtering is particularly valuable in analyses where the number of hard prongs within a jet is known, such as searches for boosted heavy particles. For example $W^{\pm}$ / Z^0 /

H^0 bosons, which typically exhibit $n_{\text{prong}} = 2$, so n_{filt} is set to 3 to account for an additional hard gluon emission. It would also be useful for top quarks, with $n_{\text{prong}} = 3$, and again we set $n_{\text{filt}} = 4$ for the additional hard gluon.

4.3.2 Trimming

Trimming (Krohn et al.) is another jet grooming technique that addresses contamination, particularly for jets formed from light partons. Trimming is very similar in spirit to filtering; a key difference, however, is that while filtering retains a fixed number of the hardest subjets, trimming removes all subjets below a p_T threshold.

Trimming employs an "outside-in" algorithm, starting with pre-identified seed jets[3]. These jets are then re-clustered into smaller subjets using a jet-finding algorithm, commonly the k_T algorithm. The k_T algorithm is particularly useful because it effectively balances energy sharing between subjets. The re-clustering is performed with a smaller radius R_{sub} compared to the original jet radius, allowing finer resolution of the jet's substructure.

Once the subjets are identified, a "softness criterion" is applied to determine which subjets are retained. Specifically, subjets are kept only if their transverse momentum p_T exceeds a certain fraction, f_{cut}, of the hard scale of the event. The hard scale – often represented as the transverse momentum of the seed jet or the event's effective mass (Λ_{hard}) – provides a reference point for distinguishing soft and hard contributions. Subjets falling below this threshold are discarded. That is, for a subjet i to remain, it must satisfy:

$$p_{Ti} > f_{\text{cut}} \Lambda_{\text{hard}}. \tag{4.6}$$

Trimming is designed to clean up jets by removing soft radiation that arises from ISR, the UE, and PU while preserving the hard core of the jet. By removing soft subjets, trimming reduces the active area of the jet. This makes trimmed jets less sensitive to contamination from soft radiation. This improves the accuracy of jet reconstruction.

4.3.3 Pruning

Pruning (Ellis et al., 2009) is a bottom-up jet grooming technique designed to reconstruct boosted heavy particles by dynamically removing spurious mergings during jet clustering (Krohn et al.). Unlike trimming, which operates top-down,

[3]Krohn et al. states that any jet clustering algorithm may be used to produce these initial seed jets, regarding the choice of algorithm as largely irrelevant.

pruning iteratively applies constraints during clustering to remove soft or asymmetric recombinations.

Starting with a seed jet, pruning re-clusters using a jet algorithm (e.g., k_T or Cambridge/Aachen) with a larger radius (Marzani et al., 2019). At each step, it imposes two conditions for recombining constituents i and j:

1. **Dynamic Radius Constraint**: Constituents must satisfy

$$\Delta R_{ij} < R_{\text{prune}} = 2 f_{\text{prune}} \frac{m_{\text{jet}}}{p_{T,\text{jet}}}, \tag{4.7}$$

   where f_{prune} is a parameter.
2. **Momentum Symmetry**: Splittings must be sufficiently symmetric:

$$\min(p_{T,i}, p_{T,j}) \geq z_{\text{prune}} p_{T,(i+j)}. \tag{4.8}$$

If neither condition holds, the softer constituent is discarded. By dynamically adjusting the pruning radius R_{prune} based on jet kinematics, pruning enhances sensitivity to jet energy and structure compared to fixed-radius methods.

Pruned jets fall into two categories (Marzani et al., 2019):

- **I-Pruning**: A soft, large-angle emission dominates and is pruned away.
- **Y-Pruning**: A symmetric hard splitting ($1 \to 2$) defines the substructure, often preferred in theory due to analytical control.

Pruning excels at identifying hard substructures from heavy particle decays but struggles with light quark jets where scale separations are less distinct. It is also more sensitive to PU and the UE, which can distort the pruning radius, misclassifying symmetric (Y-pruning) jets as dominated by soft emissions (I-pruning).

4.3.4 How does this relate to our work?

In this work we explore techniques which don't formally require jet clustering definitions. However, considerations from jet clustering and grooming inform essential aspects of our GNN definition. When preparing training data, inspiration is drawn from the motivations of I-pruning, as clusters formed without jet definitions equally benefit from removing wide angle radiation. Additionally, by considering the decay products of the hard process by the prongs characterised by traditional jet analysis, our interpretation was greatly improved, which undoubtedly shaped our methods. Most importantly, by incorporating p_T and ΔR measures into our graph neural network algorithms, we are able to make it IRC safe, drawing on inspiration from traditional jet analysis.

Chapter 5

Machine learning on graphs

ML has transformed numerous aspects of collider physics, including jet physics, where its contributions are especially impactful. Aligning the intrinsic data representations in collider physics analyses with the input structures for specific ML models has offered state-of-the-art performance in several domains. In this thesis, we are interested in the applications of Graph Neural Networks (GNNs) for particle reconstruction.

We start this chapter with a biased and non-exhaustive tour of some ML applications to HEP, and consider their suitability based on the constraints they place on data representation. This is followed by an introduction to graph representations of data. Next, we discuss the Graph Network (GN) formalism, which provides a taxonomy for understanding the internal components of GNNs, and the levels of downstream prediction which they make possible. We expand upon this by discussing three general flavours of GNN, in order of increasing expressivity, before finally introducing specific architectures which influenced our work.

5.1 Choosing models for collider data

5.1.1 Multilayer perceptron

Here we introduce a neural network architecture called a multilayer perceptron (MLP). MLPs are themselves very basic, being one of the earliest approaches in ML (Rosenblatt, 1958). An MLP is a simple, biologically inspired model of cognition in the brain. The architecture is effectively a graph structure, composed of layers of "neurons", or nodes. Nodes in any given layer are not connected to each other; however, they are connected to every node in the previous and next layers.

The first layer is known as the "input layer", which takes a vector of input values, $\mathbf{h}^{(0)} = \mathbf{x}$. The nodes in the subsequent layer are then assigned with the sum value of all of the nodes from the previous layer, weighted by scalar parameters attributed to the edge from the source nodes. These edge values are stored for each pair of nodes in a weight matrix $\mathbf{W}^{(l)}$. This effectively corresponds to a matrix multiplication of $\mathbf{W}^{(l)}\mathbf{h}^{(l-1)}$, but this does not produce the node values in layer l, $\mathbf{h}^{(l)}$. That's because matrix multiplication is a linear transform, and applying repeated linear transformations over and over again is equivalent to applying one single linear transformation. That is, we wouldn't need a multilayered approach at all, because we could just condense all of the $\mathbf{W}^{(l)}$ into one resultant matrix, and optimising an architecture like that is equivalent to fitting a straight line.

Straight line fits can be excellent tools in data analysis, but the power of MLPs is that they are Universal function approximators, and the approximations improve the deeper they get. Hence, we introduce a nonlinearity, σ, so the full equation for an MLP reads

$$\mathbf{h}^{(l)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right), \tag{5.1}$$

where we have introduced a displacement to our weighted sum, $\mathbf{b}^{(l)}$, commonly known as the bias. Therefore, "deep" MLPs, consisting of many layers, become highly nonlinear functions which we can fit to almost anything.

However, as MLPs rely on fixed-size inputs, and the fully connected structure between layers is expensive, they are rarely the right choice for modern frontier-of-research studies in ML. Instead, they are often used behind the scenes as components of more specialised architectures, including GNNs, as we shall see.

5.1.2 Nonlinear activation functions

As we have discussed, MLPs are able to achieve their remarkable expressiveness via repeated applications of linear transforms (whose parameters are learnable), followed by nonlinear activation functions.

We shall focus on two variants: the Sigmoid function, and the Rectified Linear Unit (ReLU).

The ReLU function is simply the identity for positive inputs, and zero for negative inputs, see equation 5.2.

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0, \\ 0 & x \leq 0. \end{cases} \tag{5.2}$$

This has the advantage of being simple to write down and explain. It's also easy to implement gradient-based optimisations, since its gradients are simply +1 for positive inputs, and 0 for negative inputs (this is evidently extremely computationally inexpensive, too). However, the total loss of gradient information in the negative domain can diminish its expressive power. Therefore, many ML practitioners opt to use the LeakyReLU variant instead.

LeakyReLU is identical to ReLU in the positive domain: it's just the identity transform. However, it introduces a hyperparameter α for the gradient of negative inputs, allowing the gradient in the negative domain to take a non-zero value, see equation 5.3.

$$\text{LeakyReLU}(x;\alpha) = \begin{cases} x & \text{if } x > 0, \\ \alpha x & x \leq 0. \end{cases} \tag{5.3}$$

This means that the derivative of LeakyReLU is +1 for positive inputs, and the constant α for negative inputs. This is still attractive in its simplicity, but it introduces a hyperparameter which must be chosen by the ML practitioner training the model. This will either be chosen on the basis of guesswork, trial-and-error, or systematic hyperparameter tuning, requiring the evaluation of metrics over repeated training runs, which is an expensive use of compute resources.

An interesting variant which overcomes this limitation Parameterised ReLU (PReLU) activation function. This is defined identically to LeakyReLU in equation 5.3, but PReLU treats the negative gradient α as a learnable parameter, such that the model is able to converge on an appropriate value during training.

Due to its enhanced expressiveness, in this work we favour using PReLU activation functions in latent layers of our NN architectures[1].

As we shall explore, our work is primarily concerned with binary classification problems. That is, our models transform the input data to produce yes / no (or 1 / 0) outputs. However in principle, outputs produced from repeated linear transforms and PReLU activations could take on any value, positive or negative. We need a way of mapping these outputs into the range $(0, 1)$.

The Sigmoid activation is a smooth step activation function which maps $x \in \mathbb{R}$ to $\sigma \in (0, 1)$, and is defined by the formula:

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}}. \tag{5.4}$$

[1] *Latent* or *hidden* layers are the internal layers of a NN or MLP, *ie.* any layer that is not the input or output layer.

As such, it is the de facto nonlinearity to provide binary classification scores for NN outputs.

So, in general for this work, we make use of PReLU nonlinearities in latent layers, and we apply a Sigmoid function to our outputs for binary classification tasks.

5.1.3 Jet images

The inspiring success of computer vision over the past decade has led to many fields utilising algorithms such as Convolutional Neural Networks (CNNs) for classification tasks. Representing the substructure of jets as "jet images" leads to two dimensional arrays of pixels in the $\eta - \phi$ plane (Cogan et al., 2015). These are effectively heatmaps or 2D histograms, where each bin contains the total amount of energy or p_T deposited in a corresponding calorimeter cell (Kasieczka et al., 2017); p_T is typically preferred, due to its invariance under Lorentz boosts (de Oliveira et al., 2016). These models have shown success in tagging boosted bosons (Cogan et al., 2015; de Oliveira et al., 2016), and top quarks (Macaluso and Shih, 2018; Dillon et al., 2019; Kasieczka et al., 2017). Jet tagging is the problem of associating a jet with a possible high energy parton which initiated it. A drawback of this approach is the need for extensive pre-processing, exploiting symmetries such as rotation and translation invariances, to align jets along the same axis, and keep their positions consistent. On top of this, the fixed size of 2D input arrays for jet images imposes limitations on the input data representation and memory consumption. Towers represented by the binned energy deposits are unable to distinguish between multiple particles in the same bin. Additionally, the sparsity of collider data results results in jet image representations having mostly zeroed-out bins, which limits memory efficiency. More ergonomic matches of jet representations with models have been shown to provide superior results.

5.1.4 Recurrent neural networks

As we have discussed in chapter 4.1.3, traditional sequential recombination clustering algorithms result in binary trees. These trees are often interpreted as an approximation for the series of particle splittings which resulted in the final state particles at the detector level. This sequential tree structure is ideally suited to Recurrent Neural Networks (RNNs), a technique from Natural Language Processing (NLP) which operates over variable length sequences. RNNs can thus be used as a form of feature extraction / embedding, traversing from the root node of the binary tree down to the leaves, recursively updating a global representation of the jet. This representation is naturally summarised as a fixed-length vector, which may be passed to other architectures for downstream prediction tasks.

Using RNNs in this way for $W^{\pm}$ boson tagging has been shown to offer improvements in both selection performance and efficiency (Louppe et al., 2019). However, Louppe et al. (2019) highlight that the topological structure imposed by the sequential recombination algorithm has inferior performance when compared with simply ordering the jet constituents by their value of p_T. It seems likely that this is a result of limitations in the assumption that the cluster topology is a meaningful approximation to the unseen decay history. Constructing a topological history of particle decays from sequential recombination is fundamentally flawed, since we wouldn't expect the decay history to be a tree, much less a binary tree[2]. However, the sequentially recombined representation was more robust in terms of IRC safety. RNNs are not the only offering from NLP to have an impact on jet physics – recent work has explored applications of the breakthrough architecture receiving global *attention* in recent years.

5.1.5 Transformers

Recent buzz over the transformer model's ability to effectively scale to large quantities of data has not gone unnoticed in the HEP community. Particle Transformer (ParT) (Qu et al., 2024) in CMS, has shown improvements over current methods in jet tagging. ParT employs attention mechanisms and pairwise features to infer the origins of jets, while its extension, GloParT, expands this framework to a larger set of classes, including all-hadronic and semi-leptonic decays. This transformer architecture enables the authors to train over truly huge datasets, with 100M jets[3]. This takes in a point cloud data structure, with four-momenta, electric charge, and particle type (*ie.* five classes: charged hadron, neutral hadron, electron, muon, and photon). Formatted as a matrix, pairwise relationships between all particles are learned, as well as embeddings of the particles, which are combined for downstream prediction tasks. In this way, the transformer architecture employed is effectively equivalent to a graph neural network (GNN), where the data is formatted as a fully-connected graph. This is a very expensive computation, since a fully connected graph with $\mathcal{O}(N)$ nodes corresponds to $\mathcal{O}(N^2)$ pairwise relationships, where each node and pairwise relationship is itself a feature vector. When pretrained on existing datasets and subsequently fine-tuned, ParT does achieve modest improvements to current state-of-the-art methods across the board for accuracy, AUC score, and background rejection, though no significant improvement is seen on the existing ParticleNet (Qu and Gouskos, 2020) without this pretraining and fine-tuning approach. We will explore more economical uses of graph data structures via explicit uses of GNNs in depth later, albeit for the different downstream task of particle reconstruction.

[2]This is because particles may undergo more complex interaction than simple $1 \to 2$ splittings. Also, during hadronisation weakly connected cycles are introduced, which is not representable as a tree.

[3]Future scholars may find this quaint, but at the time of writing this is two orders of magnitude larger than current publicly available datasets.

At this point, we may notice that it is not straightforward to encode jets into useful representations which match the requirements of input data for ML models. JetCLR also utilises transformers, but with the more generic goal of learning expressive representations of jets in high dimensional embedding spaces (Dillon et al., 2022). This combines the transformers' architecture with a contrastive learning loss function, leading to general purpose discriminative representations. In theory, it should be possible to use these representations for a number of downstream tasks, though the authors opted to test this on top tagging. They found an improvement when compared with jet image approaches, and while the top tagging results themselves were not state-of-the-art, the key message was that symmetry-preserving general purpose representations can deliver good results, and with future work may present a strong opportunity for transfer learning.

5.1.6 Generative models

Moving away from jet tagging, ML is also making an impact on speeding up simulation pipelines. Generating detector-level data is particularly computationally challenging. Traditional simulation techniques, such as those based on Geant4 (Agostinelli et al., 2003), are resource-intensive and time-consuming. ML-based approaches, such as variational autoencoders (VAEs) and generative adversarial networks (GANs), provide a computationally efficient alternative (Carrazza and Dreyer, 2019). For example, these methods have been employed by the ATLAS collaboration to model the electromagnetic calorimeter's response, quickly simulating electromagnetic showers while preserving accuracy (Aad et al., 2024).

In this work, we will explore the role of graph neural networks in particle reconstruction. That is, we will take point cloud data of the detector-level particles following a Pythia showering and hadronisation simulation, and attempt to cluster them to reconstruct two specific high energy particles: the Higgs boson and the top quark. We will do this by classifying each detector-level particle as to whether or not they belong to the cluster for our desired reconstructed particle. Operating over point cloud data efficiently to produce classifications on each individual point is an ideal problem space for GNNs to be applied.

But what is a graph?

5.2 Graphs as flexible data structures

A graph (or network) is minimally described by two sets: a set of nodes $\mathcal{V}$, and a set of edges $\mathcal{E}$. Graphs can be used to represent data structures in many domains, from the humdrum to the esoteric.

For example, a person's social network forms a graph, in which people are nodes, and the relationships between them are edges. Understanding graphs in this context may lead to better insights about a person's values and interests, based on their attributes and the context of their community.

GPS and navigation mobile applications have become valuable utilities in many people's lives; these combine graph based representations where nodes are cities, intersections, or waypoints, and edges are roads or pathways. These edges may be weighted with physical distance in space, projected travel time, or cost of any tolls en route. Applying techniques from formal graph theory, such as Dijkstra's algorithm, benefits the lives of at least one billion people each month[4].

In biochemistry, it has long been considered useful to model the structure of a molecule with a graph, where atoms are nodes, and bonds are edges (Knisley and Knisley, 2008). In this case, a global (or graph-level) feature u may describe some emergent property of the entire network *eg*. chemical properties of a drug.

Taken together with the node and edge features, the graph is denoted in its most general form as

$$G = (\mathcal{V}, \mathcal{E}, u)\,, \tag{5.5}$$

which is often abbreviated simply to $G\,(\mathcal{V}, \mathcal{E}, u)$.

Each edge $e_{sd} \in \mathcal{E}$ represents a pairwise relationship between the two nodes it connects, $v_s \rightarrow v_d$, where $v_s, v_d \in \mathcal{V}$. In a directed graph, v_s is referred to as the *source node*, and v_d is the destination node. In an undirected graph, all nodes in $\mathcal{V}$ serve as both source and destination nodes.

Graphs can take several forms. For instance, multigraphs allow multiple edges between the same pair of nodes. A special category of graphs, called *natural graphs*, are derived directly from the inherent structure of the data they represent. A family tree, for example, is a natural graph in which nodes correspond to individuals, and edges represent familial relationships.

A tree is a type of directed graph with the following properties:

- A single root node serves as the origin of all directed paths.

[4]This is based on Google's own, now archived, metrics for iPhone users of Google Maps in 2020.

- Each node is either an ancestor or descendant of other nodes.
- A node with no descendants is called a leaf node.

If sibling nodes are allowed to share edges, the graph becomes a directed acyclic graph (DAG). Unlike trees, DAGs may contain multiple root nodes and allow more complex interconnections while preserving a lack of cycles.

The description provided by $\mathcal{V}$ and $\mathcal{E}$ is topological. Much of graph theory relates to graphs from a purely topological perspective. However, since we wish to model physical systems with our graphs, where nodes and edges represent particles and the relationships between them, we wish to enrich them with additional data. This is done by attaching *attributes* at the node, edge, or graph level, and as such these networks are called attributed graphs[5].

5.2.1 Aside: distinguishing the generation DAG *vs.* the input graph

We have now developed sufficient graph vocabulary to distinguish networks based on structural descriptors. Therefore, we take this opportunity to break the flow of this thesis and make a distinction early to help clarify later remarks.

We will utilise two different kinds of graph in our study. The purpose of this subsection is not to fully describe the structure and method of construction for these graphs in depth; this is done in other sections. However, we wish to emphasise the structural composition of these graphs here as a means of additional context, to help readers avoid confusion when it is necessary to reference both kinds of graphs in the same discussion.

The simulation history will be represented as a generation DAG, described in chapter 8.3. This is a natural, attributed multigraph. Here the particles are represented as edges, and each edge is attributed with the full range of particle properties available from our simulations, see chapter 8. Nodes represent the interactions between particles, and while some studies may attach attributes representing vertex locations in space, we choose to leave the nodes unattributed. This is a multigraph because the particles (edges) produced from a common interaction (source vertex) may go on to interact with each other immediately afterwards (at a common destination vertex). Hence two nodes may share multiple edges. The root of this graph has two outgoing edges, which are the initial protons in the collision, and the leaves of the graph represent where the particles terminate upon entering the detector, at the end of the showering and hadronisation.

[5]Once we go on to discuss machine learning on graphs, we shall call these feature vectors, but really it's the same thing.

The second graph is formed from the edges directly incident on the leaves of the generation DAG. The ancestral information is discarded, and these particles form the node set for a new graph. This is naturally a point cloud, so we construct an edge set based on the inter-particle distance between pairs of nodes. There are many approaches to doing this, where k-nearest neighbours (kNN) is a popular choice. We instead adopt the prescription of the Energy Weighted Message Passing (EWMP) network (Konar et al., 2022), but this will be explored in more depth in chapter 5.5.

For now, we simply wish to stress that, while the generation DAG is used in producing supervision labels for the input graph, it is not the input graph itself.

- The generation DAG has a natural hierarchical topology, represents the particles as edges, and includes intermediate particles which do not enter the detector.
- The input graph has no natural topological structure, represents particles as nodes, and is formed entirely of detector-level particles.

With this aside out of the way, we return to describe practical mathematical approaches for representing graphs.

5.2.2 Neighbourhood representations

The edge set $\mathcal{E}$ of a graph can be represented in several forms, each with unique advantages and drawbacks.

Rich analyses are made possible by selecting specific representations. As will be discussed in chapter 5.2.2.5, the incidence matrix may be used as a discrete differential operator over graph structures. Additionally, the Laplacian matrix encodes rich information via its eigenvector decomposition. Spectral analysis can be used to understand the graph structure, and is an effective tool in clustering graphs (Cerro et al., 2022).

In the following sections, we define and consider the functions of several edge representations. In this work, we will use the edge list and adjacency matrix representations to describe our graph structures.

5.2.2.1 Edge List

This representation consists of a $N \times 2$ matrix or a list of N tuples, where each tuple corresponds to an edge. It is compact, simple to define, and representing multigraphs may be done trivially by adding an additional entry with the same pair of vertex indices.

Edge attributes are likewise easy to include, simply by storing more than 2 values per row, where the first two continue to provide source and destination vertex indices, and subsequent values provide edge attributes. Because there is no limit on how many values can be stored in each row, the attribute may be a vector quantity of any size.

However, it can be inefficient for manipulation and traversal. The edge list is conceptually identical to the sparse matrix coordinate (COO) list representation.

5.2.2.2 Adjacency list

In this representation, each node maintains a list of its neighbours or destinations to which it links. Multigraphs are also supported by this representation. The adjacency list format is well-suited for efficient storage and traversal, especially in large or sparse networks.

Attributing edges is noticeably more complex than in the edge list format, and normally involves storing nested `tuples` and / or [`mappings`] for each entry in an adjacency row. The Python package `networkx` (Hagberg et al., 2008) provides an implementation of this approach.

5.2.2.3 Adjacency matrix

The adjacency matrix **A** is a square matrix where:

$$A_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{5.6}$$

For undirected graphs, the matrix is symmetric. Self-loops appear along the diagonal.

In directed graphs, rows represent source nodes, and columns represent destination nodes. The sum over a row gives the *out-degree* of a node, while the sum over a column gives the *in-degree*. The *degree* of a node is defined as the number of edges which are connected to it, without considering the direction of those connections, *ie.* incoming or outgoing. Thus, in-degree and out-degree are the number of edges specifically the number of incoming and outgoing edges, respectively.

Non-binary adjacency matrices can encode additional information. For instance, multigraphs can store the number of edges between nodes by allowing $A_{ij} \in \mathbb{N}_0$, and scalar edge attributes can represent weighted relationships. However, adding edge attributes and supporting multigraphs simultaneously is not possible without additional data structures; nor is storing non-scalar edge attributes.

5.2.2.4 Laplacian matrix

The Laplacian matrix $\mathbf{L}$ is defined as the difference between the degree matrix $\mathbf{D}$ and the adjacency matrix $\mathbf{A}$:

$$L_{ij} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{ and } e_{ij} \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

For directed graphs, two Laplacians can be defined: one for in-degrees ($\mathbf{L}_{\text{in}}$) and another for out-degrees ($\mathbf{L}_{\text{out}}$). Normalisation strategies help address the dominance of high-degree nodes.

The symmetric normalised Laplacian is given by:

$$L^{\text{sym}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}} = \mathbb{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}. \tag{5.8}$$

The random walk normalized Laplacian is:

$$L^{\text{rw}} = \mathbf{D}^{-1}\mathbf{L} = \mathbb{I} - \mathbf{D}^{-1}\mathbf{A}. \tag{5.9}$$

It is impossible to represent multigraphs or edge attributes using this representation alone, as the elements of the Laplacian matrix encode important information that would be obscured by such attempts.

The Laplacian may also be written as $\mathbf{L} = \nabla^T\nabla$, where ∇ is the incidence matrix.

5.2.2.5 Incidence matrix

The incidence matrix ∇ represents edges in terms of their source and destination nodes, where each row i is a node, and each column j is an edge:

$$\nabla_{ij} = \begin{cases} 1 & \text{if } i = \text{source node} \\ -1 & \text{if } i = \text{destination node} \\ 0 & \text{otherwise.} \end{cases} \tag{5.10}$$

This means that the incidence matrix – unlike the Laplacian or adjacency matrices – is not generally square[6].

It is easy for two nodes to share multiple edges in this representation, since the matrix may be extended with identical columns. It is also possible to store scalar attributes,

[6]Cycles are one case where you could always expect a square incidence matrix.

just as in the adjacency matrix example, though this is complex and may hamper the mathematical properties of the incidence matrix.

In particular, when a vector of node features is left-multiplied by the incidence matrix, the result is a vector representing edge feature differences between the source and destination nodes. This operation resembles a discrete differential operator.

5.2.3 Graph connectivity

A graph is connected if any two nodes are joined by a path. Disconnected graphs are decomposed into components, subsets of the graph where each subset is connected.

In directed graphs, connectivity is categorized into three types. In strongly connected graphs, there exists a directed path between any two nodes, respecting edge directionality. Weakly connected graphs may still find paths by disregarding edge directionality. Disconnected graphs contain multiple components with no path connecting them.

For undirected graphs, connectivity can be analysed using block-diagonal adjacency matrices, where each block corresponds to a separate component.

Due to how we form the input graphs to our models, while it's possible a given graph may be (weakly) connected, they generally have multiple disconnected components. More on this in chapter 10.2.1.

5.3 Message passing

Graphs, represented by nodes and edges, provide a flexible framework for modeling relationships within complex data. Beyond their topological structure, graphs can be enriched with attributes on nodes and edges, capturing diverse types of information—from physical properties to semantic relationships. For instance, in high-energy physics, nodes might represent particles with attributes such as four-momentum, while edges could denote interactions or distances between these particles. These attributes enable the representation of both local and global properties of a system.

In machine learning, embeddings play a crucial role in transforming graph data into vector spaces, where computational operations become feasible. An embedding maps the attributes of nodes, edges, or the entire graph into a structured vector space. This mapping ensures that similar entities are placed closer together, facilitating downstream tasks such as classification, regression, or clustering. The flexibility of

embeddings allows their application across a wide range of fields, from social network analysis to molecular chemistry.

Graphs, however, present unique challenges for machine learning. Unlike images or sequences, which have fixed grid-like structures, graphs exhibit arbitrary sizes and connectivity patterns. This variability complicates the design of models that operate directly on graph data. Additionally, graph data must be processed in a way that respects permutation invariance, the property that the output should not depend on the arbitrary ordering of nodes or edges.

Message passing, the cornerstone of Graph Neural Networks (GNNs), addresses these challenges by iteratively propagating and aggregating information across a graph. At each iteration (or layer), nodes update their embeddings by combining their own attributes with those of their neighbours. We can describe the process of defining graph neural networks, and forming downstream predictions on nodes, edges, or globally, using the graph network (GN) formalism .

5.3.1 The graph network formalism

The graph network formalism (Battaglia et al., 2018) provides a flexible framework for performing computations on graph-structured data. It unifies GNN approaches to ML under a single generalised description, including message passing neural networks (MPNNs) and non-local neural networks (NLNNs). By defining a class of functions for relational reasoning, and highlighting opportunities to customise architectural choices, it is applicable to a wide range of domains. The GN framework is not a specific model architecture, but a vocabulary for thinking about computations on graphs, which we will use to describe and compare GNN architectural choices.

The GN formalism shares the same graph description we have defined in equation 5.5, describing a graph $G = (\mathcal{V}, \mathcal{E}, u)$. Graphs in this formalism are treated as directed, attributed multigraphs, allowing for edges with directionality, attributes, and even multiple connections between nodes. Attributes can take the form of vectors, tensors, or more complex data structures, and the graph's structure can either be predefined or inferred. Connectivity is often described using adjacency matrices or by encoding sender and receiver indices for the edges. This aligns with our usage of adjacency matrices and edge lists.

The computational unit of the GN framework is the GN block, which transforms an input graph into an updated graph with the same structure but whose attributes undergo embedding. Each block defines two internal functions: an update function, and an aggregation function. There are three blocks in total corresponding to edge, node, and graph-level update, sequentially in that order. Collating all three blocks, the update functions ϕ^e, ϕ^v, and ϕ^u transform the edge, node, and graph level attributes

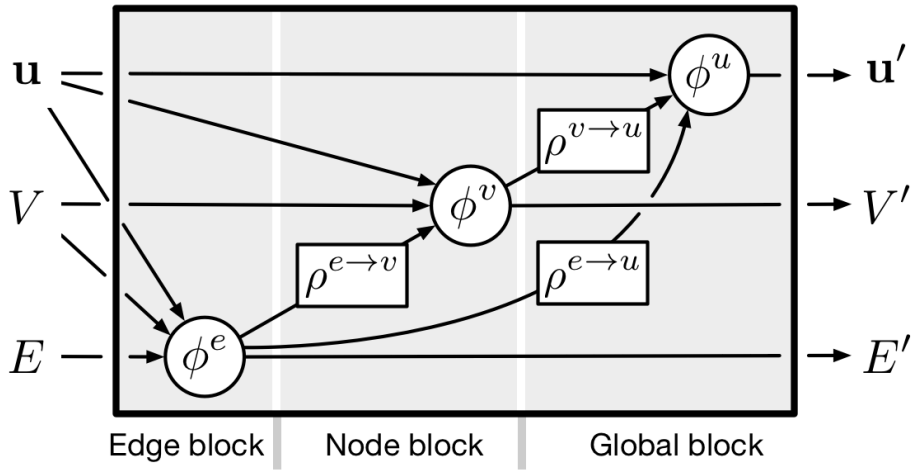


Figure 5.1: The generic Graph Network architecture, from Shlomi et al. (2021).

into latent embeddings. They have fixed input and output sizes. The aggregation functions $\rho^{e\to v}$, $\rho^{e\to u}$, and $\rho^{v\to u}$ apply a permutation invariant aggregation to reduce a variable number of incoming embeddings to a fixed-size output representation. This is important, as these pooled embeddings form so-called "messages", which are propagated to the subsequent block for the next level of embedding.

As discussed, the process starts in the edge block, see figure 5.1. ϕ^e concatenates the feature vectors of the nodes connected by the edge e_k (where k represents the edge index). Subsequently, the feature vector of both the edge itself, and global feature vector (if any) are also concatenated. The resulting vector is then mapped into its embedded representation $\mathbf{e}'_k$. The update function visits every edge in the graph structure, such that every edge has a new embedding. We may collect these edge embeddings as columns in a matrix E'.

Therefore, we have a description for the formation of new edge embeddings over the whole graph. However, the message passing algorithm is able to update node features too, by forming messages based on the nodes connected by incoming edges in its neighbourhood. This is achieved in the first step of the node block.

We start by constructing so-called messages from the incoming edge embeddings. However, an individual node may have any number of incoming edges, and we need a fixed-size message for our following update step. Additionally, there really isn't a defined ordering for neighbour nodes. Therefore, to summarise the information about the target node's neighbourhood, provided by the incoming edge embeddings, these are reduced along their feature dimension using $\rho^{e\to v}$, which is a permutation invariant aggregation function. This forms a new vector $\bar{\mathbf{e}}'_i$ whose size matches the edge embedding dimensions. It is this vector, which we call a "message", from which the algorithm message passing gets its name.

The next step of the node block is to apply the node update function ϕ^v. For a given node i, the update function ϕ^v concatenates the message received from its neighbourhood $\bar{\mathbf{e}}'_i$ with its own node feature vector, and the global feature vector (if any). In the same fashion as the update function in the edge block, it then maps the resulting vector to a new embedding $\mathbf{v}'_i$ for node i. Likewise, ϕ^v visits every node in the graph, forming a new set of latent representations for each of them. Therefore, at this stage, we have successfully performed updates to both node and edge embeddings. We may collect these node embeddings as columns in a matrix V'.

If a sufficiently expressive choice for ϕ^e is made, the edge embeddings may model features of the relationship between pairs of nodes. Node embeddings naturally update their embeddings on the basis of receiving information about their position and identity within their neighbourhoods. This is a form of information diffusion.

We could stop here. Indeed we will go no further than these two steps in our studies, since we will not be performing graph-level inferences. However, for completeness, we describe the graph block.

To obtain a global embedding of the graph, messages from both the latent representations of the edges and nodes are constructed. This is achieved simply by applying the aggregation functions $\rho^{e\to u}$ and $\rho^{v\to u}$ to perform reductions over the rows of edge and node embedding matrices E'_i and V' respectively. These form two two column vectors. The global update function then concatenates these with each other, and the currently global feature (if any). Finally, it maps the resulting vector into a new embedding for the global feature, $\mathbf{u}'$.

This series of operations may be represented by the following equations:

$$\begin{aligned} \mathbf{e}'_k &= \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) & \bar{\mathbf{e}}'_i &= \rho^{e\to v}(E'_i) \\ \mathbf{v}'_i &= \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) & \bar{\mathbf{e}}' &= \rho^{e\to u}(E') \\ \mathbf{u}' &= \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) & \bar{\mathbf{v}}' &= \rho^{v\to u}(V') \end{aligned} \tag{5.11}$$

In our work, we will use MLPs with PReLU nonlinearities for update functions, and summations for our permutation invariant aggregation functions, see chapter **??**:. This is made slightly more complex by requiring IRC safety, chapter 5.5 for the refinements we adopt in order to ensure this.

Thus, the feed-forward mechanism produces a graph with the same topology $G(\mathcal{V}, \mathcal{E}, u) \to G'(\mathcal{V}', \mathcal{E}', u')$ as the input. In GNN terms, once we have passed through all three blocks of the GN framework, we say we have applied one graph layer or GNN layer. By stacking graph layers, we form the full GNN architecture, prior to the final classification or regression layer.

For a given GNN layer, the number of nodes in the input and output layers of the update function correspond to the initial and final number of dimensions of the feature vectors transformed. This solves the problem of variable graph cardinality $|\mathcal{V}|$, since the underlying NN architecture only expects consistency in the dimension of the input feature vectors, allowing graphs of arbitrary size and connectivity to be used for training and inference. The choice of which data is represented by nodes, edges, and graph level attributes, and how these data are related, can thus be chosen to best suit the problem to which the GNN is applied.

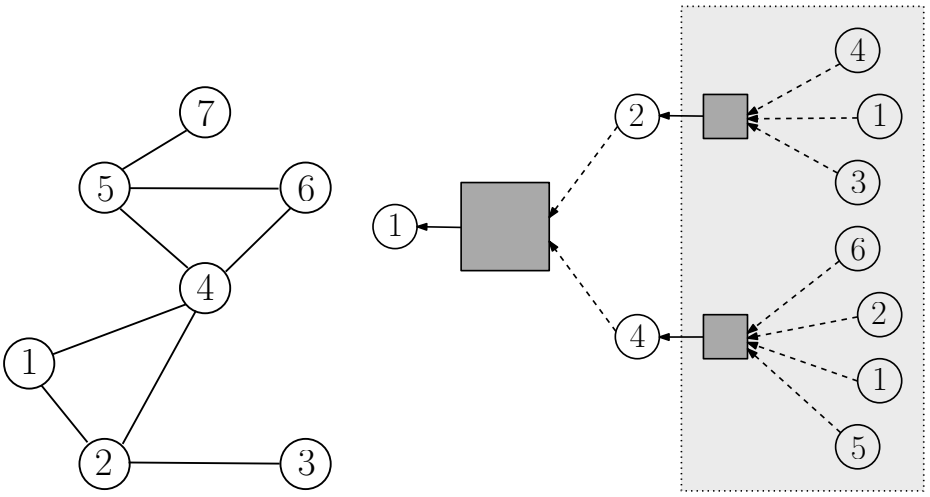
The GN formalism generalises and extends many existing graph neural network (GNN) approaches. For instance, graph convolutional networks (GCNs) can be viewed as a specific case of GN blocks without global attributes, using simple linear transformations for edge updates. Non-local neural networks (NLNNs) correspond to fully connected graphs where pairwise attention mechanisms compute edge attributes. Transformers, another powerful architecture, can be expressed as fully connected graphs with attention-based edge weighting.

5.3.2 Computational graphs

Computational graphs provide a clearer description of the node update function. They are defined as the rooted subtree around a given node of the input graph, and their structure mirrors the process of aggregating and forming a new embedding on this node. In some sense, this this reflects a bespoke NN architecture centred on each node, with different computational structures depending on the input graph topology.

Figure 5.2a shows a simple graph structure, with nodes labelled from 1 - 7. In order to update the embedding on node 1, we aggregate the neighbouring node features, and transform the result with an MLP, see chapter 5.1.1. Figure 5.2b visualises this process in a computational graph. Dashed arrows show the neighbours being aggregated to MLP blocks in grey, and solid arrows represent the resulting update operation performed on the embedding of node 1.

Applying a single GNN layer corresponds to embedding a node with its neighbours initial feature vectors; if we stack two layers, however, the neighbours we aggregate have themselves been embedded based on their own neighbourhoods. The result is that the number of layers in a GNN is equal to the number of hops a node uses within its local graph structure to perform its update operation, which widens the node's *receptive field*, *ie.* the neighbouring nodes which are used to embed it. The grey dotted region in figure 5.2b shows how the computational graph grows when increasing the number of graph layers from one to two.



(a) Input graph undergoing node embedding.

(b) Computational graph to representing update function for node 1.

Figure 5.2: Demonstrating the node update function as a rooted subtree centred at the receiving node, with grey boxes representing the shared weights at each layer. As layers are stacked, the computational graph grows, representing additional hops in the input graph.

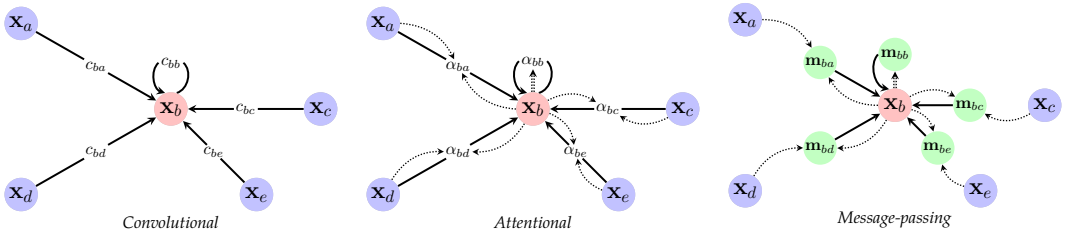
In order for a GNN to maximise overall expressiveness, it is clear the update operation for the computational graph should be as expressive as possible. In particular, this is determined by the choice of aggregation function.

5.3.3 Expressiveness is in the aggregation function

Bronstein et al. (2021) provide a useful framework for specialising the GN formalism within three "flavours". These are identified as convolutional, attentional, and message passing. Flavours are distinguished by what normalisation is applied during the aggregation function when forming messages. Each flavour is a manifestation of the same GN formalism, with the first being the most direct and general application, and the remaining two are listed in order of increasing generality and implicit computation. We outline the differences in message normalisation below, and figure 5.3 is included from the source text to visualise this.

The convolutional flavour bases normalisation on classical graph theory metrics. For instance, node degree, *ie.* how many neighbours a given node has, is used in the Graph Convolutional Network (GCN) architecture (Kipf and Welling, 2017).

The attentional flavour makes inference more implicit by parameterising the normalisation with learnable weights (Veličković et al., 2018). Messages are weighted with an attention score, which is passed through a `softmax` activation such that these attention scores form a set whose elements sum to one. These weights are learned via



(a) Sender node features are normalised by a constant.

(b) Sender node features are normalised by a learned attention score.

(c) Edge embeddings are sent as vector based messages, computed from both the sender and receiver node features.

Figure 5.3: Visualising data flow for the three flavours of GNN, based on message normalisation (Bronstein et al., 2021).

an MLP applied to the aggregate of node features between the receiving and neighbouring nodes, so the normalisation is made implicit, rather than subjecting the operation to a potentially limiting inductive bias.

In both of these cases, the weighted neighbouring node features are then aggregated, and concatenated with the receiving node's features, before ultimately being passed through the node's update function ϕ.

Finally, the message passing flavour increases the level of implicit computation by constructing incoming messages, not as the weighted feature vectors of a node's neighbours, but instead as a full-blown edge embedding feature vector, whose update function is identical to ϕ^e from equation 5.11, except without the global feature $\mathbf{u}$. These messages require no weighting, as they are entirely implicit. They can then be aggregated, concatenated with the receiving node features, and updated as normal.

Poor choice of aggregation functions may result in less expressive embeddings, or have specific failure cases when distinguishing sets of nodes.

Consider element-wise mean pooling, as used in GCN. If we treat distinct feature vectors in a node's receptive field as numbers in a multi-set, element-wise mean pooling would fail to distinguish between neighbourhoods with the same proportions of numbers, *eg.* $\{1, 2\}$ and $\{1, 1, 2, 2\}$ would result in identical embeddings for GCN. Element-wise max-pooling over neighbourhood features also fails to produce distinct embeddings between multi-sets if their unique subsets have the same mixture of feature vectors / numbers, *eg.* $\{1, 1, 2, 3\}$ and $\{1, 2, 2, 3, 3\}$ both have the unique subset $\{1, 2, 3\}$, so this would produce the same embedding. Element-wise sum-pooling manages to distinguish both of these cases, so this appears to be the best choice among non-parametric[7] aggregations (Leskovec, 2021).

[7] Parametric aggregation functions exist which elevate the expressive power of GNNs, namely GIN, see (Xu et al., 2018).

5.4 Interaction networks

The Interaction Network (IN) is designed to model the interactions and relationships between objects in a structured environment. It operates by iteratively updating edge and node embeddings to capture graph-level reasoning. This section outlines the architecture and its key components in detail.

The IN processes graph-structured data in several stages. First, the attributes of the receiving nodes, sending nodes, and edges are concatenated and passed through an MLP, which produces updated edge embeddings. Next, for each node in the graph, the updated embeddings of its incident edges are aggregated, typically through summation, to encode the influence of neighbouring edges. The aggregated edge effects are then concatenated with the node's own attributes and any external factors, and this concatenated vector is processed through another MLP to compute updated node embeddings. For tasks that require graph-level inference, the embeddings of all nodes are summed element-wise into a single global vector. This global embedding is then processed through a final MLP to generate the output for the entire graph. The steps for graph-level aggregation are task-dependent and may be omitted for applications requiring only node or edge level predictions. We do not require graph-level inference in our work.

5.4.1 Edge update block

The edge update block is responsible for updating the embeddings of edges by combining information from the attributes of sending and receiving nodes and the edge itself. For each edge, the attributes of the sending node, the receiving node, and the edge are concatenated into a single vector

$$\mathbf{z}_k = \mathbf{e}_k \oplus \mathbf{v}_{rk} \oplus \mathbf{v}_{sk}, \tag{5.12}$$

where $\mathbf{e}_k$ is the attribute vector for the edge, $\mathbf{v}_{rk}$ is the attribute vector for the receiving node, and $\mathbf{v}_{sk}$ is the attribute vector for the sending node. This process is repeated for all edges in the graph, resulting in a matrix:

$$\mathbf{Z}_e = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_n \end{bmatrix}, \tag{5.13}$$

where each column corresponds to the concatenated attributes of an edge, and n is the number of edges in the graph.

An MLP is applied independently to each column of $\mathbf{Z}_e$, mapping the concatenated attributes into a new embedding space. The output of this operation is a row matrix of updated edge embeddings.

$$\mathbf{E}' = \begin{bmatrix} \mathbf{e}'_1 & \mathbf{e}'_2 & \dots & \mathbf{e}'_n \end{bmatrix}. \tag{5.14}$$

This update step can be expressed as:

$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{rk}, \mathbf{v}_{sk}), \tag{5.15}$$

where ϕ^e is the edge update function parameterised by the MLP.

Once the updated edge embeddings have been computed, they are aggregated to compute the influence of edges on their respective receiving nodes. For a receiving node i, the embeddings of all incident edges are summed element-wise.

$$\overline{\mathbf{e}}'_i = \rho^{e \to v}(\mathbf{E}'_i) = \sum_{\{k | r_k = i\}} \mathbf{e}'_k, \tag{5.16}$$

where $\rho^{e \to v}$ represents the aggregation function and $\{k \mid r_k = i\}$ denotes the set of edges where i is the receiving node. The result of this aggregation is a matrix:

$$\overline{\mathbf{E}}' = \begin{bmatrix} \overline{\mathbf{e}}'_1 & \overline{\mathbf{e}}'_2 & \dots & \overline{\mathbf{e}}'_m \end{bmatrix}, \tag{5.17}$$

where m is the number of nodes in the graph.

5.4.2 Vertex update block

The vertex update block updates the embeddings of nodes by combining their own attributes, the aggregated edge effects, and any external effects. For each node i, a vector is constructed by concatenating the node attributes, the aggregated edge effects, and the external attributes:

$$\mathbf{z}_i = \mathbf{v}_i \oplus \overline{\mathbf{e}}'_i \oplus \mathbf{x}_i, \tag{5.18}$$

where $\mathbf{v}_i$ is the node attribute vector, $\overline{\mathbf{e}}'_i$ is the aggregated effect of all edges incident on node i and $\mathbf{x}_i$ is the vector of external attributes affecting the node. This concatenation is performed for all nodes, resulting in a matrix:

$$\mathbf{Z}_v = \begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_m \end{bmatrix}. \tag{5.19}$$

An MLP is applied independently to each column of $\mathbf{Z}_v$, mapping the concatenated attributes into a new embedding space. The output of this operation is a row matrix of updated node embeddings:

$$\mathbf{V}' = \begin{bmatrix} \mathbf{v}'_1 & \mathbf{v}'_2 & \dots & \mathbf{v}'_m \end{bmatrix}. \tag{5.20}$$

This node update step can be expressed as:

$$\mathbf{v}'_i = \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{x}_i), \tag{5.21}$$

where ϕ^v is the node update function parameterised by the MLP.

5.4.3 Global update block

For tasks that require graph-level predictions, the embeddings of all nodes are aggregated into a single vector to encode global information about the graph. This aggregation is performed using an element-wise sum over the columns of the node embedding matrix $\mathbf{V}'$:

$$\bar{\mathbf{v}}' = \rho^{v \to u}(\mathbf{V}'), \tag{5.22}$$

where $\rho^{v \to u}$ is the aggregation function. The resulting global embedding vector $\bar{\mathbf{v}}'$ is passed through a final MLP to produce the graph-level output:

$$\mathbf{u} = \phi^u(\bar{\mathbf{v}}'), \tag{5.23}$$

where ϕ^u is the global update function parameterised by the MLP. If no graph-level output is required, this block can be omitted, and the node embeddings from the vertex update block can be used directly.

5.4.4 GN formalism and flavour considerations

The IN is a highly expressive GNN architecture. In sections chapters 5.3.1, 5.3.2, 5.3.3 we presented a powerful vocabulary for describing GNN architectures. Applying this to describe the IN, we may understand it clearly in terms we have defined well.

The IN engages all three blocks of graph embedding: the edge block, the node block, and the global block. Edge embeddings are stored as the "interactions", hence the name. The IN was established with an emphasis placed on simulation of dynamic physical systems (Battaglia et al., 2016) This is why the external effect vector $\mathbf{x}_i$ is included, to account for effects such as gravity. However, if we encode this as a global feature of the graph at the input layer, functionally the network is no different to the message-passing flavour network we discussed in chapter 5.3.3, (Bronstein et al., 2021).

In chapter 10, we take inspiration from a method nominally applying INs to reconstruct $W^{\pm}$ bosons from simulation data. Since the authors omit the use of external effects vectors entirely, any potential discrepancy between INs and GNNs with message-passing flavour aggregations is rendered moot, so we may use the terms interchangeably.

5.5 Energy weighted message passing networks

5.5.1 Graph construction

Infrared and collinear (IRC) safety is a fundamental requirement in the design of graph-based methods for EWMPNs in high-energy physics (Konar et al., 2022). In this section, we outline the approach proposed by the authors of the EWMPN architecture to constructing IRC-safe graphs from a point cloud of particle momenta. This ensures that the resulting representation remains robust under infrared and collinear emissions.

Graphs are an intuitive representation of particle systems, with nodes corresponding to particles and edges encoding pairwise relationships. However, constructing such graphs in an IRC-safe manner requires addressing specific challenges. In particular, any graph construction must satisfy two critical properties: collinear safety and infrared safety.

One important consideration is the inclusion of self-loops, where each node includes itself in its neighbourhood. Without this feature, the graph can fail to preserve the original information of a particle when it undergoes a splitting process, rendering the graph IRC-unsafe. Furthermore, methods such as k-nearest neighbour (kNN) are inherently unsafe in this context, as the neighbourhood of a node can change discontinuously when a particle splits. This instability arises because kNN methods drop connections in response to changes in the local configuration of particles, failing to preserve relationships that are critical for IRC safety. By contrast, a radius-based approach ensures that connections are preserved under splitting, as illustrated in figure 5.4. In this diagram, the black arrows represent radius-based graph

connections, which are more robust when compared with the red arrows, representing kNN connections.

To formalise IRC-safe graph construction, the inclusion of neighbours for a given particle is governed by decision and threshold functions. For any pair of particles p_i and p_j, the decision function $D(p_i, p_j)$ quantifies their relationship, while the threshold function $T(p_i, p_j)$ determines whether the relationship is strong enough for p_j to be included in the neighbourhood of p_i. Formally, p_j is assigned to the neighbourhood $N[i]$ of p_i if $D(p_i, p_j) \leq T(p_i, p_j)$. The design of these functions must respect the kinematic constraints imposed by IRC safety.

Collinear safety requires that the neighbourhood relationship is preserved under the splitting of a particle into two collinear particles. This implies that if p_r and p_s are collinear fragments of an original particle, the decision and threshold conditions must satisfy:

$$D(p_i, p_r + p_s) \leq T(p_i, p_r + p_s) \iff D(p_i, p_r) \leq T(p_i, p_r) \wedge D(p_i, p_s) \leq T(p_i, p_s), \tag{5.24}$$

and similarly in the reverse direction:

$$D(p_r + p_s, p_i) \leq T(p_r + p_s, p_i) \iff D(p_r, p_i) \leq T(p_r, p_i) \wedge D(p_s, p_i) \leq T(p_s, p_i). \tag{5.25}$$

These conditions ensure that a particle's connections to other nodes in the graph remain consistent as it undergoes collinear splitting.

Infrared safety, on the other hand, demands that the addition of soft particles to the system does not remove any existing neighbours from a node's neighbourhood. If $N[i]$ represents the neighbourhood of p_i before the addition of a soft particle, and $N'[i]$ represents the neighbourhood after the perturbation, then infrared safety requires that $N[i] \subseteq N'[i]$. This property is naturally satisfied when the inclusion condition $D(p_i, p_j) \leq T(p_i, p_j)$ depends only on the 4-momenta of the two particles being compared, as the contribution of a soft particle cannot invalidate this inequality for existing neighbours.

These principles are satisfied in the collinear limit ($\Delta_{rs} \to 0$), provided the decision and threshold functions depend solely on direction-related quantities, such as pseudorapidity (η) and azimuthal angle (ϕ). A particularly simple and effective implementation is to construct graphs in the $\eta - \phi$ plane using a constant radius. In this case, the decision function is given by the angular separation ΔR_{ij} between

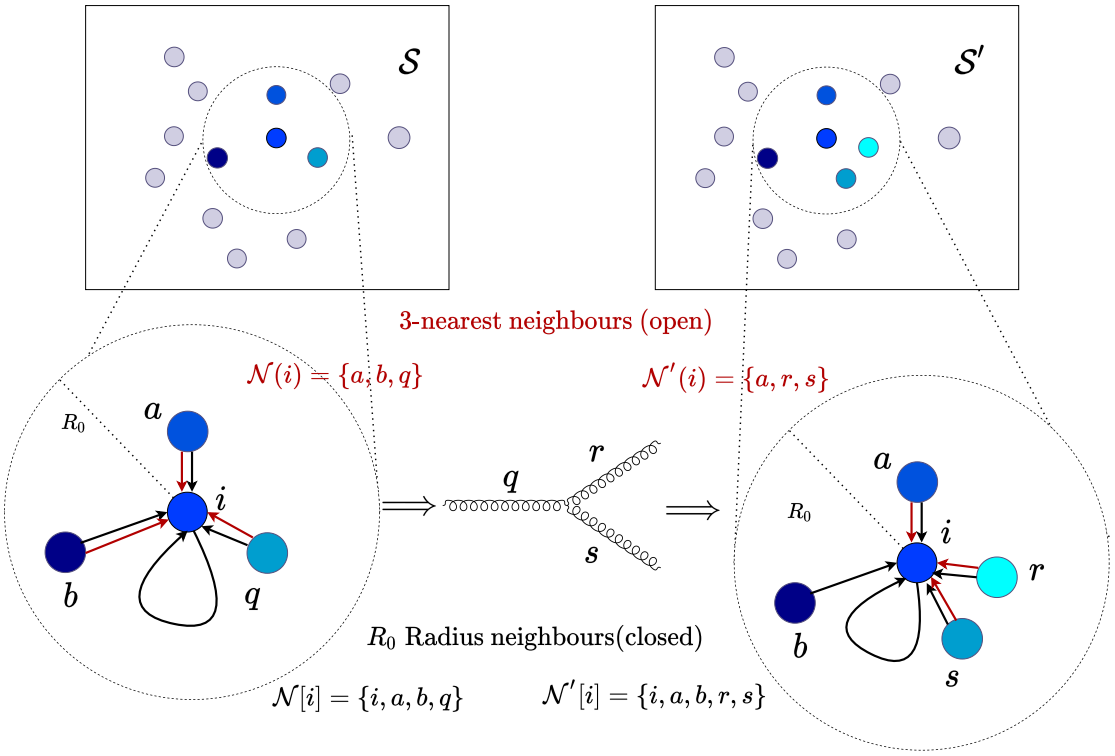


Figure 5.4: A k-nearest neighbour graph in the (η, ϕ)-plane will have a different structure when any particle q splits to r and s. The set $\mathcal{S}$ denote the particles in the jet when there is no splitting, while $\mathcal{S}'$ denotes the particles with q splitting. We show the directed edge connection to i from its three nearest neighbours with red on either side. The neighbourhood set $\mathcal{N}(i)$ has b in it, however when q splits, $\mathcal{N}'(i)$ does not contain b. Therefore, the graph's structure prevents a smooth extrapolation between the two scenarios in the infra-red and collinear limit. This is not the case for a radius graph with radius R_0 in the (η, ϕ) plane, which is shown with black connections. We also include the self-loop of i, by using the closed neighbourhood sets $\mathcal{N}(i)$ and $\mathcal{N}'(i)$, since the node i could also split into two particles (Konar et al. (2022)).

particles p_i and p_j, and the threshold function is a fixed radius R_0, such that p_j is included in $N[i]$ if $\Delta R_{ij} \leq R_0$. This radius-based approach naturally satisfies the constraints of both collinear and infrared safety, making it a practical choice for IRC-safe graph construction.

5.5.2 IRC safe message passing

The crux of the IRC safe message passing prescription outlined by Konar et al. (2022) consists of two parts. The first is to construct the node features from direction-only components of the four-momenta, and the second is to weight the sum aggregation in the message construction by relative p_T in the neighbourhood. Let's explore what this means, and why it ensures IRC safety.

Consider a node with direction-only four-momentum components for its node feature,

$$\hat{\mathbf{v}}_i^{(0)} = (\eta_i, \phi_i)\,, \tag{5.26}$$

ie. the coordinates of the particle in the $\eta - \phi$ plane, where the superscript 0 refers to the node feature at the 0th (input) layer of the GNN, and the hat over $\hat{\mathbf{v}}_i^{(0)}$ indicates that no information about vector magnitude is encoded. Now let's assume that this node is in the neighbourhood of another node j, *ie.* $i \in \mathcal{N}(j)$. The GN formalism tells us that our first step must be to form new edge embeddings; here, we shall take a simple edge update function:

$$\mathbf{e}_{ij}^{(1)} = \phi_e^{(0)}(\hat{\mathbf{v}}_j^{(0)}, \hat{\mathbf{v}}_i^{(0)}), \tag{5.27}$$

where $\phi_e^{(0)}$ is the edge update function ϕ^e in the first GNN layer, we identify edges with a subscript for their source and destination nodes instead, and adopt numerical superscripts to denote how many times a feature has been embedded, rather than the less descriptive primed notation in equation 5.11. As before, creating a message to update node j may be done naïvely via a sum aggregation

$$\overline{\mathbf{e}}_i^{(1)} \stackrel{?}{=} \sum_{i \in \mathcal{N}[j]} \mathbf{e}_k^{(1)} = \sum_{i \in \mathcal{N}[j]} \phi_e^{(0)}(\hat{\mathbf{v}}_j^{(0)}, \hat{\mathbf{v}}_i^{(0)}), \tag{5.28}$$

If the particle which the node i represents were to undergo a collinear split, the resulting two nodes in the graph would have identical node features, since by definition collinear particles travel in the same direction. In the case of equation 5.28, this would mean in the collinear pair would produce twice the contribution in the message construction than prior to the split[8]. This is not collinear safe, as the downstream observables would be sensitive to collinear splits.

However, this can be resolved by considering the conservation of momentum. The sum of transverse momenta for the split particles must equal the transverse momentum of the original particle. Let us define the transverse momentum fraction $\omega_i^{(\mathcal{N}[j])}$ of the original particle before the split in the neighbourhood of node j:

$$\omega_i^{(\mathcal{N}[j])} := \frac{p_T^i}{\sum_{k \in \mathcal{N}[j]} p_T^k}. \tag{5.29}$$

If the node were to split into $i \to r, s$, then it is a natural consequence of momentum conservation that the fraction of their transverse momenta should be conserved, *ie.*

$$\omega_i^{(\mathcal{N}[j])} = \omega_r^{(\mathcal{N}[j])} + \omega_s^{(\mathcal{N}[j])}. \tag{5.30}$$

[8] We would essentially have two incoming edges with the same embeddings from the split nodes.

Therefore, if we define the aggregation function $\rho^{e\to v}$ as the sum of the incoming edge embedding vectors, weighted by the fractions $\omega_i^{(\mathcal{N}[j])}$, collinear splits would have no effect on the final message.

$$\bar{\mathbf{e}}_j^{(1)} = \rho^{e\to v}(\mathbf{E}_j^{(1)}) := \sum_{i\in\mathcal{N}[j]} \omega_i^{(\mathcal{N}[j])} \phi_e^{(0)}(\hat{\mathbf{v}}_j^{(0)}, \hat{\mathbf{v}}_i^{(0)}), \tag{5.31}$$

Applying this p_T based weighting to the aggregation function also enforces infrared safety. This is because, if a particle q is emitted by i in the soft limit, its corresponding momentum fraction $\omega_q^{(\mathcal{N}[j])}$ is zero, by definition, and therefore it has no impact on the message construction. Therefore, infrared or collinear splits have no impact on the subsequent node update.

When a GNN layer is formed with these constraints on the input node features and aggregation function, these may be stacked to form a deep GNN, and the IRC safety is preserved. This work therefore provides a vital tool in constructing IRC safe jet observables from GNNs, and has formed a foundation for our architectural design choices.

Part II

Free and open source software contributions

Chapter 6

Performance considerations for Python in HEP

For new researchers in the field of high energy physics phenomenology, there are an overwhelming number of data formats, and tools to generate data. In the author's experience, this data is complex and disjointed, with little in the way of easing in beginners. Programs like Pythia produce large records of rich data, which extensively detail the history of the simulation, and annotate the particles produced with detailed meta-information. However, the semantic meaning of this data must be understood during the analysis, and data structures to work with this are not provided (at least, not in a satisfactory Pythonic format). `pythia8.Particle` objects may be iterated over `pythia8.Event` container objects, and their properties are accessible via getter methods, but this is poorly suited to Python's vectorised no-loop conventions. These conventions are essential in Python.

In following chapters, we introduce the ecosystem of software packages written in support of the work in this thesis. This shows how design considerations are levied to produce powerful analysis frameworks.

However, in this chapter we simply ask whether Python has sufficient performance for our needs. The answer to this question is complex, and we touch on language implementation details to guide our decisions.

6.1 Is Python fast enough?

There is debate over whether or not Python is a "slow" language. Taken at face value, the answer is obvious: yes, Python is *very* slow. It's safe to say that Guido Van Rossum's original intent when writing the Python language was far from how it is being used today. Rossum viewed his language as a convenient second language for C

programmers, for throwaway tasks. It was also targeted for beginners or students, who may never be professional programmers. Indeed, to emphasise the playful nature of the language, Rossum named Python for *Monty Python's Flying Circus*, of which he was a fan[1]. Due to these goals, the Python interpreter performs a number of high level tasks to ensure that code executes without a high standard for rigour forced on the programmer.

One such feature of Python is *garbage collection*. Garbage collection is a memory management model which automatically allocates and de-allocates memory in the heap, by keeping track of *references* to the memory via bound variable names, items in collection data structures, *eg.* `lists`, `dictionaries`, *etc.* A "reference" is simply an integer address stored on the stack (usually written in hexadecimal) which refers to a location in memory where some data on the heap is stored. Even basic implementations of garbage collection are much too complex to explore in this work, and Python's implementation is highly advanced, and evolving (the garbage collector has received important updates in both `3.12` and `3.13` releases). Suffice to say, however, that when compared with the manual memory management of languages such as C, or the borrow checker / ownership model used by Rust, garbage collection adds significant overhead, and must be evaluated constantly at runtime[2]. Another feature which improves Python's ease-of-use, but limits its performance, is its dynamic typing, and type coercion system. Unlike C, which is statically typed, Python is permissive about which data types are bound to variable names. For instance,

```
spam = 7
spam = "hello"
```

is perfectly legal Python code, and will yield the expected result of the final value of `spam` being a `string` containing the characters "hello". In C, we would require explicit variable declaration with type information, and once declared this may not be mutated implicitly; that is

```
int spam = 7;
spam = "hello";
```

would result in a compiler error. Additionally, Python's standard mutable sequence type, the `list`, may contain heterogeneous data types. As such, each item must be a full object instance, which contains metadata regarding reference count, size, type, *etc.* Storing these objects directly in a `list` in contiguous fashion becomes impractical, as

[1] `foo` and `bar` are frequently used as metasyntactic variable names, *ie.* placeholders for variable names when a programmer cannot think of anything suitable. In honour of Python's namesake, a further meta-syntactic variable `spam` is often seen in Python code, adopting the name of a 1970 Monty Python sketch.

[2] This additionally makes Python a non-deterministic language.

the items would have different sizes in memory. In order to iterate through these objects, Python `lists` instead store references to the objects, rather than the objects themselves. These objects are not guaranteed to be local to one another, so there is an inherent cost in jumping between the various positions in memory to iterate through the sequence. In the special (but quite common) case of sequences with homogeneous data types – *eg.* a `list` of `integers` – the duplication of object metadata is unnecessary. All item sizes will be identical, precluding the need for Python's user-friendly compromises on efficiency. Python data scientists have many techniques for circumventing this inefficiency. At the heart of most of these approaches is a package called `numpy` (Harris et al., 2020).

Python's active community and easy-to-use packaging and distribution systems enables users to package routines and data structures under namespaces which other users may import and use easily. These may be written in Python itself, C, or indeed many other languages with the help of specialised build tools, *eg.* `f2py` for Fortran. As such, tremendous work has been done by Python programmers to expand the functionality of the language, either by porting efficient computational libraries compiled from Fortran / C to Python – *eg.* `BLAS` (Blackford et al., 2002) and `LAPACK` (Anderson et al., 1999) – or by writing packages in pure Python (often using these efficient compiled libraries as a computational back-end).

`numpy` is a package which was created to provide a homogeneous numeric sequence data structure in Python, called `ndarray`. The `nd` prefix refers to the fact that the array may be nested with sub-arrays to form arbitrary (but non-jagged) dimensionality. The underlying implementation is in C, and it provides a contiguous block of memory, storing each value more efficiently. Additionally, `numpy` and its sibling project SciPy provide routines which are written in a blend of C, C++, Fortran, and Python, to iterate and perform vectorised calculations over the data stored in these arrays. In this way, the `numpy` vectorised approach trades memory for compute speed, in that iterating over data in Python can be done in speeds competitive with C, but this data must first be stored eagerly[3] in an `ndarray`.

Packages such as `pandas`[4] (pandas development team, 2020; Wes McKinney, 2010) further extends this functionality to heterogeneous columnar data by introducing a `DataFrame` data structure, which wraps `ndarray` instances as columns of a table, and each column may have a distinct type. Hence, utilising compiled libraries from high performance languages boosts Python's speed, at the cost of memory consumption. In this sense, Python is arguably a fast language. However, we may circumvent even this limitation, by instead compiling high performance machine code at runtime or

[3] Eager computations are performed as soon as they are defined, as opposed to lazy computations. This is useful when intermediate data is needed for IO.

[4] Polars is a recent, more performant competitor, written in Rust and ported to Python.

just-in-time (JIT), rather than ahead-of-time (AOT). Before we can consider the difference between JIT and AOT compilation, first let's consider what a compiler does.

6.2 But what is a compiler?

A compiler is a program which converts a plaintext, human-readable file of source code into a binary executable of machine-code which may be run by a computer. However, compilers and programming languages were not created at the advent of computers. In the time of John Backus's team, writing the Fortran specification and compiler in 1953, assembly language was a novelty innovation beyond hand-writing machine code, and programmers were expected to uniquely write programs for the specific machine they were operating with. Fortran was the first Universal language, in that compilers could be written for specific machines to take generic text-based Fortran code and convert them to machine code for those machines. This added layer of abstraction meant that Fortran was the first *portable, high level* language. Crucially, Backus demonstrated that the compiler could optimise the code such that it ran at competitive speeds to hand-crafted machine code or assembly, written by experts for specific machines, leading to widespread adoption.

Compilers continued to develop through the years, leading to greater portability of programs, and better optimisation routines. Today, the level of optimisations is highly sophisticated, and compilers can replace large globs of code with more efficient alternatives. For instance the following C function

Listing 6.1 Algorithmic C implementation computing the sum to n of the natural numbers.

```c
int sum_to_n(int n) {
    int total = 0;
    for (int i = 1; i < n + 1; i++) {
        total += i;
    }
    return total;
}
```

when compiled with either `clang` or `gcc`, could be expected to produce an executable whose execution time scales linearly with `n`. Indeed this is seen when optimisations are switched off. However, with optimisations on, both compilers produce a program which runs in constant time, no matter what value of `n` is passed. How is this possible? Clearly higher values of `n` would invoke more loop iterations. Not so. Modern compilers recognise routines which sum natural numbers, even raised to arbitrary powers, *eg.* sum of squares, cubes, *etc.* Having detected this, the algorithmic solution

is replaced with the analytical formula equivalent, such that the above code listing is compiled with the replacement

Listing 6.2 Analytical formula C implementation of the sum to n formula.

```c
int sum_to_n(int n) {
    return 0.5 * (n * (n + 1));
}
```

leading to the highly efficient constant-time performance.

Sophisticated optimisations such as these further lower the barrier for entry of programmers to produce highly efficient code.

Even more excitingly, writing languages has been made simpler by an additional layer of abstraction, on top of compilers. This was introduced with the release of Low Level Virtual Machine (LLVM) by Chris Lattner in 2003. Rather than having to write compilers for each individual chip architecture for any given language, LLVM enables authors of programming languages to simply convert source code to a target *Intermediate Representation* (IR) of the code. IR is agnostic to chip architecture, and code written in it may then undergo compiler optimisations before being converted to machine code for the target architecture. That is, languages which have substantially different syntax and design goals, *eg.* C and Python, could theoretically compile to the same IR, and thus benefit from the same terrific optimised machine-code speeds. The upshot of this is that creating a compiler for a language is dramatically simplified to producing IR which can be handed off to LLVM, which will then offer the same performance for all languages with equivalent IR.

Readers may protest that Python is an interpreted language, not compiled. Interpreted languages don't compile to binary executables to be run afterwards. Instead, the source code is passed to an interpreter, which converts the plaintext line-by-line into bytecode, and this is then run as a machine-code equivalent. This is another source of computational overhead, slowing down Python programs. The CPython implementation of Python (that is, the Python interpreter which is written in C) provides some optimisation for this by caching the bytecode, which prevents this step from needing repetition between running the same source code twice. However, LLVM may be used with Python by applying JIT compilers. The most popular of these is `numba`.

`numba` offers a decorator to modify Python functions inplace. The function name is then attached not to the original Python source code, but instead to a compiled version which is stored in memory. The decorator provided is called `numba.njit`, *ie.* "no-Python JIT compile", and we can reproduce the same function as that produced by listing 6.1 with compiler optimisations turned on.

Listing 6.3 Just-in-time compiled version of a Python algorithm, computing the sum to n for the natural numbers.

```
import numba as nb

@nb.njit
def sum_to_n(n: int) -> int:
    total = 0
    for i in range(1, n + 1):
        total += i
    return total
```

This is because the function *is* being compiled with the same compiler back-end (as `clang`; `gcc` does not use an LLVM back-end). Hence the performance is virtually the same as compiled C code! It also benefits from the same compiler-level optimisations, so that this algorithm is also constant time, rather than linear as the source code would suggest.

Listing 6.4 Benchmarking the JIT compiled sum of natural numbers defined in listing 6.3.

```
>>> @nb.njit
... def sum_to_n(n: int) -> int:
...     total = 0
...     for i in range(1, n + 1):
...         total += i
...     return total
...
>>> sum_to_n(100)
5050
>>> %%timeit
... sum_to_n(100)
116 ns ± 0.462 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
>>> %%timeit
... sum_to_n(1000)
117 ns ± 0.0816 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
>>> %%timeit
... sum_to_n(10000)
118 ns ± 0.159 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

Additionally, notice that by using JIT compilation, there was no need to allocate an `ndarray` to hold the intermediate values to be summed. This would have been necessary if using `numpy`, *eg.* see listing 6.5.

Passing no parameters into the `nb.njit` decorator results in this compilation

Listing 6.5 `numpy` solution for a Python algorithm to compute the sum to n for the natural numbers. The `.item()` method is used at the end to convert the final result from a `np.int32` object into a Python-native `int`.

```python
import numpy as np

def sum_to_n(n: int) -> int:
    values = np.arange(1, n + 1)  # an array of memory int * n must be stored
    return values.sum().item()  # this is then reduced to form the final sum
```

happening *lazily*[5], *ie.* the function compiles the first time it is called, automatically detecting the types of variables which are passed to it. If the function is called again with different types for the arguments, it compiles again, and the function is overloaded, dispatching the correct compiled function depending on what is passed in. This allows `numba` to circumvent Python's dynamic type system, as well as its interpreter overhead, and the memory consumption of `numpy`.

By using a considered blend of efficient Python primitives, data science libraries such as `numpy` and `pandas`, and LLVM-based JIT compilers such as `numba`, Python programs can be written with competitive performance with languages such as C or Fortran.

[5]Lazy computation refers to operations which are "put off", *ie.* performed only when the result is needed. More generally, such techniques are ubiquitous in functional programming, as chaining lazy operations on input data is equivalent to function composition.

Chapter 7

Semantic heterogeneous data structures with graphicle

In this chapter we explore how to deal with data records whose fields have different types in memory and meaning. We explore how these can be handled in heterogeneous data structures using established packages in the Python data ecosystem, and where these methods fall short. We then discuss the importance of meaning and derived properties from data, *ie.* its semantic interpretation. We explore the current offering in the Python HEP ecosystem, and where these fail to meet our needs. We finally introduce our package for processing HEP data with heterogeneous and semantic data structures, bringing the power of relational database style querying to analysing collision events.

7.1 Limitations in the current tools

We touched upon `pandas` in this previous chapter, which offers a `DataFrame` type with the ability to store data in a tabular structure. Columns in a `DataFrame` store contiguous blocks of memory with homogeneous data types. However, separate columns may have different data types, and aren't necessarily contiguous in memory. A `DataFrame` is similar in principle to a dictionary of `ndarray` objects. This comes with a great deal of high level routines, both provided by `pandas` itself, and via excellent reverse compatibility with `numpy`. However, these routines are not *semantic*, in the sense that they do not relate to the particle physics interpretations of the data, nor the topological structure of the directed acyclic graph reflecting the generation process of the data. This makes it difficult for HEP newcomers to work with data out-of-the-box, as they must manipulate the data at a low level.

Manipulating HEP data involves cross-referencing the units for dimensionful quantities associated with the particles, such as charge and 4-momentum. Additionally, the component ordering of 4-momentum must be explicitly enforced, with conversions as necessary. Status codes provide valuable data for the role a given particle plays in the data generation process, allowing precise pinpointing of important regions of the event; this, too, must be compared against numerical values provided in the manual for the event generator used. In Pythia, helicity / spin polarisation is represented with values in the range $[-1, +1]$ for spin up / down, and a sentinel value of 9 for missing data. Even more idiosyncratic is Pythia's handling of colour flow information, with colours being any unique pair of numbers, both starting from 500. Colour triplets have a single code, with the other number being 0, whereas colour octets have two unique colour codes. Ancestry can be inferred for any given particle by using the `.motherList()` method, to get a `list` of all the parent particles ID codes. If these are cross referenced over the whole event, a directed acyclic graph representing the event generation can be reconstructed. Suffice to say, this is a lot of analysis to be reproduced time and again, and there are likely many convenience calculations which could be shared between researchers.

The community has attempted to address this via Scikit-HEP (Rodrigues et al., 2020). Scikit-HEP is a meta-package, or collection of packages, written for Python which aims to provide utilities for the HEP community.

> The Scikit-HEP project (HEP stands for High Energy Physics, see more in the FAQ) is a community-driven and community-oriented project with the aim of providing Particle Physics at large with an ecosystem for data analysis in Python.
>
> – Scikit-HEP's official website (`https://scikit-hep.org/about`), 2025

Scikit-HEP's `vector` is one such package which encapsulates `numpy` arrays of 4-momentum data, and provides many convenience functions (Chopra et al., 2025). These include component conversions, shifts of reference frames using Lorentz boosts, rotations, and pairwise distance calculations. This is just one offering of Scikit-HEP, which has packages for more than just 4-momentum manipulation.

Scikit-HEP's `particle` package (Rodrigues and Schreiner) provides an object oriented approach to querying PDG (Navas et al., 2024) codes, which are the unique ID codes for both experimentally observed and theoretical particles in HEP. Particles of the same species share a number of properties, including charge, quark composition, quantum numbers, *etc.* Using Particle objects, it is possible to retrieve these properties based on PDG code, by performing a simple CSV lookup in the back-end. However, if a user wishes to convert a `numpy` array of PDG codes to a corresponding array of particle properties, the vectorisation must be done in Python. This fragments the

interfaces between `vector` and `particle`, and may have negative performance implications, due to the Python native iteration overhead discussed earlier.

While these packages provide semantic data structures to handle the low-level data, they do not form a heterogeneous whole data structure over all particle properties in the event record. In our view, this is essential to ease data manipulation. By combining this semantic data with the power of relational databases, we may filter over particle properties such as momentum components (either raw or derived), PDG properties, status codes highlighting regions of the event space, colour codes, and ancestry by traversing the DAG structure of the generation. This makes it possible to aggregate, visualise, and perform more fine-tuned analyses over the particles by drawing together multiple records describing their properties.

Since the interface design varies between packages in Scikit-HEP's ecosystem, the experience of moving between packages can at times feel inconsistent. For instance, while `vector` (as its name suggested) provides vectorised operations over `numpy` arrays, `particle` is object oriented, and glue code is needed to efficiently convert an array of PDG codes into a corresponding array of particle properties from the PDG table lookup. As such, a substantial ambition of this project was to create an ecosystem of cross-compatible packages which could fill these niches[1].

7.2 Graphicle for semantic and relational databases

The workhorse which made event analysis simple in this research project is `graphicle` (Chaplais and Cerro, 2025). `graphicle`, like `numpy`, `pandas`, and the like has two primary offerings: data structures, and optimised computations. Unlike these numerical libraries, however, `graphicle`'s use case is much more narrowly defined. Where `numpy` provides a flexible, agnostic data structure in `ndarray`, which is capable of storing data in a variety of types – even allowing users to choose between little and big Endian formats[2] – `graphicle` has the luxury narrowly defined use-cases.

`graphicle` operates over the data made available by MCEGs. In its current implementation, this is specifically tailored towards Pythia, whose particle properties are available via the getter methods shown in table 8.1 in chapter 8.

Many derived properties may be formed from this record, and there is redundancy in the getter methods provided. Kinematic variables, such as `pT()`, `pAbs()`, `theta()`, `m2()`, `mT()`, `phi()`, `y()`, `eta()`, and `tau()` can all just be viewed as components or derived quantities of the four-momentum. Therefore we discard these from `pythia8`'s output,

[1] It would be exciting in future to contribute functionality from my packages to strengthen the Scikit-HEP ecosystem overall.

[2] *ie.* whether bits for elements are stored right-to-left or left-to-right, respectively. I can't imagine a use-case for this, but that speaks to the broad church of `numpy`.

and simply store the `px()`, `py()`, `pz()`, and `e()` components in a data structure wrapping $N \times 4$ array[3], where N is the number of particles in an event. This is one of several constituent data structures.

7.2.1 Data structures: components and composites

`graphicle` contains a number of component data structures. These are designed to be composed together into composite structures, which will give us that relational database power that we wish to create.

Table 7.1: Table of contents for `graphicle`'s `data` module, providing all of the data structures.

Name	Used for	Composite?
`MomentumArray`	Four-momentum manipulation	No
`PdgArray`	PDG code querying	No
`StatusArray`	Status code querying	No
`ColorArray`	Colour code encapsulation	No
`HelicityArray`	Helicity encapsulation	No
`MaskArray`	Data filtering	No
`AdjacencyList`	Graph connectivity	No
`MaskGroup`	Combined data filtering	Yes
`ParticleSet`	Heterogeneous HEP data manipulation	Yes
`Graphicle`	Graph representation of heterogeneous HEP data	Yes

Table 7.1 shows all data structures provided by `graphicle`. Let's explore the first of these, the `MomentumArray`.

`MomentumArray` is `graphicle`'s offering which most directly competes with the `vector` library. Consider the interaction $g\ b \to t\ W^-$ (see figure 9.4a). Here, the W^- is forced to decay leptonically, and the t decays via $t \to b\ W^+$, where the W^+ is forced to decay hadronically. If we take the `MomentumArray` representing the descendants of the t quark, the string representation is given as (with units in GeV):

```
>>> pmu
MomentumArray([[-1.90988416e+00 -2.08095918e+00  2.67532643e-01  2.84061650e+00]
               [-1.41892981e+01 -1.46714989e+01  8.12430669e-01  2.04326411e+01]
               [-4.15116398e-02  3.60800480e-02  1.94391688e-02  1.51270130e-01]
               [-8.44262475e-01  1.84254494e+00 -1.26666802e+00  2.39409245e+00]
```

[3]Note: we are using C style ordering of elements, rather than Fortan-style. That is, each row has 4 columns of contiguously stored double precision floating point elements.

```
               [-1.37997801e-01 -3.43958973e-01 -6.37885172e-01  7.50818518e-01]
               [-4.02493216e+00  4.42015152e+00 -1.14437913e+01  1.29119192e+01]
               [-4.74430344e+00  4.75247002e+00 -1.15878903e+01  1.34259592e+01]
               [-1.90057890e+01  1.95837893e+01 -4.90489486e+01  5.61375501e+01]
               [-1.27303878e+00 -2.09602321e+00 -7.37363827e-01  2.56459085e+00]
               [-1.12919800e+00 -1.43683279e+00  1.11655574e-03  1.83277320e+00]
               [ 9.04198168e-02  3.35037021e+00 -2.87754592e+01  2.89703242e+01]
               [-3.30682373e-01 -4.72901704e-01  1.66710535e-02  5.93923027e-01]
               [-2.93236757e+00 -2.99024210e+00  5.46341788e-02  4.19079850e+00]
               [ 2.06850232e-01  8.75941985e-01 -1.02024517e+01  1.02420742e+01]
               [ 9.58677284e-03  2.96448872e-01 -3.98256939e+00  3.99359898e+00]
               [ 2.56434723e-01  8.31808059e+00 -7.23750308e+01  7.28519135e+01]
               [ 2.23461504e+00  6.79048477e+00 -6.44295947e+01  6.48250571e+01]
               [-8.47364580e-01  6.34245996e+00 -6.01502031e+01  6.04916150e+01]
               [ 5.07499721e-02  9.16407184e-01 -4.89153873e+00  4.97885615e+00]
               [-2.19343857e-01  1.17296156e+00 -1.23581089e+01  1.24163717e+01]],
              dtype=[('x', '<f8'), ('y', '<f8'), ('z', '<f8'), ('e', '<f8')])
>>> len(pmu)
20
```

This looks very similar to the structured `ndarray` which it wraps, giving the named fields in the `dtype` keyword argument. In `MomentumArray` instances, each row here is considered a single element, so this has a length of 20. The transverse momentum and pseudorapidity are accessible via properties of the `MomentumArray` instance:

```
>>> pmu.pt  # units of GeV
array([ 2.82454397, 20.41051346,  0.05499987,  2.02675874,  0.37060919,
        5.9781116 ,  6.71523539, 27.29001314,  2.45233379,  1.82745084,
        3.35159011,  0.57705013,  4.18811741,  0.9000341 ,  0.29660384,
        8.32203241,  7.14871931,  6.39881435,  0.91781136,  1.19329399])
>>> pmu.eta  # units of GeV
array([ 9.45760563e-02,  3.97940153e-02,  3.46466913e-01, -5.90120175e-01,
       -1.31150974e+00, -1.40463063e+00, -1.31373396e+00, -1.34913958e+00,
       -2.96322786e-01,  6.10990809e-04, -2.84660962e+00,  2.88861131e-02,
        1.30446747e-02, -3.12303779e+00, -3.29181603e+00, -2.85939115e+00,
       -2.89485061e+00, -2.93669656e+00, -2.37510501e+00, -3.03306490e+00])
```

7.2.2 Masking case study: momentum and PDG codes

As discussed in chapter 2.5, we will assume in this work that a rough approximation to the detector response can be made by applying cuts of $p_T > 0.5$ GeV and $|\eta| < 2.5$.

We can apply these cuts by `numpy`-style subscripting with boolean arrays:

```
>>> pmu_cut = pmu[np.abs(pmu.eta) < 2.5]
>>> len(pmu_cut)
13
>>> pmu_cut = pmu_cut[pmu_cut.pt > 0.5]
>>> len(pmu_cut)
11
```

So, the first cut of $|\eta| < 2.5$ eliminates 7 particles, and the following cut of $p_T > 0.5$ GeV eliminates two more. The resulting `MomentumArray` can be passed to the `numpy` Universal Function (UFunc) "sum", with an axis of 0 to indicate we wish to sum over the rows (`numpy` views the `MomentumArray` as an unstructured $N \times 4$ `ndarray`). We can then find the mass by accessing the `mass` property of the resulting `MomentumArray`, containing one element, which is the combined sum of all of the t quark constituents. Comparing before and after the cuts are applied yields:

```
>>> np.sum(pmu, axis=0).mass  # units in GeV
array([170.81136533])
>>> np.sum(pmu_cut, axis=0).mass
array([79.42239856])
```

So in this particular instance, applying the detector-level cuts has caused the mass calculation to suffer substantially. In fact, it's likely here that, since the cut mass is so close to the known mass of the W^+ boson, the b quark was not within the detector's $|\eta| < 2.5$ window, and so only the W^+ constituents remained. In any case, it was possible to use the derived properties of the raw four-momenta provided by `MomentumArray` to filter the momentum data.

We can apply these same masks to the next data structure in the list, the `PdgArray`. The `Scikit-HEP` equivalent offering for `PdgArray` is given by the `particle` package (Rodrigues and Schreiner). In fact, the underlying data for `PdgArray` comes from the CSV tables provided by `particle` itself. `PdgArrays` provide particle properties by species via instance attributes, *eg.* `name`, `charge`, *etc.*

```
>>> pdg
PdgArray([ -211   321  -211   211  -211   211  2112 -2212   211  -211   211   211
           -211    22    22   -14    13  -321   211   211],
         dtype=int32)
>>> pdg.name
array(['pi-', 'K+', 'pi-', 'pi+', 'pi-', 'pi+', 'n', 'p~', 'pi+', 'pi-',
```

```
       'pi+', 'pi+', 'pi-', 'gamma', 'gamma', 'nu(mu)~', 'mu-', 'K-',
       'pi+', 'pi+'], dtype='<U7')
>>> pdg.charge
array([-1.,  1., -1.,  1., -1.,  1.,  0., -1.,  1., -1.,  1.,  1., -1.,
        0.,  0.,  0., -1., -1.,  1.,  1.])
```

This can also be filtered on the momentum data to crudely simulate the CMS detectors. The previous method of applying cuts sequentially to the variable `pmu_cut` has several drawbacks. One of which is the fact that it's hard to repeat, as the data is being changed inplace. Using the same variable name with data at different locations in the same source code – excluding use for iteration or accumulation variables – is called variable shadowing, and is considered a programming anti-pattern, for the reason shown above and many others. It would be much better to have a single filter which could be applied and reused for `pmu`, `pdg`, and any other data pertaining to the particles we are considering. Fortunately, we can combine the filters using a logical `AND` operation between the η and p_T masks.

```
>>> cuts = np.logical_and(np.abs(pmu.eta) < 2.5, pmu.pt > 0.5)
... pdg_cut = pdg[cuts]
...
>>> len(pdg_cut)
11
>>> pdg_cut.quarks
array(['Ud', 'uS', 'uD', 'uD', 'udd', 'UUD', 'uD', 'Ud', 'uD', 'Ud', 'uD'],
      dtype='<U3')
```

The `cuts` mask can now be used on any data structure representing the same particle properties.

Except this still has drawbacks. One immediate drawback of using logical `AND` or `OR` operations to reduce boolean masks is that it is not possible to introspect how the final result was produced. How many masks were combined? Which of them was more permissive / restrictive? What was their semantic meaning? For example, what kinematic filters were applied, and were other particle properties such as charge, status code, or colour taken into account? None of this information can be encoded and retrieved from a single boolean `ndarray`. Nor is it convenient, scalable, or computationally efficient to keep several descriptively named masks in scope to be reduced every time a combined mask is required.

```
>>> pdg_cut = pdg[np.logical_and(np.abs(pmu.eta) < 2.5, pmu.pt > 0.5)]  # No!
>>> pmu_cut = pmu[np.logical_and(np.abs(pmu.eta) < 2.5, pmu.pt > 0.5)]  # Awful.
```

It is here we introduce the concrete implementations of `graphicle`'s `ABC` called `MaskBase`. In order to motivate why `MaskBase` was defined, and the purpose of its two concrete implementations – `MaskArray` and `MaskGroup` – we must first explain the Composite Pattern (Gamma et al., 1994; Freeman et al., 2004).

7.2.3 The Composite pattern for `MaskGroups`

The Composite Pattern is an Object Oriented Programming (OOP) structural design pattern, see chapter 7.2.4. In generic terms, it defines an `ABC` which is called *Component*. Concrete implementations of Component are *Leaf* and *Composite*. By sharing a uniform interface enforced by the Component `ABC`, Leaf and Composite instances may be treated identically by client code.

To illustrate, consider a digital sketching tool. An image may be created by overlaying shapes, curves, line segments, *etc.* which we consider to be the leaves. However, any subset or composite of the objects comprising the whole image may be considered an image in its own right, including individual parts. The largest composite is, of course, the whole image. However, seasoned users of tools like GIMP (The GIMP Development Team, 2025) routinely form persistent composites of these leaves in groups. Leaves within these groups have their relative positions, size, and aspect ratio locked relative to each other, and thus behave as a single element rather than a collection. Additionally, these groups may be further included in higher level groups which include other composites or leaf elements. This structural hierarchy gives control to the image creator to name, manipulate, and export meaningful portions of the image different organisational levels. This tree-like, hierarchical organisation is the hallmark of the Composite Pattern.

Returning to our `MaskArray` and `MaskGroup` data structures, we can use these concepts to a structured approach to querying our data sets. Here, `MaskArray` is the leaf element, `MaskGroup` is a composite, and they share `MaskBase` as the component interface unifying the two. `MaskGroup` is a `mapping` with `string` keys, and values which are instances of `MaskBase`, *ie.* another `MaskGroup` or a `MaskArray`. The common behaviour between both `MaskArray` and `MaskGroup` is that instances of either can be passed as a subscript to `graphicle` data structures (and, in fact, `numpy` arrays - try it!) as masks. `MaskGroups` contain an `agg_op` parameter which is an `enum` with possible members `AND`, `OR`, and `NONE`. This tells the `MaskGroup` how to reduce the masks contained within it. The names of the members are self-explanatory, with the caveat that `NONE` prevents any reduction occurring, effectively breaking the ability to use it as a subscript. Hence, this `agg_op` is seldom used.

So, our previous example can be improved by organising the η and p_T cuts in a `MaskGroup`.

Listing 7.1 Demonstrating how kinematic cuts can be combined in a composite `MaskGroup` structure. Notice that the leaf masks can always be retrieved by subscripting the `MaskGroup` with their `string` keys.

```
>>> cuts = gcl.MaskGroup(  # a kinematic mask
...     {"eta": np.abs(pmu.eta) < 2.5, "pt": pmu.pt > 0.5},
...     agg_op="AND"
... )
...
>>> cuts  # showing the MaskGroup repr
MaskGroup(masks=["eta", "pt"], agg_op=AND)
>>> pdg_cut = pdg[cuts]
... pmu_cut = pmu[cuts]
>>> len(pmu_cut)
11
>>> len(pdg_cut)  # same particles for pdg and pmu
11
>>> pts = pmu_cut.pt.tolist()
... names = pdg_cut.name.tolist()
...
>>> for name, pt in zip(names, pts):
...     print(f"{name=}, {pt=:.3e} GeV")
...
name='pi-', pt=2.825e+00 GeV
name='K+', pt=2.041e+01 GeV
name='pi+', pt=2.027e+00 GeV
name='pi+', pt=5.978e+00 GeV
name='n', pt=6.715e+00 GeV
name='p~', pt=2.729e+01 GeV
name='pi+', pt=2.452e+00 GeV
name='pi-', pt=1.827e+00 GeV
name='pi+', pt=5.771e-01 GeV
name='pi-', pt=4.188e+00 GeV
name='pi+', pt=9.178e-01 GeV
>>> len(pdg[cuts["eta"]])  # apply individual mask
13
```

In listing 7.1 we combine both η and p_T kinematic cuts into a single `MaskGroup` object. This acts as a container, and the individual masks (automatically cast from boolean `ndarray` objects into `MaskArrays`) are accessible via Python's standard `mapping` interface. Just as in the Composite Pattern example explained earlier, though, `MaskGroups` can be nested.

For instance, in listing 7.2, we create a mask called `"leptons"` to identify the μ^- and $\bar{\nu}_\mu$ leptons in our original cluster. Using `MaskGroups`, we can make this a sibling in our hierarchy to the original kinematic cuts, which we call `"kinematic"`. We can also set the `agg_op` to `OR`, which has the effect of applying our original kinematic cuts, and then

adding the leptons back in (which we can see from listing 7.1 has filtered out). Using this method, we can investigate how much of a difference these leptonic detector-level particles have on the final mass calculation. Printing out the `MaskGroup`, the nested tree structure is apparent.

Listing 7.2 Demonstrating nesting capabilities by creating a `MaskGroup` with a lepton whitelisting mask, combined with a `MaskGroup` of p_T and η kinematic cuts. The string representation of `MaskGroup` displays the tree structure of the hierarchy.

```
>>> # names -> PDG codes to find leptons
... dict(zip(pdg.name.tolist(), pdg.serialize()))
{'pi-': -211,
 'K+': 321,
 'pi+': 211,
 'n': 2112,
 'p~': -2212,
 'gamma': 22,
 'nu(mu)~': -14,
 'mu-': 13,
 'K-': -321}
>>> mask = gcl.MaskGroup(  # leptons are -14 and 13
...     {
...         "leptons": np.isin(np.abs(pdg), [13, 14]),
...         "kinematic": cuts  # nest MaskGroup
...     },
...     agg_op="OR"  # OR => leptons added after cuts
... )
>>> print(mask)
MaskGroup(agg_op=OR)
|-- leptons
`-- kinematic
   |-- eta
   `-- pt
>>> pmu_sum = np.sum(pmu[mask], axis=0)
... pmu_sum
MomentumArray([[ -47.84195626   22.87547043 -214.62844125  259.98069093]],
              dtype=[('x', '<f8'), ('y', '<f8'), ('z', '<f8'), ('e', '<f8')])
>>> pmu_sum.mass
array([136.79346453])
```

Note from listing 7.2 that `.serialize()` is `graphicle`'s equivalent of `ndarray.tolist()`. For non-composite data structures `.serialize()` will output a `list`, and composite objects output `dictionaries`. By converting to pure-Python data structures, users may then easily use libraries of their choice, such as the standard module [`json`], to encode the data for plaintext IO. See listing 7.3 for an example of this.

Listing 7.3 Plaintext IO example using the standard [`json`] library in tandem with `MaskGroup`'s `.serialize()` method. Other `graphicle` objects may be stored and retrieved in identical fashion.

```
>>> import json
...
... with open("maskgroup.json", "wt") as f:
...     json.dump(mask.serialize(), f)
...
... with open("maskgroup.json", "rt") as f:
...     mask_read = gcl.MaskGroup(json.load(f), agg_op="OR")
...
... # recursively check mask data and hierarchy structure are unchanged:
... mask_read.equal_to(mask)
...
True
```

7.2.4 Programming paradigms and their role in `graphicle`

Here we take a brief sidestep from our demonstration of `graphicle`'s data structures to discuss the software design and engineering considerations which underpin them. The design and implementation of `graphicle`'s data structures and algorithms are deeply rooted in two complementary programming paradigms: object-oriented programming (OOP) and functional programming (FP). These paradigms are leveraged to achieve a balance between modularity, reusability, and robustness, enabling the creation of high-performance tools tailored for particle physics data analysis.

7.2.4.1 Object-oriented programming

OOP focuses on encapsulating data within objects and associating methods (functions) that operate on that data. In Python, this paradigm is implemented through the use of classes, which serve as blueprints for creating objects. OOP excels in scenarios where data persistence, modular design, and intuitive interfaces are paramount.

In `graphicle`, OOP is predominantly employed for structuring data types. For example, the `MomentumArray` class provides a wrapper around $N \times 4$ [`numpy.ndarray`] instances, representing particle momenta. By defining an abstract base class (ABC) called `ArrayBase`, `graphicle` ensures that all concrete implementations (e.g., `MomentumArray`, `PdgArray`) adhere to a consistent interface. This interface supports critical operations such as iteration, equality checks, and integration with external libraries like `numpy`.

The choice to inherit from `collections.abc.Sequence` underscores the principle of "composition over inheritance." This design philosophy minimises tightly coupled

code while leveraging Python's standard library to ensure that `graphicle`'s data structures integrate seamlessly with the broader Python ecosystem. For example:

```
>>> import collections.abc as cla
>>> isinstance([1, 2, 3], cla.Sequence)
True
>>> isinstance(graphicle.MomentumArray.from_spherical_uniform(5, 10.0), cla.Sequence)
True
```

By aligning with Pythonic conventions, `graphicle` achieves a high degree of usability and compatibility, making it a natural fit for scientific computing tasks.

7.2.4.2 Functional programming

FP emphasises immutability and the use of pure functions – those that produce outputs solely based on their inputs without side effects. This paradigm is particularly advantageous for implementing computational routines, as it promotes clarity and reduces the likelihood of bugs stemming from unintended state changes.

In `graphicle`, FP principles are applied to develop robust algorithms that manipulate data without modifying the underlying structures. For example, filtering operations on particle data are performed by creating new `MaskArray` instances rather than altering the original data. This approach ensures data integrity and facilitates reproducibility, which are crucial for scientific research.

7.2.4.3 Polymorphism and reusability

Polymorphism, a cornerstone of OOP, allows different data structures to share a common interface, enabling flexible and reusable code. In `graphicle`, polymorphism is realised through ABCs like `ArrayBase`. For instance, the `MomentumArray` and `PdgArray` classes both implement methods required by the `ArrayBase` interface, allowing them to be used interchangeably in higher-level operations.

```
>>> pmu = graphicle.MomentumArray.from_cartesian(px=[1.0], py=[0.0], pz=[0.0], e=[1.0])
>>> isinstance(pmu, graphicle.base.ArrayBase)
True
```

By restricting inheritance to abstract classes, `graphicle` avoids the pitfalls of tightly coupled hierarchies. This approach is consistent with the "composition over inheritance" principle advocated by the Gang of Four design patterns, ensuring flexibility and maintainability in the library's architecture.

> "Object composition is defined dynamically at run-time through objects acquiring references to other objects. Composition requires objects to respect each others' interfaces, which in turn requires carefully designed interfaces that don't stop you from using one object with many others."
>
> – Gamma et al. (1994)

7.2.4.4 Blending paradigms for optimal design

The interplay of OOP and FP in `graphicle` exemplifies a thoughtful synthesis of these paradigms. While OOP provides the structural backbone for data representation, FP ensures that operations on this data remain predictable and robust. This dual approach empowers researchers to focus on extracting insights from particle physics data without being encumbered by implementation details.

By adhering to these paradigms, `graphicle` not only achieves technical excellence but also serves as a model for designing scientific software that is both powerful and user-friendly.

7.2.5 Topological information with AdjacencyList

Analysis with `graphicle` differs from other approaches in the central importance it places on representing collision events as graphs. We use MCEGs to simulate the history of successive decays in particle collision events. As we have mentioned, this ancestry can be represented topologically as a Directed Acyclic Graph (DAG). In our work, we represent the particles as edges in this graph, and the interactions between particles as nodes, similar to the approach used for Feynman diagrams. We encode this data in an `AdjacencyList`[4].

The data exposed by the public interface is a structured `ndarray` with two fields: `src` and `dst`. The source vertex (`src`) is the interaction from which the particle was produced[5], and the destination vertex (`dst`) is the next interaction the particle participates in. This kind of graph data representation is called the Coordinate format (COO).

However, despite the public interface, `AdjacencyList` more than a wrapper around `ndarray`. The choice to expose and instantiate from `ndarray` is to enable a consistent experience with `graphicle`'s `ArrayBase` implementations. Despite this, most routines associated with `AdjacencyList` make use of its underlying `scipy.sparse` data, in the

[4] This is actually a misnomer. In chapter 5.2.2 we defined adjacency list and edge list representations of graphs. `AdjacencyList` as `graphicle` defines it is actually an edge list as we define it here.

[5] Produced is a flexible term here. If a particle radiates a photon or gluon, but its species remains unchanged, we refer to the incoming particle as having been destroyed, and two new particles are produced.

form of private `coo_array` or `csr_array` attributes. Both of these `scipy.sparse` arrays are computed lazily, but then cached for re-use to maximise efficiency.

Using `scipy.sparse` offloads expensive sparse matrix and graph traversal routines to compiled and optimised C++. `graphicle` makes extensive use of graph traversal with Breadth First Search, provided by `scipy.sparse.csgraph.breadth_first_tree()`. `graphicle` wraps this routine with [`graphicle.select.vertex_descendant()`], which takes an `AdjacencyList` and the `integer` index of one vertex expressed within its COO formatted list, and outputs a `MaskArray` over the whole event, identifying the topological descendants of the vertex. This `MaskArray` can be used in tandem with other particle properties to refine and aggregate properties of the event space rooted on the DAG of a specific ancestor. In chapter 9.1 we explore how this may be used to reconstruct the mass of particles from the hard process.

`AdjacencyList` also exposes `MaskArray` instances as properties to select the roots and leaves of the DAG it represents. This can be especially useful if the `final` `MaskArray` identifying the remaining particles is missing (see chapter 7.2.6), or if showering / hadronisation are not completed during the simulation. In both cases, the `.leaves` property is useful. In the former case, it would be identical to `final`; in the latter, `final` would be `False` throughout, so `.leaves` would identify the last particles produced in the simulation.

7.2.6 Heterogeneous data composites

So far we have seen an example of a homogeneous composite data structure: the `MaskGroup`. It's homogeneous in the sense that the actual data being held is all just sequences of boolean data. We have then used our masks to filter over `graphicle`s sequential data structures individually. This is an awkward setup.

It is often necessary to create algorithms which use multiple particle properties in tandem. Explicitly masking each data structure, requires the programmer to manually keep track of all of them. This is a burden, and is likely to introduce mistakes[6], or friction to collaboration. An ad-hoc solution might be to organise them into some container type, like a `list`, `tuple`, or `dictionary`. However, this would require iterating over the items in the container in order to apply masks over the whole data set. Importantly, such a generic approach would also miss the opportunity to create specialised routines, made possible by having access to a rich variety of particle properties simultaneously. This is where the `ParticleSet` composite type comes in.

[6]The choice of isolating event regions in `graphicle` using boolean masks offers some protection: if a mask is passed into a dataset referring to different particles, it is unlikely that the error will silently pass, since the number of particles in the data structure and the mask must match exactly. Hence, this usually throws a `IndexError`, and can be caught.

`ParticleSet` is a data structure which takes 6 optional parameters in its initialiser, see listing 7.4. These are the first 6 data structures in table 7.1 – all of which implement `ArrayBase` – including a `MaskArray` passed by keyword as `final`, which identifies the final particles at the end of the simulation[7]. When a mask is passed to a `ParticleSet` instance, a new `ParticleSet` object will be returned, with all populated fields masked accordingly, see listing 7.5 and listing 7.6.

Listing 7.4 `ParticleSet` initialiser function signature.

```
gcl.ParticleSet(
    pdg: graphicle.data.PdgArray = NOTHING,
    pmu: graphicle.data.MomentumArray = NOTHING,
    color: graphicle.data.ColorArray = NOTHING,
    helicity: graphicle.data.HelicityArray = NOTHING,
    status: graphicle.data.StatusArray = NOTHING,
    final: graphicle.data.MaskArray = NOTHING,
) -> None
```

`ParticleSet` can further be extended by adding topological data.

As mentioned in the caption of listing 7.7, the sign of the `src` and `dst` value encodes information about graph structure. This isn't strictly necessary, since the graph structure would remain unchanged using unsigned integer indices. However, it allows easy consistency checks. For instance, notice that the root node (`src` = 0) produces two protons, with a centre-of-mass energy of 13 TeV, as expected for the LHC. Additionally, it can be seen that that leaf nodes (`dst` > 0) correspond to rows with a `final` value of `True`, since in a full showering and hadronisation simulation, the leaves are the final state particles entering the detector. This DAG adjacency information is given by passing an `AdjacencyList` along with the `ParticleSet` instance to the `Graphicle` constructor. However, the data in a fully populated `Graphicle` instance contains more information about event structure than topology alone.

`StatusArray` contains `pythia8`'s status codes for each particle. In `pythia` version 8.312 (current at time of writing), there are 74 status codes, reserved between 0 - 199, with user defined status codes reserved at values of over 200. These encode detailed information about the purpose of a particle's creation in the record. Similarly to `graphicle`'s convention for `dst` vertices, negative signs represent intermediate particles, positive signs represent the still remaining particles. We will omit the sign when quoting values of status code, since it is usually implied, or available from other data in the particle record. Particles with status codes of 12 represent the "incoming beam", whereas values between 21 - 29 refer to the hard event. 71 - 79 marks particles which are produced in preparation for hadronisation. Using these ranges, we are able to query the particle record with precision.

[7] As opposed to the intermediate particles which will never make it to the detector.

Listing 7.5 `ParticleSet`'s string representation shows the data as a table, inspired by the `DataFrame` representations in `pandas`. The output has been split into two tables for document formatting purposes, but the unmodified output gives contiguous columns.

```
>>> print(particles)
name            px          py          pz      energy
p         0.00E+00    0.00E+00    6.50E+03    6.50E+03
p         0.00E+00    0.00E+00   -6.50E+03    6.50E+03
b~       -0.00E+00    0.00E+00    1.28E+02    1.28E+02
g         0.00E+00   -0.00E+00   -3.93E+02    3.93E+02
W+        3.96E+01   -9.81E+00    8.85E+01    1.26E+02
...     ...         ...         ...         ...
K(S)0    -2.88E+00   -4.92E+00    5.30E+00    7.80E+00
gamma    -4.14E-01   -9.25E-01    1.18E+00    1.55E+00
gamma    -4.97E-01   -8.17E-01    1.03E+00    1.41E+00
pi+      -2.20E+00   -3.55E+00    4.05E+00    5.82E+00
pi-      -6.81E-01   -1.37E+00    1.25E+00    1.98E+00

color    anticolor    helicity    status  final
    0            0           9       -12  False
    0            0           9       -12  False
    0          501           1       -21  False
  501          503           1       -21  False
    0            0           0       -22  False
  ...          ...         ...       ...  ...
    0            0           9       -91  False
    0            0           9        91  True
    0            0           9        91  True
    0            0           9        91  True
    0            0           9        91  True

[3153 particles × 10 attributes]
```

`StatusArray` provides a `MaskGroup` identifying various parts of the hard process via its `.hard_mask` property. This is aliased by `Graphicle`, so can be either accessed as `StatusArray.hard_mask` or `Graphicle.hard_mask`, as shown in listing 7.8. If passed as a subscript to a `Graphicle` instance, since the `MaskGroup` uses an `agg_op` of `OR`, all particles in the hard process will be provided:

```
>>> print(graph[graph.hard_mask])
name            px          py          pz      energy     color
b~       -0.00E+00    0.00E+00    1.28E+02    1.28E+02         0
g         0.00E+00   -0.00E+00   -3.93E+02    3.93E+02       501
W+        3.96E+01   -9.81E+00    8.85E+01    1.26E+02         0
t~       -3.96E+01    9.81E+00   -3.53E+02    3.95E+02         0
e+        1.58E+01    1.94E+00    7.59E+01    7.75E+01         0
```

Listing 7.6 Showing how all data contained within a `ParticleSet` instance can be masked simultaneously. This is the same data as shown in listing 7.5.

```
>>> print(particles[particles.final])
name            px          py          pz      energy
e+        1.58E+01    1.94E+00    7.59E+01    7.75E+01
nu(e)     3.10E+01    5.01E+01    1.10E+01    5.99E+01
pi+       2.07E-01    3.23E-01   -1.64E+01    1.64E+01
pi+      -6.41E-01    4.45E-01    9.24E-01    1.22E+00
K-       -6.15E-03   -8.42E-01   -3.82E+02    3.82E+02
...     ...         ...         ...         ...
pi+      -2.19E-01    1.17E+00   -1.24E+01    1.24E+01
gamma    -4.14E-01   -9.25E-01    1.18E+00    1.55E+00
gamma    -4.97E-01   -8.17E-01    1.03E+00    1.41E+00
pi+      -2.20E+00   -3.55E+00    4.05E+00    5.82E+00
pi-      -6.81E-01   -1.37E+00    1.25E+00    1.98E+00

color    anticolor    helicity    status  final
    0            0           1        23  True
    0            0          -1        23  True
    0            0           9        82  True
    0            0           9        84  True
    0            0           9        84  True
  ...          ...         ...       ...  ...
    0            0           9        91  True
    0            0           9        91  True
    0            0           9        91  True
    0            0           9        91  True
    0            0           9        91  True

[768 particles × 10 attributes]
```

```
nu(e)     3.10E+01   5.01E+01   1.10E+01  5.99E+01        0
W-       -5.06E+01   6.54E+00  -7.36E+01  1.18E+02        0
b~        1.77E+00   2.90E+01  -2.65E+02  2.66E+02        0
d        -2.17E+01  -2.38E+01   1.51E+00  3.22E+01      502
u~       -2.89E+01   3.04E+01  -7.51E+01  8.60E+01        0

anticolor    helicity    status  final      src    dst
      501           1       -21  False       -6     -3
      503           1       -21  False       -7     -3
        0           0       -22  False       -3     -4
      503           0       -22  False       -3     -5
        0           1        23  True     -1212   1533
        0          -1        23  True     -1212   1534
        0           0       -22  False    -1508  -1535
```

Listing 7.7 The string `repr` of a `Graphicle` instance called `graph`. Notice the addition of the `src` and `dst` columns compared to listing 7.5. `src` vertices with a value of `0` represent the root of the event generation. Intermediate vertices are signalled with a negative sign. `dst` vertices with a positive sign indicate that the particle in the corresponding row is a leaf in the generation DAG; therefore for full event simulations, these are the particles incident on the detector wall.

```
>>> print(graph)
name            px          py          pz      energy    color
p         0.00E+00    0.00E+00    6.50E+03    6.50E+03        0
p         0.00E+00    0.00E+00   -6.50E+03    6.50E+03        0
b~       -0.00E+00    0.00E+00    1.28E+02    1.28E+02        0
g         0.00E+00   -0.00E+00   -3.93E+02    3.93E+02      501
W+        3.96E+01   -9.81E+00    8.85E+01    1.26E+02        0
...     ...         ...         ...         ...             ...
K(S)0    -2.88E+00   -4.92E+00    5.30E+00    7.80E+00        0
gamma    -4.14E-01   -9.25E-01    1.18E+00    1.55E+00        0
gamma    -4.97E-01   -8.17E-01    1.03E+00    1.41E+00        0
pi+      -2.20E+00   -3.55E+00    4.05E+00    5.82E+00        0
pi-      -6.81E-01   -1.37E+00    1.25E+00    1.98E+00        0

anticolor    helicity    status  final      src    dst
        0           9       -12  False        0     -1
        0           9       -12  False        0     -2
      501           1       -21  False       -6     -3
      503           1       -21  False       -7     -3
        0           0       -22  False       -3     -4
      ...         ...       ...  ...        ...    ...
        0           9       -91  False    -2813  -2821
        0           9        91  True     -2815   2822
        0           9        91  True     -2815   2823
        0           9        91  True     -2821   2824
        0           9        91  True     -2821   2825

[3153 particles × 12 attributes]
```

```
      813           1       -23  False    -1508  -1536
        0          -1       -23  False    -1535  -1537
      502           1       -23  False    -1535  -1538

[10 particles × 12 attributes]
```

Listing 7.8 - listing 7.11 shows how `MaskGroup` can break down the hard process of the event by accessing its internal `MaskArrays`. Notice that both the t quark and its decay product W^+ boson are marked as "intermediate" in listing 7.10. This is because the W^+ boson ultimately decays hadronically into a quark / anti-quark pair, which is given in the "outgoing" particles.

Listing 7.8 Selecting the hard process from the collision event. `StatusArray` has a `.hard_mask` property, which `Graphicle` aliases. The provided `MaskGroup` may be subscripted to select more refined regions, see listing 7.9 - listing 7.11.

```
>>> print(graph.hard_mask)  # alias of graph.status.hard_mask
MaskGroup(agg_op=OR)
|-- incoming
|-- intermediate
|-- outgoing
`-- outgoing_nonperturbative_diffraction
```

Listing 7.9 Isolating the "incoming" particles from the hard process over the full event record.

```
>>> print(graph[graph.hard_mask["incoming"]])
name           px          py          pz    energy    color
b~      -0.00E+00   0.00E+00   1.28E+02  1.28E+02        0
g        0.00E+00  -0.00E+00  -3.93E+02  3.93E+02      501

anticolor    helicity    status  final      src    dst
      501           1       -21  False       -6     -3
      503           1       -21  False       -7     -3

[2 particles × 12 attributes]
```

Listing 7.10 Isolating the "intermediate" particles from the hard process over the full event record.

```
>>> print(graph[graph.hard_mask["intermediate"]])
name           px          py          pz    energy    color
W+       3.96E+01  -9.81E+00   8.85E+01  1.26E+02        0
t~      -3.96E+01   9.81E+00  -3.53E+02  3.95E+02        0
W-      -5.06E+01   6.54E+00  -7.36E+01  1.18E+02        0

anticolor    helicity    status  final      src    dst
        0           0       -22  False       -3     -4
      503           0       -22  False       -3     -5
        0           0       -22  False    -1508  -1535

[3 particles × 12 attributes]
```

7.3 Beyond data structures: graphicle's modules

`graphicle` offers a great deal beyond these data structures themselves. Routines for traversing the generation DAG are available, along with forming matrices, performing transforms to test IRC safety, and calculating event shape observables. A small subset

Listing 7.11 Isolating the "outgoing" particles from the hard process over the full event record.

```
>>> print(graph[graph.hard_mask["outgoing"]])
name           px          py          pz    energy    color
e+       1.58E+01    1.94E+00    7.59E+01  7.75E+01        0
nu(e)    3.10E+01    5.01E+01    1.10E+01  5.99E+01        0
b~       1.77E+00    2.90E+01   -2.65E+02  2.66E+02        0
d       -2.17E+01   -2.38E+01    1.51E+00  3.22E+01      502
u~      -2.89E+01    3.04E+01   -7.51E+01  8.60E+01        0

anticolor    helicity    status  final      src    dst
        0           1        23  True     -1212   1533
        0          -1        23  True     -1212   1534
      813           1       -23  False    -1508  -1536
        0          -1       -23  False    -1535  -1537
      502           1       -23  False    -1535  -1538

[5 particles × 12 attributes
```

of these are used in the examples shown in chapter 9. For more information, an API reference provides information on `graphicle`'s functionality, with detailed descriptions, type information, and examples.

Chapter 8

Data generation with `showerpipe`

We have explored the data structures and analysis techniques for analysing simulated HEP data, but little has been said about how these data were obtained. For this, we owe a great deal to the authors of Pythia, who have put in the work to make a reasonably feature-complete port of their C++ program into the Python package, `pythia8`. This package reproduces the C++ interface well, which creates some friction in Python.

8.1 Using `pythia8` directly

8.1.1 Generating data

In order to perform a particle simulation using `pythia8`, we first setup our MCEG by instantiating a `pythia8.Pythia` object. This takes an optional parameter of `xmlDir`, containing Pythia's internal config data in the form of XML. Usually this is set as an environment variable, so it can be omitted. Once the instance is created, one must pass either a `.cmnd` file containing Pythia's run settings, or perform a sequence of `.readString()` method calls on the `pythia8.Pythia` instance constructed.

Listing 8.1 Pythia settings `.cmnd` file example.

```
PartonLevel:ISR = on
PartonLevel:FSR = on
PartonLevel:MPI = on
ColourReconnection:mode = 1
BeamRemnants:remnantMode = 1
Beams:frameType = 4
Beams:LHEF = /tmp/tmp9o6e4c6w.lhe
```

Listing 8.1 is an example of a setup which requires a Les Houches Event (LHE) file (Alwall et al., 2007) to be passed to the `pythia8.Pythia` constructor. LHE files contain simulated data of the hard process, and in our case we provide it with files produced by MadGraph5 (Alwall et al., 2011). Pythia does have the capacity to generate its own hard events, but this is more limited and less sophisticated than the calculations performed by MadGraph5. Once this has been passed, we call the somewhat confusingly named[1] `.init()` method to start the event generator.

In listing 8.1, we show as an example the file we use in our work to simulate data. The settings are fairly standard. We enable initial state and final state radiation, as well as multi-parton interactions (these are just settings which increase the complexity of the simulation, but improve the physicality of the results). We could have omitted `ColourReconnection:mode = 1` and `BeamRemnants:remnantMode = 1`, but these use newer models based on closely matching QCD, so we switch those on for good measure. `Beams:frameType = 4` and the `Beams:LHEF` lines tell Pythia we would like to use a pre-computed LHE input file for the hard interaction calculation, and gives the path to the LHE file, respectively. For more information on using Pythia, introductions, manuals, and worksheets can be found at https://pythia.org/.

Mutable properties then expose event data to the user. These are effectively containers with a constant address in memory, but whose contents get refreshed every time a new event is generated. Events are generated by calling the `.next()` method on the `Pythia` instance. `.next()` returns `True` or `False` depending on whether or not an event was actually generated. This should only happen in the case that a user passes an LHE file in the `.cmnd`-style settings, as otherwise `Pythia` is capable of producing new events indefinitely. However, once the end of a LHE file is reached, events are no longer generated, and this is signalled via the output of `.next()`.

A property of `Pythia` called `.event` then exposes a `pythia8.Event` object. This is an `iterable` of `pythia8.Particle` instances. These expose particle properties by a large number of methods. A non-exhaustive list of these methods is provided in table 8.1.

Table 8.1: Displaying the getter methods for particle properties provided by `pythia8.Particle`.

Attribute	Description
`px()`, `py()`, `pz()`, `e()`	The Cartesian four-momentum components of the particle in the lab rest frame
`pAbs()`, `pT()`	The magnitude of 3-momentum and transverse momentum, respectively
`m2()`, `mT()`	The squared mass and transverse mass, respectively

[1] Confusing in that the object itself has already been initialised via its constructor.

Attribute	Description
`theta()`, `phi()`, `y()`, `eta()`	The polar, azimuthal, rapidity, and pseudorapidity angles, respectively
`tau()`	The lab-frame lifetime in mm / c
`id()`	The PDG code identifying particle species
`status()`	The status code of the particle, identifying its role in the event generation
`pol()`	The spin polarisation of the particle (valid values are in range [-1, 1]; unset or unknown values are indicated with a sentinel value of 9)
`col()`, `acol()`	The colour codes of the particle; colour-singlets have both values set to 0, colour-triplets have one non-zero value, and colour-octets are indicated with two non-zero values
`isFinal()`	Identifies if the particle remains (i.e., is not intermediate) after an event has been fully simulated
`motherList()`, `daughterList()`	Collections providing indices of the immediate ancestors and descendants of the particle, respectively

This forms the low-level data basis of the derived semantics for `graphicle`'s data structures.

8.1.2 Difficulties in Python

The interface outlined above for generating and then accessing data requires a lot from a Python programmer. First initialising the generator with `pythia8.Pythia` initialiser, then seemingly having to initialise it a second time. The fact that the LHE file must be hardcoded into the `.cmnd` file makes it hard to change. However, the alternative of passing a filepath along with all other settings in successive calls to the `.readString()` method increases code verbosity and couples source code to configuration settings. This leads on to the difficulty with iteration.

The need to manually call `.next()` on the `pythia8.Pythia` is an uncomfortable fit in Python. Python's data model allows the implementation of a `.__next__()` dunder method, which would have made `pythia8.Pythia` compatible with Python's standard iteration patterns, including for-loops, and the existing built-in `next()` function. Additionally, the boolean return status on `.next()` is not Pythonic. Python signals that an `iterator` has been exhausted by throwing a `StopIteration` exception[2]. Manually

[2]There are those who question whether overloading Python's error handling apparatus for expected behaviour such as loop termination is a sensible design choice for the language, but it *is* the standard.

needing to check for a `False` return value from `.next()` makes it likely that new users will introduce errors into their data generation scripts by repeatedly calling `.next()` after the LHE file has been exhausted. Accessing the data is also problematic.

To minimise bugs in our work, we favour immutability over mutability, see chapter 7.2.4. In-place mutation of data can introduce silent errors to code, as accidental mutation invalidates data, but will likely produce plausible outputs for the same routines. Additionally, side effects highly likely when multiple objects rely on shared memory, leading to unexpected results if the data is modified in one, and therefore implicitly in the other. When `pythia8.Pythia` generates a new event, the instance of `pythia8.Event` is modified in-place. If users wish to compare successive events for debugging purposes, there is no clear interface for detaching the data from the generator to do so. Even if this were possible, it would require using `pythia8.Particle`'s methods to access the raw and derived data, which is hard work.

Visiting individual `pythia8.Particle` instances and calling methods to access all four components of momenta separately, along with colour, anti-colour, PDG, and status codes increases code verbosity and complexity. Additionally, in chapter 6.1 we covered why iterating over objects in pure Python is slow, so using this interface would have negative performance implications. The now archived `numpythia` project (Dawe et al., 2023) used to provide a Pythia interface to expose data as `numpy` arrays. This is a much more natural choice for Python data science pipelines, but unfortunately we cannot rely on deprecated projects.

The patterns we have criticised above may be perfectly idiomatic and reasonable in C++ programs. The work which has gone into Pythia is substantial, and the basis for all of the work carried out in this thesis. However, we have laid out why we believe these patterns do not translate well into idiomatic Python, *ie.* `pythia8` is not *Pythonic*.

8.2 Wrapping `pythia8` to become Pythonic with `showerpipe`

The `showerpipe` package (Chaplais, 2025c) addresses these issues by providing two wrapper interfaces to `pythia8`. These are `showerpipe.generator.PythiaGenerator` and `showerpipe.generator.PythiaEvent`[3].

8.2.1 The `PythiaGenerator` interface

`PythiaGenerator` removes some of the boilerplate from initialising the event generator. Rather than calling the object constructor and subsequently passing the settings via

[3] Ambitiously named with the Pythia prefix, as the original intention for `showerpipe` was to produce a general-purpose adapter pattern, offering the same interface for other MCEGs, like Herwig 7.0.

the .cmnd file / .readString() invocations to "initialise" the (already instantiated) generator with a final .init() call, all settings are applied in the object constructor. Additionally, PythiaGenerator's constructor may take a parameter of lhe_file, which inserts the Beams:LHEF key-value pair into the .cmnd file data, see listing 8.1. This circumvents the coupling issue of either putting all file paths into the .cmnd file on disk, or bloating code verbosity and complexity with many .readString() calls.

Once PythiaGenerator has been initialised, it uses Python's data model to provide a Pythonic iterator interface. By creating a concrete instance of [collections.abc.Iterator] and simply implementing the __next__() dunder method, PythiaGenerator becomes fully compatible with Python's builtin iteration tools. The special __len__() method is also implemented, which makes PythiaGenerator a [Sized] iterator. Upon iteration, instead of returning a boolean success status to be explicitly evaluated for the end of the simulation, a PythiaEvent instance is yielded. Once iteration is complete, the standard StopIteration exception is thrown, which exits loops gracefully and remains consistent with the response of a standard exhausted iterator when the builtin next() method is called on it.

8.2.2 The PythiaEvent interface

PythiaEvent exposes the particle data to the user. However, unlike pythia8.Event, it is not an iterable over particle objects, to be queried with individual calls to getter methods for each scalar describing the particle's data. Instead, the PythiaEvent itself has a number of properties relating to particle data. Data is exposed for all particles in the event simultaneously by the attribute access, providing the data formatted in ndarrays, see table 8.2.

Table 8.2: Attributes exposed by PythiaEvent to access data for all particles generated in an event at once, exposed as ndarrays.

Attribute	Description
pdg	Particle Data Group identification codes for the particles in the event.
pmu	Four-momenta of the particles, structured with fields for x, y, z, and e components.
color	Colour codes for particles, structured with color and anticolor fields.
helicity	Helicity eigenvalues for particles; a sentinel value of 9 indicates no eigenvalue.
status	Status codes describing each particle's creation method and role, as defined by Pythia.
final	Boolean mask identifying the final-state particles in the simulation.
edges	Directed acyclic graph (DAG) representing particle heritage, structured as a COO adjacency list.

By implementing these attributes, `PythiaEvent` satisfies `graphicle`'s `EventInterface` protocol. This means that the `ndarray` data can be seamlessly imported to a `graphicle.Graphicle` instance with the `Graphicle.from_event()` classmethod, ready for a full analysis pipeline. Using protocols in this way is a form of Dependency Injection (Freeman et al., 2004; Martin and Coplien, 2009). Both `graphicle` and `showerpipe` work effectively without dependencies on one another obeying the Single Responsibility Principle, while still enabling the benefits of strong interoperability (Hunt and Thomas, 2019; Gamma et al., 1994).

The most substantial departure from `showerpipe` merely acting as a wrapper around `pythia8` comes in the form of the `edges` attribute. Internally, `showerpipe` compares each particle's list of parents and children. Particles will have common sets of shared parents or shared children, and these are used to identify interaction vertices. This information is cross-referenced over the whole event, and produces `src` / `dst` vertex ID pairs for every particle, which forms the COO list. For a detailed definition of this algorithm, see appendix A.

Taken together, `showerpipe` and `graphicle` form a powerful data generation and analysis pipeline, whose idioms should be familiar to Python programmers.

8.3 Generation DAGs as a view on showering and hadronisation

As previously stated, the nodes in a generation DAG represent interaction vertices. Before hadronisation occurs, the dynamics of quantum chromodynamics (QCD) at high energy, governed by the property of asymptotic freedom, give rise to a largely tree-like structure. This tree structure reflects the kinematic flow of momentum and energy during the parton shower phase of a simulation, wherein particles undergo successive splittings and radiative emissions (Bierlich et al., 2022). These splittings evolve the system from a few high-energy particles to a cascade of lower-energy particles, populating the phase space of the event.

This phase of parton showering is characterised by the near-independence of partons, as they are not colour-confined. The resulting topological structure of the generation DAG mirrors the branching patterns predicted by perturbative QCD. Splittings are encoded as edges of the DAG, with the nodes representing the interactions at which partons split or radiate gluons. In this way, the tree-like pre-hadronisation structure of the DAG encapsulates the dynamics of QCD cascades, encoding not only the momentum transfer but also the subtle colour flow among partons.

However, this behaviour changes when the energies of particles fall below the hadronisation scale, $\mu \sim \Lambda_{\mathrm{QCD}}$. Here, the strong coupling constant, α_s, grows so large

that perturbative techniques break down, and colour confinement becomes unavoidable. Particles must reorganise into colour-singlet hadrons, a process governed by non-perturbative QCD dynamics. This reorganisation introduces what we call *hadronisation vertices* into the DAG. Unlike earlier vertices, hadronisation vertices represent the effective endpoints of perturbative QCD and the transition to phenomenological models such as the Lund string or cluster model, which describe the final-state hadrons (Sjöstrand et al., 2015).

In the context of hadronic decays of colour-singlet bosons like the Higgs, the generation DAG reveals crucial structural features. For example, due to the Higgs boson's intrinsic nature as a colour-singlet, its decay products are naturally paired with complementary colour charges. These charges "flow" into confined colour singlets during hadronisation, ensuring that the showers initiated by these products remain largely isolated from the rest of the event. This phenomenon is referred to as *colour-connection*, a feature that helps to explain why the momentum of a Higgs boson is relatively straightforward to reconstruct from the topological analysis of its descendant particles. The phase space occupied by Higgs descendants is effectively "closed" under the influence of colour reconnection and hadronisation.

In contrast, consider the case of top quarks, which are not colour-singlets at production. Top quarks inherit their QCD charge from the hard scattering of initial-state partons. This inherited charge is naturally colour-connected to the remnants of the initial-state spectator partons. Consequently, the top-quark showers are far more entangled with the underlying event (UE), introducing complexities into their reconstruction. The generation DAG of such an event encodes not only the primary decay dynamics of the top quark but also the intricate web of colour exchanges with the UE, a feature that challenges standard reconstruction techniques.

Thus, the generation DAG serves as an invaluable lens through which to examine the dual processes of showering and hadronisation. It captures the interplay between momentum flow and colour dynamics, offering a unique window into the topological constraints imposed by QCD. While these ideas are explored further in the following chapters on data preprocessing and clustering strategies, the generation DAG provides the critical bridge between the simulation framework and the physical observables reconstructed in detectors. By making explicit the transition from tree-like perturbative structures to the web-like connectivity of hadronisation, this framework underpins much of the subsequent analysis in this work.

It should be noted that, while `showerpipe` was developed to address the sharp edges from porting the C++ Pythia project directly to Python, it was in the light of the work presented in this thesis. Since it only exposes select attributes of the underlying Pythia interface, it may be that critical functionality is missing for researchers who use Pythia in more nuanced ways. It is hoped that, since the code is a work-in-progress, and is

released as FOSS, community engagement will guide the development process to address any such issues. But showerpipe is early in its development, and we must take care not to give across the false impression that it replaces the excellent Pythia simulator.

Chapter 9

Preprocessing techniques for generating supervision targets

9.1 Clustering via graph traversal

In listing 7.8 - listing 7.11, the `StatusArray.hard_mask` is used to locate regions inside the hard event. It is possible to combine this data from the `StatusArray` with the topological information provided by `AdjacencyList`. By identifying the intermediate and outgoing particles in the hard process, their descendants may be found by performing a Breadth-First Search (BFS) over the DAG structure.

BFS is a fundamental graph traversal algorithm that explores a graph level by level, starting from a designated source node (Moore, 1959). It systematically visits all nodes at a given distance from the source before moving on to the next level of nodes. This approach ensures that the first time a node is visited, it is via the shortest path in terms of edge count, making BFS especially useful for unweighted graphs.

The algorithm uses a first-in, first-out queue to manage the order of exploration. Nodes are enqueued as they are discovered and dequeued for exploration in the order they were added. BFS operates efficiently in linear time relative to the number of vertices and edges in the graph.

Many software libraries provide efficient implementations of BFS. For example, SciPy (Virtanen et al., 2020) includes a function called `breadth_first_tree` in its `scipy.sparse.csgraph` module, which returns a tree of predecessors resulting from a BFS traversal. This can be used to identify all descendants of a node in a directed graph or all reachable nodes in an undirected one. Graphicle uses this implementation in `graphicle.select.hierarchy()` to create a nested hierarchy of descendants of the hard interaction, traversing the entire event to the final state particles.

Naïvely, our initial assumption was that `hierarchy()` should produce jet clusterings based entirely in MC simulation truth data. This is true for a certain class of particles. High quality jet clusters may be found via BFS for colour singlet bosons (CSBs), *ie.* $W^{\pm}$, Z^0, and H^0. See listing 9.1 for an example of analysing the first H^0 event produced in a file of 1 million, showered and hadronised by `pythia8` via `showerpipe` (see chapter 8 for more on `showerpipe`).

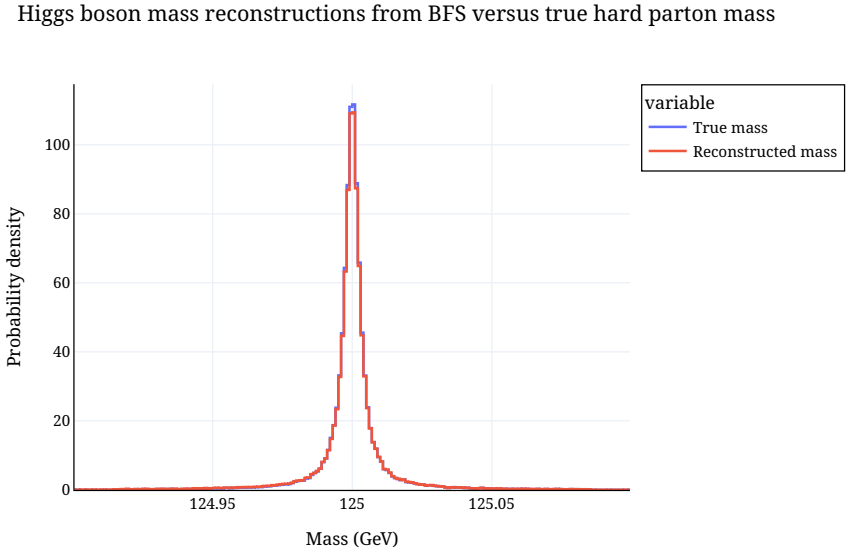
This analysis shows a remarkable agreement up to 10 significant figures between the mass generated by the BFS cluster based on the DAG *vs.* the mass of the original particle it is reconstructing. Indeed, Pythia does juggle momentum between intermediate particles for algorithmic purposes, so numerical error is expected and could account for the minor discrepancy beyond the 10th significant figure. In order to prove that this performance is consistent and predictable, in figure 9.1a we display a probability density histogram of mass calculations performed over the 100,000 events of our test dataset, and we overlay this with the histogram produced by binning the masses of the original Higgs bosons from the hard event. The histograms, network plots, and heatmaps in this work are all image exports produced by `colliderscope` (Chaplais, 2025a), an interactive visualisation library created for this work. The result is a close match at a high resolution of ± 0.5 MeV.

If we apply the same procedure to an individual top quark, however, figure 9.1b shows that the result is poor. To understand why, we must consider what the generation DAG truly represents. This was covered in chapter 8.3, but to re-iterate: whereas the DAG rooted on the Higgs boson represents a closed phase space due to the colour connection of the $b\bar{b}$ quark pair, the top quark has no such convenient advantage.

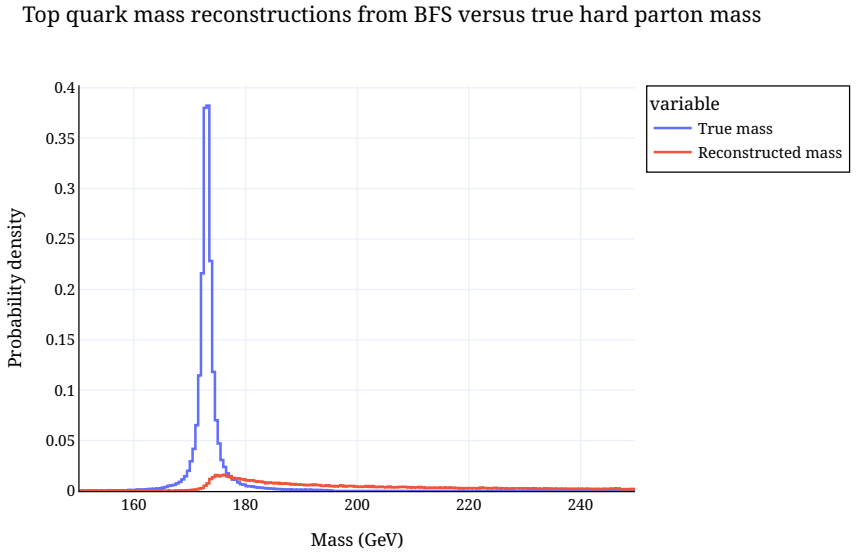
Instead, the t and consequently resulting b quark from the $t \to W^+ \, b$ decay channel are colour triplets, and connected to the UE via the spectator partons contained within the beamline protons which initiated the collision. This means that the interaction vertices exchanging colour play a more complex role. This has a particularly strong effect as the energy scale lowers the event transitions from showering into colour-reconnection and hadronisation. The interaction vertices at this point connect the descendants of the b topologically to sources of momentum from the UE. This effectively introduces noise to our momentum tracing via BFS on the DAG rooted on the b quark, spoiling the individual top quark mass reconstruction.

9.2 Improved clusters with momentum matching

As we have said, the source of noise in figure 9.1b is due to interaction vertices being a reflection of colour flow, rather than momentum flow, in events which are colour-connected to the background. Even in principle, it is impossible to circumvent



(a) Higgs boson. BFS mass reconstruction is successful. Bin width is 1 MeV.



(b) Single top quark. BFS mass reconstruction is poor. Bin width is 500 MeV.

Figure 9.1: Comparison of mass reconstructed with `graphicle.select.hierarchy()` *vs.* Monte-Carlo truth mass for partons in the hard event. `hierarchy()` approximates the momentum of the parton in the hard event by summing the momenta of the topological descendants of the parton in the final state, found via BFS. Masses were computed over the 100k events provided in the test datasets for the respective partons.

Listing 9.1 Showering and hadronising a MadGraph5-generated hard event of $p\ p \to H^0\ Z^0$, where H^0 decays hadronically via the $H^0 \to b\ \bar{b}$ channel, and Z^0 decays leptonically. The BFS mask identifying descendants of the H^0 generated by `hierarchy()` reconstructs the mass of the original hard H^0 accurately to 10 significant figures.

```
>>> import numpy as np
... import graphicle as gcl
... import showerpipe as shp
...
...
... splits = shp.lhe.split("unweighted_events.lhe.gz", 1000)
... lhe_data = next(splits)
... gen = shp.generator.PythiaGenerator("pythia-settings.cmnd", lhe_data)
... event = next(gen)
...
... graph = gcl.Graphicle.from_event(event)
... hier = gcl.select.hierarchy(graph)
... print(hier)
MaskGroup(agg_op=OR)
|-- H0
|   |-- b
|   |-- b~
|   `-- latent
`-- Z0
    |-- mu+
    |-- mu-
    `-- latent
>>> H0_cluster = graph.pmu[graph.final & hier["H0"]]
... H0_pmu = np.sum(H0_cluster, axis=0)
... H0_pmu.mass.item()  # mass of Higgs from cluster reconstruction
124.99329740563036
>>> hard_H0_pmu = graph.pmu[graph.hard_mask & (graph.pdg.name == "H0")]
... hard_H0_pmu.mass.item()  # mass of original Higgs in hard process
124.99329737099997
```

this issue fully, see chapter 3.2.2. However, some simple heuristics can significantly improve performance.

Consider how hadronisation is represented in the generation DAG for the $pp \to t\bar{t}$ process. In figure 9.2a, we observe the resulting descendants of the b quark from the t decay hadronising on a single interaction (hadronisation) vertex. Incident edges (particles) are a mixture of descendants from the hard parton we wish to reconstruct, and descendants from the UE. By superimposing the positions of the hard partons on the $\eta - \phi$ plane against a scatter plot of the detector-level particles figure 9.2b, it is apparent that distinct collimated sprays are associated with each incident particle to the vertex. That is, localised regions of the momentum space incident on the hadronisation vertex are preserved in the outgoing particles.

For ancestry-only approaches, this means collimated sprays of detector level particles aligning more closely with the UE are erroneously associated with the b quark. A cluster aligned with the hard b quark is seen directly beneath it, but to the left are tight clusters which are also included in its reconstruction, but should be discarded as the UE. In fact, figure 2.5 is an identical scatter plot from the same event as figure 9.2b. The difference is that in the former case, we had already applied the improved method of clustering, which removed this UE noise.

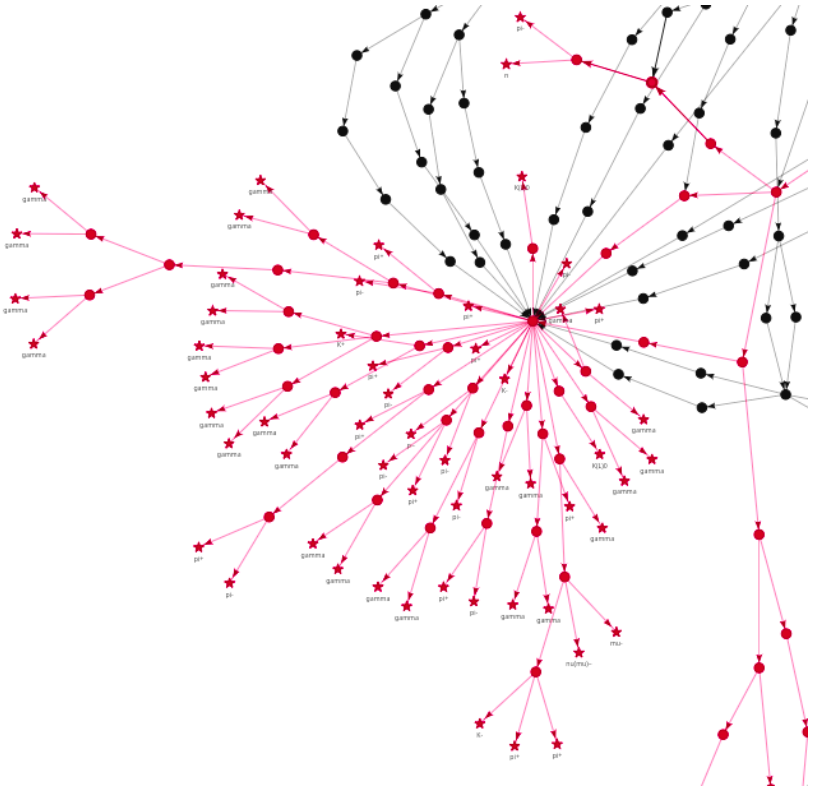
To illustrate the improvement in the reconstruction, the t quark produced in the hard process has a mass of 174.02 GeV. Combining the momenta of the detector-level constituents associated with the b, $\bar{s}$, and c quarks produced by the t's decay as in figure 9.2b results in a mass reconstruction of 1179.1 GeV for the t quark. Whereas, performing the same procedure on figure 2.5 which utilises our improved method, the mass reconstruction is 174.45 GeV. So, how do we achieve this remarkable improvement?

As we have learned, regions of momentum space can be associated with signal and UE particles separately, providing expected directions for the outgoing particles descending from these classes, respectively. By taking the direction-only components of the outgoing particles, we can form a distance matrix comparing the incoming and outgoing particles. Elements of this matrix are simply given as $\Delta R_{ij} = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}$. i is an incident particle on the hadronisation vertex, and j is a descendant of the hadronisation vertex in the final state. Note that j is unlikely to be a direct child of the hadronisation vertex, since decays following hadronisation are common.

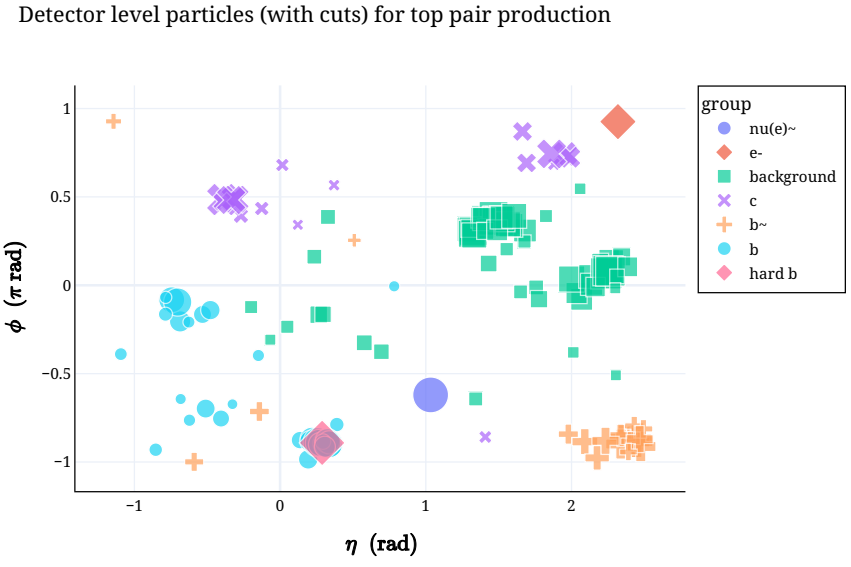
To express more mathematically, let set $\mathcal{T}$ refer to the descendant particles identified by the mask obtained via BFS of the DAG rooted on the individual t quark. Further, let set $\mathcal{I}$ refer to the particles incident to the hadronisation vertex. Finally, let set $\mathcal{F}$ refer to the final state particles in the event. By our na"ive prescription of using only topology to form clusters, we could define an individual top quark cluster in the final state as $\mathcal{T} \cap \mathcal{F}$.

Instead, we form two new sets representing signal and background classes. The signal class is defined as the top descendants incident on the hadronisation vertex, $\mathcal{S} = \mathcal{T} \cap \mathcal{I}$ (the blue incoming edges in figure 9.2a). The background class is defined as the UE descendants (the black incoming edges, *ie.* everything not descending from the top) incident on the hadronisation vertex, $\mathcal{B} = \mathcal{I} \backslash \mathcal{S}$.

In this jargon, our i and j elements from before may be defined as $i \in \mathcal{I}$ and $j \in \mathcal{T} \cap \mathcal{F}$. In the end, we wish to form a subset $\mathcal{C} \subset \mathcal{F}$ representing a cluster to reconstruct the top quark. The first step to produce this set is to populate it with elements such that



(a) Showing the generation DAG centred around a hadronisation vertex. Blue edges and vertices indicate the topological descendants of an individual top quark. Black sections of the graph structure are descended exclusively from the underlying event.



(b) Scatter plot on the $\eta - \phi$ plane for a $pp \to t\bar{t}$ production. Detector-level particles are grouped as descendants from partons in the hard process. Superimposed at (0.3, -0.9) is the "hard b", *ie.* the outgoing hard b quark produced by the t decay.

Figure 9.2: Demonstrating the overly-inclusive definition of clusters reconstructing a hard parton by simply forming them from the hard parton's descendants. Topological descendants from the t quark in the generation DAG can be seen to collimate in several distinct regions, where only one is centred on the relevant b descendant prior to hadronisation. We see that hadronisation mixes ancestry with the UE.

$$\mathcal{C}^* = \{j : \arg\min_i \Delta R_{ij} \in \mathcal{S}\}. \tag{9.1}$$

That is, the topological descendants of an individual top quark in the hadronised final state, whose momenta are more closely aligned with the top descendants before hadronisation (rather than the UE) form the cluster used to reconstruct the top momentum / mass. This process is automated by `graphicle.select.clusters()`.

Equation 9.1 defines set $\mathcal{C}^*$ as an intermediate cluster. Empirical observations of this technique showed that this was not robust for clusters containing particles with very large angular deviations ΔR_{ij}. So our next step, after $\mathcal{C}^*$ is obtained, is to exclude elements with some high cut-off value R_{cutoff} for ΔR_{ij}. `clusters()` calls `centroid_prune()` on the result, which applies this maximum angular deviation cut-off, such that

$$\mathcal{C} = \mathcal{C}^* \backslash \{k \in \mathcal{C}^* : \Delta R_{ik} > R_{\text{cutoff}}\ \forall i \in \mathcal{S}\}, \tag{9.2}$$

where R_{cutoff} is the cut-off maximum angular distance between clustered particles in the final state, to top descendants incident on the hadronisation vertex.

The result of applying equation 9.2 via `clusters()` on the top quark, rather than using `hierarchy()`, is shown in figure 9.3.

This shows a marked improvement to figure 9.1b. This result clearly does not reproduce the mass histogram of the t quark in the hard scattering, but this is fundamentally not possible. We do, however, substantially improve upon anti-k_T clusters tagged using Monte-Carlo truth data. Taken together, figure 9.1a and figure 9.3 motivates us that our data pipeline serves as a high quality feed stock to train a clustering model to exceed the performance of standard methods.

9.2.1 In praise of anti-k_T

At this stage, we have outlined the limitations of anti-k_T, and shown how simulated data may be utilised to exceed its performance in producing high resolution mass reconstructions. However, here we take a brief moment to outline our appreciation for generalised-k_T algorithms, and anti-k_T in particular. Anti-k_T provides a robust technique against soft radiation that is totally general, and by tweaking a few hyperparameters can create meaningful clusters across all different kinds of processes.

The Monte-Carlo truth based methods developed in this chapter are able to automatically create clusters to reconstruct any particle produced in a hard interaction which, in the datasets explored by our studies, exceed the fidelity provided by anti-k_T. However, these may only be applied to experimental data indirectly, by evaluating

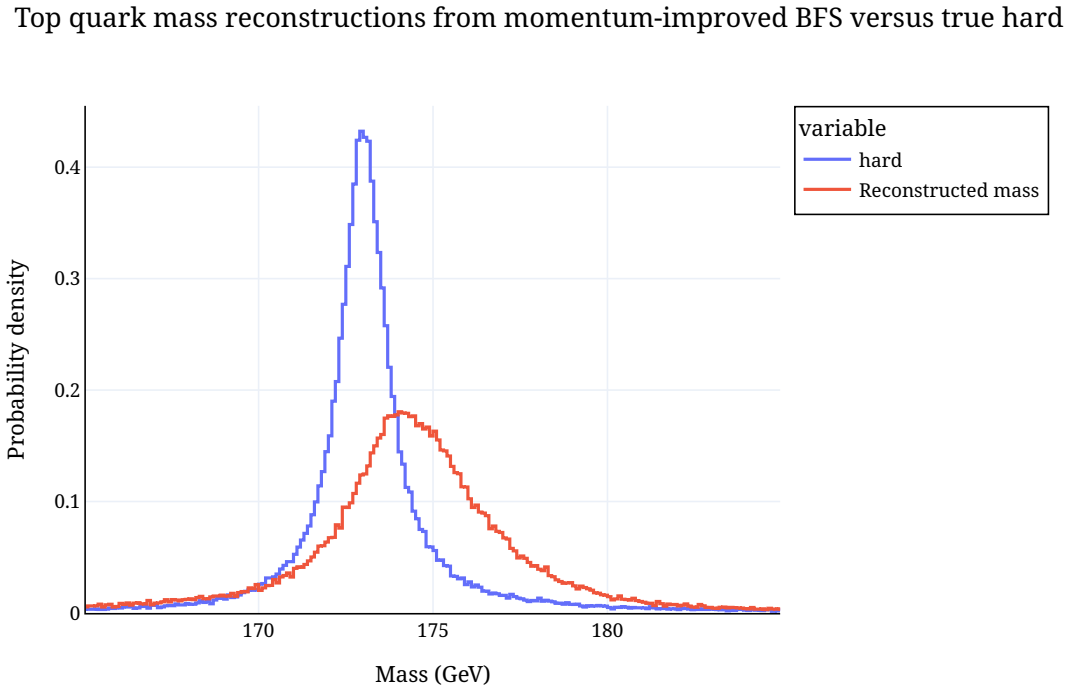


Figure 9.3: Single top quark mass reconstruction improved by matching cluster constituent directions to the top quark descendants before hadronisation, as per equation 9.2. Bin width is 100 MeV.

said data against our GNN models, trained with supervision from our Monte-Carlo truth based clusters, see chapter 10 and chapter 11. Our GNN models are trained on specific processes, and exhibit process and kinematic dependence, based on the configuration of our MCEGs. Even the quality of our Monte-Carlo truth based clusters *for the same species* of reconstructed particle exhibit process dependence, as we shall see in chapter 9.3.

We believe this latter issue may be addressed with future developments to our algorithm. Additionally, representation learning and transfer learning could be used to generalise our trained GNNs to datasets with diverse processes and kinematic profiles. However, it is important to acknowledge anti-k_T's innate generality, and we have a great deal of respect for its justified, continuing contributions to HEP analysis.

9.3 Process dependence

We have been a bit imprecise about which processes are producing our t quark up to this point. We have mentioned $pp \to t\bar{t}$, which is what we have shown in figure 2.5

(a) Interaction of a gluon and b quark producing a t quark and W^- boson.

(b) Quark anti-quark annihilation producing a H^0 boson and Z^0 boson.

Figure 9.4: Feynman diagrams depicting the processes during the proton-proton collision, which we study in this thesis using our MCEGs. Time runs from left to right, such that incoming particles are on the left, virtual mediating particles are horizontally aligned in the centre, and outgoing particles are on the right.

and figure 9.2, but it is not the process from which we have been constructing our mass histograms from.

The reason for this is not because $pp \to t\bar{t}$ gives poor reconstructions, and we wish to sweep this under the rug. In fact, this could not be further from the truth. However, when developing our ML models to reconstruct the clusters, the complexity of reconstructing two t quarks simultaneously was a challenge we wished to avoid, because if they both decayed hadronically, their detector deposits would look identical, and may overlap. To this end, we prepare a dataset with only a single t in the final state, $pp \to t\ W^-$, where the W^- is forced to decay leptonically. Specifically, the process is given by the Feynman diagram in figure 9.4a. The details of the simulated events will be described in more depth in chapter 11.

However, as we suggested before, the $pp \to t\bar{t}$ reconstruction is of very high quality. In fact, it vastly exceeds the quality of the $pp \to t\ W^-$ reconstruction, as can be seen in figure 9.5. We do not yet understand the reasons for this discrepancy. Future research to understand this difference, and to utilise the exceptionally high quality reconstruction of the top quark seen in figure 9.5 would likely yield results which exceed that which we present in this thesis, since it would significantly raise a glass ceiling in terms of supervision target quality.

To explore the package ecosystem we have presented in this section, and perform your own analyses, see the official Jupyter Notebook tutorial (Chaplais, 2025d).

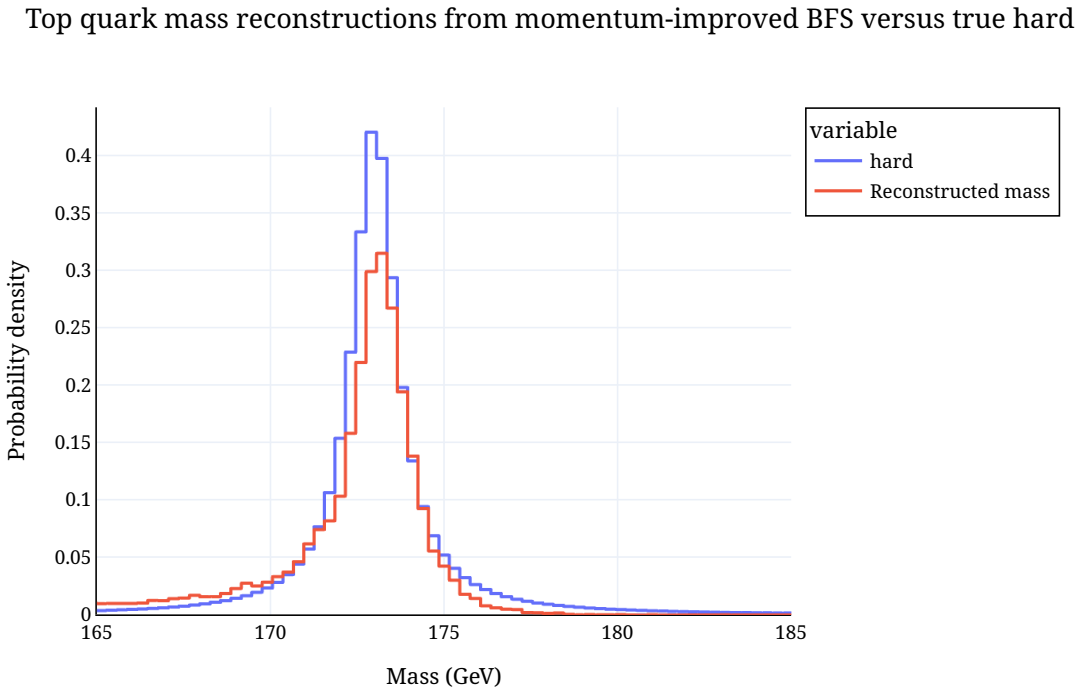


Figure 9.5: Improved single top quark mass reconstruction for $pp \to t\bar{t}$. Bin width is 300 MeV.

Part III

Experimental results

Chapter 10

Higgs boson reconstruction using EWMP networks

In this chapter, we present a GNN architecture which performs Higgs boson reconstruction from detector-level data, in a manner which is IRC safe by construction, as a node classification task. This adapts the approach of Konar et al. (2022), as explored in chapter **??**, for an EWMP network which extends the use beyond GCNs to the "message passing flavour" of GNNs, as described by Bronstein et al. (2021). In this process, we take inspiration from Ju and Nachman (2020), whose work explored the application of INs to reconstruct boosted $W^{\pm}$ bosons as a link prediction task.

We introduce an activation layer specialised for GNNs called *bright edge activation*. This enables the mapping of the link prediction task outlined by Ju and Nachman (2020) into a node prediction task, which significantly simplifies the interpretation of performance metrics.

10.1 Link prediction as clustering

10.1.1 Formulation

Ancestry information from Monte-Carlo truth data has been shown to be effective for creating supervision labels on detector-level particles, descending from the $W^{\pm}$ boson (Ju and Nachman, 2020). The authors of this method showed that performing link prediction using the IN could reconstruct superior clusters to anti-k_T.

They do this by encoding the event as a fully-connected bidirectional graph. In this prescription, nodes are given feature vectors with full four-momentum data, $\mathbf{v}_i^{(0)} = \left(p_x, p_y, p_z, E\right)$. Edge features on the input graph are not explicitly passed, but

embeddings of the edges are learned via the message passing algorithm used by the IN. No external feature vector is passed, or global embedding is produced, *ie.* only nodes and edges are embedded.

After 8 embedding steps, the authors apply a linear classification layer to obtain scalar logit scores on each edge[1]. Applying Sigmoid activations to these scores, see equation 5.4, they are compared against targets using binary cross entropy (BCE) loss.

The Binary Cross-Entropy loss is a classification loss function, defined as:

$$\mathrm{BCE}(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]. \tag{10.1}$$

Here, y_i is the true binary label (0 or 1) for the i-th example, $\hat{y}_i$ is the predicted probability (output of the sigmoid function) for the i-th example, and N is the total number of examples.

The targets are determined for each edge depending on the nodes they connect:

$$\mathrm{target}_{ij} = \begin{cases} 1 & \text{if both node } i \text{ and node } j \in \text{Higgs descendants} \\ 0 & \text{else.} \end{cases} \tag{10.2}$$

That is, there are naturally targets on the nodes identifying whether or not the corresponding particles are descendants of the Higgs / cluster consituents. To cast this as a link prediction task, the edge targets effectively apply a logical AND operation on both node targets.

10.1.2 Challenges

Formulating clustering as a link prediction task presents challenges. These are rooted in both computational considerations, and the difficulty of interpretation due to the convoluted problem formulation.

In the first case, GNNs have a high computational complexity (Thais et al., 2022; Duarte and Vlimant, 2022). This effect becomes particularly aggravated when input graphs contain many edges. The number of edges in a fully connected graph scales as $\mathcal{O}(N^2)$; if edges are encoded into high dimensional embedding spaces over many iterative GNN layers, the memory consumption of the feed-forward pass of the GNN becomes substantial, and keeping track of the operations for automatic differentiation

[1]The term "logit" is used here to mean the output of a classification layer, prior to having a sigmoid activation applied. Mathematically, the logit function is the inverse of the sigmoid function, see https://en.wikipedia.org/wiki/Logit for more.

in these cases adds a substantial additional burden[2]. In Ju and Nachman (2020)'s work, the fully-connected graph structure maximises this memory footprint. This is logical from the perspective that "true edges" are not known *ab initio*, and thus all potential edges must be evaluated to obtain a pre-computed target, but the computational expense is prohibitive without sufficient onboard memory for GPU hardware.[3].

The difficulties in evaluating model performance results from properties common in graph structured data: class imbalance. From a sample of 1,000 randomly selected events from our Higgs boson data set, the proportion of "true edges" compared with "false edges", as defined above, is approximately 3%. That is, if we were to use this method to reconstruct Higgs bosons, 97% of the targets would be negative. Many real-world link prediction tasks mirror this (Yang et al., 2014), and require substantial work to reweight the chosen loss function, and tune the optimisation hyperparameters.

On top of this, consider common classification performance metrics. A GNN which learns to classify all particles as background would achieve a 97% accuracy rate. This is clearly a catastrophic inflation in the apparent quality of a network, which may have extracted no meaningful knowledge from the data. This is worsened by the fact that such poor classifiers are a likely outcome, since severely imbalanced datasets are difficult to account for, even when proportionally weighting positive and negative contributions to the loss function.

Precision and recall are much better metrics here. Precision measures the ratio of true positive (TP) predictions to total positive predictions. Recall measures the ratio of TP predictions to the total size of the positive class.

$$\text{Precision} = \frac{\text{\# True positives}}{\text{\# True positives + \# False positives}}, \tag{10.3}$$

$$\text{Recall} = \frac{\text{\# True positives}}{\text{\# True positives + \# False negatives}}. \tag{10.4}$$

These metrics have the advantage that they intrinsically account for class imbalance, and provide a more detailed view on how the classifier is being optimised. For instance, if precision is high but recall is low, the classifier can be said to be cautiousd, only selecting a few examples to mark positive when it has high certainty for

[2]Interestingly, the sequence of differentiable operations which are performed on a tensor from backends such as PyTorch or Tensorflow are called "computational graphs". We omit this usage of the term here to avoid confusion.

[3]Indeed, initial exploratory studies performing link prediction on fully connected graphs consistently ran into memory consumption with NVIDIA Volta V100 GPUs at 16 GB RAM. This was rendered possible once we switched to NVIDIA Quadro RTX 8000 with 48 GB RAM.

correctness, leaving many ambiguous cases out. In contrast, if recall is high and precision is low, the model is unscrupulous, and will catch many true positive labels by over-zealously labelling faintly promising examples as positive, which introduces lots of background.

However, this is still confused by the fact that there is no clear understanding of what the number of pairwise targets represent in terms of cluster constituents counts. The paper quotes a recall for link prediction of 89.6%, and a precision of 90.8%, but what does this mean?[4] Both particles they connect are in a cluster, but it isn't clear that this is any more discriminating for clustering quality, since the edges have no clear meaning.

We address these concerns in input data structure, architecture, and optimisation choices.

10.2 Architecture

We implement a GNN which is IRC safe by construction, identifying Higgs cluster constituents via node prediction. The algorithm used to form embeddings over graph is heavily inspired by recent work on EWMP networks (Konar et al., 2022) described in chapter 5.5.1.

The GNN is composed of 4 energy-weighted message-passing layers, followed by a *bright edge classification* layer which performs node classification, described in chapter 10.2.2. At the input layer, the node's features are 3-vectors composed of the direction-only components of the four-momenta for each detector-level particle in the event, described more in chapter 10.2.1. The initial edge features are populated by the ΔR_{ij} values, computed between node i and node j, however preliminary investigations did not show this to produce any benefit when compared with no initial values for the edge features.

The GNN was trained to reproduce node-level targets identifying whether or not they are descended from a Higgs boson in the hard collision event, see chapter 9.

10.2.1 Input graph structure

The nodes of the input graph to the GNN encode particle data via the nodes. Each node represents a detector-level particle, and has an associated feature vector

[4]The paper actually frames these as new metrics which they call "edge efficiency" (recall) and "edge purity" (precision), but the definitions are identical to the standard classification metrics.

containing the direction-only components of the particle's four-momenta. That is,

$$\mathbf{v}_i^{(0)} = [\hat{x}_i, \hat{y}_i, \eta_i] \,, \tag{10.5}$$

where $\mathbf{v}_i^{(l)}$ represents the node embedding after passing the l-th layer of the GNN, and $\hat{x}_i = x_i / p_{Ti}$ (likewise for $\hat{y}_i$). We deviate from the original EWMP network, passing the x and y components of the momenta directly rather than handling the ϕ values, to avoid any potential issues on the periodic boundaries of ϕ.

However, we follow the EWMP network prescription for the edges, which are determined by utilising a fixed neighbourhood radius about each node, *ie.*

$$A_{ij} = \begin{cases} 1 & \Delta R_{ij} < R_0 \\ 0 & \text{else.} \end{cases} \tag{10.6}$$

The implied self-loops in this definition are allowed, as they are a necessary condition for IRC safety (Konar et al., 2022). This generally leads to a graph with disconnected subcomponents. Although this means that information diffusion for the message-passing steps are constrained within each subcomponent, it has the benefit of scaling well to events with a large number of nodes which eliminates the high memory consumption of learning of fully connected graphs (Ju and Nachman, 2020). However, this means that the GNN instead learns structural motifs within events, rather than having a global sense of the "whole event". The issue of the network losing the ability to interpret global features of the event is partially mitigated in the methods outlined in chapter 11.3.

If we concatenate the features of all nodes in the graph, we form a feature matrix $\mathbf{V}^{(l)}$, alongside the adjacency matrix $\mathbf{A}$. We will refer to the overall graph data structure passed through our GNN at a given layer as $\mathcal{G}^{(l)} = (\mathbf{V}^{(l)}, \mathbf{A})$.

The input graph $\mathcal{G}^{(0)}$ goes through a data augmentation step. Specifically, the momentum input from the node features are randomly reflected in both the z-axis, and the direction of the ϕ-axis. Additionally, the momenta of the particle point cloud is rotated about the z-axis by an angle $\delta\phi \in [0, 2\pi)$.

For a visual depiction of the input graph, with corresponding predictions of our model, see figure 10.1.

10.2.2 Bright edge activation

For the reasons outlined in chapter 10.1.2, we wish to establish particle reconstruction / clustering as a node-level prediction task. In the link prediction approach, classification is performed over all possible pairs of particles in an event, predicting

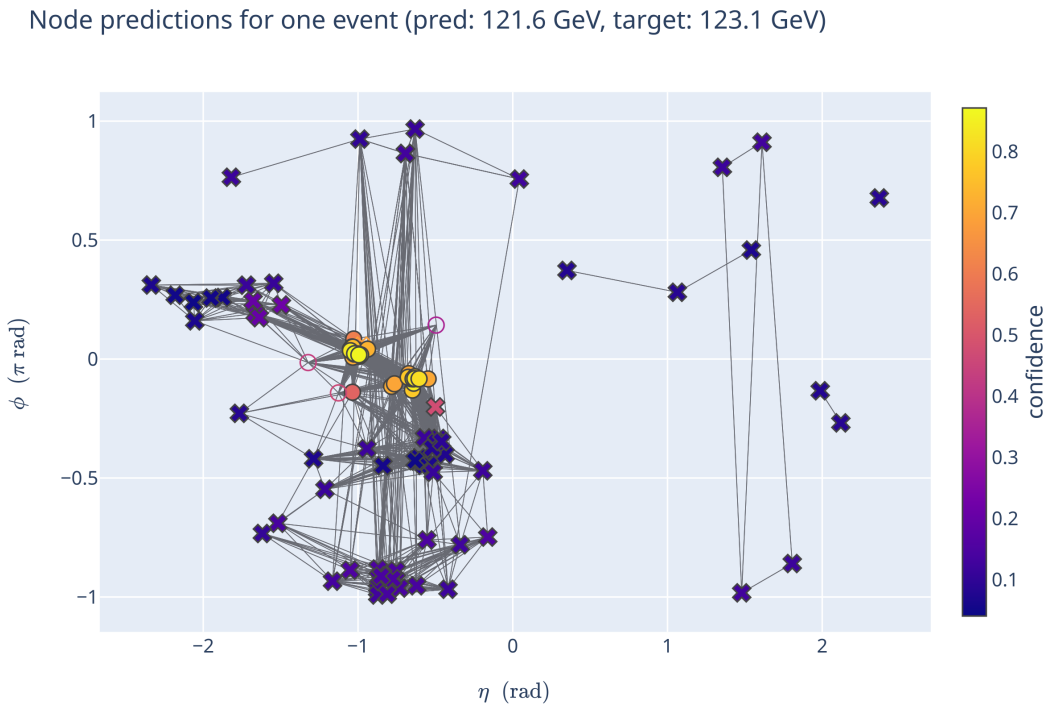


Figure 10.1: This graph shows an event which is structured as a graph input structure. Here our model's output predictions are superimposed, where the node illumination / confidence score is given by the colour. The shape of the markers indicates the supervision targets, where crosses represent UE and circles represent Higgs descendants. If the markers are filled, the model correctly predicted their class, and if the marker is empty the model incorrectly predicted the class, *eg.* an empty circle represents a false negative.

both elements in a given pair belonging to the reconstructed cluster, see equation 10.2. The memory expense of calculating differentiable high dimensional edge embeddings across a fully connected graph of order $\mathcal{O}(10^5)$ edges is substantial, and the results are hard to interpret. Preliminary studies attempted node-level prediction based on applying a classification layer to the node's final embeddings, however results at this stage were not promising.

The learned edge embeddings provide a rich description of the relationships across the graph. These relationships are more numerous than the node features, and naturally provide a higher resolution view on the graph structure. Instead of relying on the node features from normal message passing to capture this same level of detail as in the link prediction case, we develop an intuitive analogy for the edge embedding to map translate the meaning behind their activations into node activations. We name this approach *bright edge classification*.

Consider the setup of predicting the "true edge" targets, as defined in equation 10.2, from the perspective of trying to determine if a given node i is contained within the

Higgs cluster. If any incoming edge connecting the node to a neighbour j is positive, this means both nodes are members of the Higgs cluster. Since we are focusing on classifying node i, we discard considerations of node j entirely, and only consider node i, and the incoming edge $\mathbf{e}_{ij}^{(N)}$, where N is the last GNN layer of the network.

The link prediction technique involves computing classification scores for $\mathbf{e}_{ij}^{(N)}$. Using a Sigmoid activation function, if the edge classification score surpasses 0.5, both particles i and j are added to our Higgs cluster. This is repeated over the whole graph structure. Inevitably, there are a large number of duplicate particles in the Higgs cluster, due to double counting when multiple edges incident on the same node. However, from any given node's perspective, if just one incoming edge exceeds the 0.5 threshold for its score, the node must belong to the Higgs cluster.

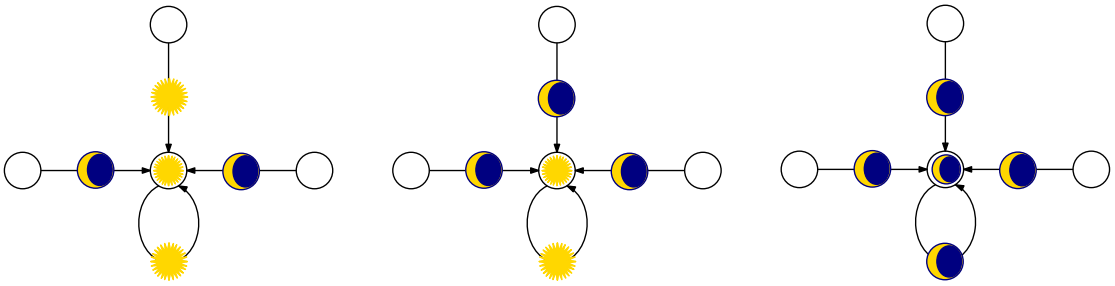
We reframe this statement by analogy with fibre optic cables incident on a photovoltaic (PV) cell. If a very sensitive PV cell, in otherwise total darkness, has several incoming fibre optic cables, and each of these may be illuminated, the cell produces a current if one or more fibre optic cables are illuminated. We say a node can have incoming "bright" or "dark" edges, and if any one edge is bright, it will "illuminate" the node. An illuminated node is taken to be a member of the positive class in node prediction; in this case, a constituent of the Higgs cluster. A node is only dark if all incoming edges are dark, see figure 10.2 for a schematic of this.

In order to make this mathematically explicit, a dark edge logit e_{ij} would be one which would give $\text{Sigmoid}(e_{ij}) < 0.5$, which implies $e_{ij} < 0$. Thus positive edge logits are bright, negative edge logits are dark. So, how do we map this to node illumination?

We could sum the incoming edges; however this would have the undesirable interpretation of dark edges having negative brightness. Dark edges could then act destructively against bright edges, which is explicitly what we are trying to avoid. This is easily solved by applying a `ReLU` activation equation 5.2 to our incoming edge logits before performing the sum. Illumination from edges is therefore always constructive, rather than destructive.

In practice, many incoming dark edges from a dense neighbourhood of particles from the UE may suggest we should be more sceptical of particles with very few weakly bright incoming edges. This is because the collimated structure of QCD emissions implies that hard partons should give rise to dense clusters in the detector.

We could apply a `LeakyReLU` to let some "negative light" in to dim the node. Recall that a hyperparameter α is chosen for the gradient of negative inputs, rather than assuming it to be zero, see equation 5.3. By selecting larger values of α, we could effectively choose how much to allow dark edges to influence the classification result on the node.



(a) A node illuminated by incoming bright edges.

(b) A node illuminated by itself via a self-loop.

(c) A dark node, which has no incident bright edges to illuminate it.

Figure 10.2: Cartoon depiction of the node illumination from incoming bright edges. If just one incoming edge is bright, then the node itself is illuminated / bright. Otherwise, the node is dark. A node may self-illuminate via a self loop.

Rather than biasing the activation layer with assumptions regarding how sensitive the output should be to incoming dark edges, we instead apply a PReLU activation to each logit in the sum. As we discussed in chapter 5.1.2, α becomes a learnable parameter, so takes on an appropriate value during training based on the data itself.

Therefore, the bright edge activation for a given node i is given by

$$y_i = \text{BrightEdgeClassifier}(i) = \text{Sigmoid}\left(\sum_{j\in\mathcal{N}(i)} \omega_j^{(\mathcal{N}[i])}\,\text{PReLU}\left[\frac{1}{K}\sum_{k=1}^{K}\phi_k(\mathbf{e}_{ji})\right]\right), \tag{10.7}$$

where ϕ_k^e are update functions which map high-dimensional edge embeddings to scalar logits. This is then mean averaged over K heads in order to improve stability, such that there are K updates for each head, inspired by multi-head attention (Veličković et al., 2018). We weight incoming edge logits by the relative p_T of the incoming node with the sum of node i's neighbourhood p_T, $\omega_j^{(\mathcal{N}[i])}$, to ensure the bright edge classifier is IRC safe.

Not only does this map the link prediction task proposed by Ju and Nachman (2020) to a node prediction task, but it makes the model's confidence for an individual node's membership within a cluster continuous and differentiable. This enables analysis of this confidence distribution, and confidence-weighted metrics to be used during training and evaluation.

10.2.3 The feed-forward mechanism and hyperparameter tuning

We train a GNN with 4 graph layers. During the edge embedding step, source and destination node features are concatenated, and passed through batch normalisation.

Dropout is applied with a probability of 6.6×10^{-35}. The result is concatenated with existing edge features, and passed through an MLP update function to produce a new edge embedding.

Dropout is a regularisation technique which prevents overfitting with specific nodes of a NN. It achieves this by randomly eliminating nodes with a certain probability during training, forcing the NN not to rely too heavily on any given node. During inference, the dropout is then removed, and all nodes are active. Remarkably, it is conceptually equivalent to training multiple NNs with different structures simultaneously, and taking the averaged result, since with dropout nodes are effectively removed from the NN, then added back at inference time.

The node embedding is computed by forming a message for each node from a p_T-weighted sum of incident edge embeddings, described by equation 5.31. The node's current feature vectors are then concatenated with the message incoming message, and a node update MLP is applied to obtain the new node embedding. The edge and node embedding steps define a single GNN layer.

These layers are stacked, and a final bright edge classification layer with 8 heads is applied to obtain the node logits. Even though the class balance is substantially more even since we formulated the task as node prediction, the majority of particles are still background. Further, many of these background particles are often easy-to-spot, since they exist in sparse neighbourhoods over wide ranges of the $\eta - \phi$ plane. Hence, we apply a focal loss equation 10.8 to address the class imbalance and de-prioritise the optimiser from responding aggressively to positive / negative predictions made with high confidence, *ie.* the easy predictions.

Focal loss is given by

$$\mathcal{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t), \tag{10.8}$$

where $p = \text{Sigmoid}(x)$, and for a target y, $p_t = p$ if $y = 1$, or $p_t = 1 - p$ if $y = 0$; likewise, $\alpha_t = \alpha$ if $y = 1$, or $\alpha_t = 1 - \alpha$ if $y = 0$. This ensures that α_t adjusts the relative loss contribution of each class, helping mitigate class imbalance; $\gamma \geq 0$, and setting it to higher values down-weights the loss so that well-classified examples (with high p_t) contribute less, concentrating learning on harder, misclassified samples. See the `torchvision` source code on `sigmoid_focal_loss()` for implementation details (maintainers and contributors, 2016).

So as training progresses and the low-hanging fruit have been picked, the model can focus on the harder classifications. We choose values of $\alpha = 0.45$ based on measured class imbalance on the Higgs dataset, and choose a value of $\gamma = 2.0$ to set the strength of focus on the challenging examples (Lin et al., 2018).

[5] This value was found via hyperparameter tuning.

We train batches of 256 graphs per step with the Adam optimiser (Kingma and Ba, 2017) at a learning rate of 3.84×10^{-2}, applying weight decay regularisation at 1.0×10^{-4} on the non-PReLU parameters. The model is trained over 80 epochs, and due to the data augmentation, the numerical representation of the node input features is different in each epoch. Model hyperparameters are comprehensively listed in table 10.1.

Table 10.1: Hyperparameters configuring our Higgs reconstruction GNN. Hyperparameters marked with * indicate these were tuned using random search.

Hyperparameter	**Value**
Learning rate*	3.84×10^{-2}
Number of GNN layers*	4
Dropout*	6.63×10^{-3}
MLP nonlinearity	PReLU
Neighbourhood radius*	1.093
Latent node dimension	64
Node MLP depth	2
Latent edge dimension	32
Edge MLP depth	3
Number of bright edge heads	8
Focal loss α	0.45
Focal loss γ	2.0
Weight decay	1.0×10^{-4}
Number of epochs	80

10.3 Dataset generation

Data was generated using Pythia8 version 8.307 and MadGraph5 version 3.5.4. Two datasets were produced, for processes $p\ p \to H^0\ Z$, and $g\ b \to t\ W^-$. Both the Higgs boson and top quark had a generation-level cut applied, such that their respective $p_T \in [500, 550]$ GeV, and their pseudorapidities satisfy $|\eta| < 2.5$. For the Higgs process, Z decays leptonically, and H decays hadronically via the $H^0 \to b\ \bar{b}$ channel. MadSpin is used to ensure that the Higgs decays with a Breit-Wigner mass distribution[6]. A Breit-Wigner distribution is a probability density function used in HEP to model resonances (Wikipedia contributors, 2025b). For the top process, W^- decays leptonically, and t decays hadronically via the $t \to b\ W^+$ channel. The

[6]MadSpin is a tool used in conjunction with MadGraph that can model particle decays while preserving spin correlation information. However, we only use it as it allows for us to preserve the physical Breit-Wigner distributions for the Higgs boson resonances

generated datasets are split into 1M:100k:100k events for training, validation, and testing, respectively.

Data was processed using various tools, including `graphicle`, see chapter 7.2, and `FastJet` (Cacciari and Salam, 2006; Cacciari et al., 2012) for comparisons with standard pipelines. Graphicle clusters are computed and stored as boolean arrays alongside the final state particle data, which does not have detector-effects applied. Rather than performing an expensive detector simulation for this exploratory study, we apply kinematic cuts to the particles such that their $p_T > 0.5$ GeV and $|\eta| < 2.5$. This is used to crudely approximate the threshold sensitivity of a detector calorimeter, and the detector's range along the beam axis. Events are stored using the `heparchy` interface to HDF5 (Chaplais, 2025b).

10.4 Results

We find the model successfully outperforms anti-k_T on our test dataset when comparing against the invariant mass reconstruction histograms[7], see figure 10.3. Anti-k_T is used to cluster the resulting b quarks from the decay channel $H^0 \to b\bar{b}$, where Monte-Carlo truth information for the location of the hard $b\bar{b}$ pair in the $\eta - \phi$ plane is used to tag the b jets. No grooming is used on the b jets.

The model is evaluated on traditional metrics describing both the classification performance, and the resulting momentum reconstruction. Precision and recall have already been described, and we additionally use F1 scores as the harmonic mean between them to determine if one is being traded off for the other. We achieve an accuracy score of 93.7%, where 65% is expected for a totally random classifier. Additionally, an impressive precision score of 95.2% is obtained by our model. The recall score is lower, at 88.7%, suggesting that the model is prioritising noise reduction over capturing the full signal. Even so, an F1 score of 91.8% is encouraging, suggesting that the imbalance between precision and recall is not severe. We compute the mass of every reconstructed cluster, and compute the mean absolute error (MAE) in mass as 24.1 GeV. These metrics are summarised in table 10.2.

The model was trained for 80 epochs over the 1M event training set, which took 24 hours on an NVIDIA GTX 1080 Ti consumer GPU, consuming on average roughly 11 GB of GPU memory. The model was tested over the 100k event test set, which took $\sim$ 2.5 minutes on an NVIDIA RTX 8000 GPU[8].

[7] There are other performance metrics for jet reconstruction, *eg.* Jet Energy Resolution, Jet Reconstruction Efficiency, Jet Matching Efficiency, *etc.* but these are beyond the scope of this work.

[8] GPU memory consumption was not recorded for any of the testing runs.

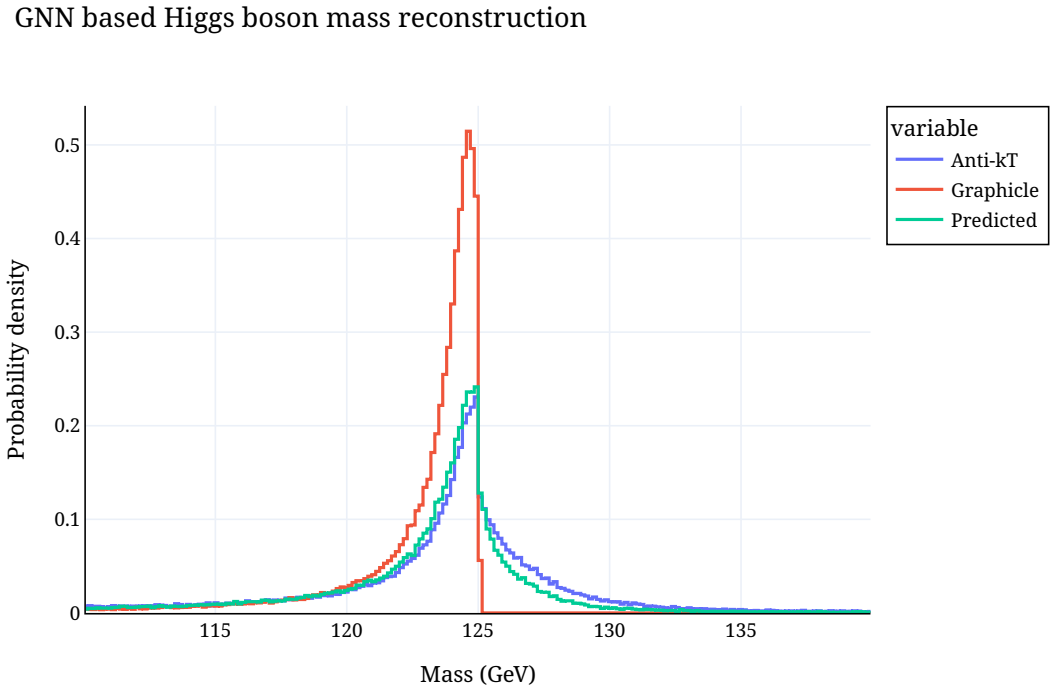


Figure 10.3: Comparison of mass reconstructions of boosted Higgs bosons. The graphicle trace represents the target based on ancestry information. The Predicted trace represents the model. Anti-k_T is shown for comparison with standard techniques.

Table 10.2: Model performance metrics, evaluated over the test dataset after training has completed in the final epoch.

Metric	**Value**
Accuracy	0.937
Precision	0.952
Recall	0.887
F1	0.918
Mass MAE (GeV)	24.1

It is more indicative to observe the histograms for the Higgs mass reconstruction. Our model shows modest improvement over anti-k_T in terms of peak sharpness, see figure 10.3. In this figure, we see three mass histograms: our model, the Monte-Carlo truth ancestry based clusters (graphicle), and anti-k_T. While the improvement is not qualitatively large, it is particularly impressive due to the limited expertise required to produce these clusters. Our model precludes the combinatoric challenges of b-jet tagging and matching to form the Higgs, instead yielding clusters in a single shot from the four-momenta across the event.

Note that the Monte-Carlo truth based "graphicle" reconstruction shows a steep cut-off beyond the 125 GeV mark. This is because, prior to applying detector level cuts on η and p_T, the Higgs peak's resolution is very fine. However, following these cuts, a range of lower mass distributions are reconstructed, which broadens the peak. This only occurs in the low mass direction, since removing cluster constituents can never result in reconstructing a higher mass. As this has elicited surprise and scepticism among several senior HEP researchers, I include a brief proof for this in Appendix B.

Finally, we show that our architecture is indeed IRC safe, by computing the confidence-weighted mass for reconstructed clusters. The mass is "confidence-weighted", in the sense that when the logits for node classification are passed through the sigmoid nonlinearity, we obtain a "confidence" value between 0 - 1. Usually a threshold of 0.5 is applied, such that if the particle i is assigned a confidence c_i, the particle is considered a Higgs constituent if $c_i > 0.5$.

If the model were a perfect classifier, it would output values of only 0 and 1, creating a vector with a size equal to the number of detector-level particles in the event, $\mathbf{c}$. The momentum of the cluster could therefore be found by taking a dot product of this vector with the matrix containing the four momenta of the corresponding particles along its rows $\mathbf{P}$, *ie.*.

$$p_\mu^{\text{cluster}} = \mathbf{c} \cdot \mathbf{P}. \tag{10.9}$$

Notice that equation 10.9 generalises intermediate values in the interval $c_i \in [0, 1]$. Therefore, in general, a confidence weighted mass can be computed from this equation, applying a Minkowski inner product to square the result and obtain the mass. We can use this to test the IRC safety of our trained model[9].

We take 10 events from the test dataset. Then, finding the hardest constituent of the Higgs cluster, we split it across a 100×100 grid, shown in figure 10.4. The constituent is split into two particles in a way which conserves total momentum, with varying energy and angular separation. This does not conserve mass. The hardest constituent is rotated within the plane generated by the hardest and softest constituents, up to a maximum angle of π radians. The energy fraction of one outgoing particle is split within the range $[0, 0.5]$, since this completely determines the energy fraction of the other particle, and the effect is symmetric about the 0.5 mark. The confidence-weighted mass is computed before and after the split, so a deviation Δm can be computed.

[9] We use the confidence-weighted mass, since this continuous value is more likely to be sensitive to effects of particle splits, rather than the discrete mask we obtain from applying a threshold to these confidence scores.

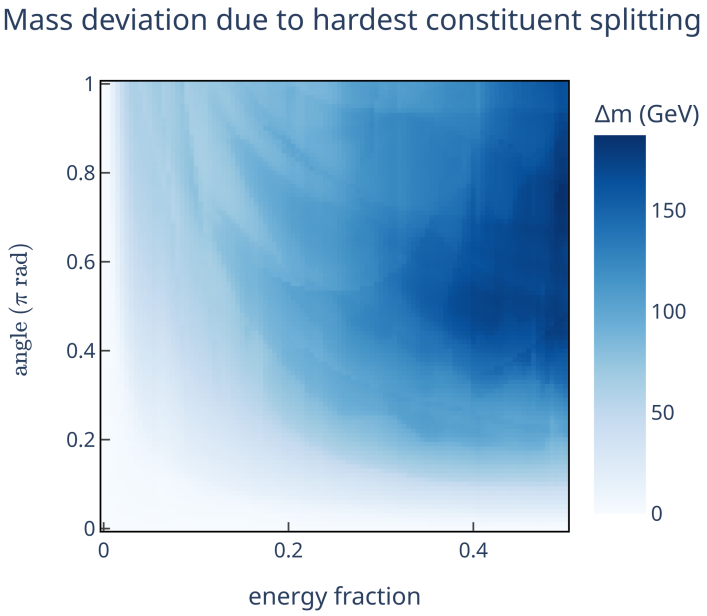


Figure 10.4: Demonstrating the deviation of Higgs boson mass reconstructions produced by our model when the hardest constituent of the Higgs cluster undergoes a split with a range of energy fractions and angles to the original particle.

Figure 10.4 shows a heatmap where each pixel is the mean average of 10 events, each evaluated by our trained GNN. The result we obtain shows that, as emissions tend towards the soft limit (the left hand side of the plot), the confidence-weighted mass deviation tends towards zero. Equally, as we tend towards the collinear limit (the bottom of the plot), confidence-weighted mass deviation also tends towards zero. Hence, we confirm that our architecture satisfies the requirements for IRC safety.

This Higgs study serves as a strong proof-of-concept for our methods. We now move our attention to see if we can apply our Monte-Carlo based approach beyond the case of colour-singlet bosons, specifically focussing on the top quark.

Chapter 11

Top quark reconstruction with cluster double sifting

Ju and Nachman (2020) state that simulation truth data can provide supervision labels for ML models to reconstruct colour-singlet boson ancestors, but go no further. They recognise that there is no unique way of associating hadronic final states with the quark / gluon degrees of freedom which generated them. In this section, we systematically respond to the difficulty of the problem raised by the authors. While we agree that it is not possible to assign supervision labels to detector-level particles identifying a unique quark / gluon ancestor, in chapter 9.2 we show that in the case of boosted top quarks, we may augment topological ancestry with heuristic applications of intermediate state four-momentum data to achieve excellent reconstructions. In this chapter, we describe how we have applied this work to our GNN architecture from chapter 10.

We find our GNN model performance to be comparable with anti-k_T when applied without architectural modification from chapter 10. While this is impressive, since the problem of tagging and jet combinatorics are superseded by the use of a single-shot GNN classifier trained on clustering, we develop a new architectural approach to improve performance. To this end, we introduce *cluster double sifting*, which provides a substantial improvement in performance for top quark mass reconstruction compared with anti-k_T.

To our knowledge, this works represents the only ML algorithm developed capable of top quark reconstruction by identifying the set of detector-level particles which descend from top quarks in $p\ p$ collision events[1]. As such, our results represent a

[1] An CNN based segmentation method of jet images (Choi et al., 2023) was brought to our attention in the late stages of editing this thesis. The paper gives little consideration to IRC safety, and due to information loss in the production of jet images, it cannot identify cluster consituents directly. Instead, the authors must rely on regression techniques to construct the kinematic variables. Strikingly, they appear

landmark step forward in applications of ML to boosted particle reconstruction, beyond colour singlet bosons.

11.1 Dataset generation

The simulation was carried out using MadGraph5 version 3.5.4 for the hard process generation and Pythia8 version 8.307 for parton showering, hadronisation, and modelling of the underlying event.

The hard scattering process was modelled as $p\ p \to t\ W^-$, where the W^- boson was forced to decay leptonically ($W^- \to \ell^-\ \nu_\ell$), while the top quark decayed via $t \to b\ W^+$. The W^+ boson was forced to decay hadronically ($W^+ \to q\ \bar{q}$). These decays were chosen to ensure a well-defined final state suitable for studying the reconstruction of boosted top quarks, without the production of significant noise from the extraneous W^- boson.

MadGraph5_aMC@NLO computed the matrix elements for the hard scattering process, ensuring an accurate description of the kinematics and dynamics of the events. The resulting partonic events were interfaced with Pythia 8.3, which performed the following tasks, whose settings are given in listing 11.1. These are the same settings as shown in listing **??**. In this case, as well, these settings are generic; they just switch on the following physics effects to enable better simulation results:

- **Parton Showering:** Initial and final state radiation (ISR/FSR) were enabled to account for QCD corrections and ensure a realistic distribution of emitted partons.
- **Hadronisation:** The Lund string model in Pythia was employed to convert partons into colour-neutral hadrons.
- **Underlying Event (UE):** Multiple Parton Interactions (MPI) were included to simulate the soft interactions accompanying the hard scattering process.
- **Colour Reconnection** and **Beam Remnants**: These features were enabled to ensure the modern QCD-based model of colour flow between partons was used.

The simulation was performed in the centre-of-mass frame with a collision energy of $\sqrt{s} = 13$ TeV, consistent with the LHC experimental setup. To reduce computational overhead and focus on the desired phase space for boosted objects, generation-level cuts were applied:

- The transverse momentum of the top quark was required to satisfy $500\ \text{GeV} < p_T < 550\ \text{GeV}$.

to have applied similar pre-processing techniques for supervision label construction. A brief comparison may be found in Appendix D.

Listing 11.1 Pythia settings for showering and hadronisation of the top quark events.

```
PartonLevel:ISR = on
PartonLevel:FSR = on
PartonLevel:MPI = on
ColourReconnection:mode = 1
BeamRemnants:remnantMode = 1
Beams:frameType = 4
```

- The pseudorapidity of the W^+ boson was constrained to $|\eta| < 2.5$.

The dataset is divided into three subsets to facilitate training, validation, and testing of our model:

- **Training set:** $1,000,000$ events
- **Validation set:** $100,000$ events
- **Test set:** $100,000$ events

The events were generated without including a detector simulation layer such as Delphes. Instead, the dataset consists of particle-level information, capturing the final state of the particles after hadronisation. The data will necessarily be less realistic without detector effects. The responses of the various components will not be modelled, and the ROOT file based data structures of tracks and towers will not be present in the output, which will mean that adapting our methods for real experimental data will require some additional work. However, this choice allows for direct evaluation of machine learning algorithms against the underlying physical processes, without introducing detector-specific effects, which we feel at this stage would be a distraction.

11.2 Applying the model without modification

We may apply the same architectural schematic from chapter 10 to the Monte-Carlo based top quark supervision labels we obtained in chapter 9. By performing another random sampling hyperparameter search, we obtain the results shown in table 11.1.

The model was trained for 25 epochs, which took 14 hours on an NVIDIA RTX 8000 GPU, consuming on average roughly 40 GB of GPU memory. The model was tested over the 100k event test set, which took $\sim$ 3.5 minutes on the same GPU hardware.

Table 11.1: Results table showing GNN performance metrics of the cluster reconstruction of the top quark from detector-level data.

Metric	Value
Accuracy	0.885
F1	0.857
Precision	0.808
Recall	0.914
Mass MAE (GeV)	79.8

The percentage of elements in the positive class in the top quark data is 13.4% over the test dataset. This means that a classifier could achieve an accuracy of 0.866 from merely learning to predict everything as the negative class. However, the other metrics for precision, recall, and mass show that this is not occurring.

While our recall score at 91.4% appears to be elevated compared with the performance of our Higgs boson reconstruction in chapter 10, the precision has fallen to 80.8% in the case of the top quark reconstruction. This is because the top quark descendant have a considerably greater overlap with UE particles compared to the Higgs. This makes optimising precision particularly difficult.

11.3 Upgrading performance using a double sift

11.3.1 First sift: considering initial results as noise removal

The disconnected graphs produced by our fixed radius definition of adjacency (see equation 10.6) makes it impossible for the message-passing algorithm to learn global event structure, regardless of how deep GNN layers are stacked. To add further insult, despite only having access to these local communities, the input feature vectors of nodes are described in terms of global coordinates. A GNN trained with our current setup must reconstruct the desired cluster, with the additional challenge that it must learn to account for its location in the $\eta - \phi$ plane, without reference to the cluster centroids or the full event context. We can overcome this challenge by defining a new approach to forming the clusters.

It is a standard pre-processing step for ML-based jet-taggers to translate their centre to $(0, 0)$ in the $\eta - \phi$ plane (Dillon et al., 2022; Komiske et al., 2019; Mikuni and Canelli, 2020; Olischläger, 2021). This enables the algorithms to learn the geometric properties of the jets from a standardised location. We propose a similar approach for particle reconstruction. Our results in chapter 11.2 show that the first pass of a GNN can remove most of the UE.

We refer to this first feed-forward pass as a first *sift*, removing much of the UE, but leaving some fine-grained contamination in the resulting cluster. Taking our trained model, we calibrate it to favour high recall over high precision in this step, by reducing the threshold for the node classification scores to below 0.5. Instead of viewing this as our end stage, we consider it to be the first pass in a two stage process. Applying this classification threshold reduces noise, while capturing a large fraction of the signal in the event.

11.3.2 Second sift: shifting and fully connecting the input

Our remaining particles form an intermediate cluster, whose momentum sum approximates the true cluster's momentum. While this may not be precise enough to reconstruct the mass well, it does provide a good baseline to re-centre our coordinates on the $\eta - \phi$ plane. We shift this cluster to be centred on $(0, 0)$, and then pass it to another GNN, whose task is to remove the fine-grained contamination of the cluster. We call this a *double sift*.

Intermediate clusters following a first sift have dramatically fewer nodes ($\mathcal{O}(10)$ *vs.* $\mathcal{O}(1000)$), so it becomes feasible to compute high-dimensional edges over a fully-connected graph.

Formally, we can write our first sift graph embedding as $\mathcal{G}_1^{(N_1)} = \text{GNN}_1(\mathcal{G}_1^{(0)})$, where N_1 is the number of message-passing layers in our GNN, and the numeric subscript refers to the fact that this is the first sift. The second sift involves restricting the included nodes to all those with a positive activation after passing the bright-edge classification layer. We then redefine the adjacency between the remaining nodes to be fully-connected, *ie.* $[\mathbf{A_2}]_{ij} = 1 \; \forall i, j \in [1, N_2]$. Encoding the same direction-only four-momenta components on the nodes to form $\mathbf{V}_2^{(0)}$, the graph passed to the double sift GNN is given as $\mathcal{G}_2^{(0)} = (\mathbf{V}_2^{(0)}, \mathbf{A}_2)$.

We propose that restricting the connectivity of the graph to a radius based node neighbourhood is appropriate in the first sift, since approximate noise removal does not require detailed structural analysis. By fully connecting the intermediate graph, the double sift GNN is able to learn global features using the whole graph, centred on the $\eta - \phi$ plane. This means that the proportion of meaningful edges in the second sift is substantially higher, enabling the GNN model to learn graph structure more effectively, and with much lower computational expense. Due to the much lower background : signal ratio, we posit that the global structure is likely to be much more characteristic for the particle we wish to reconstruct.

In order to check that this motivation is well-founded, we analyse the first sift.

11.3.3 Classifier correlations motivating the double sift approach

Our claim in the previous section effectively reduces to the assertion that we are failing to produce competitive mass reconstructions for the top quark because we are adding noise into our clusters. This is supported by the reduced precision and increased recall scores for the top quark data. To verify that this interpretation is robust, we further analyse the top quark reconstruction after the first sift. Since we have ground truth data, we are able to identify the noise introduced into the clusters, which is simply the false positive (FP) predictions.

We should be able to determine if the model is producing poor clusters due to the inclusion of FP nodes, and it is not compensating for missing cluster constituents in by subtly choosing FPs which make up the right momentum. If we simply remove the FP constituents and the model performance improves, we can determine if there is any significant gains to be enjoyed from attempting to reduce the noise in the two-stage process we described above.

Figure 11.1 shows a striking indication that we are on the correct path. If all of the FP cluster constituents are removed, effectively keeping the recall constant but boosting precision to 100%, our prediction much more closely matches our Monte-Carlo reconstruction. However, if these FP constituents have no consistent distribution, our model may still find it difficult to remove them.

In order to test this, we follow our suggested prescription in chapter 11.3, approximating the momentum of the top quark cluster after the first sift. We then shift this candidate cluster to the origin, and plot heatmaps of its FP density, TP density, and target density (centred on its own basis) on axes with the same scales.

The results in figure 11.2a show that TP and target densities have similar structures, with a well-defined dense core, surrounded by a tight halo of rapidly dropping density. The FP density is also distributed with the highest density at the centre, but it peaks at a substantially lower value compared with the TP and target densities. Importantly, the characteristic decay length is substantially larger, meaning that despite peaking at a lower value in the centre, the outer halo is continues to have a significant density at higher radial distances from the origin than the TP density.

If we plot the p_T deposits of the FP, TP, and target constituents, we see even more encouraging detail, see figure 11.2b. Here the hardness of the constituents is apparent, and we can see that the halo in the FP heatmap carries significant transverse momentum, particularly spreading across the azimuthal direction.

We can characterise the differences simply by subtracting the FP heatmap from the TP heatmap, shown in figure 11.3. Here we see the same characteristic dense core of the TP constituents. However, we note that the heatmap displays the majority of the $\eta - \phi$

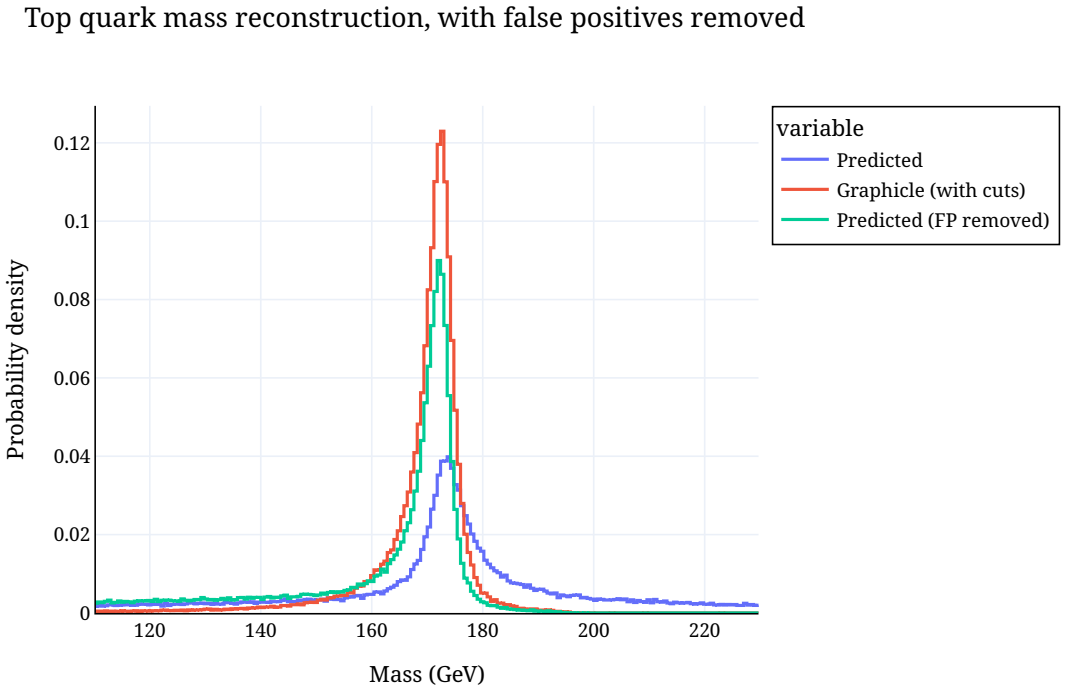
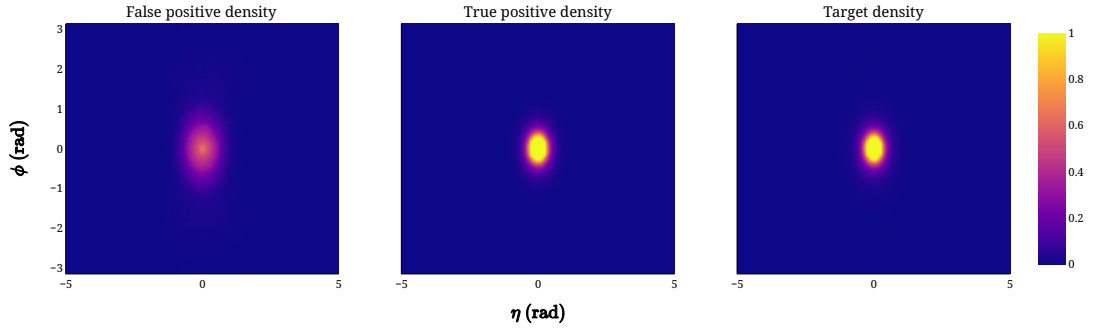


Figure 11.1: Depicting the predictions after a single application (sift) of our GNN model, superimposed with the predictions with the false positive predictions removed, and the ground truth (Graphicle).

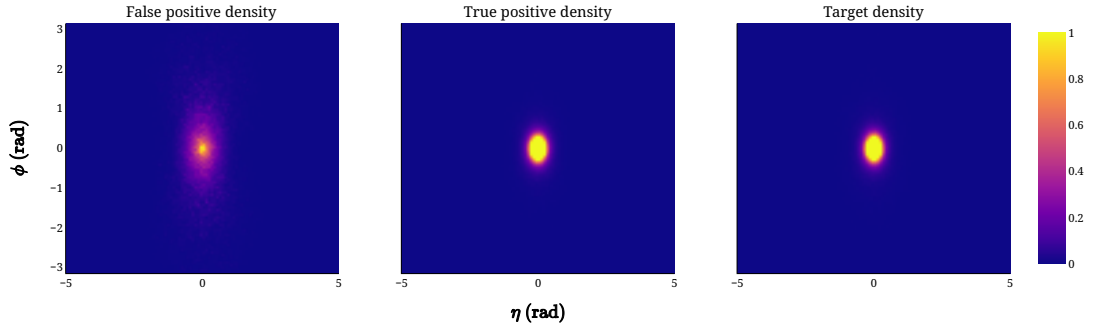
plane with zero density. In this modified figure, a negative halo – darkened with respect to the outer zero-valued background – becomes apparent around the central dense core. This is a smoking-gun indicator that by using our trained model as a first pass to sift away the majority of the UE, we can shift the origin of our cluster, exposing structure in the remaining noise we wish to remove. Simply cutting away the negative halo should remove constituents in a favourable FP : TP ratio, so it seems reasonable to expect the sophistication of our GNN to provide improved results.

11.4 Results

Now we have the impetus to do so, we apply our GNN architecture on the fully connected graphs whose origins have been shifted. Table 11.2 shows the hyperparameters of both sifts for the top reconstruction model. Between one sift layer to the next, hyperparameters were hand-tuned. Due to the reduced memory burden of the noise-reduced graphs, it was possible to increase embedding dimension and MLP depth. The classification threshold in the first sift is the value applied to produce the high recall clusters for the second sift, which uses the normal 0.5 threshold.



(a) Constituent densities across the $\eta - \phi$ plane for the re-centred first sift clusters.



(b) Heatmap of p_T deposits from the cluster constituents across the $\eta - \phi$ plane.

Figure 11.2: 2D heatmaps indicating the distribution of predicted top quark clusters filtered by classifier categories of true positive (TP) labels, false positve (FP) labels, and target supervision labels. The origins are shifted to their respective centres. The TP and FP heatmaps represent the same underlying cluster, so share an origin. The target cluster uses its own origin. These are averaged over to 100k events of the test dataset.

Table 11.2: Hyperparameters configuring our two sifts of the top reconstruction GNN. Hyperparameters marked with * indicate these were tuned using random search. Other hyperparameters were hand tuned to exploit different resource requirements.

Hyperparameter	**First sift value**	**Second sift value**
Learning rate*	9.82×10^{-5}	7.92×10^{-6}
Number of GNN layers*	4	4
Dropout*	0.279	0.279
MLP nonlinearity	PReLU	PReLU
Neighbourhood radius*	1.13	1.13
Latent node dimension	64	128
Node MLP depth	2	3
Latent edge dimension	32	128
Edge MLP depth	3	3

Hyperparameter	First sift value	Second sift value
Number of bright edge heads	8	8
Focal loss α	0.65	0.35
Focal loss γ	2.5	1.5
Weight decay	1.0×10^{-4}	1.0×10^{-4}
Number of epochs	25	16
Classification threshold	0.35	0.5

The results are displayed in table 11.3. We see no substantial change in the accuracy score, and a reduced recall, which contributes to a reduced overall F1 score. However, the precision does show a marked improvement, which contributes to an overall lower mean absolute error in the mass.

The model was trained for 15 epochs over the 1M event training set, which took 60 hours on an NVIDIA GTX 1080 Ti consumer GPU, consuming on average roughly 11 GB of GPU memory. The model was tested over the 100k event test set, which took $\sim$ 4 minutes on an NVIDIA RTX 8000 GPU.

Table 11.3: Results of the model following double sift classification.

Metric	Value
Accuracy	0.884
F1	0.849
Precision	0.840
Recall	0.858
Mass Mean Absolute Error (GeV)	50.7
Mass Median Absolute Error (GeV)	33.5

Cluster double sifting has contributed towards equalising the precision and recall scores. The improved effect on the mass reconstruction can be observed directly from figure 11.4.

We see that the mass reconstruction improves upon our single sifted result, and offers a substantially better reconstruction than anti-k_T. We believe these results show a promising future for GNN-based particle reconstruction methods. We have exceeded the performance of standard methods in reconstructing top quarks from individual constituents in the detector-level. What's more, we have done this without needing the intervention of top taggers, bottom taggers, or to solve the combinatorial challenge of matching jets to the decay products of the top quark.

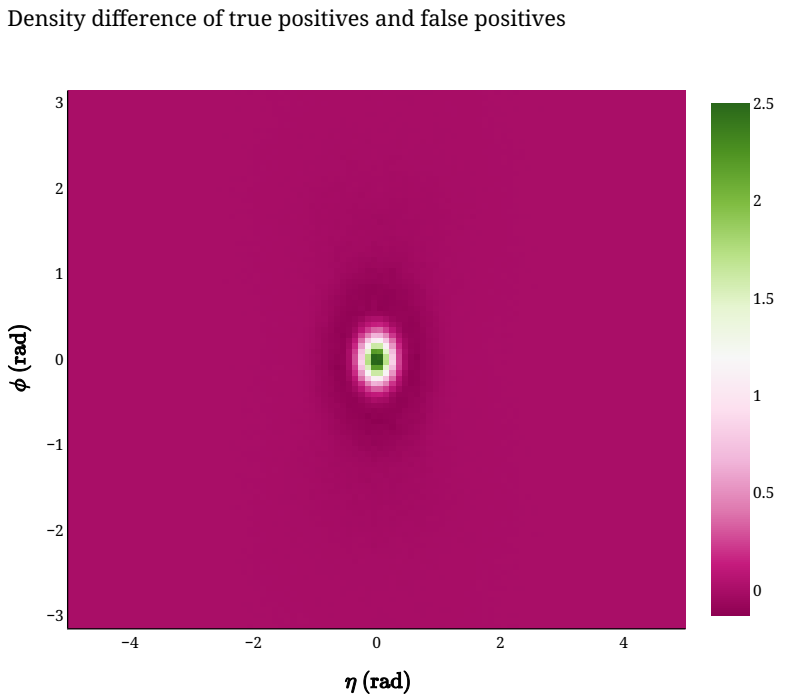


Figure 11.3: Heatmap of the difference between true positive and false negatives. Negative regions indicate locations in the $\eta - \phi$ plane where FNs are more common than TPs.

11.5 Further work

There is even some indication that it may be possible to extract error estimates from the model's confidence scores. Figure 11.5 shows a series of histograms, each displayed as a row in a heatmap. Each row represents the distribution of confidence scores of the model along the horizontal direction. These are grouped by events in which the mass reconstruction had an error within a certain range. These ranges are split up into percentiles. This is not an absolute error, but a signed relative error, so a perfect mass reconstruction would be displayed at the 50% mark. 0% - 5% refers to the extreme lower bound of mass underestimates, and 95% - 100% represents the extreme upper bound for mass overestimates.

It is hoped that the model might be able to recognise when it is fudging its predictions, and give more hedged predictions. This appears to be exactly what figure 11.5 shows. In particular, notice the low density in the central region of the heatmap. This suggests that when the mass reconstruction is performing well, the model is more confident about its predictions. The fact that this decays away as the mass reconstruction error increases both above and below the 50% mark indicates that the model recognises these are less clear cases, and adjusts its confidence to lower values accordingly.

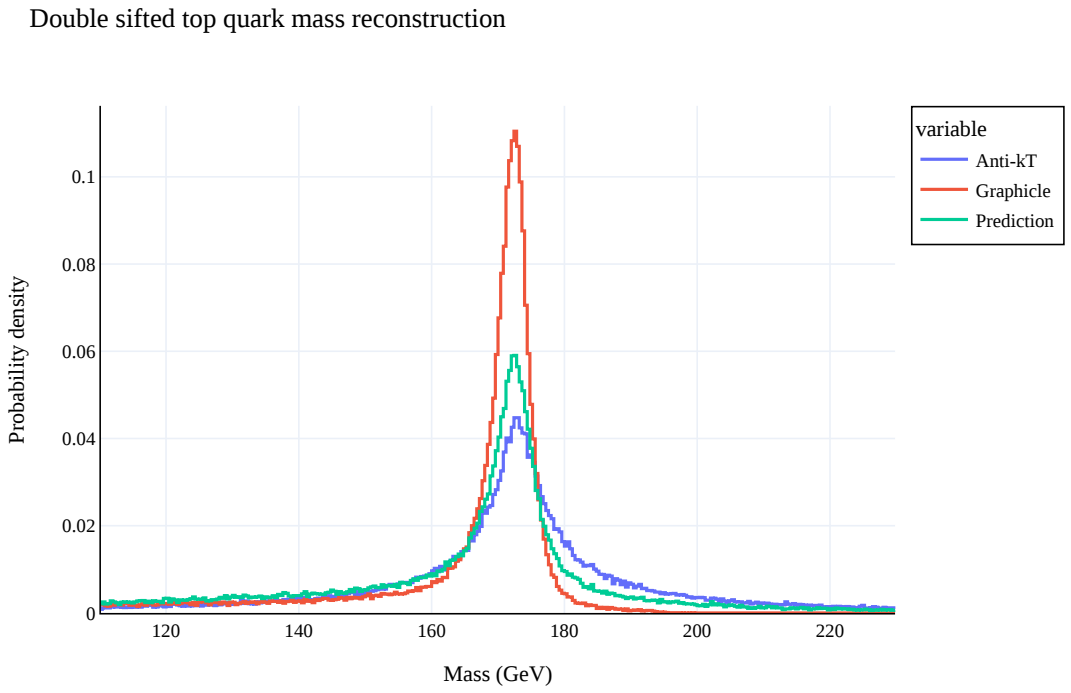


Figure 11.4: We compute histograms to perform a comparison of the top-quark mass reconstruction, over the 100,000 events of the validation dataset. Graphicle's clusters are formed using the complete event record, and then kinematic cuts of $|\eta| < 2.5$ and $p_T > 0.5$ are applied. The cutoff radius for wide angle radiation used by graphicle is 1.0. Anti-k_T is applied over the final state particles with the kinematic cuts already applied. We pass $R = 0.35$, and use a Monte-Carlo truth based tagger to identify the b and $q\,\bar{q}$ jets. Predicted shows the performance of our GNN model with a double-sifting approach, trained on the graphicle clusters. The resolution of the graphicle mass histogram shows a striking improvement to resolution, when compared with the anti-k_T approach. Our model shows that it is able to learn patterns in these improved clusters, resulting in a significant improvement in resolution to the anti-k_T clusters, as well. In particular, the double-sifting approach reduces the tail of high mass reconstructions observed both from single-sift and anti-k_T approaches.

One possibility could be to produce a heatmap such as figure 11.5 on the test dataset after training the GNN model. Then, when the model is evaluated on new examples, a histogram of its confidence scores might be matched to the closest row in the heatmap, which would approximate the error.

Regression approaches with deep learning are also possible. Again, one could produce a confidence score heatmap as in figure 11.5, storing annotating each individual event with the error. Then it would be possible to aggregate the node-level and / or edge-level features to obtain a global embedding for the event, and use this as an input to a MLP which undergoes supervised learning with the mass errors as targets.

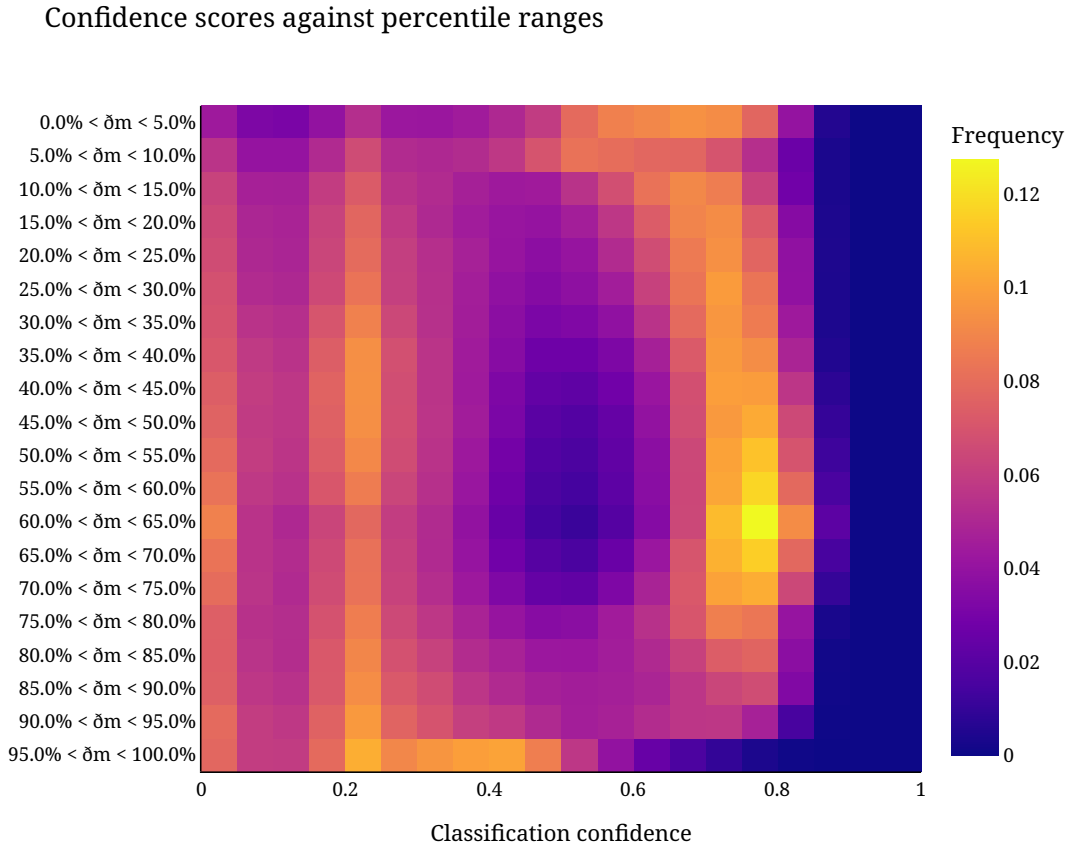


Figure 11.5: Heatmap representing the model's confidence at various levels of error, where a perfect reconstruction is at $\delta m = 50\%$. Each row are the distribution of confidence scores for the nodes in all events within that mass reconstruction error range.

It would be interesting to explore whether the application of CNNs to analyse the heatmaps directly could provide meaningful error bars on the reconstructed clusters. We leave these ideas for future work, but figure 11.5 shows great promise to provide these important experimental insights.

Chapter 12

Conclusions

This thesis has explored the application of GNNs for reconstructing boosted objects at the LHC. The structured data inherent in collider physics simulations has been utilised to produce high-fidelity labelled data, suitable for various supervised ML pipelines. GNNs in particular have demonstrated their utility by reproducing these labels on individual detector-level particles, successfully reconstructing Higgs and top quark clusters without the need for jet tagging. The work presented here has offered a exploratory analysis of the theoretical underpinnings, software contributions, and experimental validations that define this approach.

12.1 Summary of key findings

1. **Graph representations for collider data**: the introduction of graph-based data representations, specifically tailored for high-energy physics, has enabled more expressive and flexible modelling of particle interactions. By encoding detector-level particles as graph nodes and their relationships as edges, GNNs have shown significant promise in capturing the complex dependencies in collision events.
2. **State-of-the-art performance in reconstruction tasks**: utilising GNN architectures tailored for boosted object reconstruction, this thesis has demonstrated superior performance in reconstructing the mass distributions of Higgs bosons and top quarks. Techniques such as bright-edge classification and cluster double sifting have further refined the clustering process, improving downstream intelligibility, reducing noise, and enhancing resolution.
3. **Hierarchical Label Generation Using Monte Carlo Data**: The exploitation of ancestry information from Monte Carlo event generators to create high-quality supervision labels represents a significant methodological advancement. By bridging the gap between theoretical event generators and experimental

detector data, this approach has improved the training and evaluation of machine learning models in particle physics. These methods have been made easily accessible as free and open software on GitHub and the Python Package Index (PyPI), see:

- `graphicle`, (Chaplais and Cerro, 2025)
- `colliderscope`, (Chaplais, 2025a)
- `showerpipe`, (Chaplais, 2025c)
- `heparchy`, (Chaplais, 2025b)

12.2 Contributions to the Field

The development and validation of graph-based methods for collider data analysis contribute to the broader adoption of machine learning techniques in high-energy physics. Specific contributions include:

- **Software ecosystem**: the creation of `graphicle` and related tools provides an extensible framework for handling heterogeneous particle physics data. By combining semantic data structures with efficient relational querying, this ecosystem supports the needs of both experimentalists and theorists at their computational intersection.
- **Methodological innovations**: the application of IRC-safe message passing techniques to IN architectures, and novel simulation-based clustering techniques, has expanded the applicability of ML in HEP. This has provided robust solutions to longstanding challenges in event reconstruction, particularly regarding reconstruction of hard partons possessing QCD charge.
- **Theoretical insights**: by integrating principles of QCD into ML pipelines, this thesis has highlighted the potential for cross-disciplinary approaches to yield significant scientific insights.

12.3 Future Directions

While the results presented here mark significant progress, several avenues for further research remain open:

1. **Generalisation across datasets**: extending the methods to other physics processes, such as the $pp \to t\bar{t}$ decay channel, will validate their robustness and explore their generalisability. In particular exploring to what extent trained models can be fine-tuned for these different decay channels.

2. **Real-time applications**: Investigating the deployment of GNN-based algorithms in real-time event reconstruction pipelines at the LHC could enable more efficient data processing and trigger systems.
3. **Diverse training data**: Bridging the gap between simulation-based training and real detector data remains an important goal. Future work should focus on aligning with experimental tracks and towers instead of particle-level data, while incorporating systematic uncertainties and detector effects[1]. At the same time, we could make resourceful use of our existing simulated data by introducing noise. This would enable uncertainty estimates and allow us to perform robustness studies on our GNN models.
4. **Exploration of new architectures**: other research has shown that GNNs may be further extended to leverage rotational symmetry about the longitudinal axis, and provide Lorentz equivariance. These inductive biases could further enhance performance and theoretical consistency. Additionally, given that the second sift in the clustering process uses a fully connected graph, the GNN setup is analogous to a transformer architecture. Cross-pollinating the rapid advancements of transformer-based models into cluster double sifting, driven by the current boom in large language models, could offer further directions for improvement.

12.4 Final Thoughts

The application of GNNs to event reconstruction at the LHC represents a transformative shift in how particle physics leverages machine learning. This will be particularly important as we prepare for the upgraded HL-LHC, which will result in far more dense and complex data, which GNNs show promise in scaling well against (Thais et al., 2022). By encoding the rich structure of collider data into graph representations, this work has demonstrated the ability to reconstruct boosted particles without relying on traditional jet clustering algorithms at any stage of training or inference. Importantly, these methods introduced a novel approach to producing clusters of hard partons with a QCD colour charge, advancing beyond previous work that primarily focused on colour-singlet bosons. As the field continues to evolve, the methods and findings of this thesis will serve as a foundation for future innovations, advancing the search for new physics and the understanding of fundamental particles and forces.

[1] Additionally, great deal of simulated and experimental used for studies at CERN is collated on https://opendata.cern.ch/. These would serve as valuable benchmarks for future studies.

Appendix A

Converting `motherList` to COO edges

The algorithm for extracting particle ancestry as a Directed Acyclic Graph (DAG) in COO adjacency list format operates in a series of clearly defined phases. It takes as input a list of particles, each of which has methods for accessing its parents and children, and produces a COO representation of the particle ancestry structure.

Step 1: Initialization

The algorithm begins by initializing three key structures:

1. **`parents`**: A list to store the indices of all particles.
2. **`children_groups`**: A list of sorted child indices for each particle. Particles without children are assigned a unique pseudo-negative ID.
3. **`rooted_ids`**: A list of particle indices for those with no parents, representing the roots of the DAG.

Additionally, a vertex map (`vertices`) is created as a dictionary to associate groups of child particles with their corresponding parent particles.

Step 2: Group Parents and Children

For each particle in the input list:

- The particle's index is added to `parents`.
- If the particle has children, their indices are sorted and stored as a tuple in `children_groups`. Particles without children are assigned a pseudo-ID to signify their leaf status.
- Particles with no parents are added to `rooted_ids`.

A root vertex is then explicitly added to the `vertices` map, linking the collected `rooted_ids` to a special vertex ID of 0.

Step 3: Construct Vertices

Using the `children_groups` and `parents` lists, the algorithm iterates over each particle to populate the `vertices` map. This map associates each unique child group with the parent particles involved in that interaction.

Step 4: Assign Vertex IDs

Vertex IDs are assigned sequentially to each entry in the `vertices` map. Two dictionaries, `incoming_dict` and `outgoing_dict`, are created to map particle indices to their corresponding vertex IDs:

- **Incoming particles** use the vertex as their destination (`dst`).
- **Outgoing particles** use the vertex as their source (`src`).

Step 5: Generate Edges

Using the mappings from `incoming_dict` and `outgoing_dict`, the adjacency list is constructed by pairing source (`src`) and destination (`dst`) vertex IDs for each particle. This ensures that the graph representation reflects the original ordering of particles provided in the input.

Step 6: Output

The resulting edges are stored in a structured NumPy array, ensuring compatibility with downstream analyses. The COO adjacency list is returned as the final output.

Algorithm 1: Extract Particle Ancestry as a DAG in COO Format

Input: Particle list $\mathcal{P}$, where each particle has parent and child access methods.

Output: COO adjacency list $\mathcal{E}$ representing the DAG.

Initialize empty lists: `parents`, `children_groups`, `rooted_ids`;
Initialize an empty vertex map `vertices` $\leftarrow \{\}$;
foreach *particle* $p_i \in \mathcal{P}$ **do**
    Add p_i.`index`() to `parents`;
    if p_i *has children* **then**
        Add sorted child indices as a tuple to `children_groups`;
    else
        Add $-i$ to `children_groups` (pseudo-leaf ID);
    if p_i *has no parents* **then**
        Add p_i.`index`() to `rooted_ids`;
Add a root vertex to `vertices` with `rooted_ids` mapping to ID 0;
foreach (*children*, *parent*) *in* (*children_groups*, *parents*) **do**
    Append `parent` to `vertices`[`children`];
Assign vertex IDs to `vertices` and build mappings `incoming_dict` and `outgoing_dict`;
Generate edges by pairing `incoming_dict` and `outgoing_dict` for all `parents`;
Store the edges $\mathcal{E}$ as a structured array;
return $\mathcal{E}$;

Appendix B

Proof: removing cluster constituents lowers mass

Consider two momenta, $p_a = (E_a, \vec{p}_a)$ and $p_b = (E_b, \vec{p}_b)$. Their masses are m_a and m_b, respectively. The mass generated by their sum is M_{ab}.

Lemma: the mass M_{ab} generated by combining momenta is always greater than the mass of the individual constituents.

That is, removing recombined constituents never results in an increased mass, or $M_{ab} \geq m_a$ and $M_{ab} \geq m_b$.

Mass m is found by taking the difference of squares between mass and three-momentum,

$$m^2 = E^2 - |\vec{p}|^2. \tag{B.1}$$

For all timelike or lightlike particles:

$$E \geq |\vec{p}|. \tag{B.2}$$

Also,

$$|\vec{p}_a + \vec{p}_b| \leq |\vec{p}_a| + |\vec{p}_b|, \tag{B.3}$$

because there may be some component-wise cancellation in the magnitude of the sum, not present in the sum of the magnitudes of 3-momenta.

So, if

$$M_{ab}^2 = (E_a + E_b)^2 - |\vec{p}_a + \vec{p}_b|^2, \tag{B.4}$$

then

$$\begin{aligned} M_{ab}^2 &\geq (E_a + E_b)^2 - (|\vec{p}_a| + |\vec{p}_b|)^2 \\ &\geq E_a^2 - |\vec{p}_a|^2 + E_b^2 - |\vec{p}_b|^2 + 2(E_a E_b - |\vec{p}_a||\vec{p}_b|). \end{aligned} \tag{B.5}$$

But we notice that $E_a E_b - |\vec{p}_a||\vec{p}_b| \geq 0$, due to equation B.2, which implies:

$$M_{ab}^2 \geq m_a^2 + m_b^2, \tag{B.6}$$

and consequently that both conditions $M_{ab}^2 \geq m_a^2$ and $M_{ab}^2 \geq m_b^2$ are simultaneously satisfied.

This generalises to the mass generated by combining any number of particles, since the momentum p_a may be split an arbitrary number of times.

Appendix C

Performance benchmarks for graphicle

While exploring the FOSS contributions of this thesis, we have taken into consideration design, performance, and numerical stability. While it would be difficult to enumerate all of the decisions we have made, it is useful to consider the strengths and trade-offs present in our main analysis package, `graphicle`. To that end, we explore the computation of mass in a `MomentumArray` instance.

To start, let's write a basic `numpy` implementation of a mass calculation for four-momentum. The equation for mass is

$$m^2 = p_\mu p^\mu = E^2 - |\mathbf{p}|^2, \tag{C.1}$$

so we can write a function like so

```python
import numpy as np
import numpy.typing as npt

def calculate_mass_numpy(
    px: npt.NDArray[np.float64],
    py: npt.NDArray[np.float64],
    pz: npt.NDArray[np.float64],
    pe: npt.NDArray[np.float64],
) -> npt.NDArray[np.float64]:
    spatial_mag_sq = (px * px) + (py * py) + (pz * pz)
    energy_sq = pe * pe
    mass_sq = energy_sq - spatial_mag_sq
    sign = np.sign(mass_sq)
    return sign * np.sqrt(np.abs(energy_sq - spatial_mag_sq))
```

Notice that here we have used `np.sign()` to extract the signs of the squared difference. We have noticed a convention in HEP software packages propagate the signs of the squared mass to the mass itself, rather than representing it as an imaginary number, and we follow suit. That is, if $E^2 - |\mathbf{p}|^2 < 0$, then the mass will be given as $-\sqrt{E^2 - |\mathbf{p}|^2}$.

Let's get some dummy four-momentum data to test this on. We can do this using `MomentumArray.from_spherical_uniform()`, which samples randomly from a spherical uniform distribution.

```
import graphicle as gcl

# generate 10,000 four momenta, with a maximum energy of 100 GeV
# and a rate of massless particles of 1%
pmu = gcl.MomentumArray.from_spherical_uniform(
    size=10_000, max_energy=100.0, massless=0.01
)
```

We can use the `%%timeit` magic command in a Jupyter notebook to see how fast our `numpy` implementation is.

```
%%timeit
mass = calculate_mass_numpy(pmu.x, pmu.y, pmu.z, pmu.energy)
```

63.1 µs ± 136 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)

So our pure numpy solution took 63.1 μs. Before we can compare this against `graphicle`'s implementation, we must ensure that they produce similar results. We can do this with `np.isclose()`. Note that here we set an absolute tolerance of 10^{-5} GeV, which means we are taking differences of ~ 10 keV to be negligible. Given that electrons have a mass of 511 keV, this is a reasonable choice.

When we access `MomentumArray.mass`, under the hood a similar computation is performed by `graphicle`, and the resulting array is exposed as an attribute.

```
mass_gcl = pmu.mass
mass_np = calculate_mass_numpy(pmu.x, pmu.y, pmu.z, pmu.energy)

if np.all(np.isclose(mass_gcl, mass_np, atol=1.0e-5)):
    print("The two mass calculations are very similar.")
```

The two mass calculations are very similar.

So within our tolerance, the mass calculations from our `numpy` implementation and `graphicle` are the same. Now we time our `graphicle` implementation.

```
%%timeit
mass = pmu.mass
```

8.66 ns ± 0.108 ns per loop (mean ± std. dev. of 7 runs, 100,000,000 loops each)

`graphicle` appears to be 4 orders of magntiude faster, with a speed of 8.66 ns; however, this is because `MomentumArray` automatically caches many of its attributes after first computation, to prevent duplicate computations. While this does improve performance in a sense, timing the calculation is complicated. We can resolve this by clearing the cache, by deleting the `mass` attribute.

```
%%timeit
mass = pmu.mass
del pmu.mass
```

34.6 µs ± 83.5 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)

Clearing the cache gives a more accurate view on the performance of the `graphicle` implementation. So, `graphicle` achieves a 45% speedup to our basic `numpy` implementation. This is possible with `numba` JIT compilation.

If we JIT compile our basic implementation with `numba`, we can see how the speed is affected.

```
import numba as nb

@nb.njit
def calculate_mass_jit(
    px: npt.NDArray[np.float64],
    py: npt.NDArray[np.float64],
    pz: npt.NDArray[np.float64],
    pe: npt.NDArray[np.float64],
) -> npt.NDArray[np.float64]:
    spatial_mag_sq = (px * px) + (py * py) + (pz * pz)
```

```
    energy_sq = pe * pe
    mass_sq = energy_sq - spatial_mag_sq
    sign = np.sign(mass_sq)
    return sign * np.sqrt(np.abs(energy_sq - spatial_mag_sq))


%%timeit
mass = calculate_mass_jit(pmu.x, pmu.y, pmu.z, pmu.energy)
```

52.9 µs ± 18.9 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)

As you can see, we do achieve a speedup of about 16% with JIT compilation, but this is substantially less than with `graphicle`. This occurs because our JIT compiled function operates on arrays; `spatial_mag_sq`, `energy_sq`, `mass_sq`, `sign`, and the return line are all allocated as new arrays to hold the computed values in memory. However, in `graphicle`, we write our implementation using for-loops, enabling element-wise computation. This allows us to perform only one array allocation, performing multiple computations on a single element in memory before storing it as an output array element.

However, we can actually achieve an identical result in `numba` using the `vectorize` decorator instead of `njit`. `vectorize` creates what is known as a `numpy` UFunc. UFuncs are short for "Universal Functions", which operate on `numpy` arrays in an element-wise fashion. They allow advanced features such as array broadcasting, and automatic type casting. Other useful methods exist on UFuncs, like `np.ufunc.outer`, which performs the outer product when the UFunc is `np.multiply`, outer difference when the UFunc is `np.subtract`, *etc.*

Applying `vectorize` to our `calculate_mass` function will therefore convert it into a UFunc, although since we have 4 input parameters, methods such as `np.ufunc.outer` won't be available. The speed is shown below.

```
@nb.vectorize
def calculate_mass_vec(
    px: npt.NDArray[np.float64],
    py: npt.NDArray[np.float64],
    pz: npt.NDArray[np.float64],
    pe: npt.NDArray[np.float64],
) -> npt.NDArray[np.float64]:
    spatial_mag_sq = (px * px) + (py * py) + (pz * pz)
    energy_sq = pe * pe
    mass_sq = energy_sq - spatial_mag_sq
```

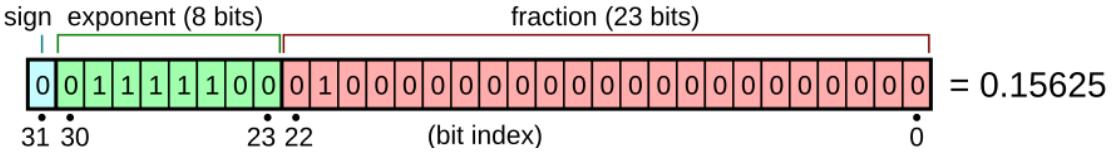


Figure C.1: Depicting the bit layout of a 32-bit floating point number (Wikipedia contributors (2025a)).

```
    sign = np.sign(mass_sq)
    return sign * np.sqrt(np.abs(energy_sq - spatial_mag_sq))


%%timeit
mass = calculate_mass_vec(pmu.x, pmu.y, pmu.z, pmu.energy)
```

20.7 µs ± 21.7 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)

As promised, this yields a much better speedup of about 67%. This is, in fact, much better than the 45% speedup from `graphicle`. This reduction in speed is a trade-off we made while writing `graphicle`, in favour of numerical stability.

In our computational work, we store numbers with floating point representations. The IEEE 754 standard defines `float32` numbers using 32-bits, with 1 bit to store the sign of the number (positive or negative), 23 bits to store the mantissa (which is a fraction in base 2), and 8 bits to store the exponent (also in base 2). Figure C.1 visually shows this layout, with thanks to Wikipedia contributors (2025a).

This is effectively standard form scientific notation, *eg.* representing $x = 123.45$ as $x = 1.2345 \times 10^3$, but rather than being written in base 10, it is in base 2. If I were to subtract a number y with the same number of significant figures as x, but a different exponent, we are likely to encounter a loss of information, *eg.*
$1.2345 \times 10^3 - 5.6789 \times 10^{-1} = 1.2288 \times 10^3$. In this operation, we've lost all digits for the positions between $10^{-3} - 10^{-5}$ in y.

This is often tolerable for numbers with large differences in exponential scale, since we rarely need accuracy to very large numbers of signicant figures. However, the problems gets much worse as the numbers become more similar. In fact, for numbers with the same exponent, and only very small differences in their mantissa, it becomes *catastrophic*.

Formally, the loss of precision that occurs when two very similar numbers are subtracted from one another is called Catastrophic Cancellation. If two numbers are close together in floating point representations, a large number of their mantissa bits will be identical (left-to-right). When they are subtracted, the identical digits become zeroed out, and the remaining digits are shifted left, until the first bit of the mantissa is

nonzero (1). The exponent is also changed to reflect the reduction in magnitude of the number.

Therefore, if the mantissas of two `float32` numbers were identical up to the 16th bit, the 7 bits at the end of the mantissa are shifted left during the subtraction, and the exponent is reduced by 16. It follows that what fills the 16 spaces now to the right of the 7 significant bits is not signicant; it's noise. Reducing the number of significant bits in the mantissa from 23 to 7 is a substantial loss in precision, hence the name "Catastrophic Cancellation". Note that we are actually using 64 bit floating point numbers, `float64`, but the principle is the same.

In our work, we study very high energy particles. For such particles, a large fraction of their energy is made up of their spatial momentum. This means that equation **??** might result in Catastrophic Cancellation.

Additionally, loss of precision due to differences in exponential scales are common when squaring and taking their sum / difference, since existing differences in exponential scale get exaggerated when squared.

The fix is simple. We rewrite the mass formula as

$$m = \sqrt{E - |\mathbf{p}|}\sqrt{E + |\mathbf{p}|} \tag{C.2}$$

Of course, if E and $|\mathbf{p}|$ are extremely close, we can't avoid Catastrophic Cancellation. But it is less likely for less extreme cases, because by performing the difference operation in linear space, rather than quadratic space, we avoid rounding errors. Additionally, by square rooting the difference and sum prior to taking their product, we avoid quadratic space altogether, which again protects our result from rounding error.

An equivalent operation in `numpy` could be:

```
@nb.njit
def magnitude(x: float, y: float, z: float) -> float:
    max_component = max(abs(x), abs(y), abs(z))
    max_recip = 1.0 / max_component
    x *= max_recip
    y *= max_recip
    z *= max_recip
    return max_component * np.sqrt(x * x + y * y + z * z)


@nb.vectorize
```

```
def calculate_mass_stable(
    px: npt.NDArray[np.float64],
    py: npt.NDArray[np.float64],
    pz: npt.NDArray[np.float64],
    pe: npt.NDArray[np.float64],
) -> npt.NDArray[np.float64]:
    spatial_mag = magnitude(px, py, pz)
    diff = pe - spatial_mag
    sign = np.sign(diff)
    return sign * np.sqrt(np.abs(diff) * (pe + spatial_mag))
```

Where we have found the magnitude of a three-vector via another numerical trick; we divide all components by the largest component. The result is that the largest component become 1, and all other components are fractional. Therefore, the sum of squares of the components will be between 1 - 3, and likely just a small fraction above 1. This avoids rounding error, and has the additional benefit that approximations of $\sqrt{1+x}$ are more accurate when x is small (since square roots are often implemented as a truncated Taylor expansion).

Timing this gives

```
%%timeit
mass = calculate_mass_stable(pmu.x, pmu.y, pmu.z, pmu.energy)
```

65 µs ± 147 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)

This results in a similar performance as our pure `numpy` implementation. `graphicle` improves upon this by using for-loops explicitly with `njit`, reducing the overhead introduced by `vectorize`. Additionally, cached versions of intermediate variables such as `spatial_mag` (which is often used for computing other attributes during analyses) improves performance.

Ultimately, `graphicle` attempts to trade-off performance for numerical stability. The specific choices we made when writing `graphicle` make it difficult to benchmark it against other libraries (such as Scikit-HEP's Vector), for each individual routine we have written, and how we have tailored it to intended use-cases. However, a detailed analysis of this would be excellent for future work.

Appendix D

Top quark reconstruction with ancestry information using jet images

In the late stages of drafting this thesis, one recent ML algorithm was uncovered to attempt top quark reconstruction from the set of detector-level particles in collision events (Choi et al., 2023). Detailed analysis and results comparisons in the main body of this thesis were not possible, as this work came to our attention late in the editing phase approaching submission, but we congratulate the authors for their work. Their model yields high performance and the areas of top jet tagging and mass prediction via regression, further vindicating the usage of ancestry-based Monte-Carlo simulation knowledge in the preparation of supervision labels for reconstructing colour charged partons. We hope that our contribution will bring additional attention to these techniques for the community at large, and more resourceful use of Monte-Carlo ancestry information will continue to bridge the gap between theory and experiments via computational simulations.

Despite this, we briefly highlight some limitations of their approach which our method improves upon.

The authors provide supervision labels by using a similar approach to ours, using Monte-Carlo truth based ancestry, overcoming ambiguous parentage detector-level particles descending from the top quark by assigning it if its close to the top in $\eta - \phi$ space. However, the authors add ambiguous final state particles to top quark jets until a desired mass has been reached, namely $1.05 \times m_{\text{top}}$, where m_{top} is the accepted mass of the top quark from existing theory and measurement. This is in contrast to our approach, which does not enforce consistency using prior knowledge, avoiding bias.

The model applied frames particle reconstruction as an image segmentation problem, learning masks over the images. This is an interesting and exciting use case for jet images, however it comes with limitations. Namely, it does not provide the set of constituents of the top quark from the final state particles in the model's readout, so reconstructing momentum via summation is impossible. The authors instead use a regression model to predict the mass based on the masks segmenting the images, using the mass obtained from Monte-Carlo truth. This, however, is also biased from the prior knowledge of the top mass which they explicitly enforce in the construction of their Monte-Carlo truth clusters.

While the authors do not explore infrared and collinear safety considerations in their calculations, collinear safety seems likely due to collinear splits in the $\eta - \phi$ plane being captured within the same mask. However, it seems unlikely that infrared safety can be intrinsically ensured with this technique, since without p_T cuts, soft emissions would change the jet image (though this may be mitigated with p_T weighting).

D.1 Disclaimer

We would like to note that our methods – refining ancestry with momentum comparisons of detector-level cluster constituents *vs.* the hard partons – were published with the official release of `graphicle` v0.2.4 on February 23rd 2023, prior to the first appearance of Choi et al. (2023)'s pre-publication on arXiv. Our method was subsequently presented later that same year, at PyHEP23 (Chaplais, 2023), while demonstrating our package ecosystem. Our approach was developed concurrently, but independently, to that of Choi et al. (2023).

Prior to both works, to the best of our knowledge, simulation-based ancestry had not been used in this way to produce reconstructions for coloured partons, such as the top quark. We are excited to see further adoption of such techniques, and how these might bridge the gap between the theory encoded in simulations with experimental analysis.

References

The CMS electromagnetic calorimeter project: Technical Design Report. Technical design report. CMS. CERN, Geneva, 1997. URL https://cds.cern.ch/record/349375.

The CMS hadron calorimeter project: Technical Design Report. Technical design report. CMS. CERN, Geneva, 1997. URL https://cds.cern.ch/record/357153.

G. Aad, B. Abbott, D. C. Abbott, A. Abed Abud, K. Abeling, D. K. Abhayasinghe, S. H. Abidi, A. Aboulhorma, H. Abramowicz, H. Abreu, Y. Abulaiti, A. C. Abusleme Hoffman, B. S. Acharya, B. Achkar, L. Adam, C. Adam Bourdarios, L. Adamczyk, L. Adamek, S. V. Addepalli, J. Adelman, A. Adiguzel, S. Adorni, T. Adye, A. A. Affolder, Y. Afik, M. N. Agaras, J. Agarwala, A. Aggarwal, C. Agheorghiesei, J. A. Aguilar-Saavedra, A. Ahmad, F. Ahmadov, W. S. Ahmed, X. Ai, G. Aielli, I. Aizenberg, M. Akbiyik, T. P. A. Åkesson, A. V. Akimov, K. Al Khoury, G. L. Alberghi, J. Albert, P. Albicocco, M. J. Alconada Verzini, S. Alderweireldt, M. Aleksa, I. N. Aleksandrov, C. Alexa, T. Alexopoulos, A. Alfonsi, F. Alfonsi, M. Alhroob, B. Ali, S. Ali, M. Aliev, G. Alimonti, C. Allaire, B. M. M. Allbrooke, P. P. Allport, A. Aloisio, F. Alonso, C. Alpigiani, E. Alunno Camelia, M. Alvarez Estevez, M. G. Alviggi, Y. Amaral Coutinho, A. Ambler, L. Ambroz, C. Amelung, D. Amidei, S. P. Amor Dos Santos, S. Amoroso, K. R. Amos, C. S. Amrouche, V. Ananiev, C. Anastopoulos, N. Andari, T. Andeen, J. K. Anders, S. Y. Andrean, A. Andreazza, S. Angelidakis, A. Angerami, A. V. Anisenkov, A. Annovi, C. Antel, M. T. Anthony, E. Antipov, M. Antonelli, D. J. A. Antrim, F. Anulli, M. Aoki, J. A. Aparisi Pozo, M. A. Aparo, L. Aperio Bella, C. Appelt, N. Aranzabal, V. Araujo Ferraz, C. Arcangeletti, A. T. H. Arce, E. Arena, J-F. Arguin, S. Argyropoulos, J.-H. Arling, A. J. Armbruster, O. Arnaez, H. Arnold, Z. P. Arrubarrena Tame, G. Artoni, H. Asada, K. Asai, S. Asai, N. A. Asbah, E. M. Asimakopoulou, J. Assahsah, K. Assamagan, R. Astalos, R. J. Atkin, M. Atkinson, N. B. Atlay, H. Atmani, P. A. Atmasiddha, K. Augsten, S. Auricchio, V. A. Austrup, G. Avner, G. Avolio, M. K. Ayoub, G. Azuelos, D. Babal, H. Bachacou, K. Bachas, A. Bachiu, F. Backman, A. Badea, P. Bagnaia, M. Bahmani, A. J. Bailey, V. R. Bailey, J. T. Baines, C. Bakalis, O. K. Baker, P. J. Bakker, E. Bakos, D. Bakshi Gupta, S. Balaji, R. Balasubramanian, E. M. Baldin, P. Balek, E. Ballabene, F. Balli, L. M. Baltes, W. K. Balunas, J. Balz, E. Banas, M. Bandieramonte, A. Bandyopadhyay, S. Bansal, L. Barak, E. L. Barberio,

D. Barberis, M. Barbero, G. Barbour, K. N. Barends, T. Barillari, M-S. Barisits, J. Barkeloo, T. Barklow, R. M. Barnett, P. Baron, A. Baroncelli, G. Barone, A. J. Barr, L. Barranco Navarro, F. Barreiro, J. Barreiro Guimarães da Costa, U. Barron, S. Barsov, F. Bartels, R. Bartoldus, G. Bartolini, A. E. Barton, P. Bartos, A. Basalaev, A. Basan, M. Baselga, I. Bashta, A. Bassalat, M. J. Basso, C. R. Basson, R. L. Bates, S. Batlamous, J. R. Batley, B. Batool, M. Battaglia, M. Bauce, F. Bauer, P. Bauer, A. Bayirli, J. B. Beacham, T. Beau, P. H. Beauchemin, F. Becherer, P. Bechtle, H. P. Beck, K. Becker, C. Becot, A. J. Beddall, V. A. Bednyakov, C. P. Bee, L. J. Beemster, T. A. Beermann, M. Begalli, M. Begel, A. Behera, J. K. Behr, C. Beirao Da Cruz E Silva, J. F. Beirer, F. Beisiegel, M. Belfkir, G. Bella, L. Bellagamba, A. Bellerive, P. Bellos, K. Beloborodov, K. Belotskiy, N. L. Belyaev, D. Benchekroun, Y. Benhammou, D. P. Benjamin, M. Benoit, J. R. Bensinger, S. Bentvelsen, L. Beresford, M. Beretta, D. Berge, E. Bergeaas Kuutmann, N. Berger, B. Bergmann, J. Beringer, S. Berlendis, G. Bernardi, C. Bernius, F. U. Bernlochner, T. Berry, P. Berta, A. Berthold, I. A. Bertram, O. Bessidskaia Bylund, S. Bethke, A. Betti, A. J. Bevan, S. Bhatta, D. S. Bhattacharya, P. Bhattarai, V. S. Bhopatkar, R. Bi, R. Bi, R. M. Bianchi, O. Biebel, R. Bielski, N. V. Biesuz, M. Biglietti, T. R. V. Billoud, M. Bindi, A. Bingul, C. Bini, S. Biondi, A. Biondini, C. J. Birch-sykes, G. A. Bird, M. Birman, T. Bisanz, D. Biswas, A. Bitadze, K. Bjørke, I. Bloch, C. Blocker, A. Blue, U. Blumenschein, J. Blumenthal, G. J. Bobbink, V. S. Bobrovnikov, M. Boehler, D. Bogavac, A. G. Bogdanchikov, C. Bohm, V. Boisvert, P. Bokan, T. Bold, M. Bomben, M. Bona, M. Boonekamp, C. D. Booth, A. G. Borbély, H. M. Borecka-Bielska, L. S. Borgna, G. Borissov, D. Bortoletto, D. Boscherini, M. Bosman, J. D. Bossio Sola, K. Bouaouda, J. Boudreau, E. V. Bouhova-Thacker, D. Boumediene, R. Bouquet, A. Boveia, J. Boyd, D. Boye, I. R. Boyko, J. Bracinik, N. Brahimi, G. Brandt, O. Brandt, F. Braren, B. Brau, J. E. Brau, W. D. Breaden Madden, K. Brendlinger, R. Brener, L. Brenner, R. Brenner, S. Bressler, B. Brickwedde, D. Britton, D. Britzger, I. Brock, G. Brooijmans, W. K. Brooks, E. Brost, P. A. Bruckman de Renstrom, B. Brüers, D. Bruncko, A. Bruni, G. Bruni, M. Bruschi, N. Bruscino, L. Bryngemark, T. Buanes, Q. Buat, P. Buchholz, A. G. Buckley, I. A. Budagov, M. K. Bugge, O. Bulekov, B. A. Bullard, S. Burdin, C. D. Burgard, A. M. Burger, B. Burghgrave, J. T. P. Burr, C. D. Burton, J. C. Burzynski, E. L. Busch, V. Büscher, P. J. Bussey, J. M. Butler, C. M. Buttar, J. M. Butterworth, W. Buttinger, C. J. Buxo Vazquez, A. R. Buzykaev, G. Cabras, S. Cabrera Urbán, D. Caforio, H. Cai, Y. Cai, V. M. M. Cairo, O. Cakir, N. Calace, P. Calafiura, G. Calderini, P. Calfayan, G. Callea, L. P. Caloba, D. Calvet, S. Calvet, T. P. Calvet, M. Calvetti, R. Camacho Toro, S. Camarda, D. Camarero Munoz, P. Camarri, M. T. Camerlingo, D. Cameron, C. Camincher, M. Campanelli, A. Camplani, V. Canale, A. Canesse, M. Cano Bret, J. Cantero, Y. Cao, F. Capocasa, M. Capua, A. Carbone, R. Cardarelli, J. C. J. Cardenas, F. Cardillo, T. Carli, G. Carlino, B. T. Carlson, E. M. Carlson, L. Carminati, M. Carnesale, S. Caron, E. Carquin, S. Carrá, G. Carratta, J. W. S. Carter, T. M. Carter, D. Casadei, M. P. Casado, A. F. Casha, E. G. Castiglia,

F. L. Castillo, L. Castillo Garcia, V. Castillo Gimenez, N. F. Castro, A. Catinaccio, J. R. Catmore, V. Cavaliere, N. Cavalli, V. Cavasinni, E. Celebi, F. Celli, M. S. Centonze, K. Cerny, A. S. Cerqueira, A. Cerri, L. Cerrito, F. Cerutti, A. Cervelli, S. A. Cetin, Z. Chadi, D. Chakraborty, M. Chala, J. Chan, W. S. Chan, W. Y. Chan, J. D. Chapman, B. Chargeishvili, D. G. Charlton, T. P. Charman, M. Chatterjee, S. Chekanov, S. V. Chekulaev, G. A. Chelkov, A. Chen, B. Chen, B. Chen, C. Chen, H. Chen, H. Chen, J. Chen, J. Chen, S. Chen, S. J. Chen, X. Chen, X. Chen, Y. Chen, C. L. Cheng, H. C. Cheng, A. Cheplakov, E. Cheremushkina, E. Cherepanova, R. Cherkaoui El Moursli, E. Cheu, K. Cheung, L. Chevalier, V. Chiarella, G. Chiarelli, G. Chiodini, A. S. Chisholm, A. Chitan, Y. H. Chiu, M. V. Chizhov, K. Choi, A. R. Chomont, Y. Chou, E. Y. S. Chow, T. Chowdhury, L. D. Christopher, M. C. Chu, X. Chu, J. Chudoba, J. J. Chwastowski, D. Cieri, K. M. Ciesla, V. Cindro, A. Ciocio, F. Cirotto, Z. H. Citron, M. Citterio, D. A. Ciubotaru, B. M. Ciungu, A. Clark, P. J. Clark, J. M. Clavijo Columbie, S. E. Clawson, C. Clement, L. Clissa, Y. Coadou, M. Cobal, A. Coccaro, R. F. Coelho Barrue, R. Coelho Lopes De Sa, S. Coelli, H. Cohen, A. E. C. Coimbra, B. Cole, J. Collot, P. Conde Muiño, S. H. Connell, I. A. Connelly, E. I. Conroy, F. Conventi, H. G. Cooke, A. M. Cooper-Sarkar, F. Cormier, L. D. Corpe, M. Corradi, E. E. Corrigan, F. Corriveau, M. J. Costa, F. Costanza, D. Costanzo, B. M. Cote, G. Cowan, J. W. Cowley, K. Cranmer, S. Crépé-Renaudin, F. Crescioli, M. Cristinziani, M. Cristoforetti, V. Croft, G. Crosetti, A. Cueto, T. Cuhadar Donszelmann, H. Cui, Z. Cui, A. R. Cukierman, W. R. Cunningham, F. Curcio, P. Czodrowski, M. M. Czurylo, M. J. Da Cunha Sargedas De Sousa, J. V. Da Fonseca Pinto, C. Da Via, W. Dabrowski, T. Dado, S. Dahbi, T. Dai, C. Dallapiccola, M. Dam, G. D'amen, V. D'Amico, J. Damp, J. R. Dandoy, M. F. Daneri, M. Danninger, V. Dao, G. Darbo, S. Darmora, A. Dattagupta, S. D'Auria, C. David, T. Davidek, D. R. Davis, B. Davis-Purcell, I. Dawson, K. De, R. De Asmundis, M. De Beurs, S. De Castro, N. De Groot, P. de Jong, H. De la Torre, A. De Maria, A. De Salvo, U. De Sanctis, M. De Santis, A. De Santo, J. B. De Vivie De Regie, D. V. Dedovich, J. Degens, A. M. Deiana, J. Del Peso, F. Del Rio, F. Deliot, C. M. Delitzsch, M. Della Pietra, D. Della Volpe, A. Dell'Acqua, L. Dell'Asta, M. Delmastro, P. A. Delsart, S. Demers, M. Demichev, S. P. Denisov, L. D'Eramo, D. Derendarz, F. Derue, P. Dervan, K. Desch, K. Dette, C. Deutsch, P. O. Deviveiros, F. A. Di Bello, A. Di Ciaccio, L. Di Ciaccio, A. Di Domenico, C. Di Donato, A. Di Girolamo, G. Di Gregorio, A. Di Luca, B. Di Micco, R. Di Nardo, C. Diaconu, F. A. Dias, T. Dias Do Vale, M. A. Diaz, F. G. Diaz Capriles, M. Didenko, E. B. Diehl, S. Díez Cornell, C. Diez Pardos, C. Dimitriadi, A. Dimitrievska, W. Ding, J. Dingfelder, I-M. Dinu, S. J. Dittmeier, F. Dittus, F. Djama, T. Djobava, J. I. Djuvsland, D. Dodsworth, C. Doglioni, J. Dolejsi, Z. Dolezal, M. Donadelli, B. Dong, J. Donini, A. D'Onofrio, M. D'Onofrio, J. Dopke, A. Doria, M. T. Dova, A. T. Doyle, E. Drechsler, E. Dreyer, A. S. Drobac, D. Du, T. A. du Pree, F. Dubinin, M. Dubovsky, E. Duchovni, G. Duckeck, O. A. Ducu, D. Duda, A. Dudarev, M. D'uffizi, L. Duflot,

M. Dührssen, C. Dülsen, A. E. Dumitriu, M. Dunford, S. Dungs, K. Dunne, A. Duperrin, H. Duran Yildiz, M. Düren, A. Durglishvili, B. Dutta, B. L. Dwyer, G. I. Dyckes, M. Dyndal, S. Dysch, B. S. Dziedzic, B. Eckerova, M. G. Eggleston, E. Egidio Purcino De Souza, L. F. Ehrke, G. Eigen, K. Einsweiler, T. Ekelof, Y. El Ghazali, H. El Jarrari, A. El Moussaouy, V. Ellajosyula, M. Ellert, F. Ellinghaus, A. A. Elliot, N. Ellis, J. Elmsheuser, M. Elsing, D. Emeliyanov, A. Emerman, Y. Enari, I. Ene, J. Erdmann, A. Ereditato, P. A. Erland, M. Errenst, M. Escalier, C. Escobar, E. Etzion, G. Evans, H. Evans, M. O. Evans, A. Ezhilov, S. Ezzarqtouni, F. Fabbri, L. Fabbri, G. Facini, V. Fadeyev, R. M. Fakhrutdinov, S. Falciano, P. J. Falke, S. Falke, J. Faltova, Y. Fan, Y. Fang, G. Fanourakis, M. Fanti, M. Faraj, A. Farbin, A. Farilla, T. Farooque, S. M. Farrington, F. Fassi, D. Fassouliotis, M. Faucci Giannelli, W. J. Fawcett, L. Fayard, O. L. Fedin, G. Fedotov, M. Feickert, L. Feligioni, A. Fell, D. E. Fellers, C. Feng, M. Feng, M. J. Fenton, A. B. Fenyuk, S. W. Ferguson, J. Ferrando, A. Ferrari, P. Ferrari, R. Ferrari, D. Ferrere, C. Ferretti, F. Fiedler, A. Filipčič, E. K. Filmer, F. Filthaut, M. C. N. Fiolhais, L. Fiorini, F. Fischer, W. C. Fisher, T. Fitschen, I. Fleck, P. Fleischmann, T. Flick, L. Flores, M. Flores, L. R. Flores Castillo, F. M. Follega, N. Fomin, J. H. Foo, B. C. Forland, A. Formica, A. C. Forti, E. Fortin, A. W. Fortman, M. G. Foti, L. Fountas, D. Fournier, H. Fox, P. Francavilla, S. Francescato, M. Franchini, S. Franchino, D. Francis, L. Franco, L. Franconi, M. Franklin, G. Frattari, A. C. Freegard, P. M. Freeman, W. S. Freund, E. M. Freundlich, D. Froidevaux, J. A. Frost, Y. Fu, M. Fujimoto, E. Fullana Torregrosa, J. Fuster, A. Gabrielli, A. Gabrielli, P. Gadow, G. Gagliardi, L. G. Gagnon, G. E. Gallardo, E. J. Gallas, B. J. Gallop, R. Gamboa Goni, K. K. Gan, S. Ganguly, J. Gao, Y. Gao, F. M. Garay Walls, B. Garcia, C. García, J. E. García Navarro, J. A. García Pascual, M. Garcia-Sciveres, R. W. Gardner, D. Garg, R. B. Garg, S. Gargiulo, C. A. Garner, V. Garonne, S. J. Gasiorowski, P. Gaspar, G. Gaudio, P. Gauzzi, I. L. Gavrilenko, A. Gavrilyuk, C. Gay, G. Gaycken, E. N. Gazis, A. A. Geanta, C. M. Gee, J. Geisen, M. Geisen, C. Gemme, M. H. Genest, S. Gentile, S. George, W. F. George, T. Geralis, L. O. Gerlach, P. Gessinger-Befurt, M. Ghasemi Bostanabad, M. Ghneimat, A. Ghosal, A. Ghosh, A. Ghosh, B. Giacobbe, S. Giagu, N. Giangiacomi, P. Giannetti, A. Giannini, S. M. Gibson, M. Gignac, D. T. Gil, B. J. Gilbert, D. Gillberg, G. Gilles, N. E. K. Gillwald, L. Ginabat, D. M. Gingrich, M. P. Giordani, P. F. Giraud, G. Giugliarelli, D. Giugni, F. Giuli, I. Gkialas, P. Gkountoumis, L. K. Gladilin, C. Glasman, G. R. Gledhill, M. Glisic, I. Gnesi, Y. Go, M. Goblirsch-Kolb, D. Godin, S. Goldfarb, T. Golling, M. G. D. Gololo, D. Golubkov, J. P. Gombas, A. Gomes, A. J. Gomez Delegido, R. Goncalves Gama, R. Gonçalo, G. Gonella, L. Gonella, A. Gongadze, F. Gonnella, J. L. Gonski, R. Y. González Andana, S. González de la Hoz, S. Gonzalez Fernandez, R. Gonzalez Lopez, C. Gonzalez Renteria, R. Gonzalez Suarez, S. Gonzalez-Sevilla, G. R. Gonzalvo Rodriguez, L. Goossens, N. A. Gorasia, P. A. Gorbounov, B. Gorini, E. Gorini, A. Gorišek, A. T. Goshaw, M. I. Gostkin, C. A. Gottardo, M. Gouighri, V. Goumarre, A. G. Goussiou, N. Govender, C. Goy,

I. Grabowska-Bold, K. Graham, E. Gramstad, S. Grancagnolo, M. Grandi, V. Gratchev, P. M. Gravila, F. G. Gravili, H. M. Gray, C. Grefe, I. M. Gregor, P. Grenier, K. Grevtsov, C. Grieco, A. A. Grillo, K. Grimm, S. Grinstein, J.-F. Grivaz, S. Groh, E. Gross, J. Grosse-Knetter, C. Grud, A. Grummer, J. C. Grundy, L. Guan, W. Guan, C. Gubbels, J. G. R. Guerrero Rojas, F. Guescini, R. Gugel, A. Guida, T. Guillemin, S. Guindon, F. Guo, J. Guo, L. Guo, Y. Guo, R. Gupta, S. Gurbuz, G. Gustavino, M. Guth, P. Gutierrez, L. F. Gutierrez Zagazeta, C. Gutschow, C. Guyot, C. Gwenlan, C. B. Gwilliam, E. S. Haaland, A. Haas, M. Habedank, C. Haber, H. K. Hadavand, A. Hadef, S. Hadzic, M. Haleem, J. Haley, J. J. Hall, G. D. Hallewell, L. Halser, K. Hamano, H. Hamdaoui, M. Hamer, G. N. Hamity, J. Han, K. Han, L. Han, L. Han, S. Han, Y. F. Han, K. Hanagaki, M. Hance, D. A. Hangal, M. D. Hank, R. Hankache, E. Hansen, J. B. Hansen, J. D. Hansen, P. H. Hansen, K. Hara, D. Harada, T. Harenberg, S. Harkusha, Y. T. Harris, P. F. Harrison, N. M. Hartman, N. M. Hartmann, Y. Hasegawa, A. Hasib, S. Haug, R. Hauser, M. Havranek, C. M. Hawkes, R. J. Hawkings, S. Hayashida, D. Hayden, C. Hayes, R. L. Hayes, C. P. Hays, J. M. Hays, H. S. Hayward, F. He, Y. He, Y. He, M. P. Heath, V. Hedberg, A. L. Heggelund, N. D. Hehir, C. Heidegger, K. K. Heidegger, W. D. Heidorn, J. Heilman, S. Heim, T. Heim, B. Heinemann, J. G. Heinlein, J. J. Heinrich, L. Heinrich, J. Hejbal, L. Helary, A. Held, S. Hellesund, C. M. Helling, S. Hellman, C. Helsens, R. C. W. Henderson, L. Henkelmann, A. M. Henriques Correia, H. Herde, Y. Hernández Jiménez, H. Herr, M. G. Herrmann, T. Herrmann, G. Herten, R. Hertenberger, L. Hervas, N. P. Hessey, H. Hibi, E. Higón-Rodriguez, S. J. Hillier, I. Hinchliffe, F. Hinterkeuser, M. Hirose, S. Hirose, D. Hirschbuehl, B. Hiti, O. Hladik, J. Hobbs, R. Hobincu, N. Hod, M. C. Hodgkinson, B. H. Hodkinson, A. Hoecker, J. Hofer, D. Hohn, T. Holm, M. Holzbock, L. B. A. H. Hommels, B. P. Honan, J. Hong, T. M. Hong, Y. Hong, J. C. Honig, A. Hönle, B. H. Hooberman, W. H. Hopkins, Y. Horii, L. A. Horyn, S. Hou, J. Howarth, J. Hoya, M. Hrabovsky, A. Hrynevich, T. Hryn'ova, P. J. Hsu, S.-C. Hsu, Q. Hu, S. Hu, Y. F. Hu, D. P. Huang, X. Huang, Y. Huang, Y. Huang, Z. Hubacek, M. Huebner, F. Huegging, T. B. Huffman, M. Huhtinen, S. K. Huiberts, R. Hulsken, N. Huseynov, J. Huston, J. Huth, R. Hyneman, S. Hyrych, G. Iacobucci, G. Iakovidis, I. Ibragimov, L. Iconomidou-Fayard, P. Iengo, R. Iguchi, T. Iizawa, Y. Ikegami, A. Ilg, N. Ilic, H. Imam, T. Ingebretsen Carlson, G. Introzzi, M. Iodice, V. Ippolito, M. Ishino, W. Islam, C. Issever, S. Istin, H. Ito, J. M. Iturbe Ponce, R. Iuppa, A. Ivina, J. M. Izen, V. Izzo, P. Jacka, P. Jackson, R. M. Jacobs, B. P. Jaeger, C. S. Jagfeld, G. Jäkel, K. Jakobs, T. Jakoubek, J. Jamieson, K. W. Janas, G. Jarlskog, A. E. Jaspan, T. Javůrek, M. Javurkova, F. Jeanneau, L. Jeanty, J. Jejelava, P. Jenni, S. Jézéquel, J. Jia, X. Jia, Z. Jia, Y. Jiang, S. Jiggins, J. Jimenez Pena, S. Jin, A. Jinaru, O. Jinnouchi, H. Jivan, P. Johansson, K. A. Johns, C. A. Johnson, D. M. Jones, E. Jones, R. W. L. Jones, T. J. Jones, J. Jovicevic, X. Ju, J. J. Junggeburth, A. Juste Rozas, S. Kabana, A. Kaczmarska, M. Kado, H. Kagan, M. Kagan, A. Kahn, A. Kahn, C. Kahra, T. Kaji, E. Kajomovitz,

N. Kakati, C. W. Kalderon, A. Kamenshchikov, N. J. Kang, Y. Kano, D. Kar, K. Karava, M. J. Kareem, E. Karentzos, I. Karkanias, S. N. Karpov, Z. M. Karpova, V. Kartvelishvili, A. N. Karyukhin, E. Kasimi, C. Kato, J. Katzy, S. Kaur, K. Kawade, K. Kawagoe, T. Kawaguchi, T. Kawamoto, G. Kawamura, E. F. Kay, F. I. Kaya, S. Kazakos, V. F. Kazanin, Y. Ke, J. M. Keaveney, R. Keeler, G. V. Kehris, J. S. Keller, A. S. Kelly, D. Kelsey, J. J. Kempster, J. Kendrick, K. E. Kennedy, O. Kepka, B. P. Kerridge, S. Kersten, B. P. Kerševan, S. Ketabchi Haghighat, M. Khandoga, A. Khanov, A. G. Kharlamov, T. Kharlamova, E. E. Khoda, T. J. Khoo, G. Khoriauli, J. Khubua, M. Kiehn, A. Kilgallon, E. Kim, Y. K. Kim, N. Kimura, A. Kirchhoff, D. Kirchmeier, C. Kirfel, J. Kirk, A. E. Kiryunin, T. Kishimoto, D. P. Kisliuk, C. Kitsaki, O. Kivernyk, M. Klassen, C. Klein, L. Klein, M. H. Klein, M. Klein, U. Klein, P. Klimek, A. Klimentov, F. Klimpel, T. Klingl, T. Klioutchnikova, F. F. Klitzner, P. Kluit, S. Kluth, E. Kneringer, T. M. Knight, A. Knue, D. Kobayashi, R. Kobayashi, M. Kocian, T. Kodama, P. Kodyš, D. M. Koeck, P. T. Koenig, T. Koffas, N. M. Köhler, M. Kolb, I. Koletsou, T. Komarek, K. Köneke, A. X. Y. Kong, T. Kono, N. Konstantinidis, B. Konya, R. Kopeliansky, S. Koperny, K. Korcyl, K. Kordas, G. Koren, A. Korn, S. Korn, I. Korolkov, N. Korotkova, B. Kortman, O. Kortner, S. Kortner, W. H. Kostecka, V. V. Kostyukhin, A. Kotsokechagia, A. Kotwal, A. Koulouris, A. Kourkoumeli-Charalampidi, C. Kourkoumelis, E. Kourlitis, O. Kovanda, R. Kowalewski, W. Kozanecki, A. S. Kozhin, V. A. Kramarenko, G. Kramberger, P. Kramer, M. W. Krasny, A. Krasznahorkay, J. A. Kremer, J. Kretzschmar, K. Kreul, P. Krieger, F. Krieter, S. Krishnamurthy, A. Krishnan, M. Krivos, K. Krizka, K. Kroeninger, H. Kroha, J. Kroll, J. Kroll, K. S. Krowpman, U. Kruchonak, H. Krüger, N. Krumnack, M. C. Kruse, J. A. Krzysiak, A. Kubota, O. Kuchinskaia, S. Kuday, D. Kuechler, J. T. Kuechler, S. Kuehn, T. Kuhl, V. Kukhtin, Y. Kulchitsky, S. Kuleshov, M. Kumar, N. Kumari, M. Kuna, A. Kupco, T. Kupfer, O. Kuprash, H. Kurashige, L. L. Kurchaninov, Y. A. Kurochkin, A. Kurova, E. S. Kuwertz, M. Kuze, A. K. Kvam, J. Kvita, T. Kwan, K. W. Kwok, C. Lacasta, F. Lacava, H. Lacker, D. Lacour, N. N. Lad, E. Ladygin, B. Laforge, T. Lagouri, S. Lai, I. K. Lakomiec, N. Lalloue, J. E. Lambert, S. Lammers, W. Lampl, C. Lampoudis, E. Lançon, U. Landgraf, M. P. J. Landon, V. S. Lang, J. C. Lange, R. J. Langenberg, A. J. Lankford, F. Lanni, K. Lantzsch, A. Lanza, A. Lapertosa, J. F. Laporte, T. Lari, F. Lasagni Manghi, M. Lassnig, V. Latonova, T. S. Lau, A. Laudrain, A. Laurier, M. Lavorgna, S. D. Lawlor, Z. Lawrence, M. Lazzaroni, B. Le, B. Leban, A. Lebedev, M. LeBlanc, T. LeCompte, F. Ledroit-Guillon, A. C. A. Lee, G. R. Lee, L. Lee, S. C. Lee, L. L. Leeuw, B. Lefebvre, H. P. Lefebvre, M. Lefebvre, C. Leggett, K. Lehmann, G. Lehmann Miotto, W. A. Leight, A. Leisos, M. A. L. Leite, C. E. Leitgeb, R. Leitner, K. J. C. Leney, T. Lenz, S. Leone, C. Leonidopoulos, A. Leopold, C. Leroy, R. Les, C. G. Lester, M. Levchenko, J. Levêque, D. Levin, L. J. Levinson, D. J. Lewis, B. Li, B. Li, C. Li, C-Q. Li, H. Li, H. Li, H. Li, J. Li, K. Li, L. Li, M. Li, Q. Y. Li, S. Li, T. Li, X. Li, Z. Li, Z. Li, Z. Li, Z. Li, Z. Liang, M. Liberatore, B. Liberti, K. Lie, J. Lieber

Marin, K. Lin, R. A. Linck, R. E. Lindley, J. H. Lindon, A. Linss, E. Lipeles, A. Lipniacka, T. M. Liss, A. Lister, J. D. Little, B. Liu, B. X. Liu, D. Liu, J. B. Liu, J. K. K. Liu, K. Liu, M. Liu, M. Y. Liu, P. Liu, Q. Liu, X. Liu, Y. Liu, Y. Liu, Y. L. Liu, Y. W. Liu, M. Livan, J. Llorente Merino, S. L. Lloyd, E. M. Lobodzinska, P. Loch, S. Loffredo, T. Lohse, K. Lohwasser, M. Lokajicek, J. D. Long, I. Longarini, L. Longo, R. Longo, I. Lopez Paz, A. Lopez Solis, J. Lorenz, N. Lorenzo Martinez, A. M. Lory, A. Lösle, X. Lou, X. Lou, A. Lounis, G. C. Louppe, J. Love, P. A. Love, J. J. Lozano Bahilo, G. Lu, M. Lu, S. Lu, Y. J. Lu, H. J. Lubatti, C. Luci, F. L. Lucio Alves, A. Lucotte, F. Luehring, I. Luise, O. Lukianchuk, O. Lundberg, B. Lund-Jensen, N. A. Luongo, M. S. Lutz, D. Lynn, H. Lyons, R. Lysak, E. Lytken, F. Lyu, V. Lyubushkin, T. Lyubushkina, H. Ma, L. L. Ma, Y. Ma, D. M. Mac Donell, G. Maccarrone, J. C. MacDonald, R. Madar, W. F. Mader, J. Maeda, T. Maeno, M. Maerker, V. Magerl, J. Magro, D. J. Mahon, C. Maidantchik, A. Maio, K. Maj, O. Majersky, S. Majewski, N. Makovec, V. Maksimovic, B. Malaescu, Pa. Malecki, V. P. Maleev, F. Malek, D. Malito, U. Mallik, C. Malone, S. Maltezos, S. Malyukov, J. Mamuzic, G. Mancini, J. P. Mandalia, I. Mandić, L. Manhaes de Andrade Filho, I. M. Maniatis, M. Manisha, J. Manjarres Ramos, D. C. Mankad, K. H. Mankinen, A. Mann, A. Manousos, B. Mansoulie, S. Manzoni, A. Marantis, G. Marchiori, M. Marcisovsky, L. Marcoccia, C. Marcon, M. Marinescu, M. Marjanovic, Z. Marshall, S. Marti-Garcia, T. A. Martin, V. J. Martin, B. Martin dit Latour, L. Martinelli, M. Martinez, P. Martinez Agullo, V. I. Martinez Outschoorn, P. Martinez Suarez, S. Martin-Haugh, V. S. Martoiu, A. C. Martyniuk, A. Marzin, S. R. Maschek, L. Masetti, T. Mashimo, J. Masik, A. L. Maslennikov, L. Massa, P. Massarotti, P. Mastrandrea, A. Mastroberardino, T. Masubuchi, T. Mathisen, A. Matic, N. Matsuzawa, J. Maurer, B. Maček, D. A. Maximov, R. Mazini, I. Maznas, M. Mazza, S. M. Mazza, C. Mc Ginn, J. P. Mc Gowan, S. P. Mc Kee, T. G. McCarthy, W. P. McCormack, E. F. McDonald, A. E. McDougall, J. A. Mcfayden, G. Mchedlidze, M. A. McKay, R. P. Mckenzie, D. J. Mclaughlin, K. D. McLean, S. J. McMahon, P. C. McNamara, R. A. McPherson, J. E. Mdhluli, S. Meehan, T. Megy, S. Mehlhase, A. Mehta, B. Meirose, D. Melini, B. R. Mellado Garcia, A. H. Melo, F. Meloni, A. Melzer, E. D. Mendes Gouveia, A. M. Mendes Jacques Da Costa, H. Y. Meng, L. Meng, S. Menke, M. Mentink, E. Meoni, C. Merlassino, L. Merola, C. Meroni, G. Merz, O. Meshkov, J. K. R. Meshreki, J. Metcalfe, A. S. Mete, C. Meyer, J-P. Meyer, M. Michetti, R. P. Middleton, L. Mijović, G. Mikenberg, M. Mikestikova, M. Mikuž, H. Mildner, A. Milic, C. D. Milke, D. W. Miller, L. S. Miller, A. Milov, D. A. Milstead, T. Min, A. A. Minaenko, I. A. Minashvili, L. Mince, A. I. Mincer, B. Mindur, M. Mineev, Y. Minegishi, Y. Mino, L. M. Mir, M. Miralles Lopez, M. Mironova, T. Mitani, A. Mitra, V. A. Mitsou, O. Miu, P. S. Miyagawa, Y. Miyazaki, A. Mizukami, J. U. Mjörnmark, T. Mkrtchyan, M. Mlynarikova, T. Moa, S. Mobius, K. Mochizuki, P. Moder, P. Mogg, A. F. Mohammed, S. Mohapatra, G. Mokgatitswane, B. Mondal, S. Mondal, K. Mönig, E. Monnier, L. Monsonis Romero, J. Montejo Berlingen, M. Montella, F. Monticelli,

N. Morange, A. L. Moreira De Carvalho, M. Moreno Llácer, C. Moreno Martinez, P. Morettini, S. Morgenstern, D. Mori, M. Morii, M. Morinaga, V. Morisbak, A. K. Morley, L. Morvaj, P. Moschovakos, B. Moser, M. Mosidze, T. Moskalets, P. Moskvitina, J. Moss, E. J. W. Moyse, S. Muanza, J. Mueller, D. Muenstermann, R. Müller, G. A. Mullier, J. J. Mullin, D. P. Mungo, J. L. Munoz Martinez, F. J. Munoz Sanchez, M. Murin, W. J. Murray, A. Murrone, J. M. Muse, M. Muškinja, C. Mwewa, A. G. Myagkov, A. J. Myers, A. A. Myers, G. Myers, M. Myska, B. P. Nachman, O. Nackenhorst, A. Nag, K. Nagai, K. Nagano, J. L. Nagle, E. Nagy, A. M. Nairz, Y. Nakahama, K. Nakamura, H. Nanjo, R. Narayan, E. A. Narayanan, I. Naryshkin, M. Naseri, C. Nass, G. Navarro, J. Navarro-Gonzalez, R. Nayak, P. Y. Nechaeva, F. Nechansky, T. J. Neep, A. Negri, M. Negrini, C. Nellist, C. Nelson, K. Nelson, S. Nemecek, M. Nessi, M. S. Neubauer, F. Neuhaus, J. Neundorf, R. Newhouse, P. R. Newman, C. W. Ng, Y. S. Ng, Y. W. Y. Ng, B. Ngair, H. D. N. Nguyen, R. B. Nickerson, R. Nicolaidou, D. S. Nielsen, J. Nielsen, M. Niemeyer, N. Nikiforou, V. Nikolaenko, I. Nikolic-Audit, K. Nikolopoulos, P. Nilsson, H. R. Nindhito, A. Nisati, N. Nishu, R. Nisius, S. J. Noacco Rosende, T. Nobe, D. L. Noel, Y. Noguchi, I. Nomidis, M. A. Nomura, M. B. Norfolk, R. R. B. Norisam, J. Novak, T. Novak, O. Novgorodova, L. Novotny, R. Novotny, L. Nozka, K. Ntekas, E. Nurse, F. G. Oakham, J. Ocariz, A. Ochi, I. Ochoa, J. P. Ochoa-Ricoux, S. Oda, S. Oerdek, A. Ogrodnik, A. Oh, C. C. Ohm, H. Oide, R. Oishi, M. L. Ojeda, Y. Okazaki, M. W. O'Keefe, Y. Okumura, A. Olariu, L. F. Oleiro Seabra, S. A. Olivares Pino, D. Oliveira Damazio, D. Oliveira Goncalves, J. L. Oliver, M. J. R. Olsson, A. Olszewski, J. Olszowska, Ö. O. Öncel, D. C. O'Neil, A. P. O'Neill, A. Onofre, P. U. E. Onyisi, R. G. Oreamuno Madriz, M. J. Oreglia, G. E. Orellana, D. Orestano, N. Orlando, R. S. Orr, V. O'Shea, R. Ospanov, G. Otero y Garzon, H. Otono, P. S. Ott, G. J. Ottino, M. Ouchrif, J. Ouellette, F. Ould-Saada, M. Owen, R. E. Owen, K. Y. Oyulmaz, V. E. Ozcan, N. Ozturk, S. Ozturk, J. Pacalt, H. A. Pacey, A. Pacheco Pages, C. Padilla Aranda, S. Pagan Griso, G. Palacino, S. Palazzo, S. Palestini, M. Palka, J. Pan, D. K. Panchal, C. E. Pandini, J. G. Panduro Vazquez, P. Pani, G. Panizzo, L. Paolozzi, C. Papadatos, S. Parajuli, A. Paramonov, C. Paraskevopoulos, D. Paredes Hernandez, B. Parida, T. H. Park, A. J. Parker, M. A. Parker, F. Parodi, E. W. Parrish, V. A. Parrish, J. A. Parsons, U. Parzefall, B. Pascual Dias, L. Pascual Dominguez, V. R. Pascuzzi, F. Pasquali, E. Pasqualucci, S. Passaggio, F. Pastore, P. Pasuwan, J. R. Pater, A. Pathak, J. Patton, T. Pauly, J. Pearkes, M. Pedersen, R. Pedro, S. V. Peleganchuk, O. Penc, C. Peng, H. Peng, M. Penzin, B. S. Peralva, A. P. Pereira Peixoto, L. Pereira Sanchez, D. V. Perepelitsa, E. Perez Codina, M. Perganti, L. Perini, H. Pernegger, A. Perrevoort, O. Perrin, K. Peters, R. F. Y. Peters, B. A. Petersen, T. C. Petersen, E. Petit, V. Petousis, C. Petridou, A. Petrukhin, M. Pettee, N. E. Pettersson, K. Petukhova, A. Peyaud, R. Pezoa, L. Pezzotti, G. Pezzullo, T. Pham, P. W. Phillips, M. W. Phipps, G. Piacquadio, E. Pianori, F. Piazza, R. Piegaia, D. Pietreanu, A. D. Pilkington, M. Pinamonti, J. L. Pinfold, C. Pitman

Donaldson, D. A. Pizzi, L. Pizzimento, A. Pizzini, M.-A. Pleier, V. Plesanovs, V. Pleskot, E. Plotnikova, G. Poddar, R. Poettgen, R. Poggi, L. Poggioli, I. Pogrebnyak, D. Pohl, I. Pokharel, S. Polacek, G. Polesello, A. Poley, R. Polifka, A. Polini, C. S. Pollard, Z. B. Pollock, V. Polychronakos, D. Ponomarenko, L. Pontecorvo, S. Popa, G. A. Popeneciu, D. M. Portillo Quintero, S. Pospisil, P. Postolache, K. Potamianos, I. N. Potrap, C. J. Potter, H. Potti, T. Poulsen, J. Poveda, G. Pownall, M. E. Pozo Astigarraga, A. Prades Ibanez, P. Pralavorio, M. M. Prapa, J. Pretel, D. Price, M. Primavera, M. A. Principe Martin, M. L. Proffitt, N. Proklova, K. Prokofiev, G. Proto, S. Protopopescu, J. Proudfoot, M. Przybycien, D. Pudzha, P. Puzo, D. Pyatiizbyantseva, J. Qian, Y. Qin, T. Qiu, A. Quadt, M. Queitsch-Maitland, G. Rabanal Bolanos, D. Rafanoharana, F. Ragusa, J. A. Raine, S. Rajagopalan, K. Ran, V. Raskina, D. F. Rassloff, S. Rave, B. Ravina, I. Ravinovich, M. Raymond, A. L. Read, N. P. Readioff, D. M. Rebuzzi, G. Redlinger, K. Reeves, D. Reikher, A. Reiss, A. Rej, C. Rembser, A. Renardi, M. Renda, M. B. Rendel, A. G. Rennie, S. Resconi, M. Ressegotti, E. D. Resseguie, S. Rettie, B. Reynolds, E. Reynolds, M. Rezaei Estabragh, O. L. Rezanova, P. Reznicek, E. Ricci, R. Richter, S. Richter, E. Richter-Was, M. Ridel, P. Rieck, P. Riedler, M. Rijssenbeek, A. Rimoldi, M. Rimoldi, L. Rinaldi, T. T. Rinn, M. P. Rinnagel, G. Ripellino, I. Riu, P. Rivadeneira, J. C. Rivera Vergara, F. Rizatdinova, E. Rizvi, C. Rizzi, B. A. Roberts, B. R. Roberts, S. H. Robertson, M. Robin, D. Robinson, C. M. Robles Gajardo, M. Robles Manzano, A. Robson, A. Rocchi, C. Roda, S. Rodriguez Bosca, Y. Rodriguez Garcia, A. Rodriguez Rodriguez, A. M. Rodríguez Vera, S. Roe, J. T. Roemer, A. R. Roepe-Gier, J. Roggel, O. Røhne, R. A. Rojas, B. Roland, C. P. A. Roland, J. Roloff, A. Romaniouk, M. Romano, A. C. Romero Hernandez, N. Rompotis, M. Ronzani, L. Roos, S. Rosati, B. J. Rosser, E. Rossi, E. Rossi, L. P. Rossi, L. Rossini, R. Rosten, M. Rotaru, B. Rottler, D. Rousseau, D. Rousso, G. Rovelli, A. Roy, A. Rozanov, Y. Rozen, X. Ruan, A. J. Ruby, T. A. Ruggeri, F. Rühr, A. Ruiz-Martinez, A. Rummler, Z. Rurikova, N. A. Rusakovich, H. L. Russell, L. Rustige, J. P. Rutherfoord, E. M. Rüttinger, K. Rybacki, M. Rybar, E. B. Rye, A. Ryzhov, J. A. Sabater Iglesias, P. Sabatini, L. Sabetta, H.F-W. Sadrozinski, F. Safai Tehrani, B. Safarzadeh Samani, M. Safdari, S. Saha, M. Sahinsoy, A. Sahu, M. Saimpert, M. Saito, T. Saito, D. Salamani, G. Salamanna, A. Salnikov, J. Salt, A. Salvador Salas, D. Salvatore, F. Salvatore, A. Salzburger, D. Sammel, D. Sampsonidis, D. Sampsonidou, J. Sánchez, A. Sanchez Pineda, V. Sanchez Sebastian, H. Sandaker, C. O. Sander, I. G. Sanderswood, J. A. Sandesara, M. Sandhoff, C. Sandoval, D. P. C. Sankey, A. Sansoni, C. Santoni, H. Santos, S. N. Santpur, A. Santra, K. A. Saoucha, J. G. Saraiva, J. Sardain, O. Sasaki, K. Sato, C. Sauer, F. Sauerburger, E. Sauvan, P. Savard, R. Sawada, C. Sawyer, L. Sawyer, I. Sayago Galvan, C. Sbarra, A. Sbrizzi, T. Scanlon, J. Schaarschmidt, P. Schacht, D. Schaefer, U. Schäfer, A. C. Schaffer, D. Schaile, R. D. Schamberger, E. Schanet, C. Scharf, N. Scharmberg, V. A. Schegelsky, D. Scheirich, F. Schenck, M. Schernau, C. Scheulen, C. Schiavi, Z. M. Schillaci, E. J. Schioppa,

M. Schioppa, B. Schlag, K. E. Schleicher, S. Schlenker, K. Schmieden, C. Schmitt, S. Schmitt, L. Schoeffel, A. Schoening, P. G. Scholer, E. Schopf, M. Schott, J. Schovancova, S. Schramm, F. Schroeder, H-C. Schultz-Coulon, M. Schumacher, B. A. Schumm, Ph. Schune, A. Schwartzman, T. A. Schwarz, Ph. Schwemling, R. Schwienhorst, A. Sciandra, G. Sciolla, F. Scuri, F. Scutti, C. D. Sebastiani, K. Sedlaczek, P. Seema, S. C. Seidel, A. Seiden, B. D. Seidlitz, T. Seiss, C. Seitz, J. M. Seixas, G. Sekhniaidze, S. J. Sekula, L. Selem, N. Semprini-Cesari, S. Sen, V. Senthilkumar, L. Serin, L. Serkin, M. Sessa, H. Severini, S. Sevova, F. Sforza, A. Sfyrla, E. Shabalina, R. Shaheen, J. D. Shahinian, N. W. Shaikh, D. Shaked Renous, L. Y. Shan, M. Shapiro, A. Sharma, A. S. Sharma, S. Sharma, P. B. Shatalov, K. Shaw, S. M. Shaw, P. Sherwood, L. Shi, C. O. Shimmin, Y. Shimogama, J. D. Shinner, I. P. J. Shipsey, S. Shirabe, M. Shiyakova, J. Shlomi, M. J. Shochet, J. Shojaii, D. R. Shope, S. Shrestha, E. M. Shrif, M. J. Shroff, P. Sicho, A. M. Sickles, E. Sideras Haddad, O. Sidiropoulou, A. Sidoti, F. Siegert, Dj. Sijacki, F. Sili, J. M. Silva, M. V. Silva Oliveira, S. B. Silverstein, S. Simion, R. Simoniello, E. L. Simpson, N. D. Simpson, S. Simsek, S. Sindhu, P. Sinervo, V. Sinetckii, S. Singh, S. Singh, S. Sinha, S. Sinha, M. Sioli, I. Siral, S. Yu. Sivoklokov, J. Sjölin, A. Skaf, E. Skorda, P. Skubic, M. Slawinska, V. Smakhtin, B. H. Smart, J. Smiesko, S.Yu. Smirnov, Y. Smirnov, L. N. Smirnova, O. Smirnova, E. A. Smith, H. A. Smith, R. Smith, M. Smizanska, K. Smolek, A. Smykiewicz, A. A. Snesarev, H. L. Snoek, S. Snyder, R. Sobie, A. Soffer, C. A. Solans Sanchez, E.Yu. Soldatov, U. Soldevila, A. A. Solodkov, S. Solomon, A. Soloshenko, K. Solovieva, O. V. Solovyanov, V. Solovyev, P. Sommer, H. Son, A. Sonay, W. Y. Song, A. Sopczak, A. L. Sopio, F. Sopkova, V. Sothilingam, S. Sottocornola, R. Soualah, Z. Soumaimi, D. South, S. Spagnolo, M. Spalla, M. Spangenberg, F. Spanò, D. Sperlich, G. Spigo, M. Spina, S. Spinali, D. P. Spiteri, M. Spousta, E. J. Staats, A. Stabile, R. Stamen, M. Stamenkovic, A. Stampekis, M. Standke, E. Stanecka, B. Stanislaus, M. M. Stanitzki, M. Stankaityte, B. Stapf, E. A. Starchenko, G. H. Stark, J. Stark, D. M. Starko, P. Staroba, P. Starovoitov, S. Stärz, R. Staszewski, G. Stavropoulos, J. Steentoft, P. Steinberg, A. L. Steinhebel, B. Stelzer, H. J. Stelzer, O. Stelzer-Chilton, H. Stenzel, T. J. Stevenson, G. A. Stewart, M. C. Stockton, G. Stoicea, M. Stolarski, S. Stonjek, A. Straessner, J. Strandberg, S. Strandberg, M. Strauss, T. Strebler, P. Strizenec, R. Ströhmer, D. M. Strom, L. R. Strom, R. Stroynowski, A. Strubig, S. A. Stucci, B. Stugu, J. Stupak, N. A. Styles, D. Su, S. Su, W. Su, X. Su, K. Sugizaki, V. V. Sulin, M. J. Sullivan, D. M. S. Sultan, L. Sultanaliyeva, S. Sultansoy, T. Sumida, S. Sun, S. Sun, O. Sunneborn Gudnadottir, M. R. Sutton, M. Svatos, M. Swiatlowski, T. Swirski, I. Sykora, M. Sykora, T. Sykora, D. Ta, K. Tackmann, A. Taffard, R. Tafirout, J. S. Tafoya Vargas, R. H. M. Taibah, R. Takashima, K. Takeda, E. P. Takeva, Y. Takubo, M. Talby, A. A. Talyshev, K. C. Tam, N. M. Tamir, A. Tanaka, J. Tanaka, R. Tanaka, J. Tang, Z. Tao, S. Tapia Araya, S. Tapprogge, A. Tarek Abouelfadl Mohamed, S. Tarem, K. Tariq, G. Tarna, G. F. Tartarelli, P. Tas, M. Tasevsky, E. Tassi, G. Tateno, Y. Tayalati, G. N. Taylor, W. Taylor,

H. Teagle, A. S. Tee, R. Teixeira De Lima, P. Teixeira-Dias, J. J. Teoh, K. Terashi, J. Terron, S. Terzo, M. Testa, R. J. Teuscher, N. Themistokleous, T. Theveneaux-Pelzer, O. Thielmann, D. W. Thomas, J. P. Thomas, E. A. Thompson, P. D. Thompson, E. Thomson, E. J. Thorpe, Y. Tian, V. Tikhomirov, Yu.A. Tikhonov, S. Timoshenko, E. X. L. Ting, P. Tipton, S. Tisserant, S. H. Tlou, A. Tnourji, K. Todome, S. Todorova-Nova, S. Todt, M. Togawa, J. Tojo, S. Tokár, K. Tokushuku, R. Tombs, M. Tomoto, L. Tompkins, P. Tornambe, E. Torrence, H. Torres, E. Torró Pastor, M. Toscani, C. Tosciri, D. R. Tovey, A. Traeet, I. S. Trandafir, C. J. Treado, T. Trefzger, A. Tricoli, I. M. Trigger, S. Trincaz-Duvoid, D. A. Trischuk, B. Trocmé, A. Trofymov, C. Troncon, F. Trovato, L. Truong, M. Trzebinski, A. Trzupek, F. Tsai, M. Tsai, A. Tsiamis, P. V. Tsiareshka, A. Tsirigotis, V. Tsiskaridze, E. G. Tskhadadze, M. Tsopoulou, Y. Tsujikawa, I. I. Tsukerman, V. Tsulaia, S. Tsuno, O. Tsur, D. Tsybychev, Y. Tu, A. Tudorache, V. Tudorache, A. N. Tuna, S. Turchikhin, I. Turk Cakir, R. Turra, P. M. Tuts, S. Tzamarias, P. Tzanis, E. Tzovara, K. Uchida, F. Ukegawa, P. A. Ulloa Poblete, G. Unal, M. Unal, A. Undrus, G. Unel, K. Uno, J. Urban, P. Urquijo, G. Usai, R. Ushioda, M. Usman, Z. Uysal, V. Vacek, B. Vachon, K. O. H. Vadla, T. Vafeiadis, C. Valderanis, E. Valdes Santurio, M. Valente, S. Valentinetti, A. Valero, A. Vallier, J. A. Valls Ferrer, T. R. Van Daalen, P. Van Gemmeren, S. Van Stroud, I. Van Vulpen, M. Vanadia, W. Vandelli, M. Vandenbroucke, E. R. Vandewall, D. Vannicola, L. Vannoli, R. Vari, E. W. Varnes, C. Varni, T. Varol, D. Varouchas, K. E. Varvell, M. E. Vasile, L. Vaslin, G. A. Vasquez, F. Vazeille, D. Vazquez Furelos, T. Vazquez Schroeder, J. Veatch, V. Vecchio, M. J. Veen, I. Veliscek, L. M. Veloce, F. Veloso, S. Veneziano, A. Ventura, A. Verbytskyi, M. Verducci, C. Vergis, M. Verissimo De Araujo, W. Verkerke, J. C. Vermeulen, C. Vernieri, P. J. Verschuuren, M. Vessella, M. L. Vesterbacka, M. C. Vetterli, A. Vgenopoulos, N. Viaux Maira, T. Vickey, O. E. Vickey Boeriu, G. H. A. Viehhauser, L. Vigani, M. Villa, M. Villaplana Perez, E. M. Villhauer, E. Vilucchi, M. G. Vincter, G. S. Virdee, A. Vishwakarma, C. Vittori, I. Vivarelli, V. Vladimirov, E. Voevodina, M. Vogel, P. Vokac, S. Voloshynovskiy, J. Von Ahnen, E. Von Toerne, B. Vormwald, V. Vorobel, K. Vorobev, M. Vos, J. H. Vossebeld, M. Vozak, L. Vozdecky, N. Vranjes, M. Vranjes Milosavljevic, V. Vrba, M. Vreeswijk, R. Vuillermet, O. Vujinovic, I. Vukotic, S. Wada, C. Wagner, W. Wagner, S. Wahdan, H. Wahlberg, R. Wakasa, M. Wakida, V. M. Walbrecht, J. Walder, R. Walker, W. Walkowiak, A. M. Wang, A. Z. Wang, C. Wang, C. Wang, H. Wang, J. Wang, P. Wang, R.-J. Wang, R. Wang, R. Wang, S. M. Wang, S. Wang, T. Wang, W. T. Wang, W. X. Wang, X. Wang, X. Wang, X. Wang, Y. Wang, Z. Wang, Z. Wang, Z. Wang, A. Warburton, R. J. Ward, N. Warrack, A. T. Watson, M. F. Watson, G. Watts, B. M. Waugh, A. F. Webb, C. Weber, M. S. Weber, S. A. Weber, S. M. Weber, C. Wei, Y. Wei, A. R. Weidberg, J. Weingarten, M. Weirich, C. Weiser, T. Wenaus, B. Wendland, T. Wengler, N. S. Wenke, N. Wermes, M. Wessels, K. Whalen, A. M. Wharton, A. S. White, A. White, M. J. White, D. Whiteson, L. Wickremasinghe, W. Wiedenmann,

C. Wiel, M. Wielers, N. Wieseotte, C. Wiglesworth, L. A. M. Wiik-Fuchs, D. J. Wilbern, H. G. Wilkens, D. M. Williams, H. H. Williams, S. Williams, S. Willocq, P. J. Windischhofer, F. Winklmeier, B. T. Winter, M. Wittgen, M. Wobisch, A. Wolf, R. Wölker, J. Wollrath, M. W. Wolter, H. Wolters, V. W. S. Wong, A. F. Wongel, S. D. Worm, B. K. Wosiek, K. W. Woźniak, K. Wraight, J. Wu, S. L. Wu, X. Wu, Y. Wu, Z. Wu, J. Wuerzinger, T. R. Wyatt, B. M. Wynne, S. Xella, L. Xia, M. Xia, J. Xiang, X. Xiao, M. Xie, X. Xie, I. Xiotidis, D. Xu, H. Xu, H. Xu, L. Xu, R. Xu, T. Xu, W. Xu, Y. Xu, Z. Xu, Z. Xu, B. Yabsley, S. Yacoob, N. Yamaguchi, Y. Yamaguchi, H. Yamauchi, T. Yamazaki, Y. Yamazaki, J. Yan, S. Yan, Z. Yan, H. J. Yang, H. T. Yang, S. Yang, T. Yang, X. Yang, X. Yang, Y. Yang, Z. Yang, W-M. Yao, Y. C. Yap, H. Ye, J. Ye, S. Ye, X. Ye, I. Yeletskikh, M. R. Yexley, P. Yin, K. Yorita, C. J. S. Young, C. Young, M. Yuan, R. Yuan, X. Yue, M. Zaazoua, B. Zabinski, G. Zacharis, E. Zaid, T. Zakareishvili, N. Zakharchuk, S. Zambito, J. Zang, D. Zanzi, O. Zaplatilek, S. V. Zeißner, C. Zeitnitz, J. C. Zeng, D. T. Zenger, O. Zenin, T. Ženiš, S. Zenz, S. Zerradi, D. Zerwas, B. Zhang, D. F. Zhang, G. Zhang, J. Zhang, K. Zhang, L. Zhang, M. Zhang, R. Zhang, S. Zhang, T. Zhang, X. Zhang, X. Zhang, Z. Zhang, H. Zhao, P. Zhao, T. Zhao, Y. Zhao, Z. Zhao, A. Zhemchugov, Z. Zheng, D. Zhong, B. Zhou, C. Zhou, H. Zhou, N. Zhou, Y. Zhou, C. G. Zhu, C. Zhu, H. L. Zhu, H. Zhu, J. Zhu, Y. Zhu, X. Zhuang, K. Zhukov, V. Zhulanov, D. Zieminska, N. I. Zimine, S. Zimmermann, J. Zinsser, M. Ziolkowski, L. Živković, A. Zoccoli, K. Zoch, T. G. Zorbas, O. Zormpa, W. Zou, and L. Zwalinski. Deep generative models for fast photon shower simulation in atlas. *Computing and Software for Big Science*, 8(1), March 2024. ISSN 2510-2044. . URL http://dx.doi.org/10.1007/s41781-023-00106-9.

S. Abdullin, V. Abramov, B. Acharya, N. Adam, M. Adams, N. Akchurin, U. Akgun, E. Albayrak, E. W. Anderson, G. Antchev, M. Arcidy, S. Ayan, S. Aydin, T. Aziz, M. Baarmand, K. Babich, D. Baden, M. N. Bakirci, Sudeshna Banerjee, Sunanda Banerjee, R. Bard, V. Barnes, H. Bawa, G. Baiatian, G. Bencze, S. Beri, L. Berntzon, V. Bhandari, V. Bhatnagar, A. Bhatti, A. Bodek, S. Bose, T. Bose, H. Budd, K. Burchesky, T. Camporesi, K. Cankoçak, K. Carrell, S. Cerci, S. Chendvankar, Y. Chung, W. Clarida, L. Cremaldi, P. Cushman, J. Damgov, P. de Barbaro, P. Debbins, M. Deliomeroglu, A. Demianov, T. de Visser, P. V. Deshpande, J. Diaz, L. Dimitrov, S. Dugad, I. Dumanoglu, F. Duru, I. Efthymiopoulos, J. Elias, D. Elvira, I. Emeliantchik, S. Eno, A. Ershov, S. Erturk, S. Esen, E. Eskut, A. Fenyvesi, W. Fisher, J. Freeman, S. N. Ganguli, V. Gaultney, H. Gamsizkan, V. Gavrilov, V. Genchev, S. Gleyzer, I. Golutvin, P. Goncharov, T. Grassi, D. Green, A. Gribushin, B. Grinev, M. Guchait, A. Gurtu, A. Murat Güler, E. Gülmez, K. Gümüş, T. Haelen, S. Hagopian, V. Hagopian, V. Halyo, M. Hashemi, J. Hauptman, E. Hazen, A. Heering, A. Heister, A. Hunt, N. Ilyina, D. Ingram, E. Isiksal, C. Jarvis, C. Jeong, K. Johnson, J. Jones, V. Kaftanov, V. Kalagin, A. Kalinin, S. Kalmani, D. Karmgard, M. Kaur, M. Kaya, O. Kaya, A. Kayis-Topaksu, R. Kellogg, A. Khmelnikov, H. Kim, I. Kisselevich, O. Kodolova, J. Kohli, V. Kolossov, A. Korablev, Y. Korneev,

I. Kosarev, L. Kramer, A. Krinitsyn, M. R. Krishnaswamy, A. Krokhotin, V. Kryshkin, S. Kuleshov, A. Kumar, S. Kunori, A. Laasanen, V. Ladygin, E. Laird, G. Landsberg, A. Laszlo, C. Lawlor, D. Lazic, S. W. Lee, L. Levchuk, S. Linn, D. Litvintsev, L. Lobolo, S. Los, V. Lubinsky, V. Lukanin, Y. Ma, E. Machado, M. Maity, G. Majumder, J. Mans, D. Marlow, P. Markowitz, G. Martinez, K. Mazumdar, J. P. Merlo, H. Mermerkaya, G. Mescheryakov, A. Mestvirishvili, M. Miller, A. Moeller, M. Mohammadi-Najafabadi, P. Moissenz, N. Mondal, V. Mossolov, P. Nagaraj, V. S. Narasimham, E. Norbeck, J. Olson, Y. Onel, G. Onengut, C. Ozkan, H. Ozkurt, S. Ozkorucuklu, F. Ozok, S. Paktinat, A. Pal, M. Patil, A. Penzo, S. Petrushanko, A. Petrosyan, V. Pikalov, S. Piperov, V. Podrasky, A. Polatoz, A. Pompos, S. Popescu, C. Posch, A. Pozdnyakov, W. Qian, R. M. Ralich, L. Reddy, J. Reidy, E. Rogalev, Y. Roh, J. Rohlf, A. Ronzhin, R. Ruchti, A. Ryazanov, G. Safronov, D. A. Sanders, C. Sanzeni, L. Sarycheva, B. Satyanarayana, I. Schmidt, S. Sekmen, S. Semenov, V. Senchishin, S. Sergeyev, M. Serin, R. Sever, B. Singh, J. B. Singh, A. Sirunyan, A. Skuja, S. Sharma, B. Sherwood, N. Shumeiko, V. Smirnov, K. Sogut, N. Sonmez, P. Sorokin, M. Spezziga, R. Stefanovich, V. Stolin, K. Sudhakar, L. Sulak, I. Suzuki, V. Talov, K. Teplov, R. Thomas, S. Tonwar, H. Topakli, C. Tully, L. Turchanovich, A. Ulyanov, A. Vanini, I. Vankov, I. Vardanyan, F. Varela, M. Vergili, P. Verma, G. Vesztergombi, R. Vidal, A. Vishnevskiy, E. Vlassov, I. Vodopiyanov, I. Volobouev, A. Volkov, A. Volodko, L. Wang, J. Werner, M. Wetstein, D. Winn, R. Wigmans, J. Whitmore, S. X. Wu, E. Yazgan, T. Yetkin, P. Zalan, A. Zarubin, M. Zeyrek, and CMS HCAL Collaborations. Design, performance, and calibration of the CMS hadron-outer calorimeter. *The European Physical Journal C*, 57(3):653–663. ISSN 1434-6052. . URL `https://doi.org/10.1140/epjc/s10052-008-0756-6`.

V.V. Abramov, B.S. Acharya, N. Akchurin, I. Atanasov, G. Baiatian, A. Ball, S. Banerjee, S. Banerjee, P. de Barbaro, V. Barnes, G. Bencze, A. Bodek, M. Booke, H. Budd, L. Cremaldi, P. Cushman, S.R. Dugad, L. Dimitrov, A. Dyshkant, J. Elias, V.N. Evdokimov, D. Fong, J. Freeman, V. Genchev, P.I. Goncharov, D. Green, A. Gurtu, V. Hagopian, P. Iaydjiev, Yu. Korneev, A. Krinitsyn, G. Krishnaswami, M.R. Krishnaswamy, V. Kryshkin, S. Kunori, A. Laasanen, D. Lazic, L. Levchuk, L. Litov, N.K. Mondal, T. Moulik, V.S. Narasimham, A. Nemashkalo, Y. Onel, P. Petrov, Yu. Petukhov, S. Piperov, V. Popov, J. Reidy, A. Ronzhin, R. Ruchti, J.B. Singh, Q. Shen, A. Sirunyan, A. Skuja, E. Skup, P. Sorokin, K. Sudhakar, D. Summers, F. Szoncso, S.I. Tereshenko, C. Timmermans, S.C. Tonwar, L. Turchanovich, V. Tyukov, A. Volodko, A. Yukaev, A. Zaitchenko, and A. Zatserklyaniy. Studies of the response of the prototype cms hadron calorimeter, including magnetic field effects, to pion, electron, and muon beams. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 457(1): 75–100, 2001. ISSN 0168-9002. . URL `https://www.sciencedirect.com/science/article/pii/S0168900200007117`.

S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, F. Behner, L. Bellagamba, J. Boudreau, L. Broglia, A. Brunengo, H. Burkhardt, S. Chauvie, J. Chuma, R. Chytracek, G. Cooperman, G. Cosmo, P. Degtyarenko, A. Dell'Acqua, G. Depaola, D. Dietrich, R. Enami, A. Feliciello, C. Ferguson, H. Fesefeldt, G. Folger, F. Foppiano, A. Forti, S. Garelli, S. Giani, R. Giannitrapani, D. Gibin, J.J. Gómez Cadenas, I. González, G. Gracia Abril, G. Greeniaus, W. Greiner, V. Grichine, A. Grossheim, S. Guatelli, P. Gumplinger, R. Hamatsu, K. Hashimoto, H. Hasui, A. Heikkinen, A. Howard, V. Ivanchenko, A. Johnson, F.W. Jones, J. Kallenbach, N. Kanaya, M. Kawabata, Y. Kawabata, M. Kawaguti, S. Kelner, P. Kent, A. Kimura, T. Kodama, R. Kokoulin, M. Kossov, H. Kurashige, E. Lamanna, T. Lampén, V. Lara, V. Lefebure, F. Lei, M. Liendl, W. Lockman, F. Longo, S. Magni, M. Maire, E. Medernach, K. Minamimoto, P. Mora de Freitas, Y. Morita, K. Murakami, M. Nagamatu, R. Nartallo, P. Nieminen, T. Nishimura, K. Ohtsubo, M. Okamura, S. O'Neale, Y. Oohata, K. Paech, J. Perl, A. Pfeiffer, M.G. Pia, F. Ranjard, A. Rybin, S. Sadilov, E. Di Salvo, G. Santin, T. Sasaki, N. Savvas, Y. Sawada, S. Scherer, S. Sei, V. Sirotenko, D. Smith, N. Starkov, H. Stoecker, J. Sulkimo, M. Takahata, S. Tanaka, E. Tcherniaev, E. Safai Tehrani, M. Tropeano, P. Truscott, H. Uno, L. Urban, P. Urban, M. Verderi, A. Walkden, W. Wander, H. Weber, J.P. Wellisch, T. Wenaus, D.C. Williams, D. Wright, T. Yamada, H. Yoshida, and D. Zschiesche. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003. ISSN 0168-9002. . URL https://www.sciencedirect.com/science/article/pii/S0168900203013688.

J. Alwall, A. Ballestrero, P. Bartalini, S. Belov, E. Boos, A. Buckley, J.M. Butterworth, L. Dudko, S. Frixione, L. Garren, S. Gieseke, A. Gusev, I. Hinchliffe, J. Huston, B. Kersevan, F. Krauss, N. Lavesson, L. Lönnblad, E. Maina, F. Maltoni, M.L. Mangano, F. Moortgat, S. Mrenna, C.G. Papadopoulos, R. Pittau, P. Richardson, M.H. Seymour, A. Sherstnev, T. Sjöstrand, P. Skands, S.R. Slabospitsky, Z. Was, B.R. Webber, M. Worek, and D. Zeppenfeld. A standard format for les houches event files. *Computer Physics Communications*, 176(4):300–304, February 2007. ISSN 0010-4655. . URL http://dx.doi.org/10.1016/j.cpc.2006.11.010.

Johan Alwall, Michel Herquet, Fabio Maltoni, Olivier Mattelaer, and Tim Stelzer. Madgraph 5: going beyond. *Journal of High Energy Physics*, 2011(6), June 2011. ISSN 1029-8479. . URL http://dx.doi.org/10.1007/JHEP06(2011)128.

Carl D. Anderson. The positive electron. *Phys. Rev.*, 43:491–494, Mar 1933. . URL https://link.aps.org/doi/10.1103/PhysRev.43.491.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users'*

Guide. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

A. Banfi, G.P. Salam, and G. Zanderighi. Infrared-safe definition of jet flavour. *The European Physical Journal C*, 47(1):113–124, May 2006. ISSN 1434-6052. . URL http://dx.doi.org/10.1140/epjc/s2006-02552-4.

W. Bartel, L. Becker, R. Felst, D. Haidt, G. Knies, H. Krehbiel, P. Laurikainen, N. Magnussen, R. Meinke, B. Naroska, J. Olsson, D. Schmidt, G. Dietrich, T. Greenshaw, J. Hagemann, G. Heinzelmann, H. Kado, C. Kleinwort, M. Kuhlen, A. Petersen, R. Ramcke, U. Schneekloth, G. Weber, K. Ambrus, S. Bethke, A. Dieckmann, E. Elsen, J. Heintze, K. H. Hellenbrand, S. Komamiya, J. von Krogh, H. Rieseberg, H. v. d. Schmitt, L. Smolik, J. Spitzer, A. Wagner, M. Zimmer, C. K. Bowdery, A. J. Finch, F. Foster, G. Hughes, J. M. Nye, J. Allison, R. J. Barlow, J. Chrin, I. P. Duerdoth, F. K. Loebinger, A. A. Macbeth, H. E. Mills, P. G. Murphy, K. Stephens, P. Warming, R. G. Glasser, P. Hill, J. A. J. Skard, S. R. Wagner, G. T. Zorn, S. L. Cartwright, D. Clarke, R. Marshall, R. P. Middleton, K. Kawagoe, T. Mashimo, T. Takeshita, S. Yamada, and JADE Collaboration. Experimental studies on multijet production in e+e- annihilation at PETRA energies. *Zeitschrift für Physik C Particles and Fields*, 33(1):23–31. ISSN 1431-5858. . URL https://doi.org/10.1007/BF01410449.

Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics, 2016. URL https://arxiv.org/abs/1612.00222.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018. URL https://arxiv.org/abs/1806.01261.

G. L. Bayatian et al. CMS Physics: Technical Design Report Volume 1: Detector Performance and Software. 2006.

Gavin Bewick, Silvia Ferrario Ravasio, Stefan Gieseke, Stefan Kiebacher, Mohammad R. Masouminia, Andreas Papaefstathiou, Simon Plätzer, Peter Richardson, Daniel Samitz, Michael H. Seymour, Andrzej Siódmok, and James Whitehead. Herwig 7.3 release note, 2024. URL https://arxiv.org/abs/2312.05175.

Christian Bierlich, Smita Chakraborty, Nishita Desai, Leif Gellersen, Ilkka Helenius, Philip Ilten, Leif Lönnblad, Stephen Mrenna, Stefan Prestel, Christian T. Preuss, Torbjörn Sjöstrand, Peter Skands, Marius Utheim, and Rob Verheyen. A

comprehensive guide to the physics and usage of pythia 8.3, 2022. URL https://arxiv.org/abs/2203.11601.

L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

M. Boronat, C. Marinas, A. Frey, I. Garcia, B. Schwenker, M. Vos, and F. Wilk. Physical limitations to the spatial resolution of solid-state detectors. *IEEE Trans. Nucl. Sci.*, 62 (1):381–386, 2015. . URL https://cds.cern.ch/record/2002654. Comments: 5 pages.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021. URL https://arxiv.org/abs/2104.13478.

Jonathan M. Butterworth, Adam R. Davison, Mathieu Rubin, and Gavin P. Salam. Jet substructure as a new higgs-search channel at the large hadron collider. *Physical Review Letters*, 100(24), June 2008. ISSN 1079-7114. . URL http://dx.doi.org/10.1103/PhysRevLett.100.242001.

Manuel Bähr, Stefan Gieseke, Martyn A. Gigg, David Grellscheid, Keith Hamilton, Oluseyi Latunde-Dada, Simon Plätzer, Peter Richardson, Michael H. Seymour, Alexander Sherstnev, and Bryan R. Webber. Herwig++ physics and manual. *The European Physical Journal C*, 58(4):639–707, November 2008. ISSN 1434-6052. . URL http://dx.doi.org/10.1140/epjc/s10052-008-0798-9.

M. Cacciari and G. P. Salam. Phys. *Lett. B*, 641, 2006. 57 [hep-ph/0512210.

M. Cacciari, G. P. Salam, and G. Soyez. Eur. *Phys. J. C*, 72, 2012. 1896 [arXiv:1111.6097 [hep-ph.

Matteo Cacciari, Juan Rojo, Gavin P Salam, and Gregory Soyez. Quantifying the performance of jet definitions for kinematic reconstruction at the lhc. *Journal of High Energy Physics*, 2008(12):032–032, December 2008. ISSN 1029-8479. . URL http://dx.doi.org/10.1088/1126-6708/2008/12/032.

Stefano Carrazza and Frédéric A. Dreyer. Lund jet images from generative and cycle-consistent adversarial networks. *The European Physical Journal C*, 79(11), November 2019. ISSN 1434-6052. . URL http://dx.doi.org/10.1140/epjc/s10052-019-7501-1.

Antonio Cassese. The cms pixel detector for the high luminosity lhc. *Journal of Instrumentation*, 18, 2022. URL https://api.semanticscholar.org/CorpusID:253027178.

S. Catani, Yu.L. Dokshitzer, M. Olsson, G. Turnock, and B.R. Webber. New clustering algorithm for multijet cross sections in e+e- annihilation. *Physics Letters B*, 269(3):

432–438, 1991. ISSN 0370-2693. . URL https://www.sciencedirect.com/science/article/pii/037026939190196W.

Giorgio Cerro, Srinandan Dasmahapatra, Henry A. Day-Hall, Billy Ford, Stefano Moretti, and Claire H. Shepherd-Themistocleous. Spectral clustering for jet physics. *Journal of High Energy Physics*, 2022(2), February 2022. ISSN 1029-8479. . URL http://dx.doi.org/10.1007/JHEP02(2022)165.

Amit Chakraborty, Srinandan Dasmahapatra, Henry Day-Hall, Billy Ford, Shubhani Jain, Stefano Moretti, Emmanuel Olaiya, and Claire Shepherd-Themistocleous. Revisiting jet clustering algorithms for new higgs boson searches in hadronic final states, 2022. URL https://arxiv.org/abs/2008.02499.

Jacan Chaplais. Unified and semantic processing for heterogeneous monte-carlo data, October 2023. URL https://doi.org/10.5281/zenodo.10014777.

Jacan Chaplais. Colliderscope: particle physics data made beautiful and interactive, June 2025a. URL https://github.com/jacanchaplais/colliderscope.

Jacan Chaplais. Heparchy: hierarchical HEP data storage with HDF5, June 2025b. URL https://github.com/jacanchaplais/heparchy.

Jacan Chaplais. Showerpipe: Pythonic pipeline for Monte-Carlo showering and hadronisation, June 2025c. URL https://github.com/jacanchaplais/showerpipe.

Jacan Chaplais. jacanchaplais/hep-tutorial: 0.1.0, June 2025d. URL https://doi.org/10.5281/zenodo.15777952.

Jacan Chaplais and Giorgio Cerro. Graphicle: semantic and heterogeneous data structures for particle physics, June 2025. URL https://github.com/jacanchaplais/graphicle.

Sang Kwan Choi, Jinmian Li, Cong Zhang, and Rao Zhang. Automatic detection of boosted higgs boson and top quark jets in an event image. *Phys. Rev. D*, 108:116002, Dec 2023. . URL https://link.aps.org/doi/10.1103/PhysRevD.108.116002.

Saransh Chopra, Henry Schreiner, Eduardo Rodrigues, Jonas Eschle, and Jim Pivarski. Vector: JIT-compilable mathematical manipulations of ragged Lorentz vectors. *Journal of Open Source Software*, 10(109):7791, May 2025. . URL https://joss.theoj.org/papers/10.21105/joss.07791.

F Close. The quark parton model. *Reports on Progress in Physics*, 42(8):1285, aug 1979. . URL https://dx.doi.org/10.1088/0034-4885/42/8/001.

Josh Cogan, Michael Kagan, Emanuel Strauss, and Ariel Schwartzman. Jet-images: computer vision inspired techniques for jet tagging. *Journal of High Energy Physics*, 2015(2), February 2015. ISSN 1029-8479. . URL http://dx.doi.org/10.1007/JHEP02(2015)118.

Noel Dawe, Eduardo Rodrigues, Henry Schreiner, Matthieu Marinangeli, Samuel Meehan, and delgadoandrea. scikit-hep/numpythia: Version 1.3.0, January 2023. URL https://doi.org/10.5281/zenodo.7510415.

Luke de Oliveira, Michael Kagan, Lester Mackey, Benjamin Nachman, and Ariel Schwartzman. Jet-images — deep learning edition. *Journal of High Energy Physics*, 2016(7), July 2016. ISSN 1029-8479. . URL http://dx.doi.org/10.1007/JHEP07(2016)069.

Barry Dillon, Gregor Kasieczka, Hans Olischlager, Tilman Plehn, Peter Sorrenson, and Lorenz Vogel. Symmetries, safety, and self-supervision. *SciPost Physics*, 12(6), June 2022. ISSN 2542-4653. . URL http://dx.doi.org/10.21468/SciPostPhys.12.6.188.

Barry M. Dillon, Darius A. Faroughy, and Jernej F. Kamenik. Uncovering latent jet substructure. *Phys. Rev. D*, 100:056002, Sep 2019. . URL https://link.aps.org/doi/10.1103/PhysRevD.100.056002.

Paul A. M. Dirac. The quantum theory of the electron. *Proc. Roy. Soc. Lond. A*, 117: 610–624, 1928. .

Yu.L Dokshitzer, G.D Leder, S Moretti, and B.R Webber. Better jet clustering algorithms. *Journal of High Energy Physics*, 1997(08):001–001, August 1997. ISSN 1029-8479. . URL http://dx.doi.org/10.1088/1126-6708/1997/08/001.

Javier Duarte and Jean-Roch Vlimant. *Graph Neural Networks for Particle Tracking and Reconstruction*, page 387–436. WORLD SCIENTIFIC, February 2022. ISBN 9789811234033. . URL http://dx.doi.org/10.1142/9789811234033_0012.

Arthur Stanley Eddington. *The Philosophy of Physical Science*. University of Michigan Press, Ann Arbor, 1939.

R. K. Ellis, W. J. Stirling, and B. R. Webber. *QCD and Collider Physics*. Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology. Cambridge University Press, 1996.

Stephen D. Ellis and Davison E. Soper. Successive combination jet algorithm for hadron collisions. *Physical Review D*, 48(7):3160–3166, October 1993. ISSN 0556-2821. . URL http://dx.doi.org/10.1103/PhysRevD.48.3160.

Stephen D. Ellis, Zoltan Kunszt, and Davison E. Soper. One-jet inclusive cross section at order α_s^3. gluons only. *Phys. Rev. D*, 40:2188–2222, Oct 1989. . URL https://link.aps.org/doi/10.1103/PhysRevD.40.2188.

Stephen D. Ellis, Christopher K. Vermilion, and Jonathan R. Walsh. Techniques for improved heavy particle searches with jet substructure. *Phys. Rev. D*, 80:051501, Sep 2009. . URL https://link.aps.org/doi/10.1103/PhysRevD.80.051501.

Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra. *Head First Design Patterns*. O' Reilly & Associates, Inc., 2004. ISBN 0596007124.

Jason Gallicchio and Matthew D. Schwartz. Seeing in color: Jet superstructure. *Physical Review Letters*, 105(2), July 2010. ISSN 1079-7114. . URL http://dx.doi.org/10.1103/PhysRevLett.105.022001.

Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, 1994. ISBN 0201633612.

M. Gell-Mann. A schematic model of baryons and mesons. *Physics Letters*, 8(3): 214–215, February 1964. .

N. N. Das Gupta and S. K. Ghosh. A report on the wilson cloud chamber and its applications in physics. *Rev. Mod. Phys.*, 18:225–290, Apr 1946. . URL https://link.aps.org/doi/10.1103/RevModPhys.18.225.

Aric Hagberg, Pieter Swart, and Daniel Chult. Exploring network structure, dynamics, and function using networkx. 06 2008. .

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. . URL https://doi.org/10.1038/s41586-020-2649-2.

Andrew Hunt and David Thomas. *The Pragmatic Programmer: Your Journey to Mastery, 20th Anniversary Edition*. Addison-Wesley, 2nd edition, 2019. ISBN 9780135957059.

John E. Huth et al. Toward a standardization of jet definitions. In *1990 DPF Summer Study on High-energy Physics: Research Directions for the Decade (Snowmass 90)*, pages 0134–136, 12 1990.

Stefan Höche. Introduction to parton-shower event generators, 2015. URL https://arxiv.org/abs/1411.4085.

George Johnson. *Strange Beauty: Murray Gell-Mann and the Revolution in Twentieth-Century Physics*. 1999. footnote 108: Oppenheimer coined the term "subnuclear zoo" in a public lecture at the Rochester VI conference; Sec VIII, p 1 of the proceedings.

Xiangyang Ju and Benjamin Nachman. Supervised jet clustering with graph neural networks for lorentz boosted bosons. *Physical Review D*, 102(7), October 2020. ISSN 2470-0029. . URL http://dx.doi.org/10.1103/PhysRevD.102.075014.

Gregor Kasieczka, Tilman Plehn, Michael Russell, and Torben Schell. Deep-learning top taggers or the end of qcd? *Journal of High Energy Physics*, 2017(5), May 2017. ISSN 1029-8479. . URL http://dx.doi.org/10.1007/JHEP05(2017)006.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907.

Debra Knisley and Jeff Knisley. *Graph Theoretic Models in Chemistry and Molecular Biology*, chapter 3, pages 85–113. John Wiley & Sons, Ltd, 2008. ISBN 9780470175668. . URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470175668.ch3.

Patrick T. Komiske, Eric M. Metodiev, and Jesse Thaler. Energy flow networks: deep sets for particle jets. *Journal of High Energy Physics*, 2019(1), January 2019. ISSN 1029-8479. . URL http://dx.doi.org/10.1007/JHEP01(2019)121.

Partha Konar, Vishal S. Ngairangbam, and Michael Spannowsky. Energy-weighted message passing: an infra-red and collinear safe graph neural network algorithm. *Journal of High Energy Physics*, 2022(2), February 2022. ISSN 1029-8479. . URL http://dx.doi.org/10.1007/JHEP02(2022)060.

Danek Kotliński. The cms pixel detector. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 465 (1):46–50, 2001. ISSN 0168-9002. . URL https://www.sciencedirect.com/science/article/pii/S016890020100345X. SPD2000.

David Krohn, Jesse Thaler, and Lian-Tao Wang. Jet trimming. *Journal of High Energy Physics*, 2010(2):84. ISSN 1029-8479. . URL https://doi.org/10.1007/JHEP02(2010)084.

David Krohn, Jesse Thaler, and Lian-Tao Wang. Jets with variable r. *Journal of High Energy Physics*, 2009(06):059, jun 2009. . URL https://dx.doi.org/10.1088/1126-6708/2009/06/059.

Andrew J. Larkoski, Simone Marzani, Gregory Soyez, and Jesse Thaler. Soft drop. *Journal of High Energy Physics*, 2014(5):146. ISSN 1029-8479. . URL https://doi.org/10.1007/JHEP05(2014)146.

Andrew J. Larkoski, Ian Moult, and Benjamin Nachman. Jet substructure at the large hadron collider: A review of recent advances in theory and machine learning. *Physics Reports*, 841:1–63, January 2020. ISSN 0370-1573. . URL http://dx.doi.org/10.1016/j.physrep.2019.11.001.

Jurij Leskovec. Cs224w | machine learning with graphs, 2021. URL https://web.stanford.edu/class/cs224w/index.html.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018. URL https://arxiv.org/abs/1708.02002.

Gilles Louppe, Kyunghyun Cho, Cyril Becot, and Kyle Cranmer. Qcd-aware recursive neural networks for jet physics. *Journal of High Energy Physics*, 2019(1), January 2019. ISSN 1029-8479. . URL http://dx.doi.org/10.1007/JHEP01(2019)057.

Sebastian Macaluso and David Shih. Pulling out all the tops with computer vision and deep learning. *Journal of High Energy Physics*, 2018(10), October 2018. ISSN 1029-8479. . URL http://dx.doi.org/10.1007/JHEP10(2018)121.

TorchVision maintainers and contributors. TorchVision: PyTorch's Computer Vision library, November 2016. URL https://github.com/pytorch/vision.

Robert C. Martin and James O. Coplien. *Clean code: a handbook of agile software craftsmanship*. Prentice Hall, Upper Saddle River, NJ [etc.], 2009. ISBN 9780132350884 0132350882.

Simone Marzani, Gregory Soyez, and Michael Spannowsky. *Looking Inside Jets: An Introduction to Jet Substructure and Boosted-object Phenomenology*. Springer International Publishing, 2019. ISBN 9783030157098. . URL http://dx.doi.org/10.1007/978-3-030-15709-8.

V. Mikuni and F. Canelli. Abcnet: an attention-based method for particle tagging. *The European Physical Journal Plus*, 135(6), June 2020. ISSN 2190-5444. . URL http://dx.doi.org/10.1140/epjp/s13360-020-00497-3.

Gert Molière. Zur Theorie der Luftschauer. *Zeitschrift für Physik*, 125(4):250–268, 1948. ISSN 0044-3328. . URL https://doi.org/10.1007/BF01454894.

E. F. Moore. The shortest path through a maze. In *Proc. International Symposium on the Theory of Switching, Part II*. 1959.

Stefano Moretti, Leif Lönnblad, and Torbjörn Sjöstrand. New and old jet clustering algorithms for electron-positron events. *Journal of High Energy Physics*, 1998(08): 001–001, August 1998. ISSN 1029-8479. . URL http://dx.doi.org/10.1088/1126-6708/1998/08/001.

S. Navas et al. Review of particle physics. *Phys. Rev. D*, 110(3):030001, 2024. .

Hans Olischläger. Jetclr: Contrastive learning of jet representations, September 2021. Bachelor's thesis, Department of Physics and Astronomy, Heidelberg University.

The pandas development team. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

Particle Data Group, P A Zyla, R M Barnett, J Beringer, O Dahl, D A Dwyer, D E Groom, C J Lin, K S Lugovsky, E Pianori, D J Robinson, C G Wohl, W M Yao, K Agashe, G Aielli, B C Allanach, C Amsler, M Antonelli, E C Aschenauer, D M Asner, H Baer, Sw Banerjee, L Baudis, C W Bauer, J J Beatty, V I Belousov, S Bethke, A Bettini, O Biebel, K M Black, E Blucher, O Buchmuller, V Burkert, M A Bychkov, R N Cahn, M Carena, A Ceccucci, A Cerri, D Chakraborty, R Sekhar Chivukula, G Cowan, G D'Ambrosio, T Damour, D de Florian, A de Gouvêa, T DeGrand, P de Jong, G Dissertori, B A Dobrescu, M D'Onofrio, M Doser, M Drees, H K Dreiner, P Eerola, U Egede, S Eidelman, J Ellis, J Erler, V V Ezhela, W Fetscher, B D Fields, B Foster, A Freitas, H Gallagher, L Garren, H J Gerber, G Gerbier, T Gershon, Y Gershtein, T Gherghetta, A A Godizov, M C Gonzalez-Garcia, M Goodman, C Grab, A V Gritsan, C Grojean, M Grünewald, A Gurtu, T Gutsche, H E Haber, C Hanhart, S Hashimoto, Y Hayato, A Hebecker, S Heinemeyer, B Heltsley, J J Hernández-Rey, K Hikasa, J Hisano, A Höcker, J Holder, A Holtkamp, J Huston, T Hyodo, K F Johnson, M Kado, M Karliner, U F Katz, M Kenzie, V A Khoze, S R Klein, E Klempt, R V Kowalewski, F Krauss, M Kreps, B Krusche, Y Kwon, O Lahav, J Laiho, L P Lellouch, J Lesgourgues, A R Liddle, Z Ligeti, C Lippmann, T M Liss, L Littenberg, C Lourengo, S B Lugovsky, A Lusiani, Y Makida, F Maltoni, T Mannel, A V Manohar, W J Marciano, A Masoni, J Matthews, U G Meißner, M Mikhasenko, D J Miller, D Milstead, R E Mitchell, K Mönig, P Molaro, F Moortgat, M Moskovic, K Nakamura, M Narain, P Nason, S Navas, M Neubert, P Nevski, Y Nir, K A Olive, C Patrignani, J A Peacock, S T Petcov, V A Petrov, A Pich, A Piepke, A Pomarol, S Profumo, A Quadt, K Rabbertz, J Rademacker, G Raffelt, H Ramani, M Ramsey-Musolf, B N Ratcliff, P Richardson, A Ringwald, S Roesler, S Rolli, A Romaniouk, L J Rosenberg, J L Rosner, G Rybka, M Ryskin, R A Ryutin, Y Sakai, G P Salam, S Sarkar, F Sauli, O Schneider, K Scholberg, A J Schwartz, J Schwiening, D Scott, V Sharma, S R Sharpe, T Shutt, M Silari, T Sjöstrand, P Skands, T Skwarnicki, G F Smoot, A Soffer, M S Sozzi, S Spanier, C Spiering, A Stahl, S L Stone, Y Sumino, T Sumiyoshi, M J Syphers, F Takahashi, M Tanabashi, J Tanaka, M Taševský, K Terashi, J Terning, U Thoma, R S Thorne, L Tiator, M Titov, N P Tkachenko, D R Tovey, K Trabelsi, P Urquijo, G Valencia, R Van de Water, N Varelas, G Venanzoni, L Verde, M G Vincter, P Vogel, W Vogelsang, A Vogt, V Vorobyev, S P Wakely, W Walkowiak, C W Walter, D Wands, M O Wascko, D H Weinberg, E J Weinberg, M White, L R Wiencke, S Willocq, C L Woody, R L Workman, M Yokoyama, R Yoshida, G Zanderighi, G P Zeller, O V Zenin, R Y Zhu, S L Zhu, F Zimmermann, J Anderson, T Basaglia, V S Lugovsky, P Schaffner, and W Zheng. Review of Particle Physics. *Progress of Theoretical and Experimental Physics*, 2020(8): 083C01, 08 2020. ISSN 2050-3911. . URL `https://doi.org/10.1093/ptep/ptaa104`.

Huilin Qu and Loukas Gouskos. Jet tagging via particle clouds. *Physical Review D*, 101 (5), March 2020. ISSN 2470-0029. . URL `http://dx.doi.org/10.1103/PhysRevD.101.056019`.

Huilin Qu, Congqiao Li, and Sitian Qian. Particle transformer for jet tagging, 2024. URL https://arxiv.org/abs/2202.03772.

Chiara Rizzi. Searches for supersymmetric particles in final states with multiple top and bottom quarks with the atlas detector, 2018. URL https://cds.cern.ch/record/2646377. Presented 15 Oct 2018.

Eduardo Rodrigues and Henry Schreiner. Particle. URL https://github.com/scikit-hep/particle.

Eduardo Rodrigues et al. The Scikit HEP Project – overview and prospects. *EPJ Web Conf.*, 245:06028, 2020. .

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958. URL https://api.semanticscholar.org/CorpusID:12781225.

Gavin P. Salam. Towards jetography. *The European Physical Journal C*, 67(3–4):637–686, May 2010. ISSN 1434-6052. . URL http://dx.doi.org/10.1140/epjc/s10052-010-1314-6.

Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, January 2021. ISSN 2632-2153. . URL http://dx.doi.org/10.1088/2632-2153/abbf9a.

Shrek, 2001. Directed by Andrew Adamson and Vicky Jenson, performed by Mike Myers and Eddie Murphy.

A.M. Sirunyan, A. Tumasyan, W. Adam, E. Asilar, T. Bergauer, J. Brandstetter, E. Brondolin, M. Dragicevic, J. Erö, M. Flechl, M. Friedl, R. Frühwirth, V.M. Ghete, C. Hartl, N. Hörmann, J. Hrubec, M. Jeitler, A. König, I. Krätschmer, D. Liko, T. Matsushita, I. Mikulec, D. Rabady, N. Rad, B. Rahbaran, H. Rohringer, J. Schieck, J. Strauss, W. Waltenberger, C.-E. Wulz, O. Dvornikov, V. Makarenko, V. Mossolov, J. Suarez Gonzalez, V. Zykunov, N. Shumeiko, S. Alderweireldt, E.A. De Wolf, X. Janssen, J. Lauwers, M. Van De Klundert, H. Van Haevermaet, P. Van Mechelen, N. Van Remortel, A. Van Spilbeeck, S. Abu Zeid, F. Blekman, J. D'Hondt, N. Daci, I. De Bruyn, K. Deroover, S. Lowette, S. Moortgat, L. Moreels, A. Olbrechts, Q. Python, K. Skovpen, S. Tavernier, W. Van Doninck, P. Van Mulders, I. Van Parijs, H. Brun, B. Clerbaux, G. De Lentdecker, H. Delannoy, G. Fasanella, L. Favart, R. Goldouzian, A. Grebenyuk, G. Karapostoli, T. Lenzi, A. Léonard, J. Luetic, T. Maerschalk, A. Marinov, A. Randle-conde, T. Seva, C. Vander Velde, P. Vanlaer, D. Vannerom, R. Yonamine, F. Zenoni, F. Zhang, T. Cornelis, D. Dobur, A. Fagot, M. Gul, I. Khvastunov, D. Poyraz, S. Salva, R. Schöfbeck, M. Tytgat, W. Van Driessche, E. Yazgan, N. Zaganidis, H. Bakhshiansohi, O. Bondu, S. Brochet, G. Bruno, A. Caudron, S. De Visscher, C. Delaere, M. Delcourt, B. Francois, A. Giammanco, A. Jafari, M. Komm, G. Krintiras, V. Lemaitre, A. Magitteri,

A. Mertens, M. Musich, K. Piotrzkowski, L. Quertenmont, M. Selvaggi, M. Vidal Marono, S. Wertz, N. Beliy, W.L. Aldá Júnior, F.L. Alves, G.A. Alves, L. Brito, C. Hensel, A. Moraes, M.E. Pol, P. Rebello Teles, E. Belchior Batista Das Chagas, W. Carvalho, J. Chinellato, A. Custódio, E.M. Da Costa, G.G. Da Silveira, D. De Jesus Damiao, C. De Oliveira Martins, S. Fonseca De Souza, L.M. Huertas Guativa, H. Malbouisson, D. Matos Figueiredo, C. Mora Herrera, L. Mundim, H. Nogima, W.L. Prado Da Silva, A. Santoro, A. Sznajder, E.J. Tonelli Manganote, F. Torres Da Silva De Araujo, A. Vilela Pereira, S. Ahuja, C.A. Bernardes, S. Dogra, T.R. Fernandez Perez Tomei, E.M. Gregores, P.G. Mercadante, C.S. Moon, S.F. Novaes, Sandra S. Padula, D. Romero Abad, J.C. Ruiz Vargas, A. Aleksandrov, R. Hadjiiska, P. Iaydjiev, M. Rodozov, S. Stoykova, G. Sultanov, M. Vutova, A. Dimitrov, I. Glushkov, L. Litov, B. Pavlov, P. Petkov, W. Fang, M. Ahmad, J.G. Bian, G.M. Chen, H.S. Chen, M. Chen, Y. Chen, T. Cheng, C.H. Jiang, D. Leggat, Z. Liu, F. Romeo, M. Ruan, S.M. Shaheen, A. Spiezia, J. Tao, C. Wang, Z. Wang, H. Zhang, J. Zhao, Y. Ban, G. Chen, Q. Li, S. Liu, Y. Mao, S.J. Qian, D. Wang, Z. Xu, C. Avila, A. Cabrera, L.F. Chaparro Sierra, C. Florez, J.P. Gomez, C.F. González Hernández, J.D. Ruiz Alvarez, J.C. Sanabria, N. Godinovic, D. Lelas, I. Puljak, P.M. Ribeiro Cipriano, T. Sculac, Z. Antunovic, M. Kovac, V. Brigljevic, D. Ferencek, K. Kadija, B. Mesic, T. Susa, M.W. Ather, A. Attikis, G. Mavromanolakis, J. Mousa, C. Nicolaou, F. Ptochos, P.A. Razis, H. Rykaczewski, M. Finger, M. Finger Jr., E. Carrera Jarrin, E. El-khateeb, S. Elgammal, A. Mohamed, M. Kadastik, L. Perrini, M. Raidal, A. Tiko, C. Veelken, P. Eerola, J. Pekkanen, M. Voutilainen, J. Härkönen, T. Järvinen, V. Karimäki, R. Kinnunen, T. Lampén, K. Lassila-Perini, S. Lehti, T. Lindén, P. Luukka, J. Tuominiemi, E. Tuovinen, L. Wendland, J. Talvitie, T. Tuuva, M. Besancon, F. Couderc, M. Dejardin, D. Denegri, B. Fabbro, J.L. Faure, C. Favaro, F. Ferri, S. Ganjour, S. Ghosh, A. Givernaud, P. Gras, G. Hamel de Monchenault, P. Jarry, I. Kucher, E. Locci, M. Machet, J. Malcles, J. Rander, A. Rosowsky, M. Titov, A. Abdulsalam, I. Antropov, S. Baffioni, F. Beaudette, P. Busson, L. Cadamuro, E. Chapon, C. Charlot, O. Davignon, R. Granier de Cassagnac, M. Jo, S. Lisniak, P. Miné, M. Nguyen, C. Ochando, G. Ortona, P. Paganini, P. Pigard, S. Regnard, R. Salerno, Y. Sirois, A.G. Stahl Leiton, T. Strebler, Y. Yilmaz, A. Zabi, A. Zghiche, J.-L. Agram, J. Andrea, D. Bloch, J.-M. Brom, M. Buttignol, E.C. Chabert, N. Chanon, C. Collard, E. Conte, X. Coubez, J.-C. Fontaine, D. Gelé, U. Goerlach, A.-C. Le Bihan, P. Van Hove, S. Gadrat, S. Beauceron, C. Bernet, G. Boudoul, C.A. Carrillo Montoya, R. Chierici, D. Contardo, B. Courbon, P. Depasse, H. El Mamouni, J. Fay, S. Gascon, M. Gouzevitch, G. Grenier, B. Ille, F. Lagarde, I.B. Laktineh, M. Lethuillier, L. Mirabito, A.L. Pequegnot, S. Perries, A. Popov, V. Sordini, M. Vander Donckt, P. Verdier, S. Viret, T. Toriashvili, Z. Tsamalaidze, C. Autermann, S. Beranek, L. Feld, M.K. Kiesel, K. Klein, M. Lipinski, M. Preuten, C. Schomakers, J. Schulz, T. Verlage, A. Albert, M. Brodski, E. Dietz-Laursonn, D. Duchardt, M. Endres, M. Erdmann, S. Erdweg, T. Esch, R. Fischer, A. Güth, M. Hamer, T. Hebbeker, C. Heidemann,

K. Hoepfner, S. Knutzen, M. Merschmeyer, A. Meyer, P. Millet, S. Mukherjee, M. Olschewski, K. Padeken, T. Pook, M. Radziej, H. Reithler, M. Rieger, F. Scheuch, L. Sonnenschein, D. Teyssier, S. Thüer, V. Cherepanov, G. Flügge, B. Kargoll, T. Kress, A. Künsken, J. Lingemann, T. Müller, A. Nehrkorn, A. Nowack, C. Pistone, O. Pooth, A. Stahl, M. Aldaya Martin, T. Arndt, C. Asawatangtrakuldee, K. Beernaert, O. Behnke, U. Behrens, A.A. Bin Anuar, K. Borras, A. Campbell, P. Connor, C. Contreras-Campana, F. Costanza, C. Diez Pardos, G. Dolinska, G. Eckerlin, D. Eckstein, T. Eichhorn, E. Eren, E. Gallo, J. Garay Garcia, A. Geiser, A. Gizhko, J.M. Grados Luyando, A. Grohsjean, P. Gunnellini, A. Harb, J. Hauk, M. Hempel, H. Jung, A. Kalogeropoulos, O. Karacheban, M. Kasemann, J. Keaveney, C. Kleinwort, I. Korol, D. Krücker, W. Lange, A. Lelek, T. Lenz, J. Leonard, K. Lipka, A. Lobanov, W. Lohmann, R. Mankel, I.-A. Melzer-Pellmann, A.B. Meyer, G. Mittag, J. Mnich, A. Mussgiller, D. Pitzl, R. Placakyte, A. Raspereza, B. Roland, M.Ö. Sahin, P. Saxena, T. Schoerner-Sadenius, S. Spannagel, N. Stefaniuk, G.P. Van Onsem, R. Walsh, C. Wissing, V. Blobel, M. Centis Vignali, A.R. Draeger, T. Dreyer, E. Garutti, D. Gonzalez, J. Haller, M. Hoffmann, A. Junkes, R. Klanner, R. Kogler, N. Kovalchuk, S. Kurz, T. Lapsien, I. Marchesini, D. Marconi, M. Meyer, M. Niedziela, D. Nowatschin, F. Pantaleo, T. Peiffer, A. Perieanu, C. Scharf, P. Schleper, A. Schmidt, S. Schumann, J. Schwandt, J. Sonneveld, H. Stadie, G. Steinbrück, F.M. Stober, M. Stöver, H. Tholen, D. Troendle, E. Usai, L. Vanelderen, A. Vanhoefer, B. Vormwald, M. Akbiyik, C. Barth, S. Baur, C. Baus, J. Berger, E. Butz, R. Caspart, T. Chwalek, F. Colombo, W. De Boer, A. Dierlamm, S. Fink, B. Freund, R. Friese, M. Giffels, A. Gilbert, P. Goldenzweig, D. Haitz, F. Hartmann, S.M. Heindl, U. Husemann, F. Kassel, I. Katkov, S. Kudella, H. Mildner, M.U. Mozer, Th. Müller, M. Plagge, G. Quast, K. Rabbertz, S. Röcker, F. Roscher, M. Schröder, I. Shvetsov, G. Sieber, H.J. Simonis, R. Ulrich, S. Wayand, M. Weber, T. Weiler, S. Williamson, C. Wöhrmann, R. Wolf, G. Anagnostou, G. Daskalakis, T. Geralis, V.A. Giakoumopoulou, A. Kyriakis, D. Loukas, I. Topsis-Giotis, S. Kesisoglou, A. Panagiotou, N. Saoulidou, E. Tziaferi, I. Evangelou, G. Flouris, C. Foudas, P. Kokkas, N. Loukas, N. Manthos, I. Papadopoulos, E. Paradas, N. Filipovic, G. Pasztor, G. Bencze, C. Hajdu, D. Horvath, F. Sikler, V. Veszpremi, G. Vesztergombi, A.J. Zsigmond, N. Beni, S. Czellar, J. Karancsi, A. Makovec, J. Molnar, Z. Szillasi, M. Bartók, P. Raics, Z.L. Trocsanyi, B. Ujvari, S. Choudhury, J.R. Komaragiri, S. Bahinipati, S. Bhowmik, P. Mal, K. Mandal, A. Nayak, D.K. Sahoo, N. Sahoo, S.K. Swain, S. Bansal, S.B. Beri, V. Bhatnagar, U. Bhawandeep, R. Chawla, A.K. Kalsi, A. Kaur, M. Kaur, R. Kumar, P. Kumari, A. Mehta, M. Mittal, J.B. Singh, G. Walia, Ashok Kumar, A. Bhardwaj, B.C. Choudhary, R.B. Garg, S. Keshri, A. Kumar, S. Malhotra, M. Naimuddin, K. Ranjan, R. Sharma, V. Sharma, R. Bhattacharya, S. Bhattacharya, K. Chatterjee, S. Dey, S. Dutt, S. Dutta, S. Ghosh, N. Majumdar, A. Modak, K. Mondal, S. Mukhopadhyay, S. Nandan, A. Purohit, A. Roy, D. Roy, S. Roy Chowdhury, S. Sarkar, M. Sharan, S. Thakur, P.K. Behera,

R. Chudasama, D. Dutta, V. Jha, V. Kumar, A.K. Mohanty, P.K. Netrakanti, L.M. Pant, P. Shukla, A. Topkar, T. Aziz, S. Dugad, G. Kole, B. Mahakud, S. Mitra, G.B. Mohanty, B. Parida, N. Sur, B. Sutar, S. Banerjee, R.K. Dewanjee, S. Ganguly, M. Guchait, Sa. Jain, S. Kumar, M. Maity, G. Majumder, K. Mazumdar, T. Sarkar, N. Wickramage, S. Chauhan, S. Dube, V. Hegde, A. Kapoor, K. Kothekar, S. Pandey, A. Rane, S. Sharma, S. Chenarani, E. Eskandari Tadavani, S.M. Etesami, M. Khakzad, M. Mohammadi Najafabadi, M. Naseri, S. Paktinat Mehdiabadi, F. Rezaei Hosseinabadi, B. Safarzadeh, M. Zeinali, M. Felcini, M. Grunewald, M. Abbrescia, C. Calabria, C. Caputo, A. Colaleo, D. Creanza, L. Cristella, N. De Filippis, M. De Palma, L. Fiore, G. Iaselli, G. Maggi, M. Maggi, G. Miniello, S. My, S. Nuzzo, A. Pompili, G. Pugliese, R. Radogna, A. Ranieri, G. Selvaggi, A. Sharma, L. Silvestris, R. Venditti, P. Verwilligen, G. Abbiendi, C. Battilana, D. Bonacorsi, S. Braibant-Giacomelli, L. Brigliadori, R. Campanini, P. Capiluppi, A. Castro, F.R. Cavallo, S.S. Chhibra, G. Codispoti, M. Cuffiani, G.M. Dallavalle, F. Fabbri, A. Fanfani, D. Fasanella, P. Giacomelli, C. Grandi, L. Guiducci, S. Marcellini, G. Masetti, A. Montanari, F.L. Navarria, A. Perrotta, A.M. Rossi, T. Rovelli, G.P. Siroli, N. Tosi, S. Albergo, S. Costa, A. Di Mattia, F. Giordano, R. Potenza, A. Tricomi, C. Tuve, G. Barbagli, V. Ciulli, C. Civinini, R. D'Alessandro, E. Focardi, P. Lenzi, M. Meschini, S. Paoletti, L. Russo, G. Sguazzoni, D. Strom, L. Viliani, L. Benussi, S. Bianco, F. Fabbri, D. Piccolo, F. Primavera, V. Calvelli, F. Ferro, M.R. Monge, E. Robutti, S. Tosi, L. Brianza, F. Brivio, V. Ciriolo, M.E. Dinardo, S. Fiorendi, S. Gennai, A. Ghezzi, P. Govoni, M. Malberti, S. Malvezzi, R.A. Manzoni, D. Menasce, L. Moroni, M. Paganoni, D. Pedrini, S. Pigazzini, S. Ragazzi, T. Tabarelli de Fatis, S. Buontempo, N. Cavallo, G. De Nardo, S. Di Guida, M. Esposito, F. Fabozzi, F. Fienga, A.O.M. Iorio, G. Lanza, L. Lista, S. Meola, P. Paolucci, C. Sciacca, F. Thyssen, P. Azzi, N. Bacchetta, L. Benato, D. Bisello, A. Boletti, R. Carlin, A. Carvalho Antunes De Oliveira, P. Checchia, M. Dall'Osso, P. De Castro Manzano, T. Dorigo, A. Gozzelino, S. Lacaprara, M. Margoni, A.T. Meneguzzo, M. Passaseo, J. Pazzini, N. Pozzobon, P. Ronchese, R. Rossin, F. Simonetto, E. Torassa, S. Ventura, M. Zanetti, P. Zotto, G. Zumerle, A. Braghieri, F. Fallavollita, A. Magnani, P. Montagna, S.P. Ratti, V. Re, M. Ressegotti, C. Riccardi, P. Salvini, I. Vai, P. Vitulo, L. Alunni Solestizi, G.M. Bilei, D. Ciangottini, L. Fanò, P. Lariccia, R. Leonardi, G. Mantovani, V. Mariani, M. Menichelli, A. Saha, A. Santocchia, K. Androsov, P. Azzurri, G. Bagliesi, J. Bernardini, T. Boccali, R. Castaldi, M.A. Ciocci, R. Dell'Orso, G. Fedi, A. Giassi, M.T. Grippo, F. Ligabue, T. Lomtadze, L. Martini, A. Messineo, F. Palla, A. Rizzi, A. Savoy-Navarro, P. Spagnolo, R. Tenchini, G. Tonelli, A. Venturi, P.G. Verdini, L. Barone, F. Cavallari, M. Cipriani, D. Del Re, M. Diemoz, S. Gelli, E. Longo, F. Margaroli, B. Marzocchi, P. Meridiani, G. Organtini, R. Paramatti, F. Preiato, S. Rahatlou, C. Rovelli, F. Santanastasio, N. Amapane, R. Arcidiacono, S. Argiro, M. Arneodo, N. Bartosik, R. Bellan, C. Biino, N. Cartiglia, F. Cenna, M. Costa, R. Covarelli, A. Degano, N. Demaria, L. Finco,

B. Kiani, C. Mariotti, S. Maselli, E. Migliore, V. Monaco, E. Monteil, M. Monteno, M.M. Obertino, L. Pacher, N. Pastrone, M. Pelliccioni, G.L. Pinna Angioni, F. Ravera, A. Romero, M. Ruspa, R. Sacchi, K. Shchelina, V. Sola, A. Solano, A. Staiano, P. Traczyk, S. Belforte, M. Casarsa, F. Cossutti, G. Della Ricca, A. Zanetti, D.H. Kim, G.N. Kim, M.S. Kim, S. Lee, S.W. Lee, Y.D. Oh, S. Sekmen, D.C. Son, Y.C. Yang, A. Lee, H. Kim, J.A. Brochero Cifuentes, T.J. Kim, S. Cho, S. Choi, Y. Go, D. Gyun, S. Ha, B. Hong, Y. Jo, Y. Kim, K. Lee, K.S. Lee, S. Lee, J. Lim, S.K. Park, Y. Roh, J. Almond, J. Kim, H. Lee, S.B. Oh, B.C. Radburn-Smith, S.h. Seo, U.K. Yang, H.D. Yoo, G.B. Yu, M. Choi, H. Kim, J.H. Kim, J.S.H. Lee, I.C. Park, G. Ryu, M.S. Ryu, Y. Choi, J. Goh, C. Hwang, J. Lee, I. Yu, V. Dudenas, A. Juodagalvis, J. Vaitkus, I. Ahmed, Z.A. Ibrahim, M.A.B. Md Ali, F. Mohamad Idris, W.A.T. Wan Abdullah, M.N. Yusli, Z. Zolkapli, H. Castilla-Valdez, E. De La Cruz-Burelo, I. Heredia-De La Cruz, A. Hernandez-Almada, R. Lopez-Fernandez, R. Magaña Villalba, J. Mejia Guisao, A. Sanchez-Hernandez, S. Carrillo Moreno, C. Oropeza Barrera, F. Vazquez Valencia, S. Carpinteyro, I. Pedraza, H.A. Salazar Ibarguen, C. Uribe Estrada, A. Morelos Pineda, D. Krofcheck, P.H. Butler, A. Ahmad, M. Ahmad, Q. Hassan, H.R. Hoorani, W.A. Khan, A. Saddique, M.A. Shah, M. Shoaib, M. Waqas, H. Bialkowska, M. Bluj, B. Boimska, T. Frueboes, M. Górski, M. Kazana, K. Nawrocki, K. Romanowska-Rybinska, M. Szleper, P. Zalewski, K. Bunkowski, A. Byszuk, K. Doroba, A. Kalinowski, M. Konecki, J. Krolikowski, M. Misiura, M. Olszewski, M. Walczak, P. Bargassa, C. Beirão Da Cruz E Silva, B. Calpas, A. Di Francesco, P. Faccioli, M. Gallinaro, J. Hollar, N. Leonardo, L. Lloret Iglesias, M.V. Nemallapudi, J. Seixas, O. Toldaiev, D. Vadruccio, J. Varela, S. Afanasiev, P. Bunin, M. Gavrilenko, I. Golutvin, I. Gorbunov, A. Kamenev, V. Karjavin, A. Lanev, A. Malakhov, V. Matveev, V. Palichik, V. Perelygin, S. Shmatov, S. Shulha, N. Skatchkov, V. Smirnov, N. Voytishin, A. Zarubin, L. Chtchipounov, V. Golovtsov, Y. Ivanov, V. Kim, E. Kuznetsova, V. Murzin, V. Oreshkin, V. Sulimov, A. Vorobyev, Yu. Andreev, A. Dermenev, S. Gninenko, N. Golubev, A. Karneyeu, M. Kirsanov, N. Krasnikov, A. Pashenkov, D. Tlisov, A. Toropin, V. Epshteyn, V. Gavrilov, N. Lychkovskaya, V. Popov, I. Pozdnyakov, G. Safronov, A. Spiridonov, M. Toms, E. Vlasov, A. Zhokin, T. Aushev, A. Bylinkin, M. Chadeeva, O. Markin, E. Tarkovskii, V. Andreev, M. Azarkin, I. Dremin, M. Kirakosyan, A. Leonidov, A. Terkulov, A. Baskakov, A. Belyaev, E. Boos, M. Dubinin, L. Dudko, A. Ershov, A. Gribushin, A. Kaminskiy, V. Klyukhin, O. Kodolova, I. Lokhtin, I. Miagkov, S. Obraztsov, S. Petrushanko, V. Savrin, V. Blinov, Y. Skovpen, D. Shtol, I. Azhgirey, I. Bayshev, S. Bitioukov, D. Elumakhov, V. Kachanov, A. Kalinin, D. Konstantinov, V. Krychkine, V. Petrov, R. Ryutin, A. Sobol, S. Troshin, N. Tyurin, A. Uzunian, A. Volkov, P. Adzic, P. Cirkovic, D. Devetak, M. Dordevic, J. Milosevic, V. Rekovic, J. Alcaraz Maestre, M. Barrio Luna, E. Calvo, M. Cerrada, M. Chamizo Llatas, N. Colino, B. De La Cruz, A. Delgado Peris, A. Escalante Del Valle, C. Fernandez Bedoya, J.P. Fernández Ramos, J. Flix, M.C. Fouz, P. Garcia-Abia, O. Gonzalez

Lopez, S. Goy Lopez, J.M. Hernandez, M.I. Josa, E. Navarro De Martino, A. Pérez-Calero Yzquierdo, J. Puerta Pelayo, A. Quintario Olmeda, I. Redondo, L. Romero, M.S. Soares, J.F. de Trocóniz, M. Missiroli, D. Moran, J. Cuevas, C. Erice, J. Fernandez Menendez, I. Gonzalez Caballero, J.R. González Fernández, E. Palencia Cortezon, S. Sanchez Cruz, I. Suárez Andrés, P. Vischia, J.M. Vizan Garcia, I.J. Cabrillo, A. Calderon, E. Curras, M. Fernandez, J. Garcia-Ferrero, G. Gomez, A. Lopez Virto, J. Marco, C. Martinez Rivero, F. Matorras, J. Piedra Gomez, T. Rodrigo, A. Ruiz-Jimeno, L. Scodellaro, N. Trevisani, I. Vila, R. Vilar Cortabitarte, D. Abbaneo, E. Auffray, G. Auzinger, P. Baillon, A.H. Ball, D. Barney, P. Bloch, A. Bocci, C. Botta, T. Camporesi, R. Castello, M. Cepeda, G. Cerminara, Y. Chen, A. Cimmino, D. d'Enterria, A. Dabrowski, V. Daponte, A. David, M. De Gruttola, A. De Roeck, E. Di Marco, M. Dobson, B. Dorney, T. du Pree, D. Duggan, M. Dünser, N. Dupont, A. Elliott-Peisert, P. Everaerts, S. Fartoukh, G. Franzoni, J. Fulcher, W. Funk, D. Gigi, K. Gill, M. Girone, F. Glege, D. Gulhan, S. Gundacker, M. Guthoff, P. Harris, J. Hegeman, V. Innocente, P. Janot, J. Kieseler, H. Kirschenmann, V. Knünz, A. Kornmayer, M.J. Kortelainen, K. Kousouris, M. Krammer, C. Lange, P. Lecoq, C. Lourenço, M.T. Lucchini, L. Malgeri, M. Mannelli, A. Martelli, F. Meijers, J.A. Merlin, S. Mersi, E. Meschi, P. Milenovic, F. Moortgat, S. Morovic, M. Mulders, H. Neugebauer, S. Orfanelli, L. Orsini, L. Pape, E. Perez, M. Peruzzi, A. Petrilli, G. Petrucciani, A. Pfeiffer, M. Pierini, A. Racz, T. Reis, G. Rolandi, M. Rovere, H. Sakulin, J.B. Sauvan, C. Schäfer, C. Schwick, M. Seidel, A. Sharma, P. Silva, P. Sphicas, J. Steggemann, M. Stoye, Y. Takahashi, M. Tosi, D. Treille, A. Triossi, A. Tsirou, V. Veckalns, G.I. Veres, M. Verweij, N. Wardle, H.K. Wöhri, A. Zagozdzinska, W.D. Zeuner, W. Bertl, K. Deiters, W. Erdmann, R. Horisberger, Q. Ingram, H.C. Kaestli, D. Kotlinski, U. Langenegger, T. Rohe, S.A. Wiederkehr, F. Bachmair, L. Bäni, L. Bianchini, B. Casal, G. Dissertori, M. Dittmar, M. Donegà, C. Grab, C. Heidegger, D. Hits, J. Hoss, G. Kasieczka, W. Lustermann, B. Mangano, M. Marionneau, P. Martinez Ruiz del Arbol, M. Masciovecchio, M.T. Meinhard, D. Meister, F. Micheli, P. Musella, F. Nessi-Tedaldi, F. Pandolfi, J. Pata, F. Pauss, G. Perrin, L. Perrozzi, M. Quittnat, M. Rossini, M. Schönenberger, A. Starodumov, V.R. Tavolaro, K. Theofilatos, R. Wallny, T.K. Aarrestad, C. Amsler, L. Caminada, M.F. Canelli, A. De Cosa, S. Donato, C. Galloni, A. Hinzmann, T. Hreus, B. Kilminster, J. Ngadiuba, D. Pinna, G. Rauco, P. Robmann, D. Salerno, C. Seitz, Y. Yang, A. Zucchetta, V. Candelise, T.H. Doan, Sh. Jain, R. Khurana, M. Konyushikhin, C.M. Kuo, W. Lin, A. Pozdnyakov, S.S. Yu, Arun Kumar, P. Chang, Y.H. Chang, Y. Chao, K.F. Chen, P.H. Chen, F. Fiori, W.-S. Hou, Y. Hsiung, Y.F. Liu, R.-S. Lu, M. Miñano Moya, E. Paganis, A. Psallidas, J.f. Tsai, B. Asavapibhop, G. Singh, N. Srimanobhas, N. Suwonjandee, A. Adiguzel, S. Damarseckin, Z.S. Demiroglu, C. Dozen, E. Eskut, S. Girgis, G. Gokbulut, Y. Guler, I. Hos, E.E. Kangal, O. Kara, A. Kayis Topaksu, U. Kiminsu, M. Oglakci, G. Onengut, K. Ozdemir, S. Ozturk, A. Polatoz, B. Tali, S. Turkcapar, I.S. Zorbakir,

C. Zorbilmez, B. Bilin, S. Bilmis, B. Isildak, G. Karapinar, M. Yalvac, M. Zeyrek, E. Gülmez, M. Kaya, O. Kaya, E.A. Yetkin, T. Yetkin, A. Cakir, K. Cankocak, S. Sen, B. Grynyov, L. Levchuk, P. Sorokin, R. Aggleton, F. Ball, L. Beck, J.J. Brooke, D. Burns, E. Clement, D. Cussans, H. Flacher, J. Goldstein, M. Grimes, G.P. Heath, H.F. Heath, J. Jacob, L. Kreczko, C. Lucas, D.M. Newbold, S. Paramesvaran, A. Poll, T. Sakuma, S. Seif El Nasr-storey, D. Smith, V.J. Smith, K.W. Bell, A. Belyaev, C. Brew, R.M. Brown, L. Calligaris, D. Cieri, D.J.A. Cockerill, J.A. Coughlan, K. Harder, S. Harper, E. Olaiya, D. Petyt, C.H. Shepherd-Themistocleous, A. Thea, I.R. Tomalin, T. Williams, M. Baber, R. Bainbridge, O. Buchmuller, A. Bundock, S. Casasso, M. Citron, D. Colling, L. Corpe, P. Dauncey, G. Davies, A. De Wit, M. Della Negra, R. Di Maria, P. Dunne, A. Elwood, D. Futyan, Y. Haddad, G. Hall, G. Iles, T. James, R. Lane, C. Laner, L. Lyons, A.-M. Magnan, S. Malik, L. Mastrolorenzo, J. Nash, A. Nikitenko, J. Pela, B. Penning, M. Pesaresi, D.M. Raymond, A. Richards, A. Rose, E. Scott, C. Seez, S. Summers, A. Tapper, K. Uchida, M. Vazquez Acosta, T. Virdee, J. Wright, S.C. Zenz, J.E. Cole, P.R. Hobson, A. Khan, P. Kyberd, I.D. Reid, P. Symonds, L. Teodorescu, M. Turner, A. Borzou, K. Call, J. Dittmann, K. Hatakeyama, H. Liu, N. Pastika, R. Bartek, A. Dominguez, A. Buccilli, S.I. Cooper, C. Henderson, P. Rumerio, C. West, D. Arcaro, A. Avetisyan, T. Bose, D. Gastler, D. Rankin, C. Richardson, J. Rohlf, L. Sulak, D. Zou, G. Benelli, D. Cutts, A. Garabedian, J. Hakala, U. Heintz, J.M. Hogan, O. Jesus, K.H.M. Kwok, E. Laird, G. Landsberg, Z. Mao, M. Narain, S. Piperov, S. Sagir, E. Spencer, R. Syarif, R. Breedon, D. Burns, M. Calderon De La Barca Sanchez, S. Chauhan, M. Chertok, J. Conway, R. Conway, P.T. Cox, R. Erbacher, C. Flores, G. Funk, M. Gardner, W. Ko, R. Lander, C. Mclean, M. Mulhearn, D. Pellett, J. Pilot, S. Shalhout, M. Shi, J. Smith, M. Squires, D. Stolp, K. Tos, M. Tripathi, M. Bachtis, C. Bravo, R. Cousins, A. Dasgupta, A. Florent, J. Hauser, M. Ignatenko, N. Mccoll, D. Saltzberg, C. Schnaible, V. Valuev, M. Weber, E. Bouvier, K. Burt, R. Clare, J. Ellison, J.W. Gary, S.M.A. Ghiasi Shirazi, G. Hanson, J. Heilman, P. Jandir, E. Kennedy, F. Lacroix, O.R. Long, M. Olmedo Negrete, M.I. Paneva, A. Shrinivas, W. Si, H. Wei, S. Wimpenny, B. R. Yates, J.G. Branson, G.B. Cerati, S. Cittolin, M. Derdzinski, R. Gerosa, A. Holzner, D. Klein, V. Krutelyov, J. Letts, I. Macneill, D. Olivito, S. Padhi, M. Pieri, M. Sani, V. Sharma, S. Simon, M. Tadel, A. Vartak, S. Wasserbaech, C. Welke, J. Wood, F. Würthwein, A. Yagil, G. Zevi Della Porta, N. Amin, R. Bhandari, J. Bradmiller-Feld, C. Campagnari, A. Dishaw, V. Dutta, M. Franco Sevilla, C. George, F. Golf, L. Gouskos, J. Gran, R. Heller, J. Incandela, S.D. Mullin, A. Ovcharova, H. Qu, J. Richman, D. Stuart, I. Suarez, J. Yoo, D. Anderson, J. Bendavid, A. Bornheim, J. Bunn, J. Duarte, J.M. Lawhorn, A. Mott, H.B. Newman, C. Pena, M. Spiropulu, J.R. Vlimant, S. Xie, R.Y. Zhu, M.B. Andrews, T. Ferguson, M. Paulini, J. Russ, M. Sun, H. Vogel, I. Vorobiev, M. Weinberg, J.P. Cumalat, W.T. Ford, F. Jensen, A. Johnson, M. Krohn, S. Leontsinis, T. Mulholland, K. Stenson, S.R. Wagner, J. Alexander, J. Chaves, J. Chu, S. Dittmer, K. Mcdermott, N. Mirman, J.R.

Patterson, A. Rinkevicius, A. Ryd, L. Skinnari, L. Soffi, S.M. Tan, Z. Tao, J. Thom, J. Tucker, P. Wittich, M. Zientek, D. Winn, S. Abdullin, M. Albrow, G. Apollinari, A. Apresyan, S. Banerjee, L.A.T. Bauerdick, A. Beretvas, J. Berryhill, P.C. Bhat, G. Bolla, K. Burkett, J.N. Butler, H.W.K. Cheung, F. Chlebana, S. Cihangir, M. Cremonesi, V.D. Elvira, I. Fisk, J. Freeman, E. Gottschalk, L. Gray, D. Green, S. Grünendahl, O. Gutsche, D. Hare, R.M. Harris, S. Hasegawa, J. Hirschauer, Z. Hu, B. Jayatilaka, S. Jindariani, M. Johnson, U. Joshi, B. Klima, B. Kreis, S. Lammel, J. Linacre, D. Lincoln, R. Lipton, M. Liu, T. Liu, R. Lopes De Sá, J. Lykken, K. Maeshima, N. Magini, J.M. Marraffino, S. Maruyama, D. Mason, P. McBride, P. Merkel, S. Mrenna, S. Nahn, V. O'Dell, K. Pedro, O. Prokofyev, G. Rakness, L. Ristori, E. Sexton-Kennedy, A. Soha, W.J. Spalding, L. Spiegel, S. Stoynev, J. Strait, N. Strobbe, L. Taylor, S. Tkaczyk, N.V. Tran, L. Uplegger, E.W. Vaandering, C. Vernieri, M. Verzocchi, R. Vidal, M. Wang, H.A. Weber, A. Whitbeck, Y. Wu, D. Acosta, P. Avery, P. Bortignon, D. Bourilkov, A. Brinkerhoff, A. Carnes, M. Carver, D. Curry, S. Das, R.D. Field, I.K. Furic, J. Konigsberg, A. Korytov, J.F. Low, P. Ma, K. Matchev, H. Mei, G. Mitselmakher, D. Rank, L. Shchutska, D. Sperka, L. Thomas, J. Wang, S. Wang, J. Yelton, S. Linn, P. Markowitz, G. Martinez, J.L. Rodriguez, A. Ackert, T. Adams, A. Askew, S. Bein, S. Hagopian, V. Hagopian, K.F. Johnson, T. Kolberg, T. Perry, H. Prosper, A. Santra, R. Yohay, M.M. Baarmand, V. Bhopatkar, S. Colafranceschi, M. Hohlmann, D. Noonan, T. Roy, F. Yumiceva, M.R. Adams, L. Apanasevich, D. Berry, R.R. Betts, R. Cavanaugh, X. Chen, O. Evdokimov, C.E. Gerber, D.A. Hangal, D.J. Hofman, K. Jung, J. Kamin, I.D. Sandoval Gonzalez, H. Trauger, N. Varelas, H. Wang, Z. Wu, M. Zakaria, J. Zhang, B. Bilki, W. Clarida, K. Dilsiz, S. Durgut, R.P. Gandrajula, M. Haytmyradov, V. Khristenko, J.-P. Merlo, H. Mermerkaya, A. Mestvirishvili, A. Moeller, J. Nachtman, H. Ogul, Y. Onel, F. Ozok, A. Penzo, C. Snyder, E. Tiras, J. Wetzel, K. Yi, B. Blumenfeld, A. Cocoros, N. Eminizer, D. Fehling, L. Feng, A.V. Gritsan, P. Maksimovic, J. Roskes, U. Sarica, M. Swartz, M. Xiao, C. You, A. Al-bataineh, P. Baringer, A. Bean, S. Boren, J. Bowen, J. Castle, L. Forthomme, S. Khalil, A. Kropivnitskaya, D. Majumder, W. Mcbrayer, M. Murray, S. Sanders, R. Stringer, J.D. Tapia Takaki, Q. Wang, A. Ivanov, K. Kaadze, Y. Maravin, A. Mohammadi, L.K. Saini, N. Skhirtladze, S. Toda, F. Rebassoo, D. Wright, C. Anelli, A. Baden, O. Baron, A. Belloni, B. Calvert, S.C. Eno, C. Ferraioli, J.A. Gomez, N.J. Hadley, S. Jabeen, G.Y. Jeng, R.G. Kellogg, J. Kunkle, A.C. Mignerey, F. Ricci-Tam, Y.H. Shin, A. Skuja, M.B. Tonjes, S.C. Tonwar, D. Abercrombie, B. Allen, A. Apyan, V. Azzolini, R. Barbieri, A. Baty, R. Bi, K. Bierwagen, S. Brandt, W. Busza, I.A. Cali, M. D'Alfonso, Z. Demiragli, G. Gomez Ceballos, M. Goncharov, D. Hsu, Y. Iiyama, G.M. Innocenti, M. Klute, D. Kovalskyi, K. Krajczar, Y.S. Lai, Y.-J. Lee, A. Levin, P.D. Luckey, B. Maier, A.C. Marini, C. Mcginn, C. Mironov, S. Narayanan, X. Niu, C. Paus, C. Roland, G. Roland, J. Salfeld-Nebgen, G.S.F. Stephans, K. Tatar, D. Velicanu, J. Wang, T.W. Wang, B. Wyslouch, A.C. Benvenuti, R.M. Chatterjee, A. Evans, P. Hansen, S. Kalafut, S.C.

Kao, Y. Kubota, Z. Lesko, J. Mans, S. Nourbakhsh, N. Ruckstuhl, R. Rusack, N. Tambe, J. Turkewitz, J.G. Acosta, S. Oliveros, E. Avdeeva, K. Bloom, D.R. Claes, C. Fangmeier, R. Gonzalez Suarez, R. Kamalieddin, I. Kravchenko, A. Malta Rodrigues, J. Monroy, J.E. Siado, G.R. Snow, B. Stieger, M. Alyari, J. Dolen, A. Godshalk, C. Harrington, I. Iashvili, J. Kaisen, D. Nguyen, A. Parker, S. Rappoccio, B. Roozbahani, G. Alverson, E. Barberis, A. Hortiangtham, A. Massironi, D.M. Morse, D. Nash, T. Orimoto, R. Teixeira De Lima, D. Trocino, R.-J. Wang, D. Wood, S. Bhattacharya, O. Charaf, K.A. Hahn, N. Mucia, N. Odell, B. Pollack, M.H. Schmitt, K. Sung, M. Trovato, M. Velasco, N. Dev, M. Hildreth, K. Hurtado Anampa, C. Jessop, D.J. Karmgard, N. Kellams, K. Lannon, N. Marinelli, F. Meng, C. Mueller, Y. Musienko, M. Planer, A. Reinsvold, R. Ruchti, N. Rupprecht, G. Smith, S. Taroni, M. Wayne, M. Wolf, A. Woodard, J. Alimena, L. Antonelli, B. Bylsma, L.S. Durkin, S. Flowers, B. Francis, A. Hart, C. Hill, W. Ji, B. Liu, W. Luo, D. Puigh, B.L. Winer, H.W. Wulsin, S. Cooperstein, O. Driga, P. Elmer, J. Hardenbrook, P. Hebda, D. Lange, J. Luo, D. Marlow, T. Medvedeva, K. Mei, I. Ojalvo, J. Olsen, C. Palmer, P. Piroué, D. Stickland, A. Svyatkovskiy, C. Tully, S. Malik, A. Barker, V.E. Barnes, S. Folgueras, L. Gutay, M.K. Jha, M. Jones, A.W. Jung, A. Khatiwada, D.H. Miller, N. Neumeister, J.F. Schulte, X. Shi, J. Sun, F. Wang, W. Xie, N. Parashar, J. Stupak, A. Adair, B. Akgun, Z. Chen, K.M. Ecklund, F.J.M. Geurts, M. Guilbaud, W. Li, B. Michlin, M. Northup, B.P. Padley, J. Roberts, J. Rorie, Z. Tu, J. Zabel, B. Betchart, A. Bodek, P. de Barbaro, R. Demina, Y.t. Duh, T. Ferbel, M. Galanti, A. Garcia-Bellido, J. Han, O. Hindrichs, A. Khukhunaishvili, K.H. Lo, P. Tan, M. Verzetti, A. Agapitos, J.P. Chou, Y. Gershtein, T.A. Gómez Espinosa, E. Halkiadakis, M. Heindl, E. Hughes, S. Kaplan, R. Kunnawalkam Elayavalli, S. Kyriacou, A. Lath, R. Montalvo, K. Nash, M. Osherson, H. Saka, S. Salur, S. Schnetzer, D. Sheffield, S. Somalwar, R. Stone, S. Thomas, P. Thomassen, M. Walker, A.G. Delannoy, M. Foerster, J. Heideman, G. Riley, K. Rose, S. Spanier, K. Thapa, O. Bouhali, A. Celik, M. Dalchenko, M. De Mattia, A. Delgado, S. Dildick, R. Eusebi, J. Gilmore, T. Huang, E. Juska, T. Kamon, R. Mueller, Y. Pakhotin, R. Patel, A. Perloff, L. Perniè, D. Rathjens, A. Safonov, A. Tatarinov, K.A. Ulmer, N. Akchurin, J. Damgov, F. De Guio, C. Dragoiu, P.R. Dudero, J. Faulkner, E. Gurpinar, S. Kunori, K. Lamichhane, S.W. Lee, T. Libeiro, T. Peltola, S. Undleeb, I. Volobouev, Z. Wang, S. Greene, A. Gurrola, R. Janjam, W. Johns, C. Maguire, A. Melo, H. Ni, P. Sheldon, S. Tuo, J. Velkovska, Q. Xu, M.W. Arenton, P. Barria, B. Cox, R. Hirosky, A. Ledovskoy, H. Li, C. Neu, T. Sinthuprasith, X. Sun, Y. Wang, E. Wolfe, F. Xia, C. Clarke, R. Harr, P.E. Karchin, J. Sturdy, S. Zaleski, D.A. Belknap, J. Buchanan, C. Caillol, S. Dasu, L. Dodd, S. Duric, B. Gomber, M. Grothe, M. Herndon, A. Hervé, U. Hussain, P. Klabbers, A. Lanaro, A. Levine, K. Long, R. Loveless, G.A. Pierro, G. Polese, T. Ruggles, A. Savin, N. Smith, W.H. Smith, D. Taylor, and N. Woods. Particle-flow reconstruction and global event description with the cms detector. *Journal of Instrumentation*, 12(10):P10003, oct 2017. . URL

https://dx.doi.org/10.1088/1748-0221/12/10/P10003.

Torbjörn Sjöstrand, Stefan Ask, Jesper R. Christiansen, Richard Corke, Nishita Desai, Philip Ilten, Stephen Mrenna, Stefan Prestel, Christine O. Rasmussen, and Peter Z. Skands. An introduction to pythia 8.2. *Computer Physics Communications*, 191: 159–177, June 2015. ISSN 0010-4655. . URL http://dx.doi.org/10.1016/j.cpc.2015.01.024.

D. Skobelzyn. Die Intensitätsverteilung in dem Spektrum der γ-Strahlen von Ra C. *Zeitschrift fur Physik*, 43(5-6):354–378, May 1927. .

George Sterman and Steven Weinberg. Jets from quantum chromodynamics. *Phys. Rev. Lett.*, 39:1436–1439, Dec 1977. . URL https://link.aps.org/doi/10.1103/PhysRevLett.39.1436.

Savannah Thais, Paolo Calafiura, Grigorios Chachamis, Gage DeZoort, Javier Duarte, Sanmay Ganguly, Michael Kagan, Daniel Murnane, Mark S. Neubauer, and Kazuhiro Terao. Graph neural networks in particle physics: Implementations, innovations, and challenges, 2022. URL https://arxiv.org/abs/2203.12852.

The CMS Collaboration, S Chatrchyan, G Hmayakyan, V Khachatryan, A M Sirunyan, W Adam, T Bauer, T Bergauer, H Bergauer, M Dragicevic, J Erö, M Friedl, R Frühwirth, V M Ghete, P Glaser, C Hartl, N Hoermann, J Hrubec, S Hänsel, M Jeitler, K Kastner, M Krammer, I Magrans de Abril, M Markytan, I Mikulec, B Neuherz, T Nöbauer, M Oberegger, M Padrta, M Pernicka, P Porth, H Rohringer, S Schmid, T Schreiner, R Stark, H Steininger, J Strauss, A Taurok, D Uhl, W Waltenberger, G Walzel, E Widl, C-E Wulz, V Petrov, V Prosolovich, V Chekhovsky, O Dvornikov, I Emeliantchik, A Litomin, V Makarenko, I Marfin, V Mossolov, N Shumeiko, A Solin, R Stefanovitch, J Suarez Gonzalez, A Tikhonov, A Fedorov, M Korzhik, O Missevitch, R Zuyeuski, W Beaumont, M Cardaci, E De Langhe, E A De Wolf, E Delmeire, S Ochesanu, M Tasevsky, P Van Mechelen, J D'Hondt, S De Weirdt, O Devroede, R Goorens, S Hannaert, J Heyninck, J Maes, M U Mozer, S Tavernier, W Van Doninck, L Van Lancker, P Van Mulders, I Villella, C Wastiels, C Yu, O Bouhali, O Charaf, B Clerbaux, P De Harenne, G De Lentdecker, J P Dewulf, S Elgammal, R Gindroz, G H Hammad, T Mahmoud, L Neukermans, M Pins, R Pins, S Rugovac, J Stefanescu, V Sundararajan, C Vander Velde, P Vanlaer, J Wickens, M Tytgat, S Assouak, J L Bonnet, G Bruno, J Caudron, B De Callatay, J De Favereau De Jeneret, S De Visscher, P Demin, D Favart, C Felix, B Florins, E Forton, A Giammanco, G Grégoire, M Jonckman, D Kcira, T Keutgen, V Lemaitre, D Michotte, O Militaru, S Ovyn, T Pierzchala, K Piotrzkowski, V Roberfroid, X Rouby, N Schul, O Van der Aa, N Beliy, E Daubie, P Herquet, G Alves, M E Pol, M H G Souza, M Vaz, D De Jesus Damiao, V Oguri, A Santoro, A Sznajder, E De Moraes Gregores, R L Iope, S F Novaes, T Tomei, T Anguelov, G Antchev, I Atanasov, J Damgov, N Darmenov, L Dimitrov, V Genchev, P Iaydjiev, A Marinov,

S Piperov, S Stoykova, G Sultanov, R Trayanov, I Vankov, C Cheshkov, A Dimitrov, M Dyulendarova, I Glushkov, V Kozhuharov, L Litov, M Makariev, E Marinova, S Markov, M Mateev, I Nasteva, B Pavlov, P Petev, P Petkov, V Spassov, Z Toteva, V Velev, V Verguilov, J G Bian, G M Chen, H S Chen, M Chen, C H Jiang, B Liu, X Y Shen, H S Sun, J Tao, J Wang, M Yang, Z Zhang, W R Zhao, H L Zhuang, Y Ban, J Cai, Y C Ge, S Liu, H T Liu, L Liu, S J Qian, Q Wang, Z H Xue, Z C Yang, Y L Ye, J Ying, P J Li, J Liao, Z L Xue, D S Yan, H Yuan, C A Carrillo Montoya, J C Sanabria, N Godinovic, I Puljak, I Soric, Z Antunovic, M Dzelalija, K Marasovic, V Brigljevic, K Kadija, S Morovic, R Fereos, C Nicolaou, A Papadakis, F Ptochos, P A Razis, D Tsiakkouri, Z Zinonos, A Hektor, M Kadastik, K Kannike, E Lippmaa, M Müntel, M Raidal, L Rebane, P A Aarnio, E Anttila, K Banzuzi, P Bulteau, S Czellar, N Eiden, C Eklund, P Engstrom, A Heikkinen, A Honkanen, J Härkönen, V Karimäki, H M Katajisto, R Kinnunen, J Klem, J Kortesmaa, M Kotamäki, A Kuronen, T Lampén, K Lassila-Perini, V Lefébure, S Lehti, T Lindén, P R Luukka, S Michal, F Moura Brigido, T Mäenpää, T Nyman, J Nystén, E Pietarinen, K Skog, K Tammi, E Tuominen, J Tuominiemi, D Ungaro, T P Vanhala, L Wendland, C Williams, M Iskanius, A Korpela, G Polese, T Tuuva, G Bassompierre, A Bazan, P Y David, J Ditta, G Drobychev, N Fouque, J P Guillaud, V Hermel, A Karneyeu, T Le Flour, S Lieunard, M Maire, P Mendiburu, P Nedelec, J P Peigneux, M Schneegans, D Sillou, J P Vialle, M Anfreville, J P Bard, P Besson, E Bougamont, M Boyer, P Bredy, R Chipaux, M Dejardin, D Denegri, J Descamps, B Fabbro, J L Faure, S Ganjour, F X Gentit, A Givernaud, P Gras, G Hamel de Monchenault, P Jarry, C Jeanney, F Kircher, M C Lemaire, Y Lemoigne, B Levesy, E Locci, J P Lottin, I Mandjavidze, M Mur, J P Pansart, A Payn, J Rander, J M Reymond, J Rolquin, F Rondeaux, A Rosowsky, J Y A Rousse, Z H Sun, J Tartas, A Van Lysebetten, P Venault, P Verrecchia, M Anduze, J Badier, S Baffioni, M Bercher, C Bernet, U Berthon, J Bourotte, A Busata, P Busson, M Cerutti, D Chamont, C Charlot, C Collard, A Debraine, D Decotigny, L Dobrzynski, O Ferreira, Y Geerebaert, J Gilly, C Gregory, L Guevara Riveros, M Haguenauer, A Karar, B Koblitz, D Lecouturier, A Mathieu, G Milleret, P Miné, P Paganini, P Poilleux, N Pukhaeva, N Regnault, T Romanteau, I Semeniouk, Y Sirois, C Thiebaux, J C Vanel, A Zabi, J L Agram, A Albert, L Anckenmann, J Andrea, F Anstotz, A M Bergdolt, J D Berst, R Blaes, D Bloch, J M Brom, J Cailleret, F Charles, E Christophel, G Claus, J Coffin, C Colledani, J Croix, E Dangelser, N Dick, F Didierjean, F Drouhin, W Dulinski, J P Ernenwein, R Fang, J C Fontaine, G Gaudiot, W Geist, D Gelé, T Goeltzenlichter, U Goerlach, P Graehling, L Gross, C Guo Hu, J M Helleboid, T Henkes, M Hoffer, C Hoffmann, J Hosselet, L Houchu, Y Hu, D Huss, C Illinger, F Jeanneau, P Juillot, T Kachelhoffer, M R Kapp, H Kettunen, L Lakehal Ayat, A C Le Bihan, A Lounis, C Maazouzi, V Mack, P Majewski, D Mangeol, J Michel, S Moreau, C Olivetto, A Pallarès, Y Patois, P Pralavorio, C Racca, Y Riahi, I Ripp-Baudot, P Schmitt, J P Schunck, G Schuster, B Schwaller, M H Sigward, J L Sohler, J Speck, R Strub,

T Todorov, R Turchetta, P Van Hove, D Vintache, A Zghiche, M Ageron, J E Augustin, C Baty, G Baulieu, M Bedjidian, J Blaha, A Bonnevaux, G Boudoul, P Brunet, E Chabanat, E C Chabert, R Chierici, V Chorowicz, C Combaret, D Contardo, R Della Negra, P Depasse, O Drapier, M Dupanloup, T Dupasquier, H El Mamouni, N Estre, J Fay, S Gascon, N Giraud, C Girerd, G Guillot, R Haroutunian, B Ille, M Lethuillier, N Lumb, C Martin, H Mathez, G Maurelli, S Muanza, P Pangaud, S Perries, O Ravat, E Schibler, F Schirra, G Smadja, S Tissot, B Trocme, S Vanzetto, J P Walder, Y Bagaturia, D Mjavia, A Mzhavia, Z Tsamalaidze, V Roinishvili, R Adolphi, G Anagnostou, R Brauer, W Braunschweig, H Esser, L Feld, W Karpinski, A Khomich, K Klein, C Kukulies, K Lübelsmeyer, J Olzem, A Ostaptchouk, D Pandoulas, G Pierschel, F Raupach, S Schael, A Schultz von Dratzig, G Schwering, R Siedling, M Thomas, M Weber, B Wittmer, M Wlochal, F Adamczyk, A Adolf, G Altenhöfer, S Bechstein, S Bethke, P Biallass, O Biebel, M Bontenackels, K Bosseler, A Böhm, M Erdmann, H Faissner, B Fehr, H Fesefeldt, G Fetchenhauer, J Frangenheim, J H Frohn, J Grooten, T Hebbeker, S Hermann, E Hermens, G Hilgers, K Hoepfner, C Hof, E Jacobi, S Kappler, M Kirsch, P Kreuzer, R Kupper, H R Lampe, D Lanske, R Mameghani, A Meyer, S Meyer, T Moers, E Müller, R Pahlke, B Philipps, D Rein, H Reithler, W Reuter, P Rütten, S Schulz, H Schwarthoff, W Sobek, M Sowa, T Stapelberg, H Szczesny, H Teykal, D Teyssier, H Tomme, W Tomme, M Tonutti, O Tsigenov, J Tutas, J Vandenhirtz, H Wagner, M Wegner, C Zeidler, F Beissel, M Davids, M Duda, G Flügge, M Giffels, T Hermanns, D Heydhausen, S Kalinin, S Kasselmann, G Kaussen, T Kress, A Linn, A Nowack, L Perchalla, M Poettgens, O Pooth, P Sauerland, A Stahl, D Tornier, M H Zoeller, U Behrens, K Borras, A Flossdorf, D Hatton, B Hegner, M Kasemann, R Mankel, A Meyer, J Mnich, C Rosemann, C Youngman, W D Zeuner, F Bechtel, P Buhmann, E Butz, G Flucke, R H Hamdorf, U Holm, R Klanner, U Pein, N Schirm, P Schleper, G Steinbrück, R Van Staa, R Wolf, B Atz, T Barvich, P Blüm, F Boegelspacher, H Bol, Z Y Chen, S Chowdhury, W De Boer, P Dehm, G Dirkes, M Fahrer, U Felzmann, M Frey, A Furgeri, E Gregoriev, F Hartmann, F Hauler, S Heier, K Kärcher, B Ledermann, S Mueller, Th Müller, D Neuberger, C Piasecki, G Quast, K Rabbertz, A Sabellek, A Scheurer, F P Schilling, H J Simonis, A Skiba, P Steck, A Theel, W H Thümmel, A Trunov, A Vest, T Weiler, C Weiser, S Weseler, V Zhukov, M Barone, G Daskalakis, N Dimitriou, G Fanourakis, C Filippidis, T Geralis, C Kalfas, K Karafasoulis, A Koimas, A Kyriakis, S Kyriazopoulou, D Loukas, A Markou, C Markou, N Mastroyiannopoulos, C Mavrommatis, J Mousa, I Papadakis, E Petrakou, I Siotis, K Theofilatos, S Tzamarias, A Vayaki, G Vermisoglou, A Zachariadou, L Gouskos, G Karapostoli, P Katsas, A Panagiotou, C Papadimitropoulos, X Aslanoglou, I Evangelou, P Kokkas, N Manthos, I Papadopoulos, F A Triantis, G Bencze, L Boldizsar, G Debreczeni, C Hajdu, P Hidas, D Horvath, P Kovesarki, A Laszlo, G Odor, G Patay, F Sikler, G Veres, G Vesztergombi, P Zalan, A Fenyvesi, J Imrek, J Molnar, D Novak, J Palinkas,

G Szekely, N Beni, A Kapusi, G Marian, B Radics, P Raics, Z Szabo, Z Szillasi, Z L Trocsanyi, G Zilizi, H S Bawa, S B Beri, V Bhandari, V Bhatnagar, M Kaur, J M Kohli, A Kumar, B Singh, J B Singh, S Arora, S Bhattacharya, S Chatterji, S Chauhan, B C Choudhary, P Gupta, M Jha, K Ranjan, R K Shivpuri, A K Srivastava, R K Choudhury, D Dutta, M Ghodgaonkar, S Kailas, S K Kataria, A K Mohanty, L M Pant, P Shukla, A Topkar, T Aziz, Sunanda Banerjee, S Bose, S Chendvankar, P V Deshpande, M Guchait, A Gurtu, M Maity, G Majumder, K Mazumdar, A Nayak, M R Patil, S Sharma, K Sudhakar, B S Acharya, Sudeshna Banerjee, S Bheesette, S Dugad, S D Kalmani, V R Lakkireddi, N K Mondal, N Panyam, P Verma, H Arfaei, M Hashemi, M Mohammadi Najafabadi, A Moshaii, S Paktinat Mehdiabadi, M Felcini, M Grunewald, K Abadjiev, M Abbrescia, L Barbone, P Cariola, F Chiumarulo, A Clemente, A Colaleo, D Creanza, N De Filippis, M De Palma, G De Robertis, G Donvito, R Ferorelli, L Fiore, M Franco, D Giordano, R Guida, G Iaselli, N Lacalamita, F Loddo, G Maggi, M Maggi, N Manna, B Marangelli, M S Mennea, S My, S Natali, S Nuzzo, G Papagni, C Pinto, A Pompili, G Pugliese, A Ranieri, F Romano, G Roselli, G Sala, G Selvaggi, L Silvestris, P Tempesta, R Trentadue, S Tupputi, G Zito, G Abbiendi, W Bacchi, C Battilana, A C Benvenuti, M Boldini, D Bonacorsi, S Braibant-Giacomelli, V D Cafaro, P Capiluppi, A Castro, F R Cavallo, C Ciocca, G Codispoti, M Cuffiani, I D'Antone, G M Dallavalle, F Fabbri, A Fanfani, S Finelli, P Giacomelli, V Giordano, M Giunta, C Grandi, M Guerzoni, L Guiducci, S Marcellini, G Masetti, A Montanari, F L Navarria, F Odorici, A Paolucci, G Pellegrini, A Perrotta, A M Rossi, T Rovelli, G P Siroli, G Torromeo, R Travaglini, G P Veronese, S Albergo, M Chiorboli, S Costa, M Galanti, G Gatto Rotondo, N Giudice, N Guardone, F Noto, R Potenza, M A Saizu, G Salemi, C Sutera, A Tricomi, C Tuve, L Bellucci, M Brianzi, G Broccolo, E Catacchini, V Ciulli, C Civinini, R D'Alessandro, E Focardi, S Frosali, C Genta, G Landi, P Lenzi, A Macchiolo, F Maletta, F Manolescu, C Marchettini, L Masetti, S Mersi, M Meschini, C Minelli, S Paoletti, G Parrini, E Scarlini, G Sguazzoni, L Benussi, M Bertani, S Bianco, M Caponero, D Colonna, L Daniello, F Fabbri, F Felli, M Giardoni, A La Monaca, B Ortenzi, M Pallotta, A Paolozzi, C Paris, L Passamonti, D Pierluigi, B Ponzio, C Pucci, A Russo, G Saviano, P Fabbricatore, S Farinon, M Greco, R Musenich, S Badoer, L Berti, M Biasotto, S Fantinel, E Frizziero, U Gastaldi, M Gulmini, F Lelli, G Maron, S Squizzato, N Toniolo, S Traldi, S Banfi, R Bertoni, M Bonesini, L Carbone, G B Cerati, F Chignoli, P D'Angelo, A De Min, P Dini, F M Farina, F Ferri, P Govoni, S Magni, M Malberti, S Malvezzi, R Mazza, D Menasce, V Miccio, L Moroni, P Negri, M Paganoni, D Pedrini, A Pullia, S Ragazzi, N Redaelli, M Rovere, L Sala, S Sala, R Salerno, T Tabarelli de Fatis, V Tancini, S Taroni, A Boiano, F Cassese, C Cassese, A Cimmino, B D'Aquino, L Lista, D Lomidze, P Noli, P Paolucci, G Passeggio, D Piccolo, L Roscilli, C Sciacca, A Vanzanella, P Azzi, N Bacchetta, L Barcellan, M Bellato, M Benettoni, D Bisello, E Borsato, A Candelori, R Carlin, L Castellani, P Checchia, L Ciano, A Colombo,

E Conti, M Da Rold, F Dal Corso, M De Giorgi, M De Mattia, T Dorigo, U Dosselli, C Fanin, G Galet, F Gasparini, U Gasparini, A Giraldo, P Giubilato, F Gonella, A Gresele, A Griggio, P Guaita, A Kaminskiy, S Karaevskii, V Khomenkov, D Kostylev, S Lacaprara, I Lazzizzera, I Lippi, M Loreti, M Margoni, R Martinelli, S Mattiazzo, M Mazzucato, A T Meneguzzo, L Modenese, F Montecassiano, A Neviani, M Nigro, A Paccagnella, D Pantano, A Parenti, M Passaseo, R Pedrotta, M Pegoraro, G Rampazzo, S Reznikov, P Ronchese, A Sancho Daponte, P Sartori, I Stavitskiy, M Tessaro, E Torassa, A Triossi, S Vanini, S Ventura, L Ventura, M Verlato, M Zago, F Zatti, P Zotto, G Zumerle, P Baesso, G Belli, U Berzano, S Bricola, A Grelli, G Musitelli, R Nardò, M M Necchi, D Pagano, S P Ratti, C Riccardi, P Torre, A Vicini, P Vitulo, C Viviani, D Aisa, S Aisa, F Ambroglini, M M Angarano, E Babucci, D Benedetti, M Biasini, G M Bilei, S Bizzaglia, M T Brunetti, B Caponeri, B Checcucci, R Covarelli, N Dinu, L Fanò, L Farnesini, M Giorgi, P Lariccia, G Mantovani, F Moscatelli, D Passeri, A Piluso, P Placidi, V Postolache, R Santinelli, A Santocchia, L Servoli, D Spiga, P Azzurri, G Bagliesi, G Balestri, A Basti, R Bellazzini, L Benucci, J Bernardini, L Berretta, S Bianucci, T Boccali, A Bocci, L Borrello, F Bosi, F Bracci, A Brez, F Calzolari, R Castaldi, U Cazzola, M Ceccanti, R Cecchi, C Cerri, A S Cucoanes, R Dell'Orso, D Dobur, S Dutta, F Fiori, L Foà, A Gaggelli, S Gennai, A Giassi, S Giusti, D Kartashov, A Kraan, L Latronico, F Ligabue, S Linari, T Lomtadze, G A Lungu, G Magazzu, P Mammini, F Mariani, G Martinelli, M Massa, A Messineo, A Moggi, F Palla, F Palmonari, G Petragnani, G Petrucciani, A Profeti, F Raffaelli, D Rizzi, G Sanguinetti, S Sarkar, G Segneri, D Sentenac, A T Serban, A Slav, P Spagnolo, G Spandre, R Tenchini, S Tolaini, G Tonelli, A Venturi, P G Verdini, M Vos, L Zaccarelli, S Baccaro, L Barone, A Bartoloni, B Borgia, G Capradossi, F Cavallari, A Cecilia, D D'Angelo, I Dafinei, D Del Re, E Di Marco, M Diemoz, G Ferrara, C Gargiulo, S Guerra, M Iannone, E Longo, M Montecchi, M Nuccetelli, G Organtini, A Palma, R Paramatti, F Pellegrino, S Rahatlou, C Rovelli, F Safai Tehrani, A Zullo, G Alampi, N Amapane, R Arcidiacono, S Argiro, M Arneodo, R Bellan, F Benotto, C Biino, S Bolognesi, M A Borgia, C Botta, A Brasolin, N Cartiglia, R Castello, G Cerminara, R Cirio, M Cordero, M Costa, D Dattola, F Daudo, G Dellacasa, N Demaria, G Dughera, F Dumitrache, R Farano, G Ferrero, E Filoni, G Kostyleva, H E Larsen, C Mariotti, M Marone, S Maselli, E Menichetti, P Mereu, E Migliore, G Mila, V Monaco, M Musich, M Nervo, M M Obertino, R Panero, A Parussa, N Pastrone, C Peroni, G Petrillo, A Romero, M Ruspa, R Sacchi, M Scalise, A Solano, A Staiano, P P Trapani, D Trocino, V Vaniev, A Vilela Pereira, A Zampieri, S Belforte, F Cossutti, G Della Ricca, B Gobbo, C Kavka, A Penzo, Y E Kim, S K Nam, D H Kim, G N Kim, J C Kim, D J Kong, S R Ro, D C Son, S Y Park, Y J Kim, J Y Kim, I T Lim, M Y Pac, S J Lee, S Y Jung, J T Rhee, S H Ahn, B S Hong, Y K Jeng, M H Kang, H C Kim, J H Kim, T J Kim, K S Lee, J K Lim, D H Moon, I C Park, S K Park, M S Ryu, K-S Sim, K J Son, S J Hong, Y I Choi, H Castilla Valdez, A Sanchez Hernandez, S Carrillo Moreno,

A Morelos Pineda, A Aerts, P Van der Stok, H Weffers, P Allfrey, R N C Gray, M Hashimoto, D Krofcheck, A J Bell, N Bernardino Rodrigues, P H Butler, S Churchwell, R Knegjens, S Whitehead, J C Williams, Z Aftab, U Ahmad, I Ahmed, W Ahmed, M I Asghar, S Asghar, G Dad, M Hafeez, H R Hoorani, I Hussain, N Hussain, M Iftikhar, M S Khan, K Mehmood, A Osman, H Shahzad, A R Zafar, A Ali, A Bashir, A M Jan, A Kamal, F Khan, M Saeed, S Tanwir, M A Zafar, J Blocki, A Cyz, E Gladysz-Dziadus, S Mikocki, M Rybczynski, J Turnau, Z Wlodarczyk, P Zychowski, K Bunkowski, M Cwiok, H Czyrkowski, R Dabrowski, W Dominik, K Doroba, A Kalinowski, K Kierzkowski, M Konecki, J Krolikowski, I M Kudla, M Pietrusinski, K Pozniak, W Zabolotny, P Zych, R Gokieli, L Goscilo, M Górski, K Nawrocki, P Traczyk, G Wrochna, P Zalewski, K T Pozniak, R Romaniuk, W M Zabolotny, R Alemany-Fernandez, C Almeida, N Almeida, A S Araujo Vila Verde, T Barata Monteiro, M Bluj, S Da Mota Silva, A David Tinoco Mendes, M Freitas Ferreira, M Gallinaro, M Husejko, A Jain, M Kazana, P Musella, R Nobrega, J Rasteiro Da Silva, P Q Ribeiro, M Santos, P Silva, S Silva, I Teixeira, J P Teixeira, J Varela, G Varner, N Vaz Cardoso, I Altsybeev, K Babich, A Belkov, I Belotelov, P Bunin, S Chesnevskaya, V Elsha, Y Ershov, I Filozova, M Finger, M Finger Jr, A Golunov, I Golutvin, N Gorbounov, I Gramenitski, V Kalagin, A Kamenev, V Karjavin, S Khabarov, V Khabarov, Y Kiryushin, V Konoplyanikov, V Korenkov, G Kozlov, A Kurenkov, A Lanev, V Lysiakov, A Malakhov, I Melnitchenko, V V Mitsyn, K Moisenz, P Moisenz, S Movchan, E Nikonov, D Oleynik, V Palichik, V Perelygin, A Petrosyan, E Rogalev, V Samsonov, M Savina, R Semenov, S Sergeev, S Shmatov, S Shulha, V Smirnov, D Smolin, A Tcheremoukhine, O Teryaev, E Tikhonenko, A Urkinbaev, S Vasil'ev, A Vishnevskiy, A Volodko, N Zamiatin, A Zarubin, P Zarubin, E Zubarev, N Bondar, Y Gavrikov, V Golovtsov, Y Ivanov, V Kim, V Kozlov, V Lebedev, G Makarenkov, F Moroz, P Neustroev, G Obrant, E Orishchin, A Petrunin, Y Shcheglov, A Shchetkovskiy, V Sknar, V Skorobogatov, I Smirnov, V Sulimov, V Tarakanov, L Uvarov, S Vavilov, G Velichko, S Volkov, A Vorobyev, D Chmelev, D Druzhkin, A Ivanov, V Kudinov, O Logatchev, S Onishchenko, A Orlov, V Sakharov, V Smetannikov, A Tikhomirov, S Zavodthikov, Yu Andreev, A Anisimov, V Duk, S Gninenko, N Golubev, D Gorbunov, M Kirsanov, N Krasnikov, V Matveev, A Pashenkov, A Pastsyak, V E Postoev, A Sadovski, A Skassyrskaia, Alexander Solovey, Anatoly Solovey, D Soloviev, A Toropin, S Troitsky, A Alekhin, A Baldov, V Epshteyn, V Gavrilov, N Ilina, V Kaftanov, V Karpishin, I Kiselevich, V Kolosov, M Kossov, A Krokhotin, S Kuleshov, A Oulianov, A Pozdnyakov, G Safronov, S Semenov, N Stepanov, V Stolin, E Vlasov, V Zaytsev, E Boos, M Dubinin, L Dudko, A Ershov, G Eyyubova, A Gribushin, V Ilyin, V Klyukhin, O Kodolova, N A Kruglov, A Kryukov, I Lokhtin, L Malinina, V Mikhaylin, S Petrushanko, L Sarycheva, V Savrin, L Shamardin, A Sherstnev, A Snigirev, K Teplov, I Vardanyan, A M Fomenko, N Konovalova, V Kozlov, A I Lebedev, N Lvova, S V Rusakov, A Terkulov, V Abramov, S Akimenko,

A Artamonov, A Ashimova, I Azhgirey, S Bitioukov, O Chikilev, K Datsko, A Filine, A Godizov, P Goncharov, V Grishin, A Inyakin, V Kachanov, A Kalinin, A Khmelnikov, D Konstantinov, A Korablev, V Krychkine, A Krinitsyn, A Levine, I Lobov, V Lukanin, Y Mel'nik, V Molchanov, V Petrov, V Petukhov, V Pikalov, A Ryazanov, R Ryutin, V Shelikhov, V Skvortsov, S Slabospitsky, A Sobol, A Sytine, V Talov, L Tourtchanovitch, S Troshin, N Tyurin, A Uzunian, A Volkov, S Zelepoukine, V Lukyanov, G Mamaeva, Z Prilutskaya, I Rumyantsev, S Sokha, S Tataurschikov, I Vasilyev, P Adzic, I Anicin, M Djordjevic, D Jovanovic, D Maletic, J Puzovic, N Smiljkovic, E Aguayo Navarrete, M Aguilar-Benitez, J Ahijado Munoz, J M Alarcon Vega, J Alberdi, J Alcaraz Maestre, M Aldaya Martin, P Arce, J M Barcala, J Berdugo, C L Blanco Ramos, C Burgos Lazaro, J Caballero Bejar, E Calvo, M Cerrada, M Chamizo Llatas, J J Chercoles Catalán, N Colino, M Daniel, B De La Cruz, A Delgado Peris, C Fernandez Bedoya, A Ferrando, M C Fouz, D Francia Ferrero, J Garcia Romero, P Garcia-Abia, O Gonzalez Lopez, J M Hernandez, M I Josa, J Marin, G Merino, A Molinero, J J Navarrete, J C Oller, J Puerta Pelayo, J C Puras Sanchez, J Ramirez, L Romero, C Villanueva Munoz, C Willmott, C Yuste, C Albajar, J F de Trocóniz, I Jimenez, R Macias, R F Teixeira, J Cuevas, J Fernández Menéndez, I Gonzalez Caballero, J Lopez-Garcia, H Naves Sordo, J M Vizan Garcia, I J Cabrillo, A Calderon, D Cano Fernandez, I Diaz Merino, J Duarte Campderros, M Fernandez, J Fernandez Menendez, C Figueroa, L A Garcia Moral, G Gomez, F Gomez Casademunt, J Gonzalez Sanchez, R Gonzalez Suarez, C Jorda, P Lobelle Pardo, A Lopez Garcia, A Lopez Virto, J Marco, R Marco, C Martinez Rivero, P Martinez Ruiz del Arbol, F Matorras, P Orviz Fernandez, A Patino Revuelta, T Rodrigo, D Rodriguez Gonzalez, A Ruiz Jimeno, L Scodellaro, M Sobron Sanudo, I Vila, R Vilar Cortabitarte, M Barbero, D Goldin, B Henrich, L Tauscher, S Vlachos, M Wadhwa, D Abbaneo, S M Abbas, I Ahmed, S Akhtar, M I Akhtar, E Albert, M Alidra, S Ashby, P Aspell, E Auffray, P Baillon, A Ball, S L Bally, N Bangert, R Barillère, D Barney, S Beauceron, F Beaudette, G Benelli, R Benetta, J L Benichou, W Bialas, A Bjorkebo, D Blechschmidt, C Bloch, P Bloch, S Bonacini, J Bos, M Bosteels, V Boyer, A Branson, H Breuker, R Bruneliere, O Buchmuller, D Campi, T Camporesi, A Caner, E Cano, E Carrone, A Cattai, J P Chatelain, M Chauvey, T Christiansen, M Ciganek, S Cittolin, J Cogan, A Conde Garcia, H Cornet, E Corrin, M Corvo, S Cucciarelli, B Curé, D D'Enterria, A De Roeck, T de Visser, C Delaere, M Delattre, C Deldicque, D Delikaris, D Deyrail, S Di Vincenzo, A Domeniconi, S Dos Santos, G Duthion, L M Edera, A Elliott-Peisert, M Eppard, F Fanzago, M Favre, H Foeth, R Folch, N Frank, S Fratianni, M A Freire, A Frey, A Fucci, W Funk, A Gaddi, F Gagliardi, M Gastal, M Gateau, J C Gayde, H Gerwig, A Ghezzi, D Gigi, K Gill, A S Giolo-Nicollerat, J P Girod, F Glege, W Glessing, R Gomez-Reino Garrido, R Goudard, R Grabit, J P Grillet, P Gutierrez Llamas, E Gutierrez Mlot, J Gutleber, R Hall-wilton, R Hammarstrom, M Hansen, J Harvey, A Hervé, J Hill, H F Hoffmann, A Holzner, A Honma, D Hufnagel, M Huhtinen, S D

Ilie, V Innocente, W Jank, P Janot, P Jarron, M Jeanrenaud, P Jouvel, R Kerkach, K Kloukinas, L J Kottelat, J C Labbé, D Lacroix, X Lagrue, C Lasseur, E Laure, J F Laurens, P Lazeyras, J M Le Goff, M Lebeau, P Lecoq, F Lemeilleur, M Lenzi, N Leonardo, C Leonidopoulos, M Letheren, M Liendl, F Limia-Conde, L Linssen, C Ljuslin, B Lofstedt, R Loos, J A Lopez Perez, C Lourenco, A Lyonnet, A Machard, R Mackenzie, N Magini, G Maire, L Malgeri, R Malina, M Mannelli, A Marchioro, J Martin, F Meijers, P Meridiani, E Meschi, T Meyer, A Meynet Cordonnier, J F Michaud, L Mirabito, R Moser, F Mossiere, J Muffat-Joly, M Mulders, J Mulon, E Murer, P Mättig, A Oh, A Onnela, M Oriunno, L Orsini, J A Osborne, C Paillard, I Pal, G Papotti, G Passardi, A Patino-Revuelta, V Patras, B Perea Solano, E Perez, G Perinic, J F Pernot, P Petagna, P Petiot, P Petit, A Petrilli, A Pfeiffer, C Piccut, M Pimiä, R Pintus, M Pioppi, A Placci, L Pollet, H Postema, M J Price, R Principe, A Racz, E Radermacher, R Ranieri, G Raymond, P Rebecchi, J Rehn, S Reynaud, H Rezvani Naraghi, D Ricci, M Ridel, M Risoldi, P Rodrigues Simoes Moreira, A Rohlev, G Roiron, G Rolandi, P Rumerio, O Runolfsson, V Ryjov, H Sakulin, D Samyn, L C Santos Amaral, H Sauce, E Sbrissa, P Scharff-Hansen, P Schieferdecker, W D Schlatter, B Schmitt, H G Schmuecker, M Schröder, C Schwick, C Schäfer, I Segoni, P Sempere Roldán, S Sgobba, A Sharma, P Siegrist, C Sigaud, N Sinanis, T Sobrier, P Sphicas, M Spiropulu, G Stefanini, A Strandlie, F Szoncsó, B G Taylor, O Teller, A Thea, E Tournefier, D Treille, P Tropea, J Troska, E Tsesmelis, A Tsirou, J Valls, I Van Vulpen, M Vander Donckt, F Vasey, M Vazquez Acosta, L Veillet, P Vichoudis, G Waurick, J P Wellisch, P Wertelaers, M Wilhelmsson, I M Willers, M Winkler, M Zanetti, W Bertl, K Deiters, P Dick, W Erdmann, D Feichtinger, K Gabathuler, Z Hochman, R Horisberger, Q Ingram, H C Kaestli, D Kotlinski, S König, P Poerschke, D Renker, T Rohe, T Sakhelashvili, A Starodumov, V Aleksandrov, F Behner, I Beniozef, B Betev, B Blau, A M Brett, L Caminada, Z Chen, N Chivarov, D Da Silva Di Calafiori, S Dambach, G Davatz, V Delachenal, R Della Marina, H Dimov, G Dissertori, M Dittmar, L Djambazov, M Dröge, C Eggel, J Ehlers, R Eichler, M Elmiger, G Faber, K Freudenreich, J F Fuchs, G M Georgiev, C Grab, C Haller, J Herrmann, M Hilgers, W Hintz, Hans Hofer, Heinz Hofer, U Horisberger, I Horvath, A Hristov, C Humbertclaude, B Iliev, W Kastli, A Kruse, J Kuipers, U Langenegger, P Lecomte, E Lejeune, G Leshev, C Lesmond, B List, P D Luckey, W Lustermann, J D Maillefaud, C Marchica, A Maurisset, B Meier, P Milenovic, M Milesi, F Moortgat, I Nanov, A Nardulli, F Nessi-Tedaldi, B Panev, L Pape, F Pauss, E Petrov, G Petrov, M M Peynekov, D Pitzl, T Punz, P Riboni, J Riedlberger, A Rizzi, F J Ronga, P A Roykov, U Röser, D Schinzel, A Schöning, A Sourkov, K Stanishev, S Stoenchev, F Stöckli, H Suter, P Trüb, S Udriot, D G Uzunova, I Veltchev, G Viertel, H P von Gunten, S Waldmeier-Wicki, R Weber, M Weber, J Weng, M Wensveen, F Wittgenstein, K Zagoursky, E Alagoz, C Amsler, V Chiochia, C Hoermann, C Regenfus, P Robmann, T Rommerskirchen, A Schmidt, S Steiner, D Tsirigkas, L Wilke, S Blyth,

Y H Chang, E A Chen, A Go, C C Hung, C M Kuo, S W Li, W Lin, P Chang, Y Chao, K F Chen, Z Gao, G W S Hou, Y B Hsiung, Y J Lei, S W Lin, R S Lu, J G Shiu, Y M Tzeng, K Ueno, Y Velikzhanin, C C Wang, M-Z Wang, S Aydin, A Azman, M N Bakirci, S Basegmez, S Cerci, I Dumanoglu, S Erturk, E Eskut, A Kayis Topaksu, H Kisoglu, P Kurt, K Ozdemir, N Ozdes Koca, H Ozkurt, S Ozturk, A Polatöz, K Sogut, H Topakli, M Vergili, G Önengüt, H Gamsizkan, S Sekmen, M Serin-Zeyrek, R Sever, M Zeyrek, M Deliomeroglu, E Gülmez, E Isiksal, M Kaya, O Kaya, S Ozkorucuklu, N Sonmez, B Grinev, V Lyubynskiy, V Senchyshyn, L Levchuk, S Lukyanenko, D Soroka, P Sorokin, S Zub, A Anjum, N Baker, T Hauer, R McClatchey, M Odeh, D Rogulin, A Solomonides, J J Brooke, R Croft, D Cussans, D Evans, R Frazier, N Grant, M Hansen, R D Head, G P Heath, H F Heath, C Hill, B Huckvale, J Jackson, C Lynch, C K Mackay, S Metson, S J Nash, D M Newbold, A D Presland, M G Probert, E C Reid, V J Smith, R J Tapper, R Walton, E Bateman, K W Bell, R M Brown, B Camanzi, I T Church, D J A Cockerill, J E Cole, J F Connolly, J A Coughlan, P S Flower, P Ford, V B Francis, M J French, S B Galagedera, W Gannon, A P R Gay, N I Geddes, R J S Greenhalgh, R N J Halsall, W J Haynes, J A Hill, F R Jacob, P W Jeffreys, L L Jones, B W Kennedy, A L Lintern, A B Lodge, A J Maddox, Q R Morrissey, P Murray, G N Patrick, C A X Pattison, M R Pearson, S P H Quinton, G J Rogers, J G Salisbury, A A Shah, C H Shepherd-Themistocleous, B J Smith, M Sproston, R Stephenson, S Taghavi, I R Tomalin, M J Torbet, J H Williams, W J Womersley, S D Worm, F Xing, M Apollonio, F Arteche, R Bainbridge, G Barber, P Barrillon, J Batten, R Beuselinck, P M Brambilla Hall, D Britton, W Cameron, D E Clark, I W Clark, D Colling, N Cripps, G Davies, M Della Negra, G Dewhirst, S Dris, C Foudas, J Fulcher, D Futyan, D J Graham, S Greder, S Greenwood, G Hall, J F Hassard, J Hays, G Iles, V Kasey, M Khaleeq, J Leaver, P Lewis, B C MacEvoy, O Maroney, E M McLeod, D G Miller, J Nash, A Nikitenko, E Noah Messomo, M Noy, A Papageorgiou, M Pesaresi, K Petridis, D R Price, X Qu, D M Raymond, A Rose, S Rutherford, M J Ryan, F Sciacca, C Seez, P Sharp, G Sidiropoulos, M Stettler, M Stoye, J Striebig, M Takahashi, H Tallini, A Tapper, C Timlin, L Toudup, T Virdee, S Wakefield, P Walsham, D Wardrope, M Wingham, Y Zhang, O Zorba, C Da Via, I Goitom, P R Hobson, D C Imrie, I Reid, C Selby, O Sharif, L Teodorescu, S J Watts, I Yaselli, E Hazen, A Heering, A Heister, C Lawlor, D Lazic, E Machado, J Rohlf, L Sulak, F Varela Rodriguez, S X Wu, A Avetisyan, T Bose, L Christofek, D Cutts, S Esen, R Hooper, G Landsberg, M Narain, D Nguyen, T Speer, K V Tsang, R Breedon, M Case, M Chertok, J Conway, P T Cox, J Dolen, R Erbacher, Y Fisyak, E Friis, G Grim, B Holbrook, W Ko, A Kopecky, R Lander, F C Lin, A Lister, S Maruyama, D Pellett, J Rowe, M Searle, J Smith, A Soha, M Squires, M Tripathi, R Vasquez Sierra, C Veelken, V Andreev, K Arisaka, Y Bonushkin, S Chandramouly, D Cline, R Cousins, S Erhan, J Hauser, M Ignatenko, C Jarvis, B Lisowski, C Matthey, B Mohr, J Mumford, S Otwinowski, Y Pischalnikov, G Rakness, P Schlein, Y Shi, B Tannenbaum, J Tucker, V Valuev, R Wallny, H G Wang, X Yang,

Y Zheng, J Andreeva, J Babb, S Campana, D Chrisman, R Clare, J Ellison, D Fortin, J W Gary, W Gorn, G Hanson, G Y Jeng, S C Kao, J G Layter, F Liu, H Liu, A Luthra, G Pasztor, H Rick, A Satpathy, B C Shen, R Stringer, V Sytnik, P Tran, S Villa, R Wilken, S Wimpenny, D Zer-Zion, J G Branson, J A Coarasa Perez, E Dusinberre, R Kelley, M Lebourgeois, J Letts, E Lipeles, B Mangano, T Martin, M Mojaver, J Muelmenstaedt, M Norman, H P Paar, A Petrucci, H Pi, M Pieri, A Rana, M Sani, V Sharma, S Simon, A White, F Würthwein, A Yagil, A Affolder, A Allen, C Campagnari, M D'Alfonso, A Dierlamm, J Garberson, D Hale, J Incandela, P Kalavase, S A Koay, D Kovalskyi, V Krutelyov, S Kyre, J Lamb, S Lowette, M Nikolic, V Pavlunin, F Rebassoo, J Ribnik, J Richman, R Rossin, Y S Shah, D Stuart, S Swain, J R Vlimant, D White, M Witherell, A Bornheim, J Bunn, J Chen, G Denis, P Galvez, M Gataullin, I Legrand, V Litvine, Y Ma, R Mao, D Nae, I Narsky, H B Newman, T Orimoto, C Rogan, S Shevchenko, C Steenberg, X Su, M Thomas, V Timciuc, F van Lingen, J Veverka, B R Voicu, A Weinstein, R Wilkinson, Y Xia, Y Yang, L Y Zhang, K Zhu, R Y Zhu, T Ferguson, D W Jang, S Y Jun, M Paulini, J Russ, N Terentyev, H Vogel, I Vorobiev, M Bunce, J P Cumalat, M E Dinardo, B R Drell, W T Ford, K Givens, B Heyburn, D Johnson, U Nauenberg, K Stenson, S R Wagner, L Agostino, J Alexander, F Blekman, D Cassel, S Das, J E Duboscq, L K Gibbons, B Heltsley, C D Jones, V Kuznetsov, J R Patterson, D Riley, A Ryd, S Stroiney, W Sun, J Thom, J Vaughan, P Wittich, C P Beetz, G Cirino, V Podrasky, C Sanzeni, D Winn, S Abdullin, M A Afaq, M Albrow, J Amundson, G Apollinari, M Atac, W Badgett, J A Bakken, B Baldin, K Banicz, L A T Bauerdick, A Baumbaugh, J Berryhill, P C Bhat, M Binkley, I Bloch, F Borcherding, A Boubekeur, M Bowden, K Burkett, J N Butler, H W K Cheung, G Chevenier, F Chlebana, I Churin, S Cihangir, W Dagenhart, M Demarteau, D Dykstra, D P Eartly, J E Elias, V D Elvira, D Evans, I Fisk, J Freeman, I Gaines, P Gartung, F J M Geurts, L Giacchetti, D A Glenzinski, E Gottschalk, T Grassi, D Green, C Grimm, Y Guo, O Gutsche, A Hahn, J Hanlon, R M Harris, T Hesselroth, S Holm, B Holzman, E James, H Jensen, M Johnson, U Joshi, B Klima, S Kossiakov, K Kousouris, J Kowalkowski, T Kramer, S Kwan, C M Lei, M Leininger, S Los, L Lueking, G Lukhanin, S Lusin, K Maeshima, J M Marraffino, D Mason, P McBride, T Miao, S Moccia, N Mokhov, S Mrenna, S J Murray, C Newman-Holmes, C Noeding, V O'Dell, M Paterno, D Petravick, R Pordes, O Prokofyev, N Ratnikova, A Ronzhin, V Sekhri, E Sexton-Kennedy, I Sfiligoi, T M Shaw, E Skup, R P Smith, W J Spalding, L Spiegel, M Stavrianakou, G Stiehr, A L Stone, I Suzuki, P Tan, W Tanenbaum, L E Temple, S Tkaczyk, L Uplegger, E W Vaandering, R Vidal, R Wands, H Wenzel, J Whitmore, E Wicklund, W M Wu, Y Wu, J Yarba, V Yarba, F Yumiceva, J C Yun, T Zimmerman, D Acosta, P Avery, V Barashko, P Bartalini, D Bourilkov, R Cavanaugh, S Dolinsky, A Drozdetskiy, R D Field, Y Fu, I K Furic, L Gorn, D Holmes, B J Kim, S Klimenko, J Konigsberg, A Korytov, K Kotov, P Levchenko, A Madorsky, K Matchev, G Mitselmakher, Y Pakhotin, C Prescott,

L Ramond, P Ramond, M Schmitt, B Scurlock, J Stasko, H Stoeck, D Wang, J Yelton, V Gaultney, L Kramer, L M Lebolo, S Linn, P Markowitz, G Martinez, J L Rodriguez, T Adams, A Askew, O Atramentov, M Bertoldi, W G D Dharmaratna, Y Gershtein, S V Gleyzer, S Hagopian, V Hagopian, C J Jenkins, K F Johnson, H Prosper, D Simek, J Thomaston, M Baarmand, L Baksay, S Guragain, M Hohlmann, H Mermerkaya, R Ralich, I Vodopiyanov, M R Adams, I M Anghel, L Apanasevich, O Barannikova, V E Bazterra, R R Betts, C Dragoiu, E J Garcia-Solis, C E Gerber, D J Hofman, R Hollis, A Iordanova, S Khalatian, C Mironov, E Shabalina, A Smoron, N Varelas, U Akgun, E A Albayrak, A S Ayan, R Briggs, K Cankocak, W Clarida, A Cooper, P Debbins, F Duru, M Fountain, E McCliment, J P Merlo, A Mestvirishvili, M J Miller, A Moeller, C R Newsom, E Norbeck, J Olson, Y Onel, L Perera, I Schmidt, S Wang, T Yetkin, E W Anderson, H Chakir, J M Hauptman, J Lamsa, B A Barnett, B Blumenfeld, C Y Chien, G Giurgiu, A Gritsan, D W Kim, C K Lae, P Maksimovic, M Swartz, N Tran, P Baringer, A Bean, J Chen, D Coppage, O Grachov, M Murray, V Radicci, J S Wood, V Zhukova, D Bandurin, T Bolton, K Kaadze, W E Kahl, Y Maravin, D Onoprienko, R Sidwell, Z Wan, B Dahmes, J Gronberg, J Hollar, D Lange, D Wright, C R Wuest, D Baden, R Bard, S C Eno, D Ferencek, N J Hadley, R G Kellogg, M Kirn, S Kunori, E Lockner, F Ratnikov, F Santanastasio, A Skuja, T Toole, L Wang, M Wetstein, B Alver, M Ballintijn, G Bauer, W Busza, G Gomez Ceballos, K A Hahn, P Harris, M Klute, I Kravchenko, W Li, C Loizides, T Ma, S Nahn, C Paus, S Pavlon, J Piedra Gomez, C Roland, G Roland, M Rudolph, G Stephans, K Sumorok, S Vaurynovich, E A Wenger, B Wyslouch, D Bailleux, S Cooper, P Cushman, A De Benedetti, A Dolgopolov, P R Dudero, R Egeland, G Franzoni, W J Gilbert, D Gong, J Grahl, J Haupt, K Klapoetke, I Kronkvist, Y Kubota, J Mans, R Rusack, S Sengupta, B Sherwood, A Singovsky, P Vikas, J Zhang, M Booke, L M Cremaldi, R Godang, R Kroeger, M Reep, J Reidy, D A Sanders, P Sonnek, D Summers, S Watkins, K Bloom, B Bockelman, D R Claes, A Dominguez, M Eads, M Furukawa, J Keller, T Kelly, C Lundstedt, S Malik, G R Snow, D Swanson, K M Ecklund, I Iashvili, A Kharchilava, A Kumar, M Strang, G Alverson, E Barberis, O Boeriu, G Eulisse, T McCauley, Y Musienko, S Muzaffar, I Osborne, S Reucroft, J Swain, L Taylor, L Tuura, B Gobbi, M Kubantsev, A Kubik, R A Ofierzynski, M Schmitt, E Spencer, S Stoynev, M Szleper, M Velasco, S Won, K Andert, B Baumbaugh, B A Beiersdorf, L Castle, J Chorny, A Goussiou, M Hildreth, C Jessop, D J Karmgard, T Kolberg, J Marchant, N Marinelli, M McKenna, R Ruchti, M Vigneault, M Wayne, D Wiand, B Bylsma, L S Durkin, J Gilmore, J Gu, P Killewald, T Y Ling, C J Rush, V Sehgal, G Williams, N Adam, S Chidzik, P Denes, P Elmer, A Garmash, D Gerbaudo, V Halyo, J Jones, D Marlow, J Olsen, P Piroué, D Stickland, C Tully, J S Werner, T Wildish, S Wynhoff, Z Xie, X T Huang, A Lopez, H Mendez, J E Ramirez Vargas, A Zatserklyaniy, A Apresyan, K Arndt, V E Barnes, G Bolla, D Bortoletto, A Bujak, A Everett, M Fahling, A F Garfinkel, L Gutay, N Ippolito, Y Kozhevnikov, A T Laasanen, C Liu, V Maroussov,

S Medved, P Merkel, D H Miller, J Miyamoto, N Neumeister, A Pompos, A Roy, A Sedov, I Shipsey, V Cuplov, N Parashar, P Bargassa, S J Lee, J H Liu, D Maronde, M Matveev, T Nussbaum, B P Padley, J Roberts, A Tumanov, A Bodek, H Budd, J Cammin, Y S Chung, P De Barbaro, R Demina, G Ginther, Y Gotra, S Korjenevski, D C Miner, W Sakumoto, P Slattery, M Zielinski, A Bhatti, L Demortier, K Goulianos, K Hatakeyama, C Mesropian, E Bartz, S H Chuang, J Doroshenko, E Halkiadakis, P F Jacques, D Khits, A Lath, A Macpherson, R Plano, K Rose, S Schnetzer, S Somalwar, R Stone, T L Watts, G Cerizza, M Hollingsworth, J Lazoflores, G Ragghianti, S Spanier, A York, A Aurisano, A Golyash, T Kamon, C N Nguyen, J Pivarski, A Safonov, D Toback, M Weinberger, N Akchurin, L Berntzon, K W Carrell, K Gumus, C Jeong, H Kim, S W Lee, B G Mc Gonagill, Y Roh, A Sill, M Spezziga, R Thomas, I Volobouev, E Washington, R Wigmans, E Yazgan, T Bapty, D Engh, C Florez, W Johns, T Keskinpala, E Luiggi Lopez, S Neema, S Nordstrom, S Pathak, P Sheldon, D Andelin, M W Arenton, M Balazs, M Buehler, S Conetti, B Cox, R Hirosky, M Humphrey, R Imlay, A Ledovskoy, D Phillips II, H Powell, M Ronquest, R Yohay, M Anderson, Y W Baek, J N Bellinger, D Bradley, P Cannarsa, D Carlsmith, I Crotty, S Dasu, F Feyzi, T Gorski, L Gray, K S Grogg, M Grothe, M Jaworski, P Klabbers, J Klukas, A Lanaro, C Lazaridis, J Leonard, R Loveless, M Magrans de Abril, A Mohapatra, G Ott, W H Smith, M Weinberg, D Wenman, G S Atoian, S Dhawan, V Issakov, H Neal, A Poblaguev, M E Zeller, G Abdullaeva, A Avezov, M I Fazylov, E M Gasanov, A Khugaev, Y N Koblik, M Nishonov, K Olimov, A Umaraliev, and B S Yuldashev. The cms experiment at the cern lhc. *Journal of Instrumentation*, 3(08):S08004, aug 2008. . URL `https://dx.doi.org/10.1088/1748-0221/3/08/S08004`.

The GIMP Development Team. Gnu image manipulation program (gimp), version 3.0.4. community, free software (license gplv3), 2025. URL `https://gimp.org/`. Version 3.0.4, Free Software.

Fyodor V. Tkachov. A theory of jet definition. *International Journal of Modern Physics A*, 17(21):2783–2884, August 2002. ISSN 1793-656X. . URL `http://dx.doi.org/10.1142/S0217751X02009977`.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL `https://arxiv.org/abs/1710.10903`.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris,

Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. . URL https://doi.org/10.1038/s41592-019-0686-2.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. .

Wikipedia contributors. Single-precision floating-point format, 2025a. URL https://en.wikipedia.org/wiki/Single-precision_floating-point_format. [Online; accessed 30-June-2025].

Wikipedia contributors. Relativistic breit-wigner distribution - wikipedia, the free encyclopedia, 2025b. URL https://en.wikipedia.org/w/index.php?title=Relativistic_Breit%E2%80%93Wigner_distribution. [Online; accessed 26-June-2025].

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2018. URL https://arxiv.org/abs/1810.00826.

Yang Yang, Ryan N. Lichtenwalter, and Nitesh V. Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, October 2014. ISSN 0219-3116. . URL http://dx.doi.org/10.1007/s10115-014-0789-0.

G Zweig. An SU_3 model for strong interaction symmetry and its breaking; Version 2. 1964. . URL https://cds.cern.ch/record/570209.