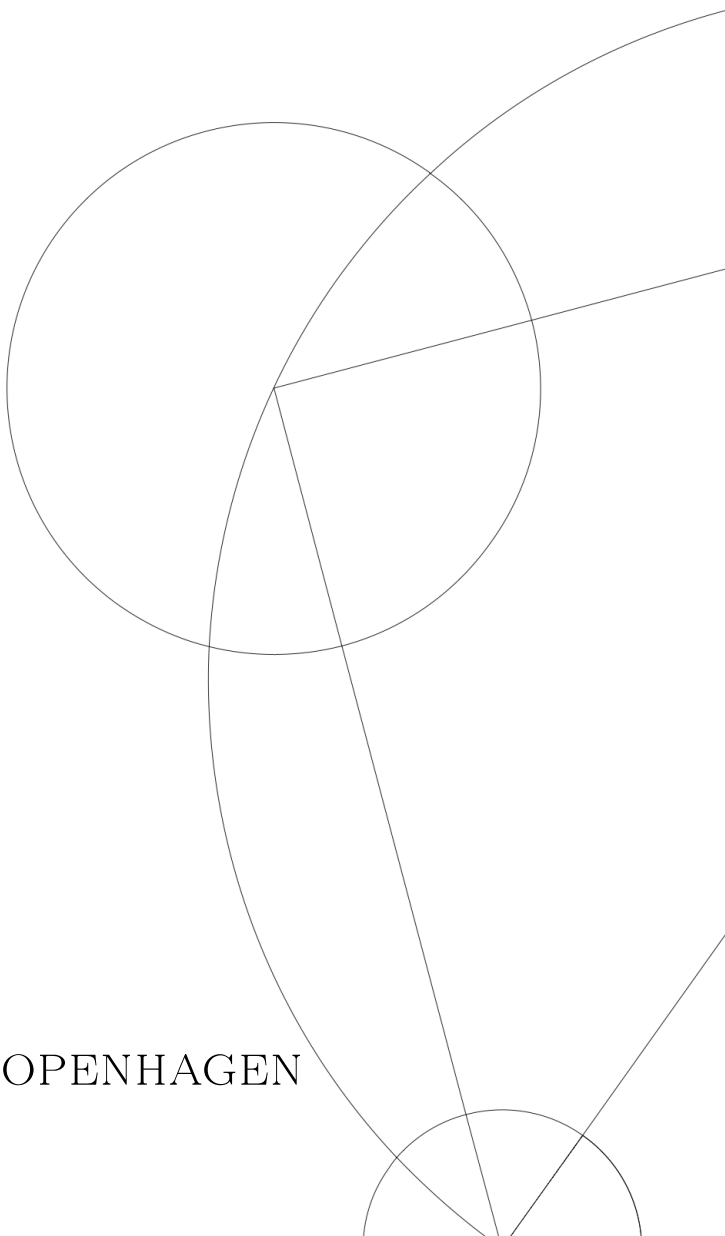
Track Reconstruction with GNN and CNN

PREP

Written by

Alejandro Maza Villalpando (HCJ888)

August 1, 2022



University of Copenhagen

Contents

1 Introduction

1.1 LHC

The Large Hadron Collider (LHC) is the largest particle collider in the world. The LHC main focus is particle physics – the study of the fundamental constituents of matter – but the physics programme at the laboratory is much broader, ranging from nuclear to high-energy physics, from studies of antimatter to the possible effects of cosmic rays on clouds [1]. In the LHC two proton beams are accelerated in opposite directions in such a way that the beams collide inside four different experiments, one of which is the A Toroidal LHC ApparatuS (ATLAS), seen in figure 1.

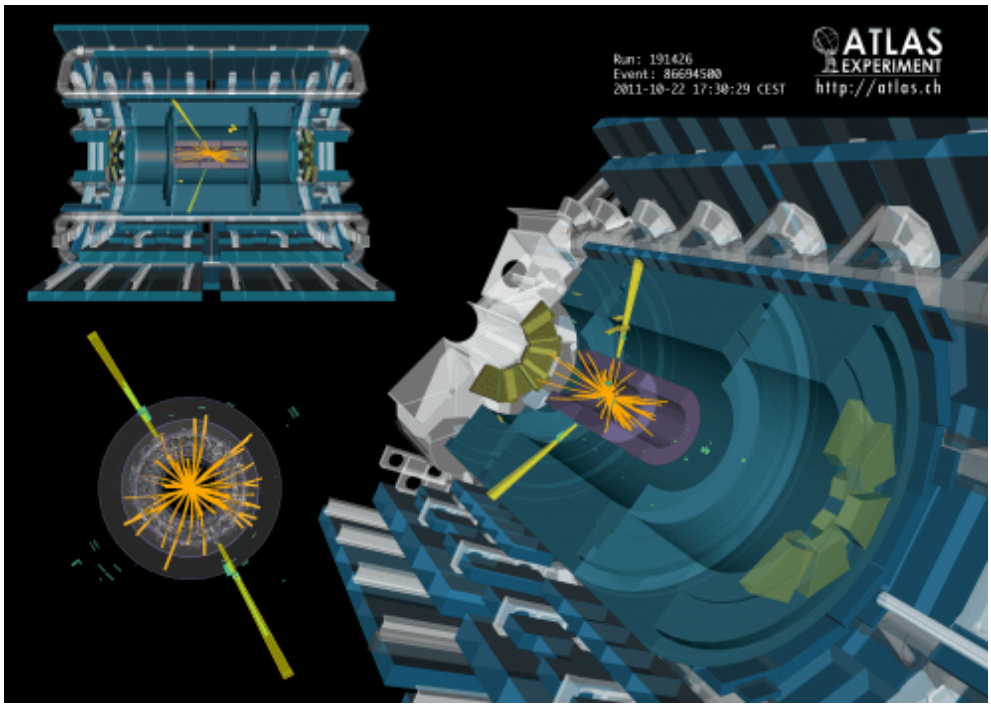


Figure 1: A two-photon event captured in the ATLAS experiment at CERN [2].

The study of the byproduct of these collisions allows physicists to discover new particles, analyse subatomic structures and observe new phenomena.

1.1.1 HL-LHC

The High-Luminosity Large Hadron Collider (HL-LHC) project aims to upgrade the performance of the LHC in order to increase the potential for discoveries after 2029. The objective is to increase the integrated luminosity by a factor of 10 beyond the LHC's design value [3]. Luminosity gives a measure of how many collisions are happening in a particle accelerator, it measures how many particles we're able to squeeze through a given space in a given time [4]. Therefore as the luminosity increase it will be possible to gather more data to analyse different phenomena. For example, the High-Luminosity LHC will produce at least 15 million Higgs bosons per year, compared to around three million from the LHC in 2017 [3].

1.1.2 ATLAS

ATLAS is a general-purpose particle physics experiment at the LHC [5]. ATLAS is a many-layered instrument designed record the high-energy particle collisions of the LHC, which take palace at a rate of over a billion interactions per second in the center of the detector [6]. The detector has the dimensions of a cylinder, 46m long, 25m in diameter, and it consists of four different detecting

subsystems wrapped concentrically in layers around the collision point to record the trajectory, momentum, and energy of particles, allowing them to be individually identified and measured [7]. This project focuses on the trigger and data acquisition system within the ATLAS detector [8], [9]. Curently the ATLAS detector is designed to observe up to 1.7 x 10^9 proton-proton collisions per second, this is more than 60 million mega bytes per second. However, only some of these events will contain interesting characteristics that might lead to new discoveries. The Trigger and Data Acquisition system ensures optimal data-taking conditions and selects the most interesting collision events for study [10]. Naturally when the HL-LHC project starts the amount of proton-proton collision will increase, making the current trigger and data acquisition system obsolete. Therefore a new trigger and data acquisition system needs to be developed. In the following sections the possibility to create such a system based on graph neural networks (GNN) is explored, as well as comparing this model with a convolutional neural network (CNN), a Extreme gradient boosting (XGB), and a multi layer perceptron (MLP).

1.2 Exploratory Data Analysis

The aim of the project is to build particle tracks based on the given data as shown in figure 2.

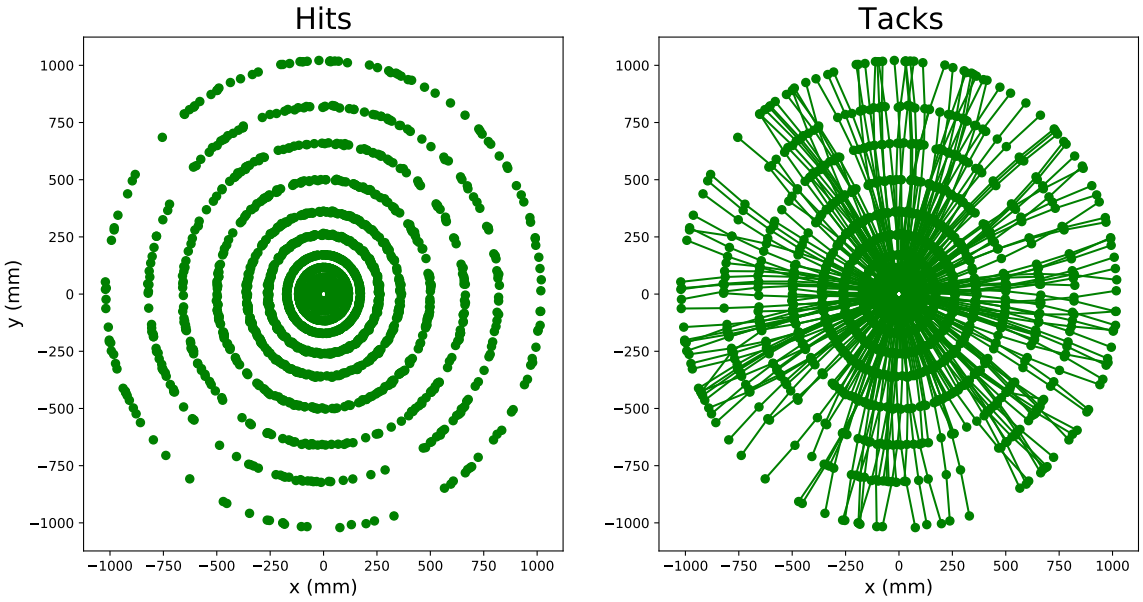


Figure 2: Left raw data, right reconstructed particle tracks.

The given data consists on a set of parameters that can be used to train the GNN, in addition to some information about the detector. It was taken from the TrackML Particle Tracking Challenge simulation data [11]. The data is divided into events, an event is considered to be a beam collision. Each event has 4 different files:

- hits
- cells
- truth
- particles

In order to minimize the latency of the algorithms only 100 events were taken into consideration.

1.2.1 Hits

The hits files have an average of $110,000 \pm 10,000$ particle hits per event. Each event contains x, y, z coordinates as well as volume id, layer id, module id, and hit id.

- Hit id, is unique for each event, and it is a identification of each recorded x, y, z, one of the particularities is that, the hit ids doesn't seam to have a specific order.
- Volume ids are a set of layers as seen in figure 4, it used to divide the detector, and to make the data more manageable.
- Layer ids, are sections on the detector where the particles leave a hit, there are used to manage the data and orientation.
- Module ids, are sections within the layers.

In order to visualize the sections were the particles hit the detector, I calculated the radius using $r = \sqrt{x^2 + y^2}$, and then plotted as seen in figure 3.

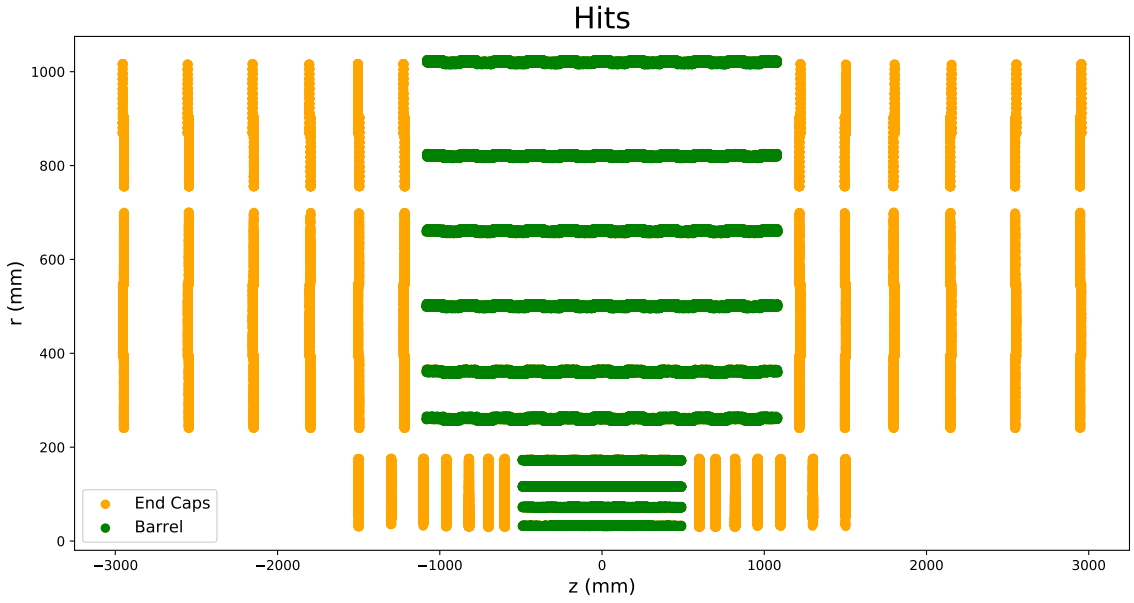


Figure 3: All hits in an event, reconstructing the trigger and data acquisition detector.

Is possible to compare figure 3 with the truth detector outline shown in figure 4, in order to understand the position of the hits as well as the volume-layer structure of the detector.

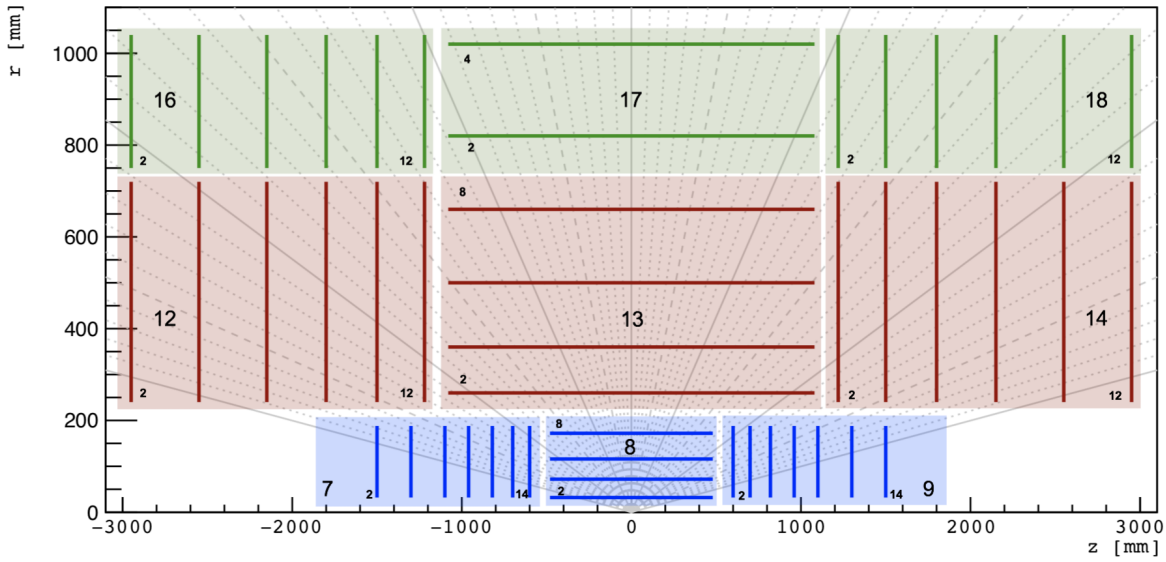


Figure 4: Trigger and data acquisition detector[11].

By analysing figure 3 and figure 4, I decided that for this project only the section defined as the barrel (volumes 8, 13 and 17) was going to be considered. The average amount of hits per layer per event in the barrel section is shown in table 1.

Average Hits per Layer	
Layer	Hits
0	7,900±900
1	6,700 ±700
2	5,900 ± 600
3	5,500 ± 500
4	6,500 ± 700
5	6,100 ± 600
6	5,800 ± 500
7	5,300 ± 400
8	4,900 ± 300
9	4,900 ± 300
Total	60,000 ± 6,000

Table 1: Average hits per layer.

The mean in table 1, and in the rest of the analysis are assumed to have a Gaussian distribution. The mean ($\bar{x}$) and standard deviation (σ) of a Gaussian distribution are given by:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{1}$$

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{2}$$

Where n is the amount of data points in the sample.

1.2.2 Cells

The cells file contains the constituent active detector cells that comprise each hit. The cells can be used to refine the hit to track association. A cell is the smallest granularity inside each detector module, and depending on the volume id a cell can be a square or a long rectangle. A cell is identified by two channel identifiers that are unique within each detector module and encode the position, much like column/row numbers of a matrix. A cell provide signal information that the detector module has recorded in addition to the position. Depending on the detector type only one of the channel identifiers is valid, e.g. for the strip detectors, and the value might have different resolution [11]. The cells files wasn't used in the project, it is just presented as an alternative method to analyse the data.

1.2.3 Truth

The truth file contains the truth coordinates x, y, z truth momentum in the coordinates p_x, p_y, p_z, weight of the particle w, as well as hit id and particle id. Using the particle id is possible to reconstruct the particle track as seen in figure 5, this information will be used to train the neural network.

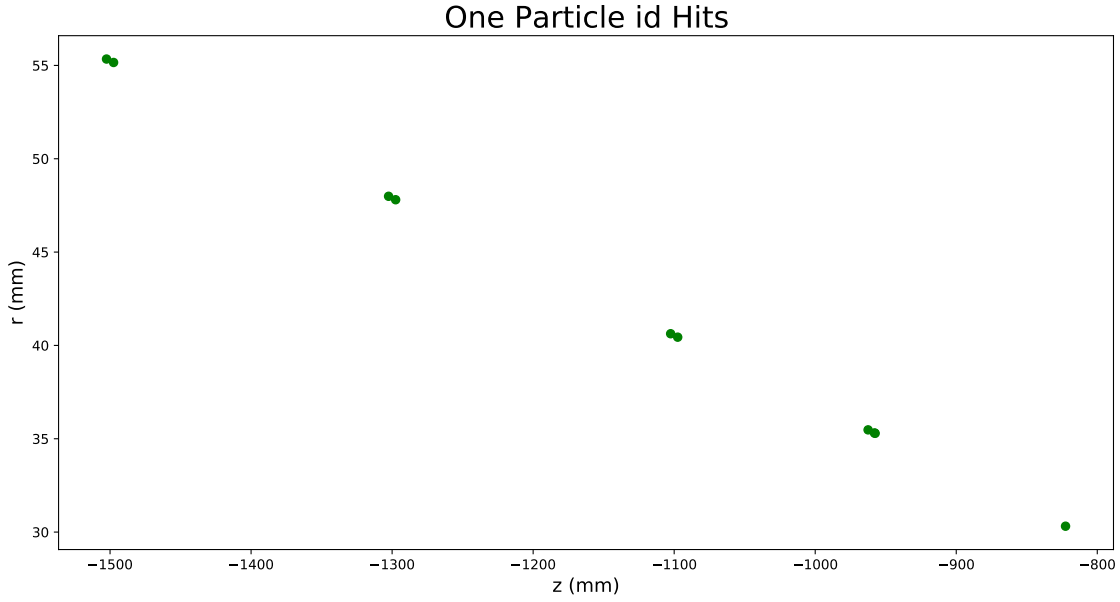


Figure 5: One particle track through the detector.

By analysing the data and as seen in figure 5 is possible to see that some of the particles have double hits in a layer, this is because the hits are registered in the modules within the layers, and in order to have an accurate measurement the modules overlap with each other causing the double hit. In addition to this some of the particles skip one layer for example a particle can have hits in layer 0,1,3,5 and 6. How to deal with this issues will be explained in Section 2.

1.2.4 Particles

The particles file contains the velocity components v_x, v_y, v_z momentum components p_x, p_y, p_z, charge q, and number of hits. The average particle amount per event is 11000 ± 1000. Using the events in the particles file is possible to obtain the distribution of hits in all the events as shown in figure 6.

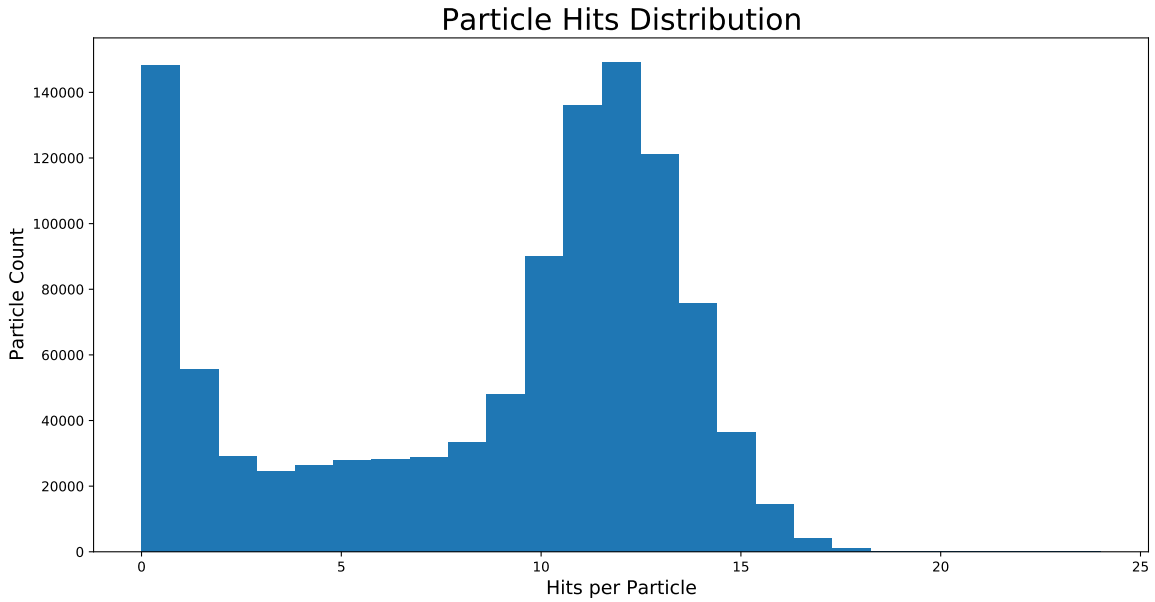


Figure 6: Hits per particle distribution.

As seen in figure 6 there are many particles whit zero hits, this data will be removed as part of the data filters in Section 2 and regarded as noise.

2 Data to Graph Format

2.1 Graphs

In order to use a GNN it is necessary to transform the data into a graph format. Graphs are mathematical objects used to model relationships between data points. A graph consist of nodes which are connected by edges. A graph can be represented by an ordered pair :

$$G = (X, E) \tag{3}$$

where X is a set of nodes, E is a set of edges, with $E \subseteq \{(x_i, x_j)|(x_i, x_j) \in X^2 \text{ and } x_i \neq x_j\}$ [12]. Graphs can be divided into two categories, directed and undirected. A directed graph is a graph which edges have a direction, meaning data flows in one direction only, and an undirected graph edges doesn't have a direction, therefore data can flow in both directions.

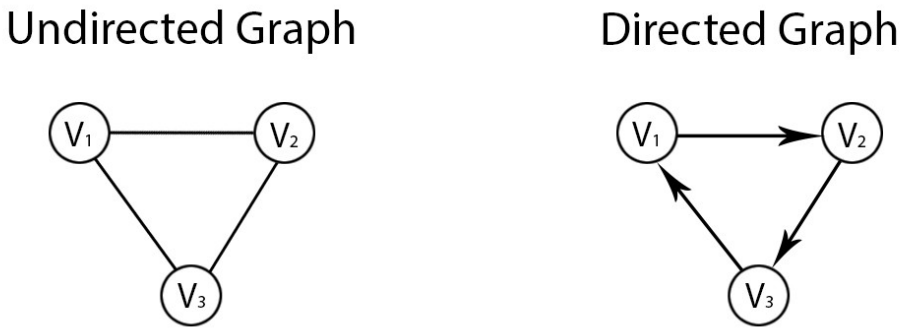


Figure 7: Undirected and directed graphs [13].

In this project the graph is directed, as the particles travel trough the detector in one direction, and the hits coordinates represent the nodes.

2.2 Data Filters

The way to create a graph from the hits is to connect hits in layer n with hits in layer $n+1$, if all possible connections between hits (edges) are considered the problem quickly becomes too big to handle. Using the information in table 1 is possible to calculate all the possible edges in the 10 layers in the order of $O(37)$. Therefore is necessary to reduce the amount of hits per layer, thus the amount of possible combinations. The first way to reduce the combinations without loosing valuable information is to use the geometry of the detector as explained in section 2.1.1

2.2.1 Section Filters

A way to reduce the amount of data handling by an individual processor, the detector can be divided into sections. In order to section the detector first is necessary to enrich the data, calculating transverse momentum $p_t = \sqrt{p_x^2 + p_y^2}$, azimuth angle $\phi = \arctan(\frac{y}{x})$, and the pseudorapidity $\eta = -\log(\frac{\theta}{2})$, where $\theta = \arctan(\frac{r}{z})$.
The detector is divided into 2 η sections and 4 ϕ sections as shown in figure 8.

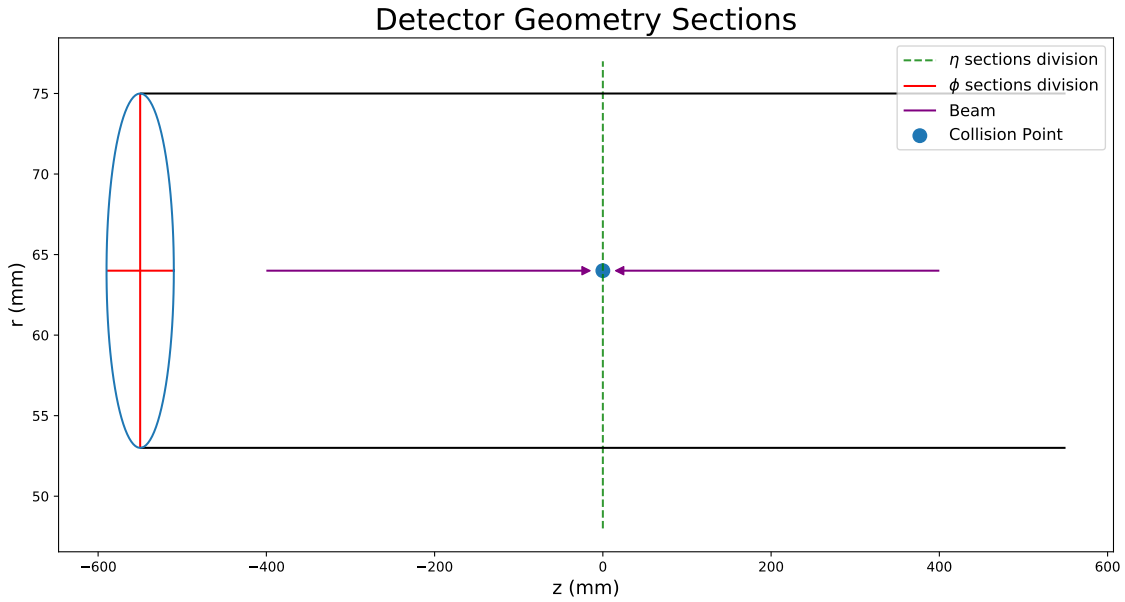


Figure 8: Sections on the detector.

In order to preserve most of the tracks an overlap of $\pm 0.26\phi$ and $\pm 0.5\eta$ on each section was included. In addition to this I decided to take into account only 8 (0,1,2,3,4,5,7,8) out of the 10 layers of the barrel, doing this wont affect the final rack reconstruction, as it is possible to extrapolate the track path even with less hits. The summary of the data after sectioning the detector is shown in table 2.

Initial Filters			
Filter	Hits Before Filter	Hits After Filter	Edges
Section	$60{,}000 \pm 6{,}000$	$9{,}600 \pm 400$	$O(24)$

Table 2: Section information.

With respect of the total amount of hits, sectioning the detector reduce by approximately 60% the amount of data, however no information is lost.

2.2.2 Initial Filters

The initial filters come from the data files and no calculation is needed in order to obtain the parameters.
The first data filter I implemented, was removing the hits id where the particle id is zero, which means that the hit did not originate from a reconstructible particle, but noise in the detector.
The second data filter I applied is to remove hits ids with weight equal zero, as this particle are consider not interesting ones.
In order to simplify the reconstruction, I assumed that particles cannot have 2 hits in the same layer, as the double hit per layer doesn't add any additional information to the particle track, Therefore I removed the second hit in each layer when the double hit is present, as the last filter in this section of the pipeline.
The summary of the amount of hist per event after the initial filter pipeline is shown in table 3.

Initial Filters			
Filter	Hits Before Filter	Hits After Filter	Edges
Noise	$9,600 \pm 400$	$7,900 \pm 400$	$O(22)$
Weight	$7,900 \pm 400$	$7,700 \pm 400$	$O(22)$
Double hit	$7,700 \pm 400$	$6,300 \pm 300$	$O(21)$

Table 3: Initial filter.

After passing the data through the initial filter pipeline $\approx 34\%$ of the data per section per event was removed.

2.2.3 Calculations Filters

The calculation filters pipeline uses the previously calculated p_t and eta. The first filter I applied in this section of the pipeline, was to remove the particles with $p_t < 1GeV$, because this particles are of no interest for the experiment. As seen in figure 9 this is the majority of the data.

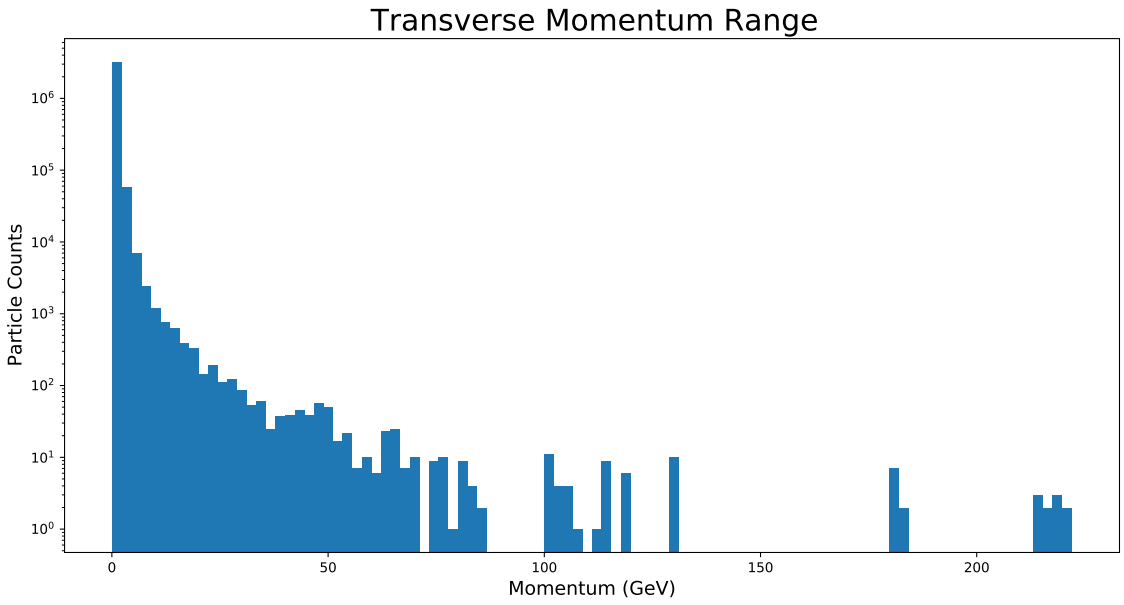


Figure 9: p_t distribution.

By applying the p_t filter and the volume filter, the majority of the data has an η in the range $-3 < \eta < 3$ as shown in figure 10. For completeness I removed the particles with an eta in the range, $3 < \eta < -3$, as this particles are not interesting for the researchers, although this filter doesn't produce any significant reduction.

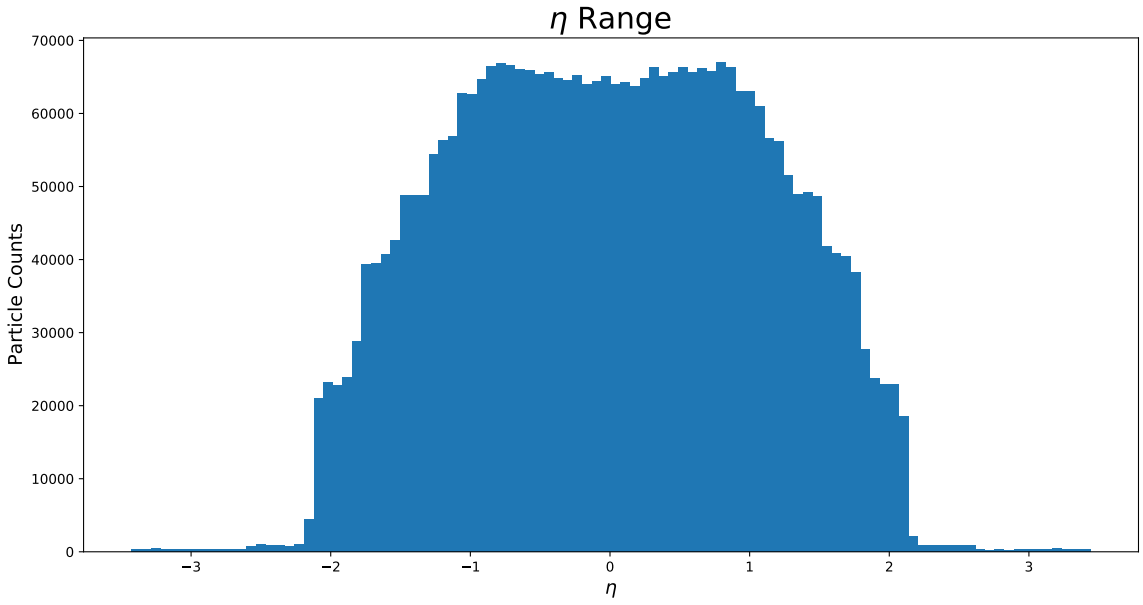


Figure 10: η distribution.

A summary of the hits per layer after the calculation filters is shown in table 4.

Calculation Filters			
Filter	Hits Before Filter	Hits After Filter	Edges
p_t	6,300±300	960±60	$O(13)$
η	960±60	960±60	$O(13)$

Table 4: Calculation filter.

After passing the data through the calculation filter pipeline approximately 85% of the data per event was removed with respect of the initial filters. As expected by analysing figure 9, approximately 99.98% of this is from the p_t filter, making it the most significant filter so far.

2.3 Node Edge Connections

In order to create a graph, I connected the hits in layer n with the hits in the layer n_{+1}. As mention previously the possible connections in the barrel was in the order of $O(37)$. After applying the previously mentioned filters and sectioning the detector the possible connections are in the order of $O(13)$. I implemented a further reduction on hits, by considering only particles with more than 4 hits leaving an average of 790 ± 50 hits per event per section with possible edges in the order $O(11)$. Furthermore not all particles have 8 hits as seen in table 5.

Percentage of Particles by Number of Hits	
Hits	Particles (%)
8	59±13
7	15±5
6	27±7
5	8±3

Table 5: Percentage of particles per section per event.

As the true data is available, hits are combined only in the layers where hits belong to the same particle, producing an average of $70,000 \pm 30,000$ edges per section.
However not all connections are physically possible and, again is possible to use the true particle tracks to calculate some parameters, and compare the distribution of these parameters between the true particles and the rest of the connections. Therefore the following calculations are necessary:

$$d\phi = \phi_{n+1} - \phi_n, \tag{4}$$

$$dz = z_{n+1} - z_n, \tag{5}$$

$$dr = r_{n+1} - r_n, \tag{6}$$

$$d\eta = \eta_{n+1} - \eta_n, \tag{7}$$

$$dR = \sqrt{d\eta^2 + d\phi^2}, \tag{8}$$

$$\phi_{slope} = \frac{\phi}{dr}, \tag{9}$$

$$z0 = z_n - r_n \frac{dz}{dr} \tag{10}$$

Where n indicates the layer number. Using this new parameters is possible to analyse the distributions of ϕ_{slope}, dR, and $z0$ of the truth connections and the rest of the connections. In order to get general parameters all data was used, as shown in figures 11, 12, and 13. This analysis produce a really useful insight in how to restrict the possible connections.

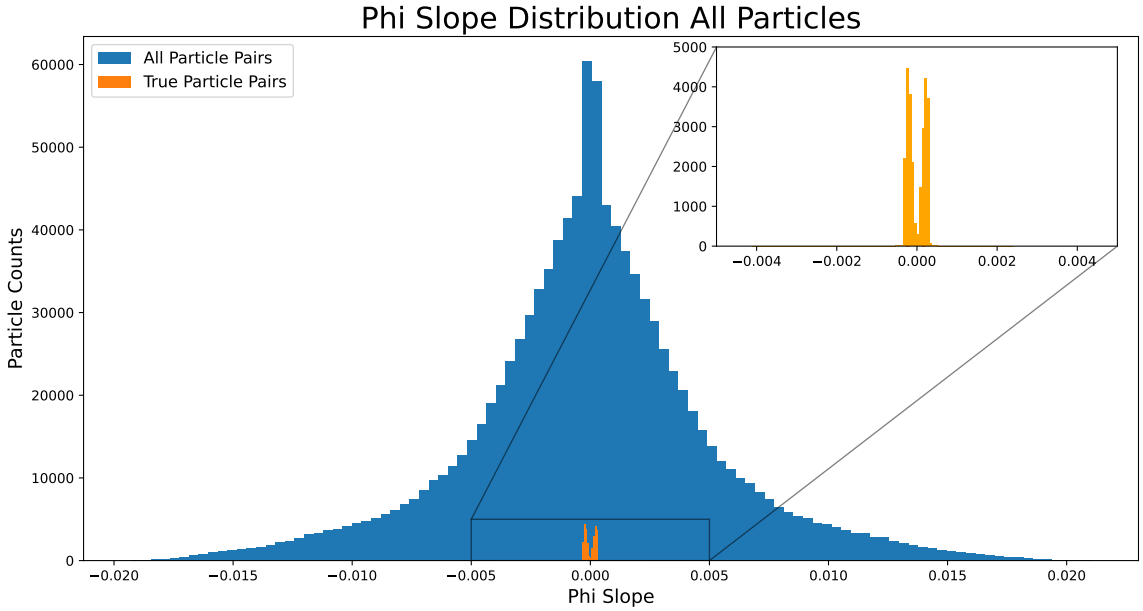


Figure 11: ϕ_{slope} distribution all data.

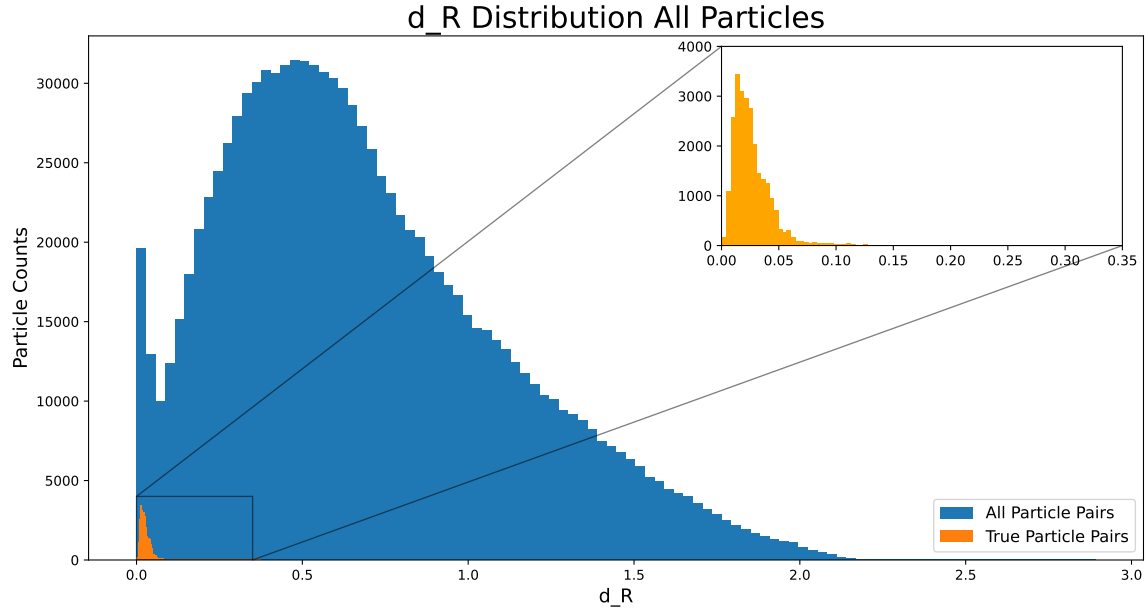


Figure 12: dR distribution all data.

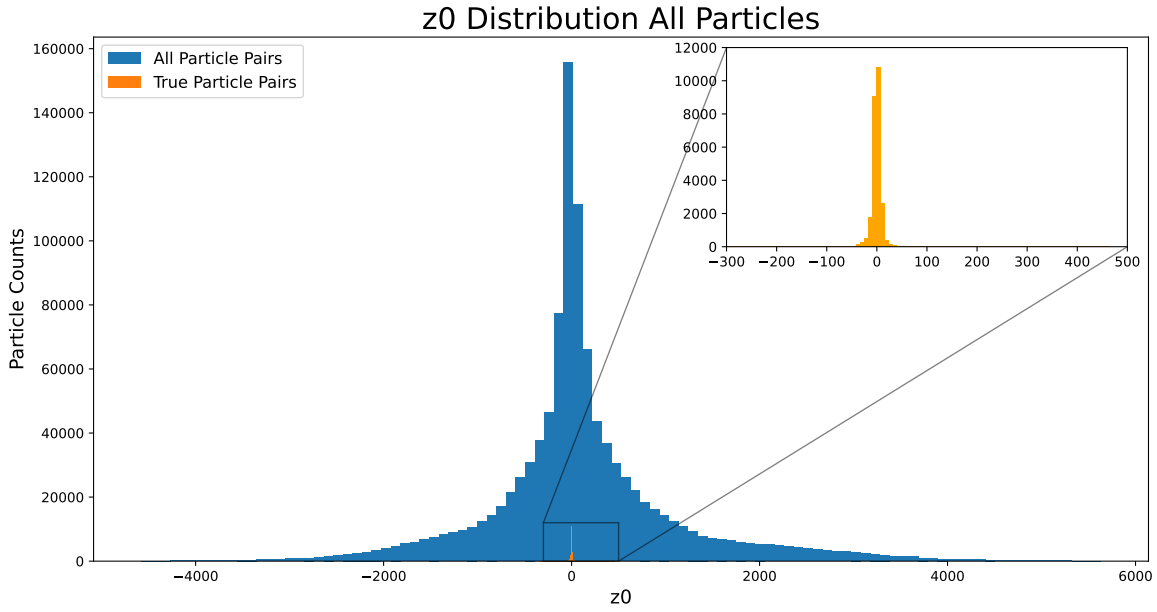


Figure 13: $z0$ distribution all data.

By analysing the distributions the possible edges are restricted to $z0 \in [-320, 520]$, $dR \in [0, 0.5]$, and $\phi_{slope} \in [-0.0055, .0055]$. Even though this ranges produce more false connection than necessary for some of the sections in an event, I concluded that this was the best approach. If more precise ranges were applied, the ratio of possible connections to not possible connections is highly inbalanced towards the true connections in some occasions, and vice versa in other occasions. Therefore by using the constant ranges, the ratio is always inbalanced towards the false connections. This issue will be fixed in Section 2.4.

After performing this connection cuts the amount of possible edged reduces to an average of $15,000 \pm 5000$ per section per event. An example of this reduction can be seen in figure 14, where the red dots

represent the hits that can have a connection, and the green hits represent the true particle track, using the section II of the detector.

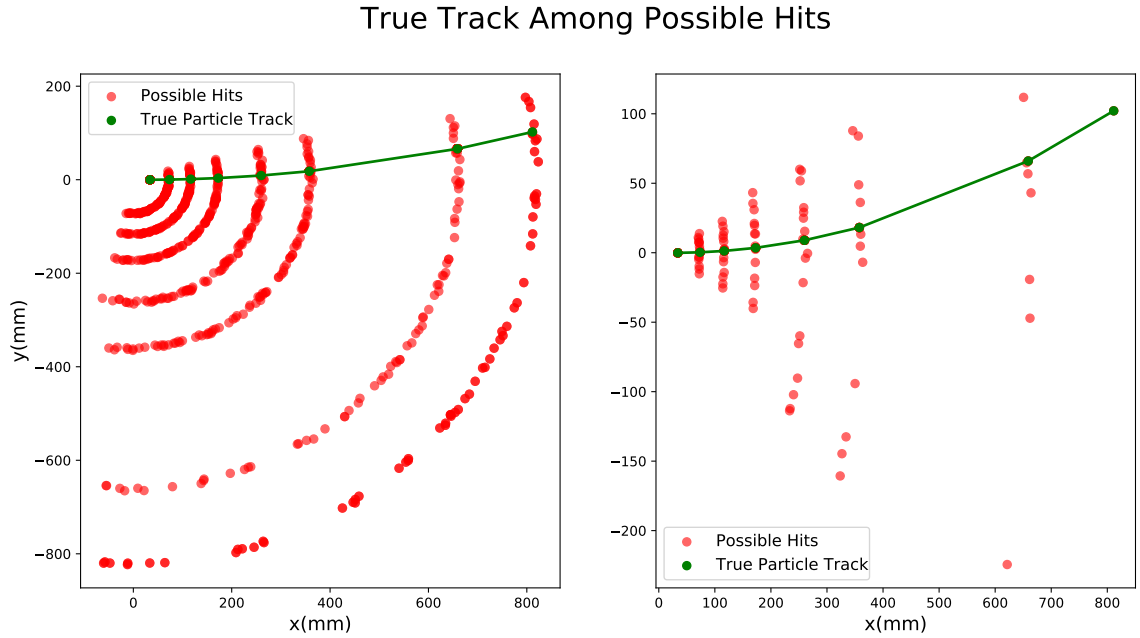


Figure 14: Left: True track among possible hits before filters. Right: True track among possible hits after filters.

2.4 Inbalanced Data Set

As mentioned in Section 2.3 the data set is highly inbalanced. In average there are $14,000 \pm 5,000$ false edges and $1,100 \pm 300$ true edges per section per event as seen in figure 15.

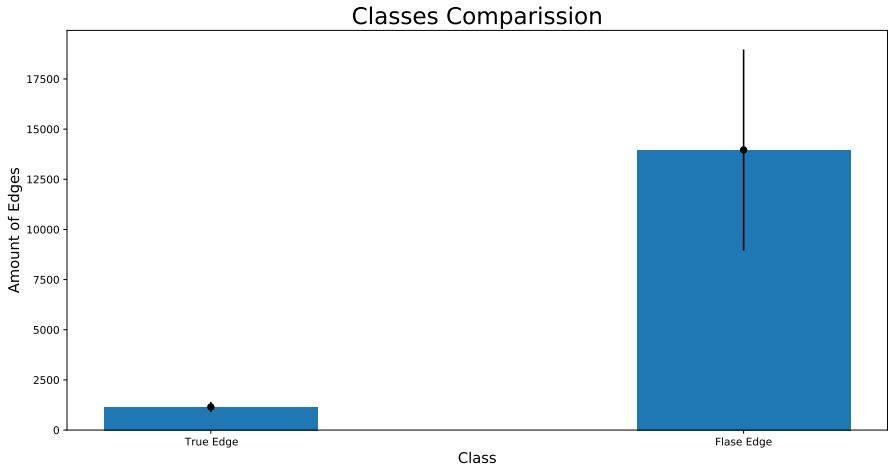


Figure 15: Ratio of true and false edges, inbalanced data set.

The ratio between true edges and false edges is 0.08 ± 0.03, meaning that $\approx 92\%$ of the data is a false edge. An inbalanced data set is a common machine learning problem [14]. This raise a problem in this case because, any machine learning model predicting a 100% of false edges, would get an accuracy of $\approx 92\%$, a model with such a high accuracy is consider to be a good model, but is not what I want, as I need a model that predict true edges. Therefore to train the GNN correctly it is necessary to balance the data, this was done using RandomUnderSampler [15] function from python

imblearn library. RandomUnderSampler function randomly delete some of the false edges until the data is balanced as shown in figure 16.

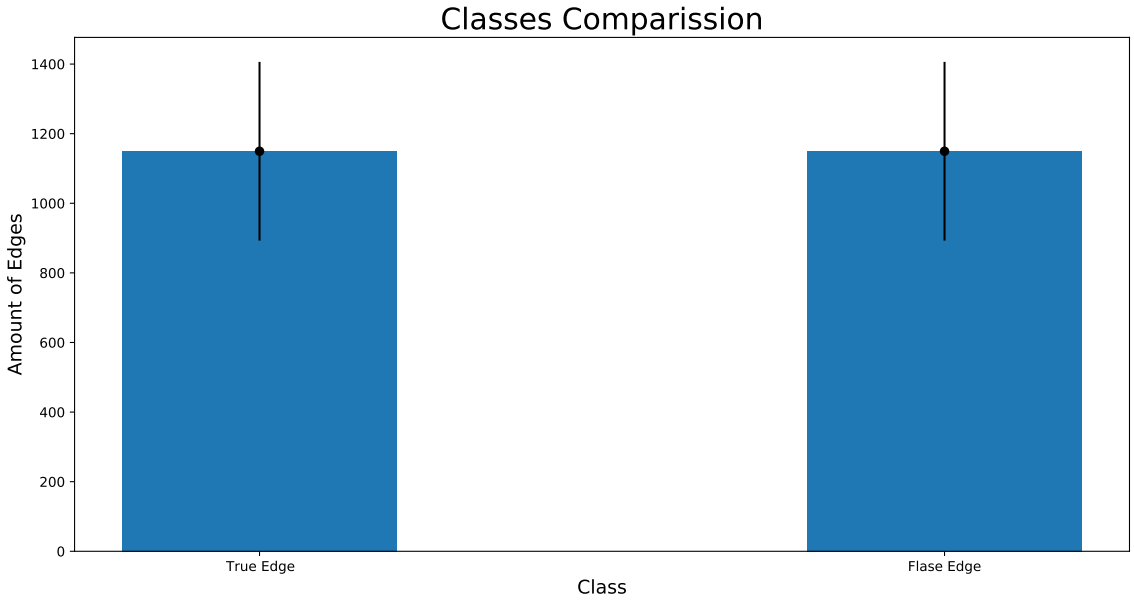


Figure 16: Ratio of true and false edges, balanced data set.

In addition to this, I relabeled hits ids in order from 0 to the amount of hits remaining. After passing the data trough the previous filters, many of the hits ids were removed. For anyone looking at the hits ids in this stage, they would see random numbers. Where as before the filters, the hit ids were a continuous set of numbers between 0 and max(hit id). The relabeling of the hits ids transform the data back to a continuous set of numbers. The hits ids as a continuous set is very important part of a GNN from Pytorch Geometric model [16], as each hit id represent an index of the node features information. If for example the max(hit id) was present as an index, the node feature information at index max(hit id) would be out of bounds.

2.5 Data in Graph Format

By obtaining the possible edges, the data can be changed to a graph format.
The node features are r, ϕ, η, x, y, and z. An edge list is a $2 \times N_{edges}$ matrix containing the hits id where is possible to have an edge. The edge features are $d\phi, dz, dr, d\eta, dR, \phi_{slope}$ and $z0$.
Finally I obtained the target vector y_{target} using the truth file, assigning a 1 where there is an edge in the edge list and a 0 where there is no edge.

3 Graph Neural Network

GNN's are a class of neural network methods designed to perform inference on data described by graphs [17]. The key design element of GNNs is the use of message passing, such that graph nodes iteratively update their representations by exchanging information with their neighbors. Since their inception, several different GNN architectures have been proposed [18],[19], each of them implementing different methods of message passing [20]. In the following sections a description of neural networks, message passing and the chosen architecture for this project will be described.

3.1 Neural Networks

First proposed in 1944 by Warren McCullough and Walter Pitts, neural networks are a means of doing machine learning, in which a computer learns to perform some task by analyzing training examples [21]. Generally neural networks models consist of layers of neurons densely interconnected. Each neuron might be connected to several neurons in the previous layer, from which it receives data, and several neurons in the next layer, to which it sends data. A diagram of a neural network can be seen in figure 17.

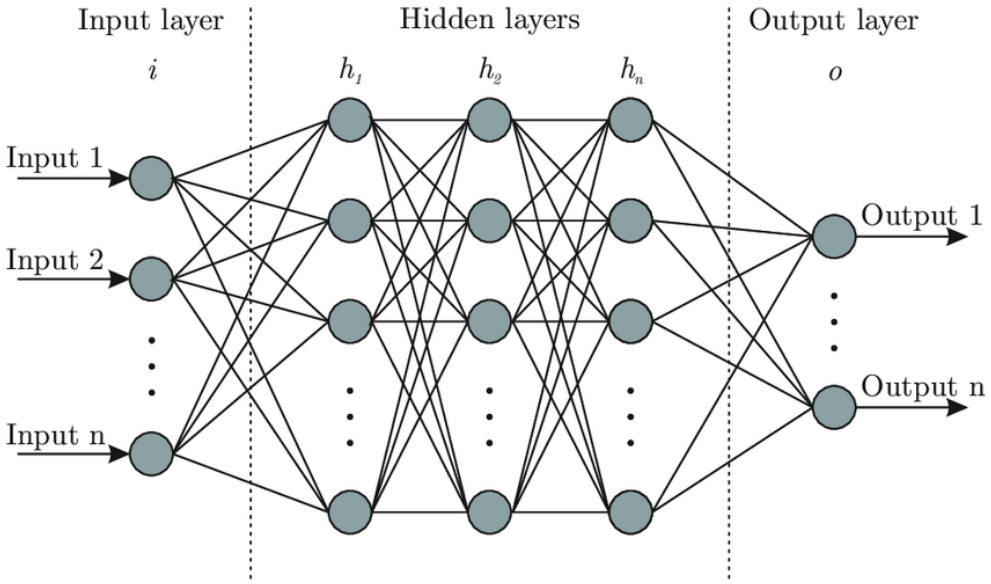


Figure 17: Neural Network [22].

Neurons have a weight w and a bias b that is adjusted during the learning process. The input data is multiplied by the weight and added to the bias. All incoming data for each neuron then is aggregated and passes through an activation function (AF) f, usual AF are ReLU, max(0,z), sigmoid, etc. The mathematical representation of the output of a neuron then becomes:

$$f(b + \sum_{i=1}^{n} x_i w_i) \tag{11}$$

At the beginning the weights and the bias get a random value, which is then adjusted by the process of back propagation. At a high level, back propagation is an algorithm that computes the gradient of weights and bias with respect to a loss function. Then an optimizer is used to find the optimal values of weights and bias in order to minimize the loss function. This process is repeated during several iterations (called epochs) until the minimum is found. A simple illustration of the back propagation is shown in figure 18.

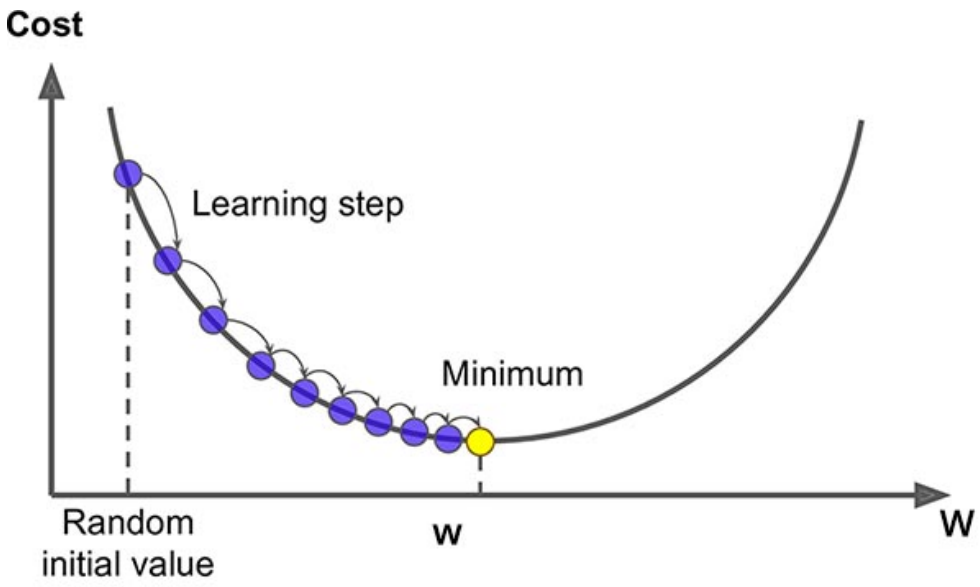


Figure 18: Gradient descend through back propagation [23].

3.2 Message Passing

In the graph, some of the nodes are interconnected with each other. Therefore the nodes information is not independent, and is necessary to obtain more information about their structure and properties. One neural network algorithm that is designed for this type of data is the message passing. A detail description of the message passing algorithm can be found in the following papers [24],[25],[26], and it can be summarized in tree steps.

First each node in the graph computes a message for each of its neighbours (nodes that have edges with), this message is a function (Φ) of the nodes features itself (X_i), the neighbour node features (X_j), and the edge features (E).

Seconly the messages are sent and each node aggregates the received message using a permutation invariant function ($\square$).

Finally each node gets updated (X_i') as a function (γ) of X_i and the aggregated message. Mathematically is possible to describe the message passing algorithm as follows:

$$X_i' = \gamma(X_i, \square_{\in N(i)} \Phi(X_i, X_j, E_{ji}))[26] \tag{12}$$

where Φ is the message function, $\square_{\in N(i)}$ is the aggregation function usually a sum, or an average, and γ is the update function.

3.3 GNN Architecture

The inspiration for the architecture of the GNN was taken from the paper [27] by G. DeZoort Et al. It consist of three neural networks R_1, R_2, and O with the following paramteres:

GNN Layers Architecture				
NN	AF	Input size	Hidden size	Output size
R_1	ReLU	10	19	4
O	ReLU	10	19	3
R_2	ReLU	19	19	1

Table 6: Initial filter.

A ReLu activation function applies the rectified linear unit function element-wise [28]:

$$ReLu(x) = max(0, x) \tag{13}$$

R_1 and O are part of the message passing algorithm. The input layer of R_1 has $X_i + X_j + E$ neurons. A forward pass in R_1 is the first stage of the message passing, the message computing.
The second stage, the aggregation, uses the output of R_1 and apply an addition.
Finally the last stage of message passing, the update, is a forward pass in O, where the input layer is the aggregated vector and the node features X.
The updated node features X' is the input of R_2. R_2 is a familiar neural network which takes X'_i, X'_j and E pass it trough the hidden layer and output 1 value through a final Sigmoid AF.
The sigmoid function is an element-wise operation that squishes any real number into a range between 0 and 1. This is a very common activation function to use as the last layer of binary classifiers because it lets you treat model predictions like probabilities that their outputs are true, i.e. $p(y = 1)$ [29].

$$sigmoid = \frac{1}{1 + exp^{-x}} \tag{14}$$

3.4 Training

The training parameters of the GNN are, an adaptive moment estimator (ADAM) optimizer, a variable learning rate of $lr = 0.005$, such that the lr decreases every 10 epochs by 10%, and a binary cross entropy as loss function given by:

$$H = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i)) \tag{15}$$

Where y is the class (true or false), and $p(y_i)$ is the probability of the edge to be true for all nodes N.
The data was divided into 60% for training, 10% for validation and, 30% for testing. The model was trained for 200 epochs. I trained 9 models, 1 for each section, and a master model that includes all the sections. The results of the training for every section can be seen in figures 19 and 20.

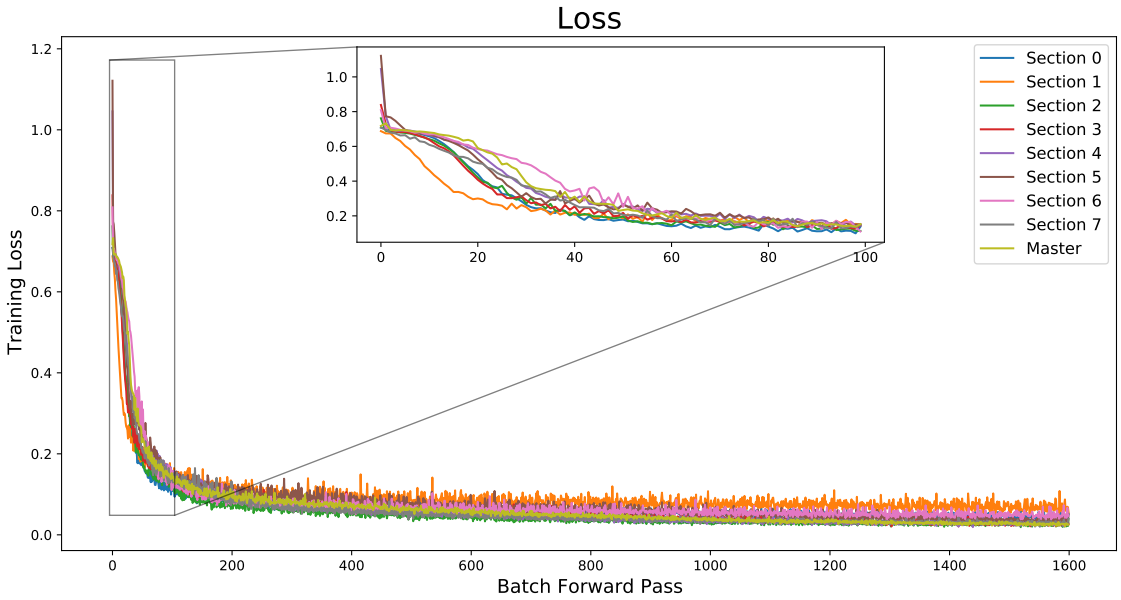


Figure 19: Loss during training per section.

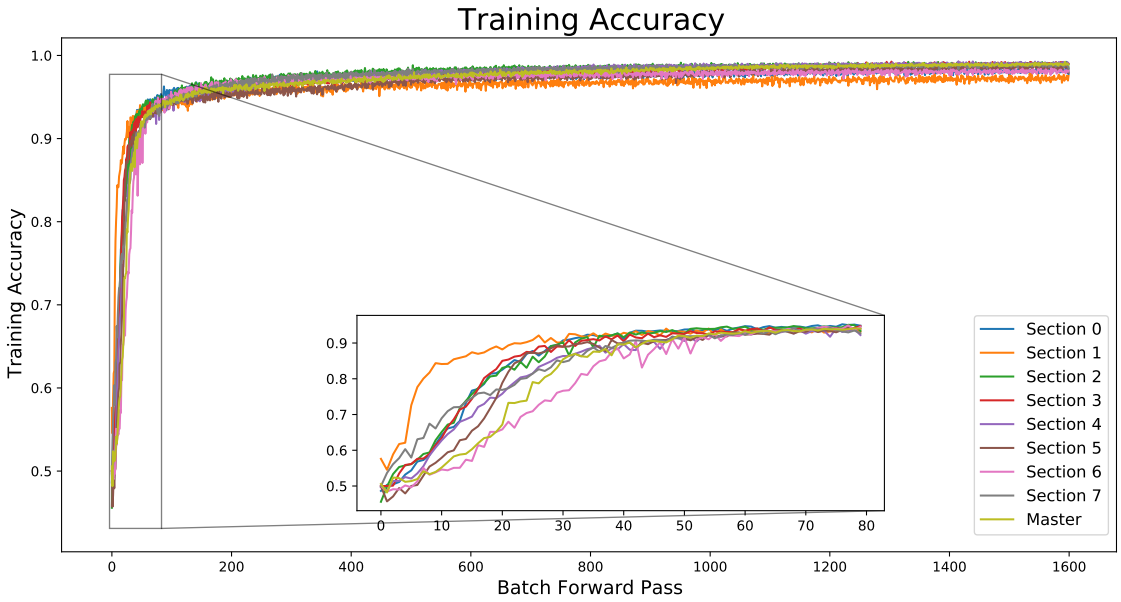


Figure 20: Accuracy during training per section.

As seen in figures 19 and 20 all models converge, but the master model seam to have bigger accuracy and smaller loss than the sections models, a further discussion of this is done in section 4.

3.5 Track Reconstruction

After the GNN is trained it outputs the probability of two hits are connected, this only produce segments of the whole track, therefore the last step to obtain a full track reconstruction is to join the segments where the hits ids match, this was done using a custom implementation of a depth first algorithm.

4 Results and Comparison

4.1 GNN Results

4.1.1 Master vs Section Models

As the project focus in comparing the best GNN models with the rest of the machine learning models, I decided to analyse the performance of the sections models and the master model. The analysis was done using the accuracy on predicting the segments, and the time each model take to make a prediction per section per event. The segment accuracy is given by:

$$SegmentAccuracy = \frac{True_{PS} + True_{NS}}{True_{PS} + True_{NS} + False_{PS} + False_{NS}} \tag{16}$$

Where $True_{PS}$ is the amount of correctly predicted real segments, $True_{NS}$, is the amount of correctly predicted fake segments, $False_{PS}$ is the amount of incorrectly predicted real segment, and $False_{NS}$ is the amount of incorrectly predicted fake segment. The segment accuracy in a few words is the ratio of correct predictions over the total predictions, if the model would predict everything correct, the ratio would be 1. The distribution of both the accuracy and time, for the sections models and the master model can be seen in figure 21.

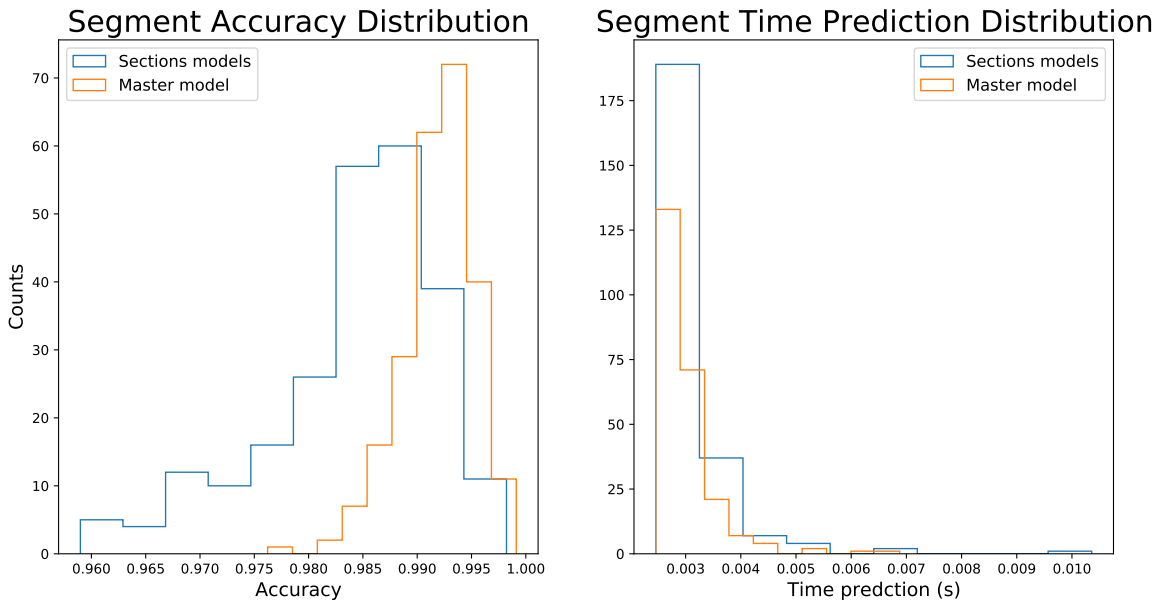


Figure 21: Left, segment accuracy. Right, time taken to predict segments.

In addition to this the mean accuracy and the mean time can bee seen in table 7.

GNN Models		
Model	Segment Accuracy	Segment Prediction Time (s)
Sections	0.984 ± 0.008	0.0031 ± 0.0007
Master	0.992 ± 0.003	0.0030 ± 0.0005

Table 7: Models Performance.

In order to analyse both of the distributions, first it was necessary to perform a test to see if they are statistically different from each other. I performed two sample Kolmogorov-Smirnov test [30], with a null hypothesis that the two distributions are identical. Results are shown in table 8.

Kolmogorov-Smirnov Results	
Distribution	P-value
Accuracy	6.9×10^{-36}
Time	0.27

Table 8: Kolmogorov-Smirnov Test .

By analysing results in table 8, is possible to reject the null hypothesis for the accuracy distribution, and there is not enough evidence to reject the null hypothesis for the time distribution. Therefore I decided to move forward with the master model as the accuracy is approximately 0.8% better, and the times are statistically same based on the analysis.

4.1.2 Time Performance

As mentioned in section 1.1, the HL-LHC will increase the luminosity up to 10 times, producing more than 60 million mega bytes per second. One of the challenges is to reconstruct the tracks as

fast as possible, to be precise the trigger system has a latency of approximately $4\mu s$ [31] .Therefore an analysis of the time it takes to process an event using the previously described method is needed. The algorithm was run on a Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz processor. With this processor, the time per process per event in a section is displayed in table 9.

Time Performance Per Event in a Section	
Process	Time (s)
Initial filters	0.0053 ± 0.0008
Calculation filters	0.0080 ± 0.0002
Edge connections	6 ± 2
Edge filters	0.015 ± 0.005
Graph data	0.015 ± 0.007
Segment prediction	0.0030 ± 0.0005
Track Reconstruction	0.23 ± 0.09
Total Track Prediction	6.276 ± 2.193

Table 9: Time Performance.

As seen in table 9, the prediction time is in the order of $O(6)$ greater than the target, this result was expected, because in order to achieve the time requirements the use of specialised hardware, and paralelization of the processes is needed. For completeness I measure the time to train the GNN master model as 228 s, making the time to complete the whole process run in $(234 \pm 2)s$.

4.1.3 Accuracy

In addition to reconstruct particle tracks with a time constrain, it is necessary to predict the most tracks as possible, in order to get a quantitative analysis on the performance of the neural networks, I used two measurement, the Segment accuracy and Track accuracy.
The track accuracy is necessary due to the fact, that the information I have is only the amount of correctly predicted tracks, incorrectly predicted tracks, and the true amount of real tracks. Therefore I calculated the ratio of the total correctly predicted tracks over the total real tracks given by:

$$TrackAccuracy = \frac{Predicted_T}{True_T} \tag{17}$$

where $Predicted_T$ is the amount of correctly predicted tracks, and $True_T$ is the amount of real tracks in the data set.
The results are shown in table 10.

GNN Accuracy	
Segment Accuracy	Track Accuracy
0.992 ± 0.003	0.97 ± 0.02

Table 10: Accuracy Performance.

Comparing the segment accuracy of the GNN master model with the results by G. DeZoort Et al. [27] 0.997 ± 0.001, it is possible to see that the results obtained by the GNN by G. DeZoort Et al. has 0.005 better accuracy, this result is not ideal, but the GNN master model is within 2σ of this result. In addition to this is possible to see that the track accuracy decrease from the segment accuracy, this is expected as the tracks with more segments tend to decrease in accuracy, due to the fact that segment accuracy is not perfect. A detailed track accuracy by number of hits is shown in table 11, where is possible to see the decline in accuracy with the increase of hits.

GNN Accuracy	
Hits	Track Accuracy
5	0.98 ± 0.01
6	0.98 ± 0.01
7	0.97 ± 0.02
8	0.94 ± 0.03

Table 11: Accuracy Performance.

An example of the predicted tracks and the true tracks can be seen in figure 22.

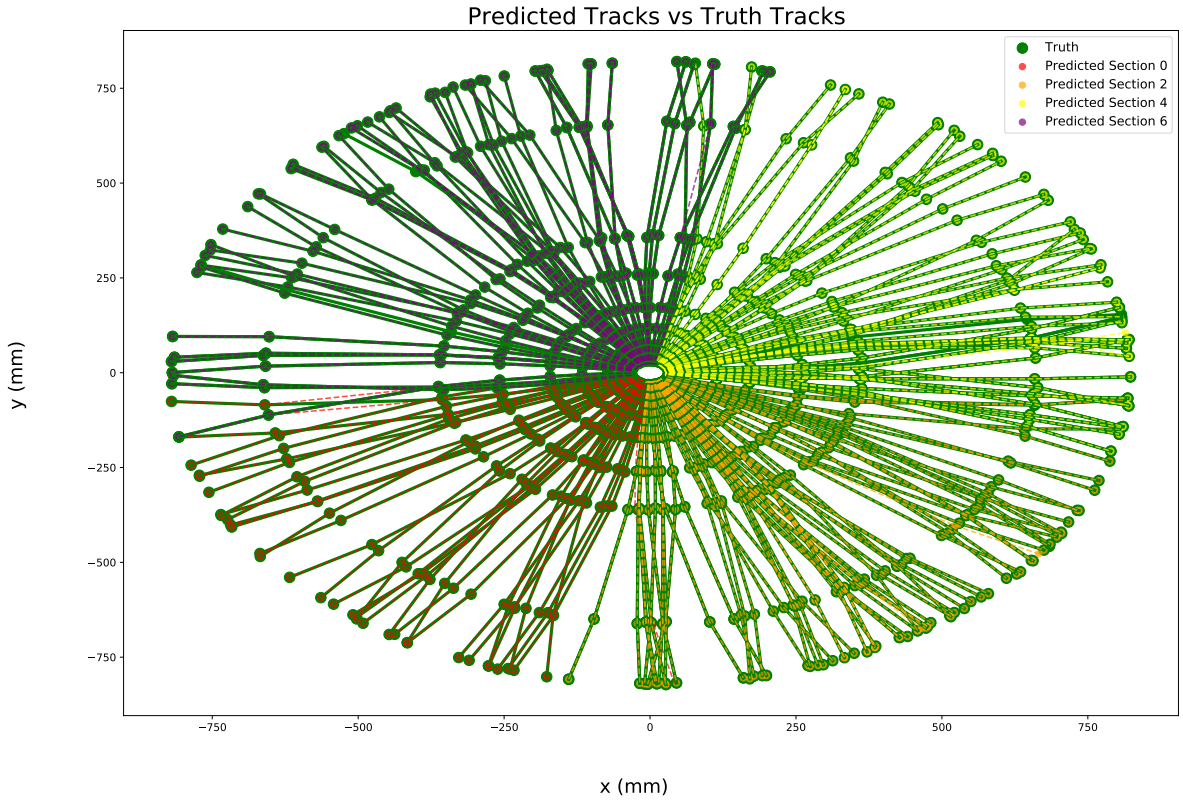


Figure 22: 4 sections plot of true tracks vs predicted tracks.

By analysing figure 22 is possible to see that the track accuracy matches the results in table 10.
In the following sections an accuracy comparison between a XGB, a MLP, and a CNN was performed.

4.2 XGB Classifier vs GNN

XGB, is a decision tree based ensemble machine learning algorithm that uses a gradient boost framework [32]. The XGB was chosen for its simplicity and power. As is very easy to create such a model, I used it as a fist test to pass the data in graph format trough a machine learning model. The XGBClassifier was trained using the default parameters [33], and root mean square error as evaluation metric. The results of the XGB model can be seen in table 12.

XGB Performance		
Segment Accuracy	Track Accuracy	Total Track Prediction (s)
0.9897 ± 0.0006	0.967 ± 0.003	6.4 ± 2.1

Table 12: XGB Performance.

Comparing the segment prediction of the XGB with the GNN, is possible to see that the GNN has 0.003 better accuracy than the XGB model, i.e. the XGB model accuracy is more than 5σ away from the GNN accuracy. similarly than for the GNN the track accuracy decline as expected. In addition to this the XGB model takes approximately 50 times longer to make a prediction. Therefore the GNN is the best of this 2 models.

4.3 MLP Classifier vs GNN

A MLP classifier is a machine learning algorithm, that has neurons stacked in multiple layers, with every neuron connected to all other neurons in next layer [34]. The MLP model was chosen as this project was done in parallel with the thesis by S. Johannsen and N. Søderberg [35], therefore a comparison with this model was a natural decision. The MLP was build uisng 2 hidden layers with 15 neurons each, and run for 200 epochs. Results of the MLP model can be seen in table 13.

MLP Performance		
Segment Accuracy	Track Accuracy	Total Track Prediction (s)
0.985 ± 0.002	0.965 ± 0.001	6.29 ± 2.02

Table 13: MLP Performance.

Comparing the segment prediction form the MLP with the one from the GNN is possible to see that the GNN has a 0.007 better accuracy than the MLP model, i.e. the MLP model accuracy is more than 3 σ away from the GNN accuracy. The lower segment accuracy produces an even lower track accuracy as expected. In addition to this the model took approximately 10 times longer to make a segment prediction. This result confirms that the GNN is a superior model than the MLP.

4.4 Comparison Summary

After obtaining the previous results is possible to make a concise summary of the accuracy and times of the previous models. The summary can be seen in table 14.

Result Comparison			
Model	Track Accuracy	Segment Accuracy	Segment prediction (s)
GNN	0.97 ± 0.02	0.992 ± 0.003	0.0030 ± 0.0005
XGB	0.967 ± 0.003	0.9897 ± 0.0006	0.14 ± 0.01
MLP	0.965 ± 0.001	0.985 ± 0.002	0.03 ± 0.01

Table 14: Models Results.

After analysing results in table 14 is possible to see that the GNN model is the best model from the previous 3, in both accuracy and prediction time.
The CNN model predicts whole particle tracks, and the data structure is completely different, therefore a different comparison is needed as seen in section 4.5.

4.5 CNN vs GNN

A CNN is a machine learning algorithm that can take an input image, and detect features in the image. The CNN comparison was done due to the fact, that a CNN is one of the possible candidate models fot the HL-LHC trigger system, therefore a comparison between models is necessary in order to choose the most efficient and accurate model.
The nature of the CNN raise a problem, as the given data only includes coordinates and images are needed. Therefore is necessary to transform the coordinates features into images, this can be done using Hough transform.

4.5.1 Hough transform

The Hough transform is an algorithm designed to detect edges on an image, using this transformation is possible to transform point hits into lines, the overlapping of these lines indicates a particle. In order to Hough transform the coordinates is necessary to find the Hough transform equation as follows.
In the ATLAS detector particles move in an electromagnetic filed according to the Lorentz force:

$$\mathbf{F} = q(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \tag{18}$$

Where q is the particle charge, $\mathbf{v}$, the velocity of the particle, $\mathbf{B}$ is the magnetic field and $\mathbf{E}$ the electric field. In this case $\mathbf{E} \approx 0$, there is a constant magnetic field $\mathbf{B}$ along z direction, and the velocity coordinate is along ϕ, resulting in:

$$\mathbf{F} = qBv\hat{r} \tag{19}$$

The particles follow a circular path, and the equation that describe the force in a circular path is:

$$\mathbf{F} = \frac{p_t v}{r}\hat{r} \tag{20}$$

Combining equations 19 and 20:

$$\frac{qB}{p_t} = \frac{1}{r} \tag{21}$$

After some manipulation [36],[37] it is possible to transform equation 21 into:

$$\frac{qA}{\rho_t} = \frac{sin(\phi_0 - \phi)}{r} \tag{22}$$

Where A is a constant $A = 3 \times 10^{-4} GeV/mm/c/e$ that includes the constant magnetic field $B = 2T$ in the ATLAS. As the areas of interest are regions where ϕ_0 is really small is possible to use the small angle approximation, obtaining the final equation:

$$\frac{qA}{\rho_t} \approx \frac{\phi_0 - \phi}{r} \tag{23}$$

In order to generate the images I used a range of ϕ_0 [-1,4], and calculate a vector $\frac{qA}{\rho_t}$ with equation 23, where r, and ϕ are the hit coordinates from the predicted tracks obtained with the GNN. In this way a point hit with coordinates r and ϕ, is transformed into a line with coordinates ϕ_0 and $\frac{qA}{\rho_t}$. The CNN takes an image of 36×36, therefore each of the values of ϕ_0 $\frac{qA}{\rho_t}$ are binned into 36 equally spaced bins. An example of the transformation is shown in figure 23.

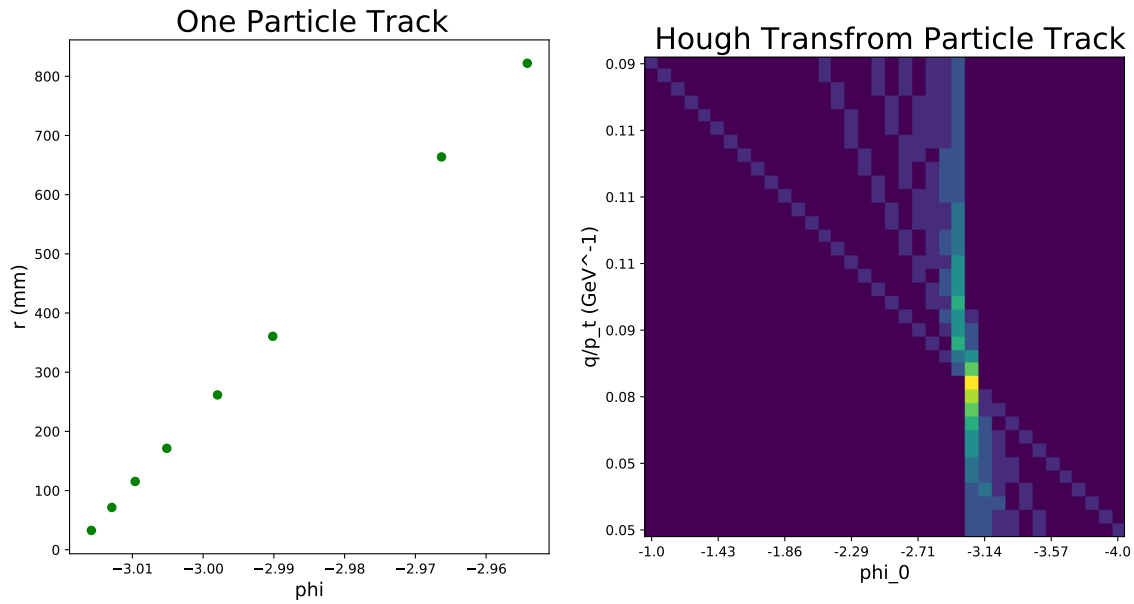


Figure 23: Left: ϕ and r space, 8 hit particle track. Right: Hough transform 8 lines particle track.

Figure 23 shows an isolated particle track, however this is not the case in an event, as there is noise from other particle tracks, as well as noise from the detector. Therefore I added some noise to the images, in order to create a more realistic scenario. I analyzed both the single particle tracks and the ones with noise.

4.5.2 CNN Results Comparison

In order to make predictions I used a pre trained CNN created by M. M. Mødekjær [36], which was optimized to recognize particle tracks with 6 to 8 hits. As the format of the CNN is different from the previous three cases, I made a comparison of the track accuracy for particles with 6, 7 and 8 hits, detected by the CNN, with the ones with the same hits detected by the GNN, results are shown in table 15.

CNN Performance			
Number of Hits	Track Accuracy CNN	Track Accuracy CNN (noise)	Track Accuracy GNN
6	0.43 ± 0.05	0.3 ± 0.1	0.98 ± 0.01
7	0.4 ± 0.1	0.3 ± 0.1	0.97 ± 0.02
8	0.4 ± 0.2	0.32 ± 0.08	0.94 ± 0.03

Table 15: CNN Performance.

As seen in table 15 the track accuracy of the CNN with noise performs worse than the track accuracy of the CNN whit-out noise. However for the three cases track accuracy of the CNN with noise is within 1σ from track accuracy of the CNN with-out noise. In addition to this the CNN performance without noise is approximately 57% to 59% worse than the GNN, and the the CNN performance with noise is approximately 66% to 70% worse than the GNN.
The time performance of the CNN is shown in table 16.

CNN Time Performance		
Number of Hits	Time Prediction (s)	Time Hough Transform (s)
6	0.54 ± 0.02	0.34 ± 0.02
7	0.57 ± 0.05	0.29 ± 0.04
8	0.50 ± 0.04	0.27 ± 0.03
Total	1.61 ± 0.07	0.90 ± 0.05

Table 16: CNN Time Performance.

The total time to make a prediction in an section per event with 6 , 7 and 8 hits with the given CNN is $(2.51 \pm 0.09)s$. Comparing the CNN time with the GNN time $(6 \pm 2)s$, the GNN time is within 2σ from the CNN time, however the CNN predication time is 2.4 times faster making the CNN process the fastest of all.

5 Future Work and Conclusions

5.1 Conclusions

5.1.1 Concusions on Performance

As shown in tables 14 and 15, the GNN model performs really well, comparing with the XGB, the MLP, and the CNN, with a segment accuracy of 0.992 ± 0.003 and track accuracy of 0.97 ± 0.02, being th best of all the models. However as mentioned in section 4.1.3 previous work show better accuracy results [27], therefore more work is needed in order to improve the accuracy of the model. The GNN, XGB and MLP have very similar results with maximum 5σ difference between them. On the other hand as seen in table 15, the CNN performs quite poorly for the pure tracks and, even worse for the tracks with noise. I conclude that this poor results, were due to the fact that the training of the CNN was done with approximately 100 images only, and using a different data set. The use of few training images seam to have caused an overfiting of the model, and even though the given data was transformed into the right format (Hough transform 36×36matrix, with 8 to 6 hits), the model was unable to correctly predict a particle in the images.

5.1.2 Concusions on Latency

As seen in table 14 and in section 4.5.2 all the processing times are far from the $4\mu s$ [31] target, with all of the times being in the order $O(6)$ bigger than the target.
The GNN, XGB, and MLP share the same pipeline with the exception of the prediction section, Therefore is possible to identify the parts of the pipelines with the highest latency. As seen in table 9 this latency is caused mainly by 2 processes, the edge connection and the track reconstruction. The edge connection part takes 95.6% of the total prediction time and the track reconstruction part takes 3.6%, together this 2 progresses take 99.2% of the total time. This result was expected as the combinatorics problem mentioned in section 1 and 2 was evident from the begging of the project, and this problem was the reason for the all the filters pipelines. However I propose some solutions for this problem in section 5.2.
Comparing the 4 models latency is possible to see that the CNN model was the best with a total track reconstruction time of $(2.51 \pm 0.09)s$, this is due to the fact that the process to predict a track using a CNN is much simpler, consisting of only 2 parts the Hough transformation and the prediction through the CNN.

5.1.3 Final Conclusions

As a final conclusion I can say that the GNN was the best model of the 4 tested in this project, with a segment accuracy of 0.992 ± 0.003, a track accuracy of 0.97 ± 0.02, and a total track prediction time of (6.276 ± 2.193)s, therefore it should be consider to perform further work on it, in particular to improve the latency, in the edge connection and track reconstruction parts. Additionally, as the CNN shows a much better performance time wise, and the work of M. M. Mødekjær [36], shows an accuracy of 0.872 ± 0.008, the CNN model should be explore in more detail.

5.2 Future Work

5.2.1 Sections

An optimal way to split the detector into sections needs to be found. Therefore a careful study of the sectioning needs to be done, in order to find the optimal section area, the amount of sections

needed, and the overlap, in such a way that reduces the processing time the most, and preserve most of the full tracks intact.

5.2.2 Initial Filters

Removing noise, or better say distinguishing noise, is an important part of the pipeline, therefore a method that distinguishes noise and removes it is necessary, this could be done with a simple neural network like the XGB. As mention in section 2.2.2 removing the double hits was necessary to reduce the amount of data, therefore a double hit filter needs to be implemented as well. One of the options here would be to use a similar GNN model as the one used, but creating edges within the layers.

5.2.3 Calculation Filters

As seen in section 2.2.3 the p_t filter is the most important. A test with the whole range of p_t is needed in order to see how the GNN performs. If the results are not satisfactory a GNN with the whole range of p_t needs to be trained.

5.2.4 Node Edge Connections

As mentioned in the conclusions there is a need to explore ways to accelerate the track reconstruction process, specially the edge connection process. In this project, the edge connection was performed using the true data, and creating an edge between layers where a connection exist. Using this method it was possible to connect hits that didn't have a hit in an adjacent layer, i.e. layer 1 connected with layer 3. Even though this method is the best method to train the GNN, in a real experiment this missing layers hits information wont be available, as is not possible to know in advance if a particle will skip a layer. Therefore a method to find hits that miss a layer needs to be found, or as an alternative, only edges between adjacent layers will be consider. The latter method will of course produce broken tracks, however broken tracks can be connected using a χ^2 fit, or, if something more powerful is needed, an edge prediction GNN [38] can be used.
The use of such edge connection algorithm will reduce the time of this process by itself, however more sophisticated algorithms can be explored. One option can be a reinforcement learning neural network, that produce a range on the adjacent layer, then the hit will only have possible edges within this range, reducing the time creating edges. Another method can be to study the module information. Modules briefly mentioned in section 1.2.1 are sections within the layers. A particle hit in certain layer module, can only have a hit in a subset of all the modules in the adjacent layer. Therefore a study of the module layout configuration can be done in order to reduce possible edge combinations.
In addition to this edge connection algorithms, this process (as well as others) can be parallelized, and run in hardware accelerators, such as GPU's or FPGA's, as done in [39], [40], [41], [42] and [43], to mention a few. This parallelization needs to be done no matter what, as this is the only way to achieve the times required for the HL-LHC.

5.2.5 GNN

Even though the GNN segment accuracy was 0.992 ± 0.003, it is necessity to get a better performance, therefore further hyper parameter tuning can be done. The hyper parameters that can be tuned to obtain a better performance are:

- The neurons in the hidden layers of R_1, R_2, and O.
- The optimizer.
- Learning rate.

- The learning rate decrease by epoch.
- Training epochs.

The previously mentioned method will improve the accuracy, but a method to reduce the latency besides paralelization needs to be explored. Such a method can be to reduce the input parameters of the GNN. As mentioned in section 2.5, I used the node features and edge features as input parameters. In order to understand the importance of these features, a feature importance analysis [44] needs to be done. If is possible to remove some of the features without affecting the accuracy, the latency will be reduce.
Additionally, I recommend that the results of the models are confirmed by a different researcher, as the possibility of overfiting, or an unnoticed bug is there, until confirmed otherwise.

5.2.6 Track Reconstruction

The current track reconstruction algorithm similarly than the edge connections algorithm, takes into account missing layers hits, making it not as efficient as possible. Same methods as mention in section 5.2.3 can be implemented to improve the efficiency, with the risk of producing broken tracks. Furthermore the most efficient way to create tracks needs to be found, as I believe there can be better ways than the one implemented. One of these methods can be use a union-find algorithm [45]. The final improvement needed, as mentioned before is the parallelization of this process.

References

[1] Physics. https://home.cern/science/physics. Accessed: 2022-06-22.

[2] The atlas experiment at cern. https://blogs.ucl.ac.uk/science/2014/07/28/picture-of-the-week-the-atlas-experiment-at-cern/. Accessed: 2022-06-22.

[3] High-Luminosity LHC. https://home.cern/science/accelerators/high-luminosity-lhc. Accessed: 2022-07-12.

[4] Luminosity?, Why don't we just say collision rate? https://home.cern/news/opinion/cern/luminosity-why-dont-we-just-say-collision-rate. Accessed: 2022-07-12.

[5] G Aad, M Ackers, FA Alberti, M Aleppo, G Alimonti, J Alonso, EC Anderssen, A Andreani, A Andreazza, JF Arguin, et al. Atlas pixel detector electronics and sensors. *Journal of instrumentation*, 3(07):P07007, 2008.

[6] The atlas experiment. https://atlas.cern/about. Accessed: 2022-06-22.

[7] The ATLAS Detector . https://atlas.cern/Discover/Detector. Accessed: 2022-07-12.

[8] Atlas high-level trigger, data-acquisition and controls : Technical design report. https://cds.cern.ch/record/616089. Accessed: 2022-06-22.

[9] Technical design report for the phase-ii upgrade of the atlas tdaq system. https://cds.cern.ch/record/2285584. Accessed: 2022-06-22.

[10] Trigger and Data Acquisition. https://atlas.cern/Discover/Detector/Trigger-DAQ. Accessed: 2022-07-12.

[11] TrackML Particle Tracking Challenge. https://www.kaggle.com/competitions/trackml-particle-identification. Accessed: 2022-07-12.

[12] Edward A Bender and S Gill Williamson. *Lists, decisions and graphs*. S. Gill Williamson, 2010.

[13] Don't understand graphs? here's why you should study graphs in computer science. https://bennettgarner.medium.com/what-the-graph-a-beginners-simple-intro-to-graphs-in-computer-science-3808d542a0e5. Accessed: 2022-06-22.

[14] A gentle introduction to imbalanced classification. https://machinelearningmastery.com/what-is-imbalanced-classification/. Accessed: 2022-06-22.

[15] Random under sampler. https://imbalanced-learn.org/stable/references/generated/imblearn.under$_sampl$ $2022 - 06 - 22$.

[16] Pyg documentation. https://pytorch-geometric.readthedocs.io/en/latest/. Accessed: 2022-06-22.

[17] Graph Neural Network and Some of GNN Applications: Everything You Need to Know. https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications. Accessed: 2022-07-16.

[18] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[19] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[20] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[21] Explained: Neural networks. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. Accessed: 2022-07-16.

[22] Facundo Bre, Juan M Gimenez, and Víctor D Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158:1429–1441, 2018.

[23] What is Gradient Descent in machine learning? https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/. Accessed: 2022-07-16.

[24] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

[25] Jiwei Li, Michel Galley, Chris Brockett, Georgios P Spithourakis, Jianfeng Gao, and Bill Dolan. A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*, 2016.

[26] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[27] Gage DeZoort, Savannah Thais, Javier Duarte, Vesal Razavimaleki, Markus Atkinson, Isobel Ojalvo, Mark Neubauer, and Peter Elmer. Charged particle tracking via edge-classifying interaction networks. *Computing and Software for Big Science*, 5(1):1–13, 2021.

[28] PyTorch, torch.nn relu. https://pytorch.org/docs/stable/generated/torch.nn.ReLU.htmltorch.nn.ReLU. Accessed: 2022-06-17.

[29] How to Use the PyTorch Sigmoid Operation, sparrow computing. https://sparrow.dev/pytorch-sigmoid/. Accessed: 2022-06-17.

[30] scipy stats ks 2samp. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks$_2$*samp.html.Ac* $2022-07-28$.

[31] Anders Ryd and Louise Skinnari. Tracking triggers for the hl-lhc. *arXiv preprint arXiv:2010.13557*, 2020.

[32] XGBoost Algorithm: Long May She Reign! https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d. Accessed: 2022-07-17.

[33] Xgboost. https://xgboost.readthedocs.io/en/stable/python/python$_i$*ntro.html*. *Accessed* : $2022-07-29$.

[34] Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis. https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141 . Accessed: 2022-07-21.

[35] Simon Johannsen Nicolai Søderberg. Use of neural networks for track finding at the high luminosity lhc with the atlas trigger system. 2022.

[36] Mikkel M. Mødekjær. Convolutional neural networks studies on fpga for the atlas detector trigger system for hl-lhc. 2022.

[37] Alessandro Gabrielli, Fabrizio Alfonsi, Alberto Annovi, Alessandra Camplani, and Alessandro Cerri. Hardware implementation study of particle tracking algorithm on fpgas. *Electronics*, 10(20):2546, 2021.

[38] Seal link prediction, explained. https://towardsdatascience.com/seal-link-prediction-explained-6237919fe575. Accessed: 2022-07-21.

[39] R Cenci, F Lazzari, P Marino, MJ Morello, G Punzi, LF Ristori, F Spinella, S Stracka, and J Walsh. Performance of a high-throughput tracking processor implemented on stratix-v fpga. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 936:344–345, 2019.

[40] Sioni Summers and Andrew Rose. Kalman filter track reconstruction on fpgas for acceleration of the high level trigger of the cms experiment at the hl-lhc. In *EPJ Web of Conferences*, volume 214, page 01003. EDP Sciences, 2019.

[41] Aneesh Heintz, Vesal Razavimaleki, Javier Duarte, Gage DeZoort, Isobel Ojalvo, Savannah Thais, Markus Atkinson, Mark Neubauer, Lindsey Gray, Sergo Jindariani, et al. Accelerated charged particle tracking with graph neural networks on fpgas. *arXiv preprint arXiv:2012.01563*, 2020.

[42] Xiaocong Ai, Georgiana Mania, Heather M Gray, Michael Kuhn, and Nicholas Styles. A gpu-based kalman filter for track fitting. *Computing and Software for Big Science*, 5(1):1–16, 2021.

[43] V Halyo, A Hunt, P Jindal, P LeGresley, and P Lujan. Gpu enhancement of the trigger to extend physics reach at the lhc. *Journal of Instrumentation*, 8(10):P10005, 2013.

[44] How to Calculate Feature Importance With Python. https://machinelearningmastery.com/calculate-feature-importance-with-python/. Accessed: 2022-07-29.

[45] Md Patwary, Mostofa Ali, Jean Blair, and Fredrik Manne. Experiments on union-find algorithms for the disjoint-set data structure. In *International Symposium on Experimental Algorithms*, pages 411–423. Springer, 2010.