University Of Milano-Bicocca

Physics Department

Master's degree in Physics

Master Thesis

Development and Applications of Graph Neural Networks for the Trigger of the LHCb Experiment

Supervisor:
Prof. Marta Calvi

Co-Supervisor:
Dr. Julián García Pardiñas

Candidate:
Martina Mozzanica

Student ID: **869374**

Academic Session 2021–2022

Abstract

This thesis presents a contribution to the development of a new machine-learning-based algorithm for improving the future Large Hadron Collider beauty (LHCb) trigger system. The LHCb experiment requires identifying interesting events during data taking based on the possible presence of signal beauty- or charm- hadron decays in the event. The future upgrades of the LHCb experiment will increase the instantaneous luminosity, leading to unprecedented challenges for the trigger system. The Deep Full Event Interpretation (DFEI) algorithm is designed to address this issue and uses deep neural networks to process information from all stable particles in each event. This thesis focuses on the development of the DFEI algorithm for b-hadron decays and expands the work by using the Geant4-based full LHCb simulation in DFEI for the first time and performing a detailed study of the DFEI performance on a selection of exclusive key signal modes in LHCb. The thesis describes the re-optimization and re-training of the DFEI algorithm on the full inclusive simulation and the use of several metrics for evaluating the performance of a classification model in extracting a signal from a larger data set.

Contents

Acronyms

AI Artificial Intelligence.

AUC Area Under the Curve.

BCE Balanced Cross Entropy.

BDT Boosted Decision Tree.

CE Cross Entropy.

DFEI Deep-learning based Full Event Interpretation.

ECAL Electromagnetic CALorimeter.

FDSP Fully Disconnected Signal Particles.

FL Focal Loss.

FPR False Positive Rate.

GEM Gas Electron Multipliers.

GN Graph Networks.

GNN Graph Neural Networks.

GPU Graphic Processing Unit.

HCAL Hadronic CALorimeter.

HLT High Level Trigger.

HPD Hybrid Photon Detectors.

IP Impact Parameter.

IT Inner Tracker.

KBE Killed Background Edges.

LCA Lowest Common Ancestor.

LHC Large Hadron Collider.

LHCb Large Hadron Collider beauty.

OP Origin Point.

OT Outer Tracker.

PDG Particle Data Group.

PER Perfect Event Reconstruction.

PMT PhotoMultiplier Tubes.

PS Pre-Shower Detector.

PSR Perfect Signal Reconstruction.

PV Primary Vertex.

RICH Ring Imaging Cherenkov.

ROC Receiver Operating Characteristic Curve.

SciFi Scintillating Fiber.

SGD Stochastic Gradient Descent.

SiPM Silicon PhotoMultiplier.

SM Standard Model.

SPD Scintillating Pad Detector.

SV Secondary Vertex.

TPR True Positive Rate.

TT Tracker Turicensis.

UT Upstream Tracker.

VELO VErtex LOcator.

WLS WaveLength Shifting.

Introduction

The Large Hadron Collider beauty (LHCb) experiment is one of the four primary experiments located along the LHC ring at CERN. This thesis describes contributions done to the development of a novel machine-learning-based algorithm aimed to improve the future LHCb trigger system.
To start, the thesis provides an overview of the physics of the b meson, from its production to its decay, based on the Standard Model. It then discusses the LHCb detector setup and its particle detection process, concluding with a description of the LHCb trigger. The identification of interesting events during data taking bases on the inference of the possible presence of signal beauty- or charm- hadron decays in the event, starting from the set of stable particles detected in the experiment. This requires the thorough analysis of high-dimensional feature spaces in real-time to classify events efficiently, a task that can be improved by the use of novel machine learning methods, such as Graph Neural Networks (GNN).

The LHCb experiment has completed Upgrade I and is preparing for Upgrade II in a decade's time. These upgrades will increase the instantaneous luminosity by five and ten times, respectively, leading to unprecedented challenges for the trigger system due to an increase in signal and background events. To address this issue, the LHCb DFEI project aims to design a Deep Full Event Interpretation (DFEI) algorithm that uses deep neural networks to process information from all the stable particles in each event. The DFEI algorithm is constructed as a sequence of Graph Neural Networks and its goal is to reconstruct b- and c-hadron decay chains while providing a good level of separation against all the other particles from the rest of the event.

The DFEI approach is designed for both b- and c-hadron reconstruction, but this thesis focuses solely on b-hadron decays. The original training and performance evaluation of the DFEI algorithm was done using solely a simplified simulation of LHCb events based on the Pythia generator, and an inclusive-b sample. This thesis expands the work in two ways: using the Geant4-based full LHCb simulation in DFEI for the first time, and performing the first detailed study of the DFEI performance on a selection of exclusive key signal modes in LHCb. Firstly, a comparison of the Fast and Full simulation methods is

performed, studying their key differences. The analysis compares the distribution of particles in the decay tree of heavy hadrons with those of from the rest of the event, and it examines the differences between the generator-level distributions and the reconstructed ones.

Later on, the edge-preselection step in DFEI is re-designed and optimised, using the knowledge of the new simulation samples. This pre-selection is a crucial aspect of the study as it helps to eliminate a substantial number of edges connecting particles that do not come from the same B hadron while preserving almost all signal particles' connections. The rest of the DFEI algorithm is subsequently re-trained in the new configuration. Various loss functions, such as cross-entropy loss and focal loss, were used for different trainings.
For performance evaluation, the DFEI output is compared with true values obtained from Monte Carlo simulations, using the perfect-reconstruction efficiency as one of the metrics. The evaluation is done both per signal and per event and to compute performance numbers for exclusive signal modes in the processing environment of DFEI, a new truth-matching algorithm is developed.

To conclude the thesis describes re-optimisation and re-training of the DFEI algorithm on the full inclusive simulation and the first detailed study of its performance focused on exclusive decay modes. The use of several metrics for evaluating the performance of a classification model in extracting a signal from a larger data set is explained.

Chapter 1

The LHCb Experiment

The LHCb (Large Hadron Collider beauty) experiment is one of the four main experiments located along the LHC ring at CERN.
In the first section of this chapter 1.1 it is given an overview of the physics behind the b meson, starting from the Standard Model to the way in which it interacts and decays. Afterwards, in the section 1.2, it is shown both the setup of the LHCb detector and how the particle detection process works. The current chapter ends, in the last section 1.3, with the description of the LHCb trigger.

1.1 Physics at LHCb

LHCb is a dedicated b and c-physics precision experiment at the LHC for new physics beyond the Standard Model through the study of very rare decays of charm and beauty-flavoured hadrons and studies of CP violation in charm and beauty hadrons, aiming to better understand the asymmetry between matter and antimatter.

The Standard Model The Standard Model (SM) explains how the basic building blocks of matter interact, governed by four fundamental forces. The way in which the building blocks, called fundamental particles, are related with three of the forces is encapsulated in the Standard Model.

The fundamental particles are divided in two groups of six particles each: quarks and leptons. It can be seen in figure 1.1 that each group consists of six particles, which are related in pairs, or "generations". The first generation contains the most stable and lighter particles (u and d quarks, e and ν_e leptons) whereas in the second and third ones belongs the heavier and less-stable particles (the other ones). These particles that make up the matter are called fermions (half-integer spin).
Owing to a phenomenon known as color confinement, quarks are never found

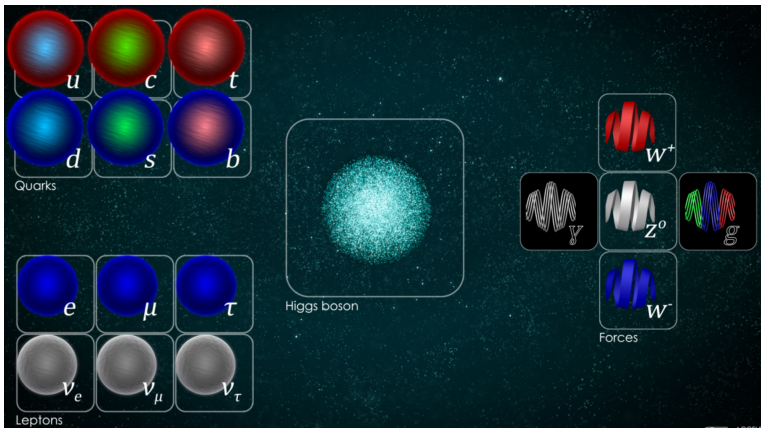


Figure 1.1: The Standard Model [3].

in isolation, but they combine to form composite particles, called hadrons, which are divided in mesons and barions (three quarks).

The three fundamental interactions described in the SM are the electromagnetic force, the weak force and the strong force. They result respectively from the exchange of force-carrier particles: the photon, the gluon and the W and Z, which belong to the bosons group (integer spin).
However, the fourth force, gravity, is not part of the Standard Model, as fitting gravity comfortably into this framework has proved to be a difficult challenge.

In 2012 the discovery of the Higgs boson confirmed the existence of the Higgs field. The particles get their mass by interacting with the Higgs field; the stronger a particle interacts with the Higgs field, the heavier the particle ends up being.

The b-hadrons The B mesons are composed of a bottom antiquark and either an up (B^+), down (B^0), strange (B_s^0) or charm quark (B_c^+). Another interesting b hadron is the Λ_b baryon, composed by a quark up, a quark down and a bottom quark. The fact that the top quark is too heavy to build hadrons makes the b hadrons the heaviest ones (the mass of the b quark is around 5 GeV).
Experimentally, b hadrons are easier to observe because tracks issued from their decays have high transverse momentum due to the high mass of the b hadrons. Their lifetime (nearly 1 ps) is relatively long and the subsequent presence of secondary vertices in the detector can be used as a tag.
Nevertheless measurements of rare signal decays that include invisible particles like neutrinos are a challenging task as they leave no direct signature in the detector. Information of the full B decays is therefore missing in these cases.

The b and c hadrons are generated at LHC during the proton-proton collision where the heavy quark prodution happens through the strong interaction resulting in a quark and anti-quark production ($c\bar{c}$ and $b\bar{b}$). The dominant production mechanism for heavy quarks in pp-interactions is the fusion process of partons (gg and $q\bar{q}$). After the $c\bar{c}/b\bar{b}$ creation, there is the hadronization process where the hadrons are produced. The hadrons then decay in the daughter particles via strong, weak or electromagnetic interaction.
An extrapolation to 14 TeV gives 102.9 mb for the total cross section and 698 μb for the $b\bar{b}$ production cross section. There are higher order diagrams influencing the $b\bar{b}$ cross section, whose contribution cannot be neglected.
The total number of inelastic collisions over a given period of time can be written as:

$$N = \sigma_{inel} \int Ldt \tag{1.1}$$

where σ_{inel} is the inelastic cross section and L is the instantaneous luminosity. In one second of running at the nominal LHCb luminosity of Run II, $L = 4 \cdot 10^{32} cm^{-2} s^{-1}$, this gives an expected number of 10^5 $b\bar{b}$ pairs. In one year this corresponds to 10^{12} $b\bar{b}$ pairs.

1.2 Detector Setup

In this section is firstly described the lhcb detector setup for Run I and II then is briefly discussed the Upgrade I, so the Run III setup. The LHCb detector must be able to reconstruct c and b events with a good resolution on the momentum, the mass, the decay length and other variables. It is also needed a good performance in the identification of different types of hadrons and leptons.

The LHCb detector, shown in figure 1.2, is a single-arm spectrometer instrumented in the forward region and designed to study decays of beauty and charm hadrons. The B mesons formed by the colliding proton beams (and the particles they decay into) stay close to the line of the beam pipe and the production $b\bar{b}$ quarks is suppressed when they have a large variation in the pseudorapidity, so that they are mostly generated both with a similar psudorapidity and in the same direction. This justify the choice of a detector with only a forward arm and with an angular acceptance of $2 < \eta < 5$ (from approximately 10 mrad to 300):

$$\eta = -\log(\tan\theta/2) \qquad pseudorapidity \tag{1.2}$$

where θ the angle between the particle three-momentum $\vec{p}$ and the positive direction of the beam axis.
The preference of the pseudorapity over the angular variable occur because the particles production as a function of the psudorapidity is nearly constant.

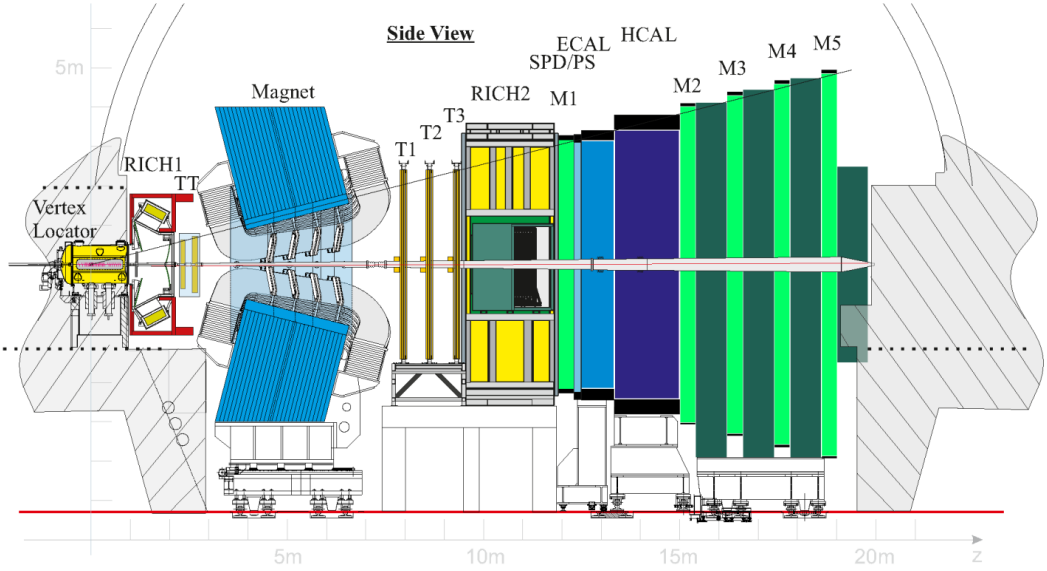


Figure 1.2: The LHCb Detector setup

VErtex LOcator (VELO) The point where the beams collide, and particles containing b and anti-b quarks are produced, is inside the VELO subdetector. Its job is to pick out B mesons from the multitude of other particles produced aiming to both a precise reconstruction and the measurement of the distance between the point where protons collide (and where B particles are created) and the point where the B particles decay, spraying out other particles that VELO can detect. It is made of silicon microstrip detectors and it can locate the position of B particles to within 10 microns [4].

Magnets A dipole magnet is used to measure the momentum (with a precision of about 0.4%) of charged particles by deflecting them in the horizontal plane. The magnet's two coils are each 7.5 m long, 4.6 m wide and 2.5 m high.

Silicon Trackers (ST) LHCb's tracking system consists of a series of four large rectangular stations, each covering an area of about 40 m^2. The Silicon Tracker consists of:

- Four-layer silicon strip Tracker Turicensis (TT) upstream of magnet
- Silicon strip Inner Tracker (IT) in the innermost region of tracking stations (T1-T3) downstream of magnet whereas gas-filled straw-tubes are employed in the outer region of the stations (Outer Tracker, OT)

The principle task of the tracking system is to provide efficient reconstruction of charged-particle tracks. These are used to determine the momenta of charged

particles and their trajectory is used to reconstruct Cherenkov rings in the RICH detectors, muons and electrons (in the calorimeter).

Ring Imaging Cherenkov (RICH) detectors At LHCb experiment two RICH detectors are responisible for identifying a range of different particles that result from the decay of B mesons, including pions, kaons and protons. RICH detectors utilize hybrid photon detectors (HPDs) to measure the positions of the emitted Cherenkov photons and it is mostly used for pion and kaon identification.
The system consists of an upstream detector (RICH-1) with silica aerogel and C4F10 gas radiators (to detect low-momentum particles), positioned directly behind the VELO, and a downstream detector (RICH-2) with a CF4 gas radiator, located behind the magnet (to detect mostly high-momentum particles).

Calorimeters The calorimeter system consists of four sub-detectors: the Scintillating Pad Detector (SPD), the Pre-Shower Detector (PS), the "shashlik"-type Electromagnetic Calorimeter (ECAL), and the scintillating tile iron plate Hadron Calorimeter (HCAL).

- The SPD determines whether particles hitting the calorimeter system are charged or neutral while the PS indicates the electromagnetic character of the particle. The SPD and PS consist of scintillating pads with a thickness of 15 mm, inter spaced with a lead converter. Light is collected using wavelength-shifting (WLS) fibers.

- The electromagnetic calorimeter is responsible for measuring the energy of lighter particles, such as electrons and photons. The ECAL employs 'shashlik' technology of alternating scintillating tiles and lead plates. The overall detector dimensions are 7.76 x 6.30 m.

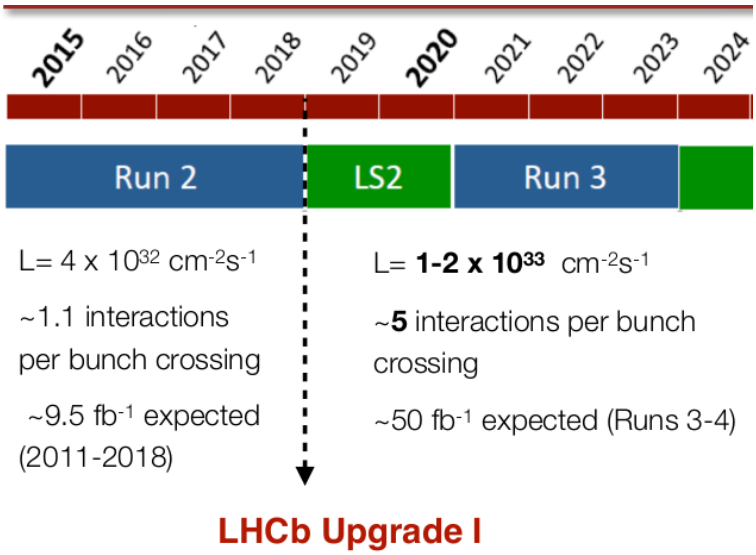
- The hadron calorimeter samples the energy of protons, neutrons and other particles containing quarks. The HCAL is positioned behind the ECAL. Its internal structure consists of thin iron plates inter spaced with scintillating tiles arranged parallel to the beam pipe. Light is collected by WLS fibers running along the detector towards the back side where the PMTs are housed.

Muon Spectrometer Located at the far end of the detector downstream of the calorimeters, the muon system comprises five rectangular 'stations' of multi-wire proportional chambers and GEM chambers in the innermost part of M1. The first station (M1) is placed in front of the calorimeter preshower, at 12.1 m from the interaction point, and is important for the transverse-momentum measurement of the muon track used in the Level-0 muon trigger.

The remaining four stations are interleaved with the muon shield at mean positions of 15.2 m (M2), 16.4 m (M3), 17.6 m (M4) and 18.8 m (M5) [6].

Before the Run 3 has started at the LHCb experiment there was the Upgrade I, briefly shown in figure 1.3. The LHCb detector needed to be upgraded to sustain the renewed experimental challenges due to the increased luminosity.

The VELO was replaced by 26 tracking layers based on 50x50 μm^2 pixel technology that ensures a better hit resolution and simpler track reconstruction. The upgraded VELO is closer to the beam axis and the particles see substantially less detector material before the intersection of the first tracking layers. These improvements provide a better decay time resolution, improve the impact parameter resolution and increase the VELO tracking efficiency especially for low momentum tracks. In order to reduce the radiation damage to the sensors, the detector layers will be opened while not taking data and closed when stable beam condition is reached.
The Upstream Tracker (UT) is essential to improve the trigger timing, and the momentum resolution. The inner sensors will be closer to the beam pipe with respect to the current tracker, in order to increase the geometrical acceptance.
The Scintillating Fiber tracker (SciFi) is used for track reconstruction after the magnet region thus providing measurement of the particle momentum. The SciFi is based on scintillating fibres. The fibres are readout by SiPMs placed on the top and on the bottom of the detector layers.
The usage of UT hits in the track extrapolation from the VELO to the SciFi detectors allows to reduce the number of fake tracks reconstructed by a factor 50-70%. The SciFi will improve the tracking efficiency, reducing in the meanwhile the amount of reconstructed fake tracks, with respect using the current detector in the Upgrade.
The optical layout of the RICH1 is modified to handle the much higher particle occupancy of the upgrade conditions. The current readout of both RICH detectors, performed by HPDs at 1 MHz rate, is replaced by Multianode PMTs working at 40 MHz.
The LHCb calorimeters and muon chambers do not undergo substantial upgrades, apart by the replacement of the front-end electronics and the removal of some stations currently used by the L0 trigger only.



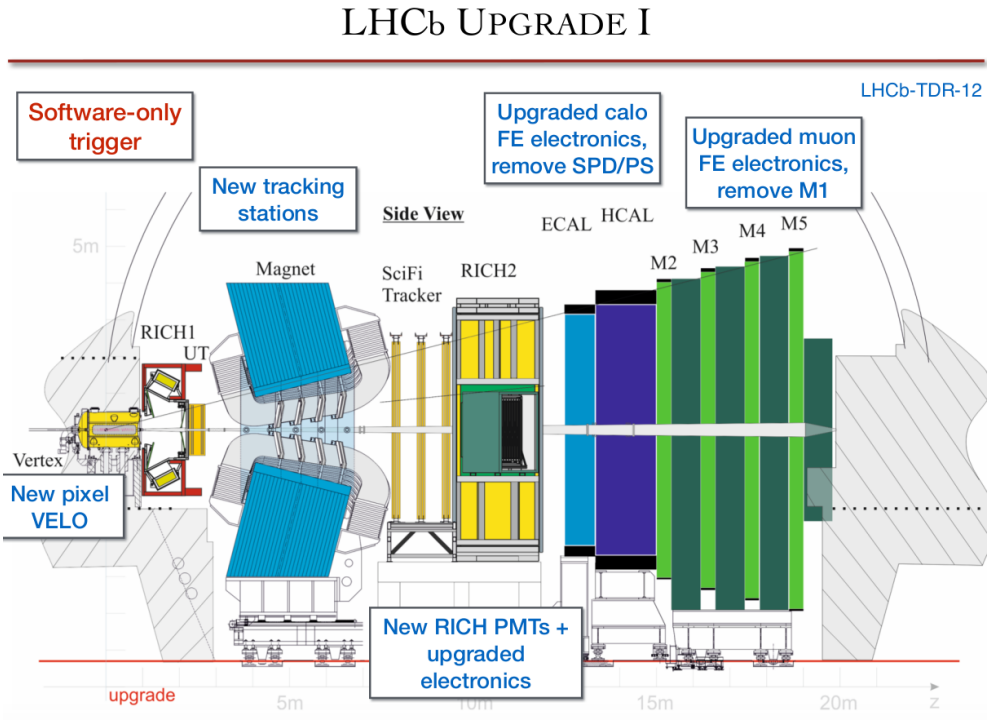


Figure 1.3: Upgrade I at LHCb.

1.3 The LHCb Trigger

All trigger systems consist of a set of algorithms that classify events as either interesting or uninteresting for further analysis, so that the data rate can be

reduced to a manageable level by keeping only parts of the events.
At energy of $\sqrt{s} = 14\ TeV$ the $b\bar{b}$ production cross section at LHCb is about 500 μb. Therefore the LHCb trigger must be able to exploit a large number of b hadrons. This requires an efficient, robust and flexible trigger in order to cope with the harsh hadronic environment.

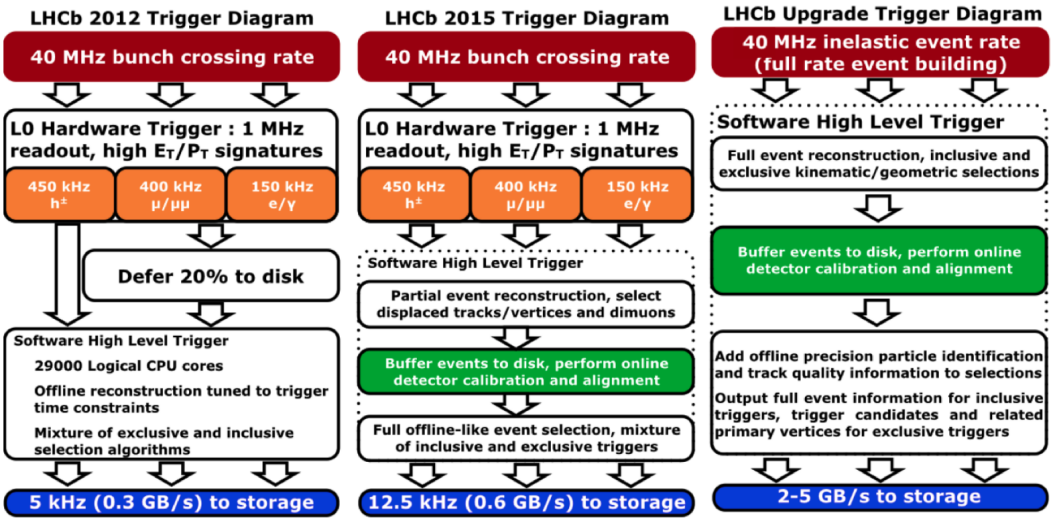


Figure 1.4: LHCb Trigger during Run I, II and III [14].

The main goal of the trigger is to reduce this rate to nearly 10 kHz, keeping the most relevant pp interactions for subsequent physics analysis. In particular, LHCb is mainly interested in heavy flavour physics and the signatures of these decays are the presence of high transverse momentum (p_T) tracks, high transverse energy (E_T) in the calorimeters and displaced vertices, because b hadrons fly on average a few centimetres before they decay. Moreover, the presence of displaced vertices implies that some of the decay products have a large displacement from the pp interaction vertex.
During Run I and II, the LHCb Trigger system had three levels of increasing complication.

First Level Trigger (L0) Events are selected by a trigger consisting of a hardware stage (L0) with 4 μs latency aiming to reduce the event rate to the maximal read-out rate of tracking sub-detectors of 1 MHz. It reconstructs and selects particles with high transverse momentum in the muon chambers or with high transverse energy in the calorimeter system.
The L0 trigger consists of three systems:

- *L0 Muon Trigger.* The L0 muon trigger looks for muon tracks with a high transverse momentum. The trigger decision is based on the two muon

candidates with the largest p_T: either the largest p_T must be above the L0Muon threshold, or the product of the largest and second largest p_T values must be above the L0DiMuon threshold.

- *The L0 Calorimeter Trigger* The Level-0 calorimeter trigger reconstructs and selects particles with a high transverse energy deposit in the calorimeters. It provides candidates for hadrons, electrons, photons and neutral pions. The transverse energy is measured in the ECAL and HCAL but more information is gained with the SPD and PS detectors. For each type, only the candidate with the highest E_T is kept.

- *The L0 Pile-Up System* The pile-up veto system is a component of the VELO and is made of two silicon planes, positioned upstream (with respect to the spectrometer) of the nominal interaction point. The pile-up veto provides a rejection of events with multiple proton-proton collisions through the reconstruction of all primary vertices in the event.

These thresholds lead to a rate of ∼450 kHz for the L0 hadron trigger, ∼400 kHz for the muon trigger and ∼150 kHz for the electromagnetic trigger. The total output rate of the L0 trigger is 1 MHz.

High Level Trigger (HLT) HLT is divided in two steps: HLT1 and HLT2, performing a partial and a full event reconstruction, respectively. In Run 2, the HLT1 tracking transverse momentum threshold was the same as the one in HLT2 for Run 1. In addition, The HLT line of Run 2 has been improved using a more powerful multivariate analysis.

- **HTL1** The HLT first level reduces the rate from the 1 MHz output of the L0 trigger to a few tens of kHz. The HLT1 looks for either one super high transverse momentum or high displacement track.
  The HLT first level consists of several parallel sequences of algorithms, called trigger alleys. Dependent on the type of the Level-0 trigger decision, a different alley is executed. The alleys enrich the B content of the events by adding information from either the VELO or the main tracker and applying cuts on the transverse momentum (p_T) and the impact parameter (IP) with respect to the primary vertex.
  This step is called "L0 confirmation". There are two alternative possibilities to confirm a L0 candidate: with tracks from the main tracking detector (T-Stations) or with tracks from the vertex locator (VELO).

- **HTL2** The HLT2 is executed on the output of HTL1. It reduces the rate to ∼12 kHz. The HLT2 full event reconstruction consists of three major steps: the track reconstruction of charged particles, the reconstruction

of neutral particles, and particle identification. The HLT2 track reconstruction exploits the full information from the tracking sub-detectors, performing additional steps of the pattern recognition which are not possible in HLT1 due to strict time constraints.
The largest contribution to the 12 kHz output rate of HLT2 is provided by topological trigger lines designed to trigger on partially reconstructed b-hadron decays. In particular, the topological trigger lines are used for all b-hadrons with at least two charged particles in the final state and a displaced decay vertex.
Two types of final selections are applied: inclusive and exclusive.
-*Inclusive selections* aim to collect decays of resonances which are likely to have been produced in B decays.
-*Exclusive selections* are specifically designed to provide the highest possible efficiency for fully reconstructed B decays of interest.

The instantaneous luminosity is increased by a factor of five from $L = 4 \cdot 10^{32} cm^{-2} s^{-1}$ in Run I and Run II to $L = 2 \cdot 10^{33} cm^{-2} s^{-1}$ in Run III. To cope with this change in luminosity a new trigger paradigm has been adopted, with the current two-stage hardware plus software trigger being replaced by a fully software trigger. Removal of the hardware trigger is expected to facilitate an increase in the trigger efficiency of purely hadronic decays modes by around a factor of two.

In the past (Runs 1 and 2) the goal of the trigger was indeed to select which events are interesting, and then the full event information would be saved. However, the events will be too large in Upgrade II to be able to store all the event information, and the goal of the trigger needs to change between "which events are interesting?" to "which parts of the event are interesting?". This is the main motivation for DFEI, and the reason why it is "unique" in the story of LHCb, since it goes beyond the simple event selection and attempts to find all the signals in the event, allowing to remove the rest of the event information. Now in Run 3 we are in an intermediate situation, so the right moment to do the first studies and attempts with DFEI.

Chapter 2

Machine Learning

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.
Machine learning has been a crucial component in the field of particle physics, leading to remarkable results in various physical sciences domains. It has also opened new avenues for tackling complex scientific challenges that involve more than simple classification. By leveraging machine learning techniques, researchers can effectively manage large data sets and gain insights into previously unknown causal factors. Pioneering studies that used neural networks and boosted decision trees (BDT) in previous-generation experiments laid the groundwork for the emergence of machine learning as an essential tool at the LHC [15].
In the case of the LHCb experiment, many decays of greatest interest do not produce striking signatures in the detector, making it necessary to thoroughly analyse high-dimensional feature spaces in real time to efficiently classify events. In the first year of LHCb data collection, the primary algorithm used for such classification has been machine-learning-based: a BDT was used for the first two years, which has since been replaced by a MatrixNet algorithm. Currently, 70% of all data retained are classified by machine-learning algorithms and all charged-particle tracks are vetted by neural networks [15].

This Chapter focuses on Neural Networks, which is a sub-field of machine learning, and deep learning, which is a sub-field of neural networks. The first section 2.1 aims to give an overview of the Neural Networks that take inspiration from the human brain through a set of algorithms. Afterwards section 2.2 describes how a Graph Neural Networks algorithm works. To conclude, the last section 2.3 focuses on the application of GNN in particles physics.

2.1 Neural Network

Neural networks are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. Neural networks are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. The general structure of the network, like which and how many layers are used, are critical for the performance of the network. Features of the network such as number of layers or the number of nodes per layer are referred to as hyperparameters. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

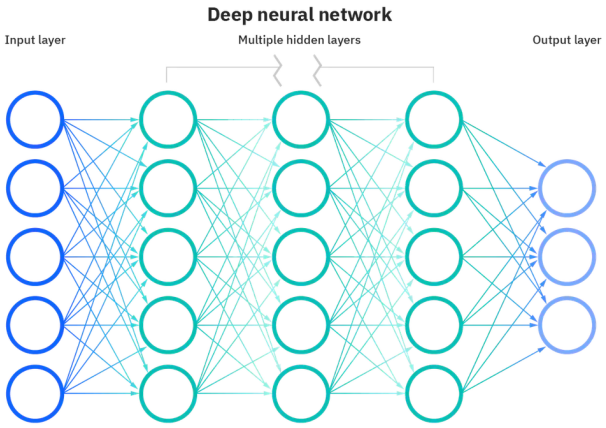


Figure 2.1: Schematic view of a Neural Network [7].

The input of a node are the values $\mathbf{x}$ and weights $\mathbf{W}$ of the previous layer nodes. The node is characterized by a bias b which is the constant added to the product of features and weights and used to offset the result. Then the output of the activation function $f(\mathbf{x})$ tells if the specific node is either active or not. More in general the whole architecture can be defined using the following Eq. 2.1:

$$\mathbf{h}^n = f^n(\mathbf{W}^{n^T} \cdot f^{n-1}(\mathbf{W}^{n-1^T} \cdot f^{n-2}(... f^1(\mathbf{W}^{1^T} \cdot \mathbf{x}^1 + b^1) ...) + b^{n-1}) + b^n) \quad (2.1)$$

The successive layers feed the information into the next one in the forward direction, therefore these networks are often called feed-forward network. The learning procedure is based on correctly adjusting these weights in order to make correct predictions. The final layer of the network, the output layer, fulfills the goal of a specific task (for example classification).

Training In order to get the right weights for meaningful predictions a Neural Networks gets trained. This is done by feeding input data to the network for which the correct prediction or label is already known. The weights $\mathbf{w}$ are adjusted by backpropagation in the training steps by a loss or cost function,

which is used to quantify the prediction quality. The aim for the training is the minimization of the loss (or cost) function J of which value usually gets lower for higher accuracies of the network's predictions. This is achieved with a gradient descent algorithm: the loss information should flow backward through the network to compute the gradient. An example of a gradient descent algorithm is the stochastic gradient descent (SGD) that performs a parameter update for each training example x^i and label y^i:

$$\mathbf{w} = \mathbf{w} - \eta \cdot \nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{x}^i; \mathbf{y}^i) \tag{2.2}$$

where η is the learning rate, the regulator of the speed of the weights adjustments. Its purpose is to prevent too big fluctuations in the weights during the training process.
The weight adjustment is done for subsets of the whole training data at once, called batches. The forward-backward procedure happens repeatedly and stops after a certain amount of iterations, that are called epochs, or when a satisfying performance is achieved.
In order to assess the unbiased performance of the network, the training data gets split into a training set and a validation set. The training set gets used to adjust the weights while the validation set is used to prevent over training, which occurs when the network starts learning the statistical fluctuations of the particular data set used for training leading to worse performances.
Any unknown sets applied to the model after the training is called testing set.

2.2 Graph Neural Networks

The term deep learning, a sub-field of neural networks, implies the usage of multiple (non-linear) inner layers in a neural network in contraposition to a shallow neural network, which typically allows to extract more abstract patterns in data. That for example allows the possibility to use low-level features as input to the network, instead of "engineered" variables.
Neural networks that operate on graphs, and structure their computations accordingly, have been developed and explored extensively for more than a decade under the name of "graph neural networks". A set of objects, and the connections between them, are naturally expressed as a graph. Hence a graph represents the relations (edges or link) between a collection of entities (nodes or vertices).

A graph networks (GN) framework defines a class of functions for relational reasoning over graph-structured representations. The main unit of computation in the GN framework is the GN block, a "graph-to-graph" module which takes a graph as input, performs computations over the structure, and returns a graph as output.

In figure 2.2, a node is denoted as v_i , an edge as e_k, and the global attributes for the graph block as u. s_k and r_k are used to indicate the indices of the sender and receiver nodes, respectively, for an edge k.

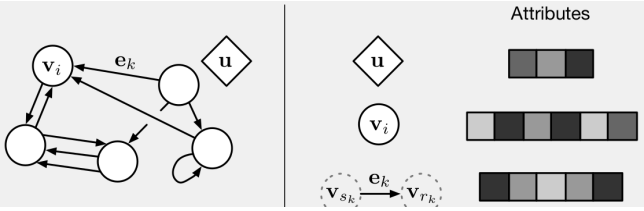


Figure 2.2: Schematic view of graph of a Neural Network [1].

A GN block contains three update functions, ϕ, and three aggregation functions, ρ:

$$e'_k = \phi_e(e_k, v_{r_k}, v_{s_k}, u) \quad \bar{e_i}' = \rho^{e\to v}(E'_i) \qquad \triangleright \textit{ Edge Block} \tag{2.3}$$

$$v'_i = \phi_v(\bar{e_i}', v_i, u) \qquad \bar{e}' = \rho^{e\to u}(\underset{i}{\cup} E'_i) \qquad \triangleright \textit{ Node Block} \tag{2.4}$$

$$u' = \phi_u(\bar{e}', \bar{v}', u) \qquad \bar{v}' = \rho^{v\to u}(V'_i) \qquad \triangleright \textit{ Global Block} \tag{2.5}$$

where E'_i is the set of resulting per-edge outputs for each node i, $\underset{i}{\cup} E'_i$ is the set of all per-edge outputs and V'_i the set of resulting per-node outputs.
The $\rho^{e\to v}$ aggregates the edge updates for edges that project to vertex i, $\rho^{e\to u}$ aggregates the edge updates and $\rho^{v\to u}$ aggregates the node updates. ϕ_u is applied once per graph, and it computes an update for the global attribute u'.
The output of the GN is the set of all edge-, node-, and graph-level outputs $G' = (\bar{u}', \bar{v}', \bar{e}')$.

The node and edge features are created and updated by means of multiple neural networks, operating on the graph. GNNs implement a form of parameterized message-passing whereby information is propagated across the graph, allowing sophisticated edge-, node-, and graph-level outputs to be computed. Within a GNN there are one or more standard neural network building blocks, typically fully connected layers, which implement the message computations and propagation functions.
In practice the ϕ_e, ϕ_v , and ϕ_u are often implemented as a simple trainable neural network. The $\rho^{e\to v}$, $\rho^{e\to u}$, and $\rho^{v\to u}$ functions are typically implemented as permutation invariant reduction operators, such as element wise sums, means, or maximums. The ρ functions must be permutation invariant if the GN block is to maintain permutation equivariance.

In figure 2.3 is explained how a Graph Neural Network works. The process is the following: the edge block computes one output for each edge e'_k, and

aggregates them by their corresponding receiving node i. The particular message function ϕ_e is dependent on the choice of architecture.

Messages are aggregated around a node using $\rho^{e\to v}$ which is then updated through ϕ_v. So that the vertex block computes one output for each node and, at the end, the edge- and node-level outputs are all aggregated in order to compute the global block.

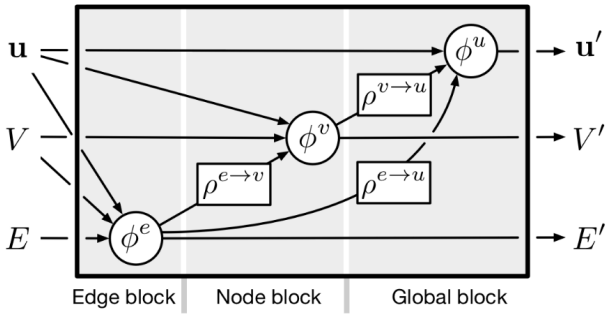


Figure 2.3: A GNN convolution process [17].

The convolutional process has an input graph, $G = (\bar{u}, \bar{v}, \bar{e})$, which is processed and a graph with the same edge structure but different attributes, $G' = (\bar{u}', \bar{v}', \bar{e}')$, is returned as output.

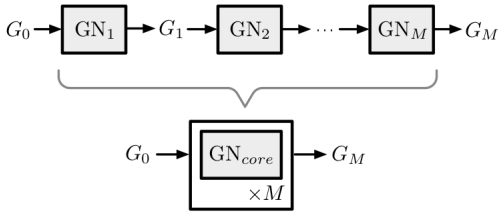


Figure 2.4: Arrangement of GN blocks. Information can be processed and propagated across the graph's structure [17].

A key design principle of graph networks is constructing complex architectures by composing GN blocks.
In the most basic form, two GN blocks, GN_1 and GN_2, can be composed by passing the output of the first as input to the second: $G_1 = GN_2(GN_1(G_0))$.

In principle an arbitrary number M of GN blocks can be composed.

2.3 Application of GNN in Particle Physics

Deep learning is often applied on high level features derived from particle physics data. At many levels the data are, by definition, sets (unordered collection) of items. If one considers relation between items (geometrical, or physical) a set transforms into a graph.
In HEP Experiments is performed a simulation that creates a "truth record" of the physics event; the "truth record" is used to train supervised learning

algorithms; and, at the end, algorithms are applied to real data that were measured by the detector.
The majority of graph-based learning algorithms in HEP focus on reconstruction (clustering) and identification (classification) tasks [17].
GNNs are used in three different ways to make predictions: at the level of the graph, node, or edge, depending on the task at hand [17].
In the following, some examples that focus on the classification task (as will be the case in the DFEI algorithm) are described.

Graph Classification A fist application of GNNs to particle physics can be the *Jets Classification* [17]. Jets or showers are sprays of stable particles that are stemming from multiple successive interaction and decays of particles, originating from a single initial object. Focusing on the b-hadrons, thanks to their the rather large lifetime and hence a significantly displaced decay vertex, identification of b-jet (b-tagging) using classical methods has been rather successful. With the advent of deep learning methods, lower level information has been used to improve the performance of b-tagging, and opened the possibility of identifying jets coming from other particles (c-hadron, top-quark, tau, etc). In particular, within the framework of the particle flow reconstruction, the event can be interpreted through a set of particle candidates.
Another application can be the *Event Classification* [17], where the term event means the capture by an experiment of the full history of a physics process. The jet tagging task presented before is part of a full event identification in collider physics.

Node Classification In a view to increase the overall probability of producing rare processes and exotic events, the particle density of bunches composing the colliding beams can be increased. The downside of this increased probability happens when there is an interesting interaction which is accompanied with other spurious interactions (pileup), considered as noise for the analysis. *Pile up Mitigation* is of prime importance for analysis at colliders. GNNs find important application also in *Calorimeter Reconstruction*, *Particle Flow Reconstruction* and *Efficiency Parametrization* [17].

Edge Classification The *Charged Particle Tracking* as well as the *Secondary Vertex Reconstruction* are one of the main examples of edge classification [17]. Regarding the former, the reconstruction of the trajectory of original particles amounts to find what set of isolated measurement (hits) belong to the same particle while, for the latter, the GNNs aims to find the decay of the intermediate particles, identified as secondary vertices within the jet.

Chapter 3

Deep Full Event Interpretation (DFEI)

The LHCb experiment has just completed its Upgrade I, which will allow it to collect data at a five-times larger instantaneous luminosity. In a decade from now, the Upgrade II of LHCb will prepare the experiment to face another ten-fold increase in instantaneous luminosity.
This implies an increase of both signal and background events (the latter implies ubsequent complexity to reconstruct events), lead to a unprecedented challenges to the trigger system of LHCb.
The LHCb trigger for Run2 can be seen as a logical OR operator between many decay-mode selection lines. The current algorithms, used to make the trigger efficiently identify signals, would be too slow to deal with the high level of particle combinatorics.
The LHCb DFEI project aims to design a *Deep Full Event Interpretation (DFEI)* algorithm to enhance the trigger performance. The term deep refers to the deep neural networks that process information from the detector and infer the heavy-hadron decays occurred in the events. In particular, the DFEI algorithm for LHCb is constructed as a sequence of Graph Neural Networks, where the final-state particles are represented as nodes. The DFEI algorithm is based on GNNs which are currently implemented using the *graph_nets* library developed by Deep Mind.
The deep learning based algorithm aims to quickly identify the signal particles, by reconstructing the b- and c-hadron decay chains in the event, as well as to provide a good level of separation against particles from the rest of the event. The DFEI approach is designed for both b- and c- hadrons reconstruction. This thesis is going to focus only on the b hadrons decays.
The first section of this chapter 3.1 shows how to perform the reconstruction of a decay tree starting from the input stable particles. Section 3.2 gives a brief explanation of the $DFEI$ algorithm while section 3.3 describes the types of simulated data that will be used in the subsequent chapters.

3.1 Decay Tree Reconstruction

The data that are going to be used are samples of events. Each event contains a variable number of proton-proton collisions. A collision is characterized by a primary vertex (PV), which is the 3 dimensional spatial point located within the region of overlap between the two opposite-direction proton beams. From each collision, a different number of particles is produced. They are divided into stable particles (electrons, muons and protons [1]) and unstable particles. The latter have short lifetimes; mostly they decay before interacting with the detector, and so their properties should be inferred through their decay products. Nevertheless there are some unstable particles, for instance among kaons and pions, whose lifetime is such that they are detected before decaying.
The properties of particles produced in collisions are known to nature, but the reconstruction of those properties must be done based on the readout of the detector. The process of inferring all properties is called *reconstruction*.

The final products of a decay process are referred to as stable particles. Unstable particles undergo further decay into either more unstable or stable particles, until the end result is a combination of only stable particles. The secondary vertices (SV) are the 3D spatial points in which the unstable particles decay.
The decay tree of a particle is defined as the chain from the point in which the particle has been produced to the point in which the decay chain ends with only stable particles. It is composed by one PV and one or more secondary vertices.

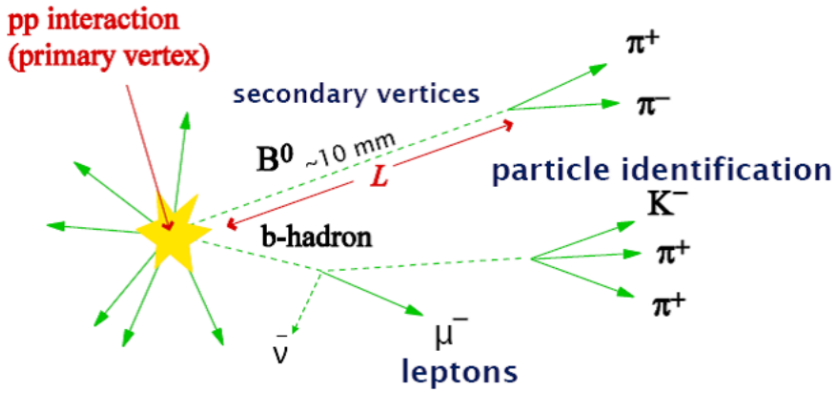


Figure 3.1: Structure of an event composed of decay trees.

[1] Neutral stable particles are also present, but their study is beyond the work in this thesis

An important output of the reconstruction is a set of tracks, each representing the trajectory of a particle through the detector. A track's momentum can also be estimated based on its curvature, which is caused by the LHCb dipole magnet. Knowing the polarity of the magnet as well as its field strength it can also be inferred the associated particle's charge.

It should be noticed that:

- not all the stable particles that are produced are detected
- The LHCb sub-detectors are located at a certain distance from the collision area. It happens frequently that the unstable particles decay before reaching the detection region. Consequently, the LHCb detector only sees the resulted combination of stable particles from the collision plus decay processes.

The reconstruction of a decay tree aims to identify and study the stable particles that were produced in the decay tree of a specific type of initial stable particle: the beauty hadrons.
In each event occur around one to five pp-collisions and on average two b hadrons are generated. Considering Run 3 conditions, the number of stable particles produced in the decay tree of a b hadron is approximately six while the number of stable particles per event seen by the LHCb detector is around 140.

The measurement of the previous quantities at LHCb is not perfect, due to different experimental effects, such as limited geometrical coverage of the detection area, "pixelated" measurements, imperfect algorithms to reconstruct the tracks from single hits in different modules, etc.
The main effects from the experimental reconstruction are:

- Inefficiencies: some of the final state particles will not be seen.
- Resolutions: the measurement of the positions for the PV and origin point of the particles, as well as the three-momentum, won't be exact; the quantities will have a small error.
- PV mis-matching: the association of a PV to a stable particle is done on the basis of which PV is closest to the trajectory of a given particle. This association can fail, for example if another PV randomly happens to be placed between the true PV and the particle.
- Ghost particles creation: not all the created objects may correspond to real particles; it is possible to create ghost tracks, hence trajectories created from combinations of random hits that together look like a real trajectory.

3.2 The DFEI algorithm

The DFEI algorithm aims to properly reconstruct b- and c-hadron decay chains in a hierarchical-clustering manner. The input to the algorithm are the reconstructed stable particles (charged tracks and neutral clusters although this thesis focuses only on charged tracks). Using Graph Neural Networks as the basic deep-learning unit, the algorithm is structured trough a chain of modules with increasing "granularity". Each of the modules filter away part of the event in order to make it easier to the next algorithm to be executed.

Graph Structure Each event is translated into a graph data structure where the nodes represent the particles and the edges represent the relations between each possible pair of particles.
Each particle is characterized by its charge q, the associated primary vertex PV coordinates and by the 3D components of both the momentum and origin point. From a combination of some of the previous quantities, one can construct derived variables, which are known to have interesting physical properties for the task at hand. One of those is the pseudorapidity η as defined in 1.2. The other variables are the *Impact Parameter* (IP), which is the closest distance between the particle's trajectory and the PV from which the stable particle has originated, and the *Transverse Momentum* (p_T), defined as $p_T = \sqrt{p_x^2 + p_y^2}$.
Regarding the relations between pairs of particles (or edges), the derived quantities that have been used are: the *Opening Angle* (θ), which is the angle between the three-momentum vectors of the two particles; the *Distance Of Closest Approach* ($DOCA$), which is the distance of closest approach between the trajectories of the two particles; the *Distance Along z* (Δz), which is the distance between the origin points of the two particles along the z direction; and the *Transverse Distance* ($trdist$), which is distance between the origin points of the two particles in a plane transverse to the sum of their three-momentum. It is defined as $FromSamePV$, a boolean variable indicating if the pair of particles come from the same PV.
The most relevant input nodes features are η, IP, p_T and q while the edges features are: θ, Δz, $trdist$, $DOCA$ and $FromSamePV$.

In figure 3.2 the algorithm of DFEI is schematized. There are three main steps of the algorithm: the *Node Pruning*, the *Edge Pruning* and the *"Topological" LCA reconstruction*. The first two constitute the prefiltering.
The input of the algorithm, after an edge pre filtering selection, is a graph with around 140 nodes, which correspond to the produced particles per events and compose around five PV, and approximately 10^4 edges.
The algorithm aims to find the b hadron ancestor, i.e. "blue particles" in fig. 3.2, for each cluster of stable particles classified as the final state of a decay

tree.

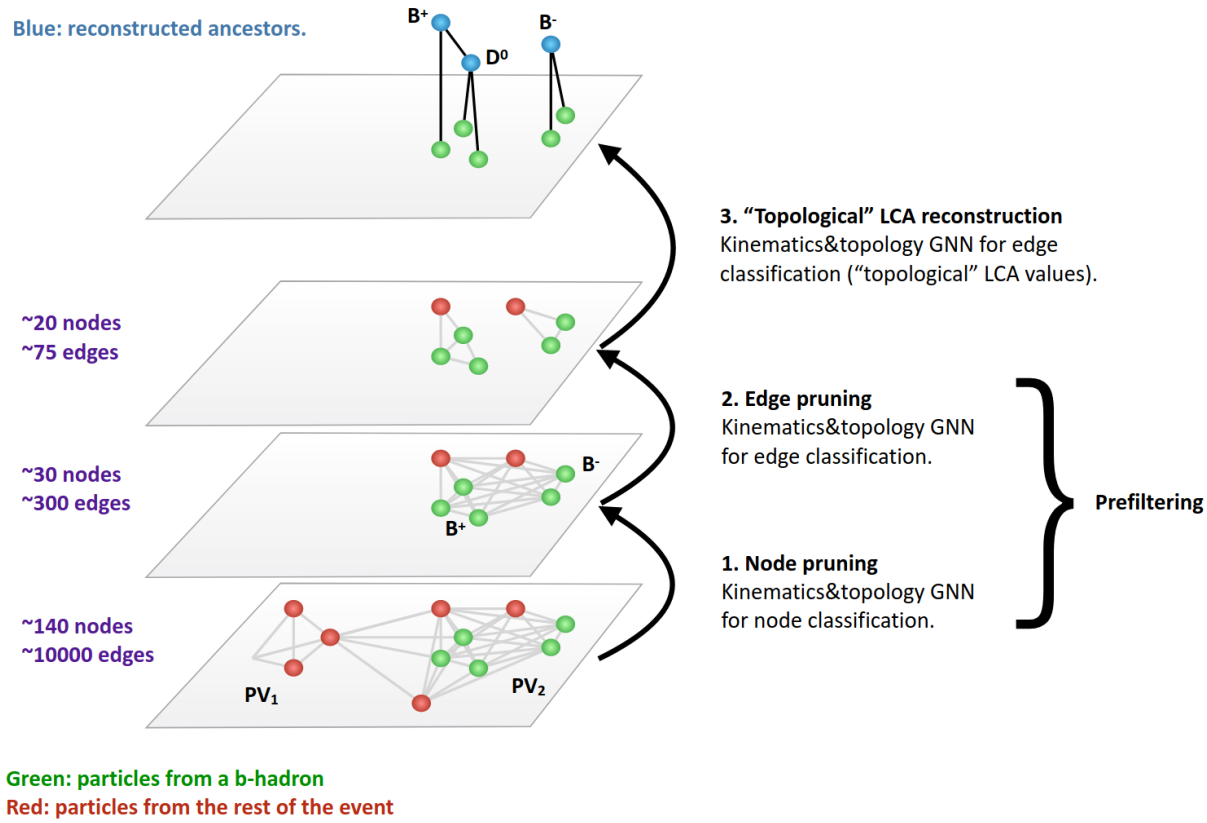


Figure 3.2: Structure of the DFEI algorithm [12].

1. **Edge pre-filtering**: This very first step aim to preliminary reduce the edges. The applied cut has been selected by investigating the edge features. The parameters taken into account were the number of Fully Disconnected Signals Particles (FDSP), which are the fraction of signal particles that would remain fully disconnected if one applies a given cut, and the number of Killed Background Edges (KBE). Aiming to obtain more than 80% KBE and less than 1% FDSP, a 2D cut of $FromSamePV + Theta$ has been chosen. It will be further described in detail in section 4.3.

2. **Node Pruning**: Since there is a large number of nodes and edges, one wants to remove some particles which clearly do not come from a b hadron. This is done by using a GNN where the target is a binary classification per node (rectangular cut): the signal are particles that come

from a b hadron while the background are particles coming from any other sources. After the node pruning filtering, the nodes are reduced around 30 while the edges are nearly 300.

3. **Edge Pruning**: The edge pruning algorithm, which is based on GNN, filters the edges and reduces the number of nodes to around 20 and the number of edges to around 75. This is done to eliminate connections between particles that are topologically distant. The algorithm is more advanced than the node pruning algorithm and helps distinguish particles produced by different b hadrons. Basically, it mainly removes links between particles that come from different primary vertices, with a loose cut-based selection approach. This algorithm is a binary classification of the edges, which considers as signal edges any pair of particles with the same b-hadron ancestor whereas the background edges are any other pair of particles.

4. **Topological LCA reconstruction**: The *LCA (lowest common ancestor)* reconstruction aims to hierarchical clustering a decay chain. It means a GNN-based full reconstruction of the decay chain, adding unstable intermediate particles. It also splits disconnected sub-graphs. The idea of this algorithm is a multi-class classification on edges: the information of the decay tree can be fully encoded using an integer for each edge. This number is the LCA and it goes from zero to three, where LCA=0 means that there is not a common ancestor between a pair of particles (unrelated pair) whereas LCA=1,2,3 is the level of the decay chain at which a pair of particles has a common ancestor.
   Since the LHCb has a good vertex resolution, it has used a *topological* way to reconstruct the LCA. This means that the decay tree is contracted by removing both very short lived resonances and resonances without enough charged descendants to allow a vertex reconstruction. Therefore they are merged with their ancestor. The output of the algorithm is a LCA matrix whose rows and columns are the particles and its entries (LCA values $0, 1, 2, 3$) can be seen as the edges.

The training cycle of the LCA module is shown in figure 3.3. This is an example of the kind of transformation it has been done, but taken from the Belle II experiment [16].

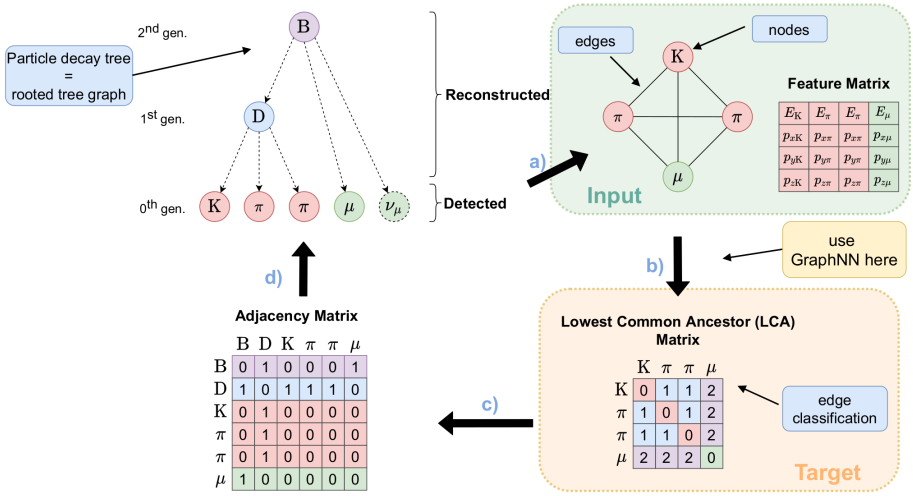


Figure 3.3: Training cycle of the LCA module [9]. An example of a decay tree is shown at the top left. A fully connected graph of the detected final state particles is built (a). This serves as the input for the graph neural network (b). The lowest common ancestor matrix, on the bottom right, functions as the training target. Based upon this, the adjacency Matrix (c), stating the structure of the tree, can be calculated. Finally, the decay tree can be reconstructed using the adjacency matrix and the feature matrix (d) [16].

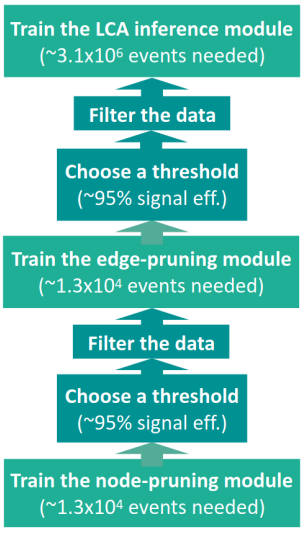


Figure 3.4: Structure of the Training of the DFEI algorithm [13].

Training A structure of the training is shown in figure 3.4. Even if different training samples and threshold values, figure 3.4 describes its main. After applying the node pruning algorithm, a threshold of 95% of signal efficiency is chosen on the output. Then the data are filtered and the training of edge pruning module is applied. The choice of the threshold is always at 95% of signal efficiency. After that, there is training of the topological LCA reconstruction.

The training has been performed in the GPU cluster of the Future SOC Lab [8]. In every training the loss function is the weighted softmax cross entropy, used to balance out the different classes while the minimizer is Adam, which is an extension to the stochastic gradient descent. The advantage of using Adam is a per-parameter learning rate adapted based on the average of recent magnitudes of the gradients for the weight.

3.3 Simulated Data

The DFEI algorithm is designed to operate on real data to be collected by LHCb in the coming years. Those data are affected by the experimental reconstruction effects and don't contain any ground truth information related to which particles come or not from the decay tree of a b hadrons, or which unstable particles were produced in each event.
The DFEI algorithm is trained in a supervised way, which requires the knowledge of the ground truth. For that reason, the training data sets consist on simulated data, in which the events emulate the real ones but all the information is available, including the unstable and invisible particles produced in the relevant decay trees.
Two kinds of simulation have been considered:

- An approximate simulation (**Fast Simulation**) based on PYTHIA8 in which the average experimental effects of inefficiency, resolution and wrong PV association have been emulated. This simulation is very fast to produce, and is the one that has been used to train the first prototype of DFEI.

- The so-called **Full simulation**: official simulation of LHCb events, in which all the interactions of the particles with the LHCb detectors and the detailed reconstruction processes are simulated. This Geant4-based simulation is much slower to produce, but significantly more realistic. *Gauss* is the LHCb simulation framework which generates the initial particles and simulates their transport through the LHCb detector by interfacing to multiple external applications. This simulation has been used, after the fast one, to train DFEI on samples much more similar to the real data.

Format of the data sets For an efficient computation, the node-level and edge-level information is tabulated and stored in two separate files (ROOT file format): the *Particles File* and the *Relations File*. Their most relevant features present in the files are described in table 3.1 and in table 3.2, respectively.

ExpandedEventNumber	unique identifier of the event in which the particle was produced
ExpandedParticleIndex	unique identifier of the particle inside its event
ParticleType	integer, equal to 0 for unstable particles, equal to 2 for b-hadrons (only available in simulation)
xProd_reco	x coordinate of the origin point of the particle
yProd_reco	y coordinate of the origin point of the particle
zProd_reco	z coordinate of the origin point of the particle
px_rec	x component of the three-momentum of the particle
py_reco	y component of the three-momentum of the particle
pz_reco	z component of the three-momentum of the particle
pt_reco	transverse momentum of the particle
associated_ip_reco	IP of the particle with respect to its associated PV
associated_xPV_reco	x coordinate of the PV associated to the particle
associated_yPV_reco	y coordinate of the PV associated to the particle
associated_zPV_reco	z coordinate of the PV associated to the particle
inGeomAcc	boolean, 0 if the particle is outside the detector's geometrical acceptance (only in simulation), 1 if it is inside
isCharged	boolean, 0 for neutral particles (not considered in this study), 1 for charged particles
charge	value of the particle's charge, 0 (only for neutral), -1 or +1
ExpandedPrimaryHeavyHadronIndex	unique identifier of the particle's b-hadron ancestor (if any, otherwise equal to -1) in the event. This quantity is only available in simulation. For particles that correspond to b-hadron ancestors themselves, this quantity is conventionally set to -1
FromPrimaryHeavyHadron	boolean, 0 if the particle has not been produced in the decay tree of a b-hadron, 1 otherwise (only available in simulation)

Table 3.1: **"Particles file"**

ExpandedEventNumber	unique identifier of the event in which the two particles were produced
FirstParticleIndex	unique identifier of the first particle of the pair inside their event
SecondParticleIndex	unique identifier of the second particle of the pair inside their event
theta_reco	opening angle between the two particles' three-momentum
DOCA_reco	DOCA between the two particles
delta_z0_reco	Delta z between the two particles
trdist_reco	"transverse distance" between the two particles
FromSameAssociatedPV_reco	boolean, 0 if the the PVs associated to the two particles are different, 1 otherwise
p1_isCharged	boolean, 0 if the first particle in the pair is not charged, 1 otherwise
p2_isCharged	boolean, 0 if the second particle in the pair is not charged, 1 otherwise
BothInGeomAcc	boolean, 0 if any of the two particles is outside the detector's geometrical acceptance, 1 otherwise
FromSameHeavyHadron	boolean, 0 if the two particles don't come from the same b hadron, 1 otherwise (only available in simulation)
LCA	lowest common ancestor (LCA) value. 0 if two particles do not have a common LCA, 1,2 or 3 depending on the LCA level
LCA_ids	lowest common ancestor (LCA) id

Table 3.2: **"Relations file"**

Chapter 4

Studies On Simulated b Hadrons Decays

In the first section, 4.1, a comparison of the Fast and Full simulation methods is performed, studying their key differences. The analysis examines important factors such as momentum, impact parameter, pseudo-rapidity, etc., and compares the distribution of particles in the decay tree of heavy hadrons with those of from the rest of the event. Additionally, it compares the differences between the generator-level distributions and the reconstructed ones.
The second section, 4.2, delves into the Truth Matching algorithm, which will be employed to estimate the true number of different b decays from various Monte Carlo simulations. This algorithm is introduced with a comprehensive overview of the b decays that will be analyzed. The information provided in this section will aid in understanding how the algorithm works and its importance in the study of B decays.

In the final section, 4.3, the edge preselection step in DFEI is re-designed and optimised, using the knowledge of the new simulation samples. This preselection is a crucial aspect of the study as it helps to eliminate a substantial number of edges connecting particles that do not come from the same B hadron while preserving almost all signal particles' connections.

The concepts and procedures outlined in the subsequent sections and chapters were implemented using Python [18] , a widely-used programming language, and ROOT [2], a data analysis framework. The use of these tools enhances the efficiency and accuracy of the study and allows for seamless integration with other programming tools.

4.1 Analysis of relevant signal properties

This section presents a preliminary analysis of some properties of relevant particles using the *Particles File* described in Section 3.3. The particles in the decay tree of heavy hadrons are referred to as "signal" particles, while the other particles are referred to as "background".
Comparing the signal and background is important for the DFEI algorithm, which aims to correctly select particles in the b hadron decay tree. It has been also compared the true Monte Carlo data with the reconstructed experimental data. The Monte Carlo data simulates the actual particle decays and interactions, while the reconstructed data takes into account various sources of uncertainty and detector effects to simulate the physical processes that occur in the experiment.

During the analysis, only stable charged particles inside the detector's geometrical acceptance will be reconstructed. This means that the particles must have a pseudorapidity between 1.9 and 4.9. The two simulations used in this analysis are based on inclusive $b\bar{b}$ samples. One is the Fast simulation using PYTHIA with approximately 50k events, and the other is the official LHCb simulation (Full) with approximately 10k events. The table in Section 4.1 highlights some relevant features per event of the two simulations. In both cases the simulation reproduces the conditions of the data-taking during Run3.

Number of:	**Fast Simulation**	**Full Simulation**
particles	145	99.9
b hadrons	1.5	1.7
particles from b hadrons	5.9	6.2
$\pi^{+/-}$	121.1	55.6
K	14.9	8.6
μ	0.3	1.0
e	0.5	12.5

Table 4.1: Relevant quantities per event for both the Fast simulation (47746 events) and the Full simulation (10066 events).

An important point that highlights the difference between the Fast and Full simulation concerns the way in which the VELO planes are positioned. The VELO consists of 52 modules positioned along the beam direction, with the full length of the detector being approximately 1000 mm. In the Fast simulation, the VELO planes are assumed to be equally spaced from each other along the beam axis with a spacing of $= 1000/52 = 19.23\ mm$. In contrast, in the Full simulation, the position of the VELO planes along the beam axis is

different; they are closer together in the central area and more spaced towards the end, which changes the measurement of the Z coordinate of the track vertex (that corresponds to the first plane it intercepts), and thus changes the distributions of variables that depend on that quantity.
In the PYTHIA simulation particles are generated but not traced in the detector. Their properties (px, p_y, p_z and origin vertices) are smeared with Gaussian resolution, which is the average one expected for the LHCb Upgrade I. In addition, there is also some probability of wrong-PV association. This is why generated and reconstructed quantities could not be the same in the Pythia sample.
In the Pythia configuration there is also a "selection" cut for all the particles to be in the pseudo-rapidity (η) range from 1.9 to 4.9, which is equivalent to requiring the particles to be within the LHCb geometrical acceptance.
On the other hand, the Full simulation accounts for more complicated effects. In the Full simulation, particles tracks are reconstructed from the hits they leave in the detector. This means that it could be possible to lose some of them, which implies not to reconstruct them leading to inefficiencies, and/or create "ghosts" (fake tracks). In addition, there are some more requirements posed on the tracks in full simulation (on momenta and direction) that reduce their number with respect to the pythia ones as it can been seen in table 4.1.

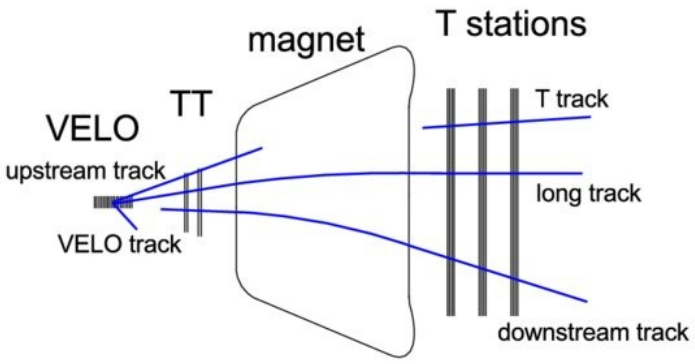


Figure 4.1: Tracks differentiation at LHCb. The long tracks are the ones with best resolution.

As shown in figure 4.1, the LHCb experiment differentiates between various types of tracks, depending on the type of information that was used to reconstruct it, that depends on the combination of sub-detectors through which a specific particle passed.

There are several algorithms to reconstruct tracks, all used in full simulation, however, for this tuning of DFEI we have chosen to use only "long tracks". Long tracks have a starting point in the VELO detector. They are the best ones in terms of resolution since their tracks cross the VELO and all the tracking stations.
Since in the full simulation only long traks have been used; so this removes tracks with certain criteria that are kept instead in the PYTHIA simulation.

Before starting at describing different properties of the fast and full inclusive $b\bar{b}$ simulations, it has been noticed, in the full simulation, some peculiar distribution of the true (not reconstructed) quantities. These are most likely due to the imperfect Geant4-based simulation which is based on Run 3 conditions. Two of them are the transverse momentum (p_T) and the pseudorapidity (η). The latter is defined as:

$$\eta = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right) \tag{4.1}$$

In the pseudorapidity background, it was noticed an high column at $\eta < 0$ most likely caused by ghosts. Since there is no true MC particle that matches those reconstructed particles, they are just fake artifacts coming from the imperfect LHCb reconstruction of particles.
Considering the p_T signal and background, there a is a peak nearly $p_T \simeq 1\ MeV$, hence at low transverse momentum. These particles are additional weird artifacts in the reconstruction. Those are typically filtered away by standard preselections in LHCb (because they are typically not relevant for the analyses), so never end up being studied. Apart from that, the full simulation is based on tracking algorithms for the particle reconstruction which are still being perfected as we progress in the commissioning of the upgraded LHCb detector, and it can be simply that they were partly misconfigured for this simulation.

Based on this statements, it has been decided to leave those low momentum particles outside the studies in the thesis.
In order to decide a cut to remove both low momentum particles and ghosts tracks, the 2d histograms of both the transverse momentum and the pseudorapidity of the true and reconstructed variables are plotted.
The plots are shown in figure 4.2.

In figure 4.2 the true negative pseudorapidity corresponds to a range of the reconstructed pseudorapidity which goes nearly from 2 to 5.5. The true low momentum particles, correspond to a reconstructed particles whose p_T logarithm goes roughly from -2 to 1.
Basing on the histograms in figure 4.2, it has been applied a 2D cut on the full inclusive $b\bar{b}$ sample on the true variables:

$$\eta > 0 \quad and \quad p_T > 25\ MeV \tag{4.2}$$

to leave detailed studies on ghost tracks "for future work", outside of the thesis. This selection will be reflected on the reconstructed ones.

Beginning with momentum and concluding with flight distance, the following will describe several properties of interest for signal and background for

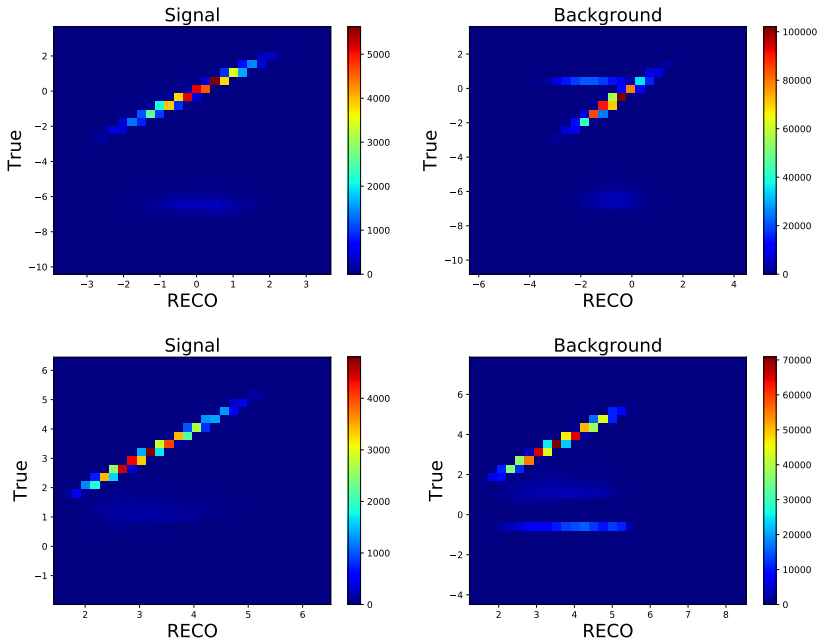


Figure 4.2: 2D histograms of the true and reconstructed pseudorapidity (bottom) and transverse momentum in log scale (top) of the full Simulation for signal and background.

both fast and full simulations.

In figures 4.3 and 4.4 are shown the total momentum and the transverse momentum, respectively, of the signal and background. The total momentum is computed as $\sqrt{p_x^2 + p_y^2 + p_z^2}$ whereas the transverse momentum p_t is the projecton of the three-momentum vector on the transverse plane (the plane perpendicular to the beam axis), i.e. $\sqrt{p_x^2 + p_y^2}$. The p_t of the signal particles could be used to identify the boost of the b hadron, as this one is proportional to the p_t of the decay products.

In both the full and fast simulations the true and reconstructed quantities are exactly the same, hence the histograms completely overlap.

In figure 4.5, a 2D plot of the x and z coordinate differences between the origin point and the primary vertex (PV) is displayed. The primary vertex is defined as the point where protons collide, hence b hadrons are created, while the origin point is the point where the b particles decay and other particles are produced.

As mentioned in section 1.2, these two quantities are reconstructed within the VELO region. The difference between them can be used to determine the

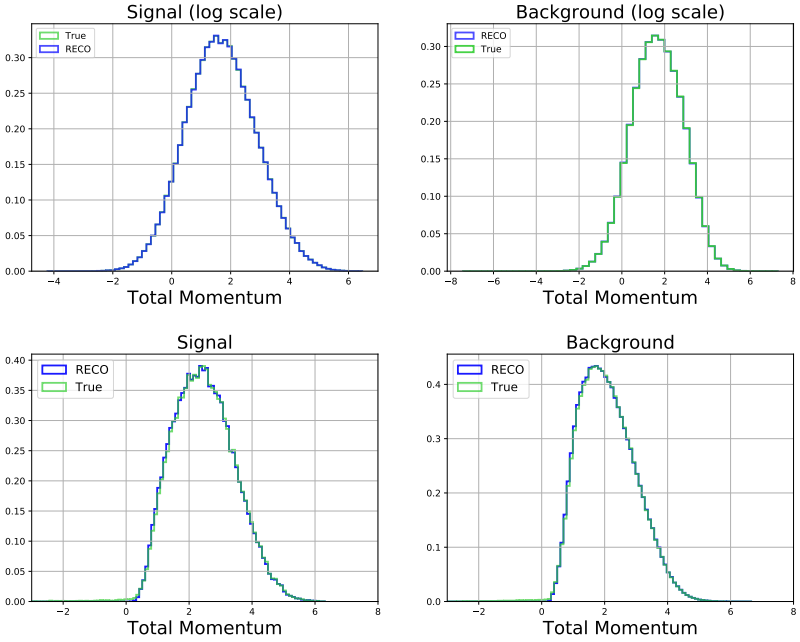


Figure 4.3: True and reconstructed total momentum of Fast Simulation (top) and Full Simulation (bottom) for signal and background.

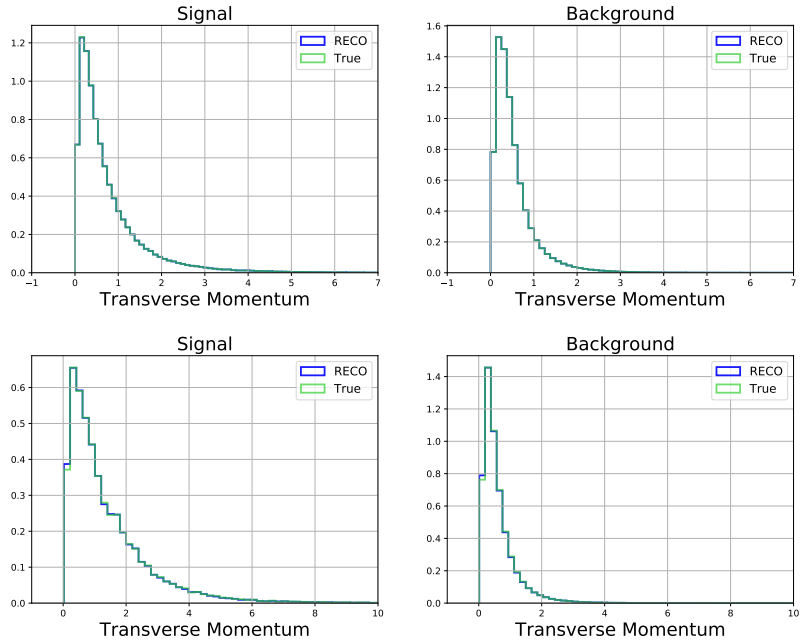


Figure 4.4: True and reconstructed transverse momentum of Fast Simulation (top) and Full Simulation (bottom) for signal and background.

decay length of the B-particle, which is a useful tool for identifying b hadrons and reconstructing their properties.
In figure 1.3 can be observed that the origin of the Cartesian coordinate system is placed at the center of the Vertex Locator sub-detector, so the expected region of interest ranges from $\sim -300mm$ to $\sim 800mm$ along the z-axis.

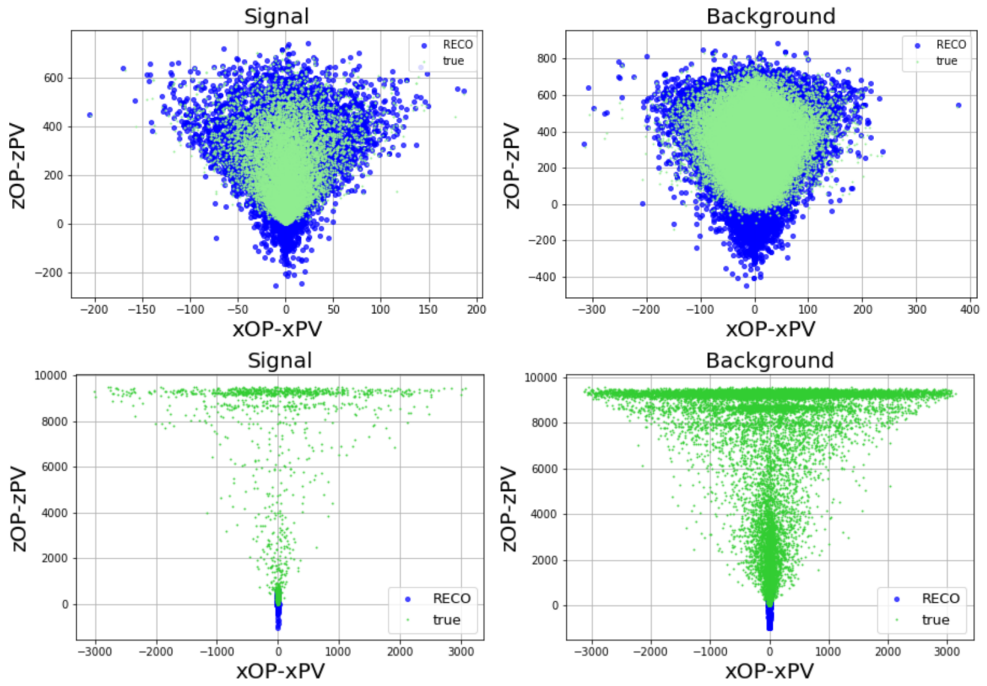


Figure 4.5: Difference in millimeters of the x and z coordinates of the primary vertex (PV) and the origin point (OP) of both Fast Simulation (top) and the Full one (bottom).

Even if in figure 4.5 it is not clear how to distinguish particles from and not from a heavy hadron, it can be noticed that the true and the reconstructed quantities overlap almost everywhere. The only relevant part where the true and reconstructed quantities do not overlap is in the full simulation, where some stripes can be observed located far from the Vertex Locator region ($\sim 8\ m$).
Looking at figure 1.3 the spatial space from 8 to 10 m correspond to the SciFi tracker meaning that while the origin point is located inside the VELO, the primary vertex is in the SciFi tracker region. As only the long tracks with primary vertices reconstructed within the VELO region are kept, there is no further reconstruction.

In figures 4.7 and 4.6 are shown the pseudorapidity η and the azimuthal

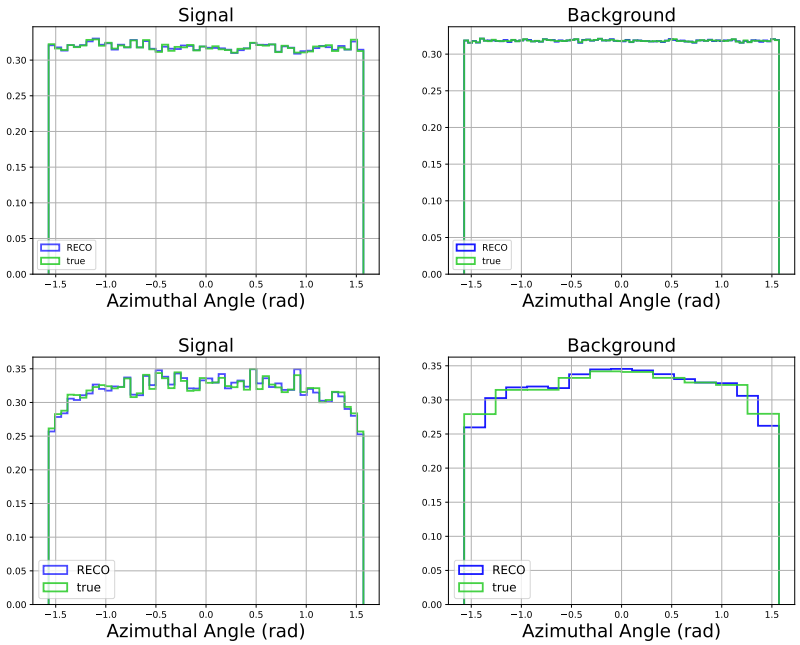


Figure 4.6: Azimuthal angle (ϕ) of the Fast simulation (top) and the Full one (bottom) both for signal and background.

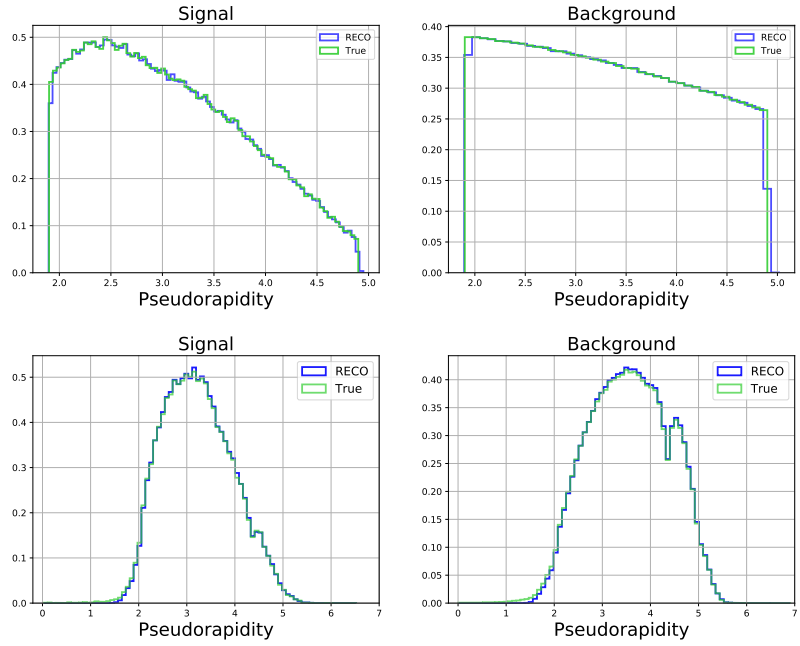


Figure 4.7: Pseudorapidity (η) of the Fast simulation (top) and the Full one (bottom) both for signal and background.

angle ϕ. The latter is defined as:

$$\phi = \arctan \frac{p_x}{p_y} \tag{4.3}$$

As mentioned in section 1.2, the lhcb detector covers a range of pseudorapidity $1.9 < \eta < 4.9$, which is the condition to be inside the detector's geometrical acceptance.

In both the full and the the Pythia-based simulation, the azimuthal angle is uniformly distributed between $-\pi/2$ and $\pi/2$ for both the signal and the background. Generally, the azimuthal angle is uniformly distributed between $-\pi$ and π, but here since the arctan function is defined between $-\pi/2$ and $\pi/2$ the angle is represented as well in this domain.
Speaking of the pseudorapidity, even if there is no much difference between signal and background, it is interesting to notice the profile variation of the Fast simulation with respect to the Full one: while the latter is well-peaked at $\eta \simeq 3/3.5$, the former decreases steadily until the upper limit of $\eta = 4.9$.
In the Pythia simulation, the true pseudorapidity always falls within the geometric acceptance range of $1.9 < \eta < 4.9$ thanks to the way in which Pythia was configured. In the Full simulation, this is not always true.

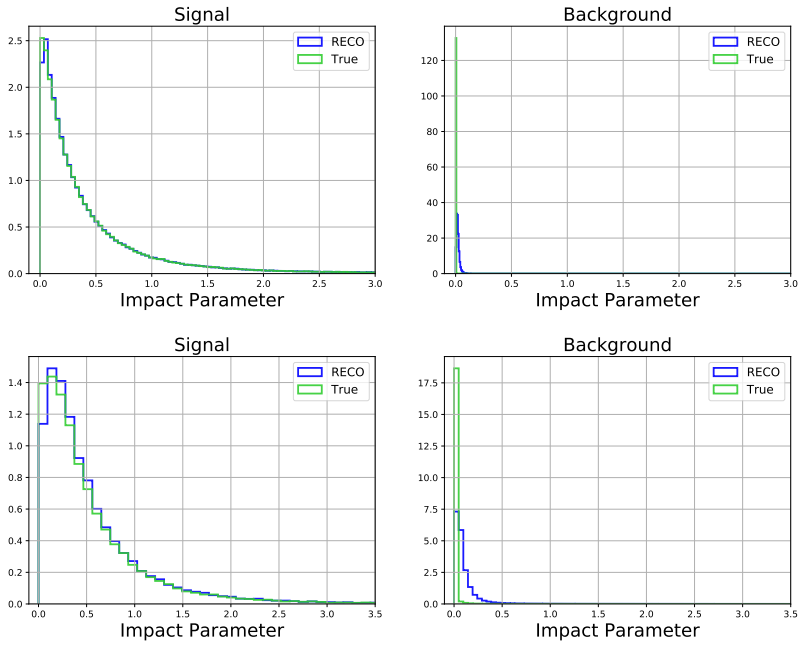


Figure 4.8: Impact Parameter (IP) of the Fast simulation (top) and the Full one (bottom) for signal and background.

The graphs in figure 4.8 show the Impact Parameter (IP) of signal and

background particles with respect to their associated primary vertices (PV). The IP represents the closest distance between the particle's trajectory and the PV from which the stable particle originated.
This parameter is often used in high-energy collision analysis to separate signal events from background events as these ones typically occur at the PV, while signal events come from a displaced vertex due to the long-lived nature of the heavy hadrons' mothers. The well-defined peak of the signal histograms and the shifted peaks of the background plots demonstrate the efficacy of IP in identifying signal particles.
In figure 4.8, the IPs of the true and reconstructed quantities of the signal match perfectly whereas, regarding the background, the true IP have a sharp peak at zero while the reconstructed IP have a broader distributions due to the reconstruction itself. The results are satisfactory because the distributions approach zero when IP approaches 0.5.

In figure 4.9 is plotted the the difference between the origin point (OP) and the PV on the z -axis:

$$\Delta z = z_{OP} - z_{PV} \tag{4.4}$$

The signal has a larger distribution than the background due to the large distances traveled by the heavy hadrons before decaying into daughter particles. The distribution in the full simulation is biased due to a pre-selection of tracks that tends to select the more displaced ones. The difference in the position of the VELO plates along the z axis between the fast and full simulations results in a mismatch in the flight distance plots. In the histogram of the fast simulation background, there is a sudden decrease in the reconstruction peak, which corresponds to the end of the Vertex Locator region.

The true distributions of the fast and the full simulations are different because of the preselection being applied on full MC (long tracks, etc.).
The background is plotted in linear scale due to the peak at zero for the true quantities.

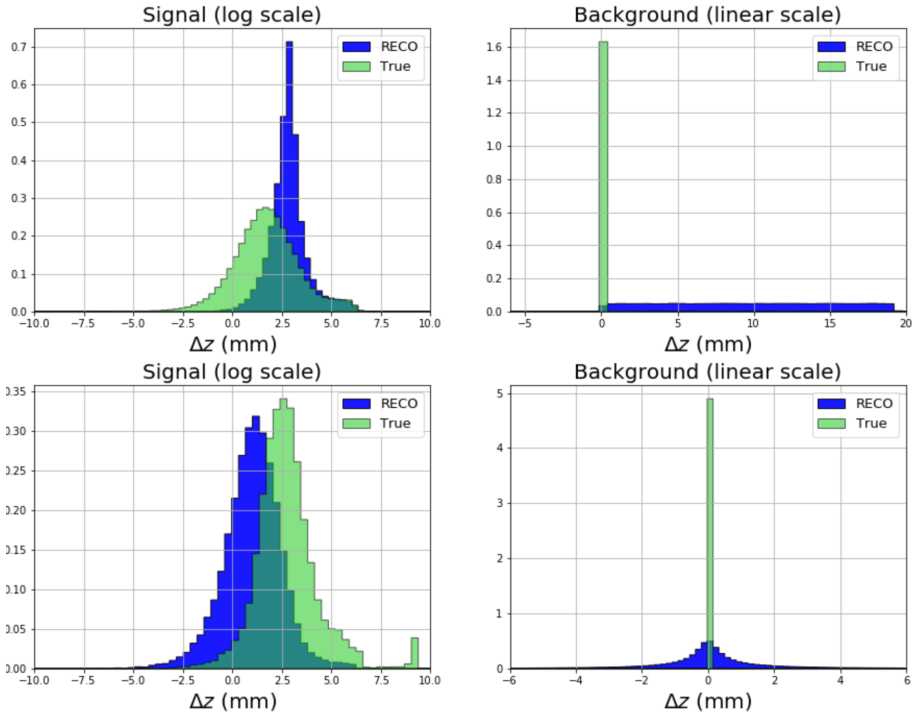


Figure 4.9: Difference of the origin point and the primary vertex on the z-axis of the Fast simulation (top) and the Full one (bottom) both for signal in logarithmic scale and background in linear scale.

4.2 Truth Matching of b hadron decay trees

The reconstruction of b hadron decay trees in high-energy collisions is a crucial step in the study of b hadron properties and decays. Therefore, it is important to evaluate the accuracy of these reconstructed trees and identify any potential biases or limitations in the reconstruction algorithms.

Truth Matching is a technique that compares reconstructed decay trees to the truth information obtained from Monte Carlo simulations. This comparison helps us determine the performance of the reconstruction algorithms and identify any discrepancies between the reconstructed and truth information. The truth matching will be used to compare the DFEI algorithm reconstruction, previously described in section 3.2, with the true decays obtained from Monte Carlo simulations.

The b hadron decays to be analyzed are listed in table 4.2 along with their branching ratios, which is the fraction of times a particle decays to a particular final state.

Common decays of b hadrons include the decay of the bottom quark into two lighter quarks, such as a down quark and an up quark. Decays in which

	Decay	Branching Ratio
a)	$B^+ \to \bar{D}^0[K^+\pi^-]\mu^+\nu_\mu$	$0.03947 \cdot 0.0230 = \mathbf{0.9 \cdot 10^{-4}}$
b)	$B^0 \to D^-[K^+\pi^-\pi^-]\mu^+\nu_\mu$	$0.0938 \cdot 0.0224 = \mathbf{2.1 \cdot 10^{-3}}$
c)	$B^0 \to K^+\pi^-$	$\mathbf{1.96 \cdot 10^{-5}}$
d)	$B_s^0 \to \mu^+\mu^-$	$\mathbf{3.0 \cdot 10^{-9}}$
e)	$\Lambda_b^0 \to \Lambda_c^+[p\ K^-\pi^+]\pi^-$	$0.0628 \cdot 0.0049 = \mathbf{3.1 \cdot 10^{-4}}$
f)	$B^0 \to K_s^0[\pi^+\pi^-]\pi^+\pi^-$	$0.6920 \cdot 5.3\ 10^{-5} = \mathbf{1.8 \cdot 10^{-5}}$
g)	$B^0 \to K_0^*[K\pi]\mu^+\mu^-$	$0.99 \cdot 9.4\ 10^{-7} = \mathbf{9.3 \cdot 10^{-7}}$
h)	$B^0 \to K_0^*[K\pi]e^+e^-$	$0.99 \cdot 1.03\ 10^{-6} = \mathbf{1.0 \cdot 10^{-6}}$
i)	$B_s^0 \to D_s^-[K^-K^+\pi^-]\pi^+$	$2.98\ 10^{-3} \cdot 0.054 = \mathbf{1.6 \cdot 10^{-4}}$
j)	$B^0 \to D^-[K^+2\pi^-]D^+[K^-2\pi^+]$	$0.094 \cdot 0.094 \cdot 2.11\ 10^{-4} = \mathbf{1.9 \cdot 10^{-6}}$
k)	$B^+ \to K^+K^-\pi^+$	$\mathbf{5.2 \cdot 10^{-6}}$
l)	$B_s^0 \to J/\psi[\mu^+\mu^-]\ \phi[K^+K^-]$	$1.04\ 10^{-3}\ 0.0596\ 0.489 = \mathbf{3.0 \cdot 10^{-5}}$

Table 4.2: Some of b hadron decays along with their respective branching ratios [19].

the hadron decay in two daughter particles is known as a two-body decay and is the most common decay of b hadrons. Other types of decays include three-body decays and four-body decays, where the b hadron decays into three or four particles.

The first two decays in table 4.2 have significantly higher branching ratios compared to the other B decays. An inclusive $b\bar{b}$ simulation with a few thousand events can reasonably provide a significant number of events for each decay.

	g)	h)
Simulation	Full	Full
events	1162	833
Number of:		
particles	45	42
B	1.7	1.7
particles from B	6.2	6.3
$\pi^\pm$	27	25
K	4.9	4.7
μ	2.3	0.3
e	1.5	3.4

Table 4.3: Relevant quantities per event for exclusive simulations with Run2 conditions. The letters refer to decays in table 4.2.

Regarding the other decays, to obtain a sufficiently large number of events is necessary to use exclusive simulations. Exclusive simulation in particle physics refers to the process of simulating only specific, predetermined final states in a particle interaction or decay, rather than simulating all possible final states (inclusive simulation).

The Fast and Full simulation samples includes, in addition to the inclusive $b\bar{b}$ simulation, some exclusive simulations. Some relevant particle properties per event of the exclusive simulations are shown in tables 4.3,

for samples obtained with Run2 conditions on the simulations, 4.4, for samples obtained with Run3 conditions on the simulations.

	c)	**c)**	**d)**	**e)**	**e)**	**f)**	**g)**	**i)**	**j)**	**k)**	**l)**
Simulation	Full	Fast	Full	Full	Fast	Full	Fast	Fast	Fast	Fast	Fast
events	800	5000	848	2922	5000	2366	5000	5000	5000	5000	5000
Number of:											
particles	99	150	98	101	255	97	151	151	151	151	150
b hadrons	1.9	1.5	2.0	1.8	1.5	1.8	1.5	1.5	1.5	1.5	1.5
particles from											
b hadrons	5.6	3.9	5.9	5.9	5.8	4.9	6.0	5.9	6.9	5.1	6.2
$\pi^{\pm}$	55	124	53	56	124	55	124	125	125	124	122
K	8.9	15.8	8.3	8.7	15.5	8.0	15.7	16.8	16.2	16.8	16.9
μ	0.8	0.2	2.4	-	-	0.8	2.2	0.2	0.2	0.2	2.1
e	12.2	0.8	12.1	-	-	11.9	0.8	0.9	0.9	0.9	0.9

Table 4.4: Relevant quantities per event for exclusive simulations with Run3 conditions. The letters refer to decays in table 4.2.

For each decay in table 4.2, a suitable truth matching algorithm has been computed based on the specific decay channel of the b hadron.

The truth matching algorithm was computed using information contained in the Particles File and Relation File described in section 3.3. Only particles within the detector's acceptance were considered in the development of the algorithm.

A final state particle is uniquely identified by its event number and particle index, while its type is determined by its ID. The ID, an unidentified number for particles, is taken from the Monte Carlo numbering scheme adopted in particle physics to give a unique identifier to each particle in an event based on its properties. The IDs in the Particles and Relations files are taken from the Particle Data Group's (PDG) Monte Carlo numbering scheme [19].

Truth Matching Algorithm The algorithm is described in detail for the decay $B \to D[K\pi\pi]\mu\nu$, but in an analogous way it is applied to the others interesting decays. Since true output of the simulation gives the features of all the particles in the decay chain, the goal of the algorithm is to group all the particles in the decay sequence of the targeted b hadron into clusters.

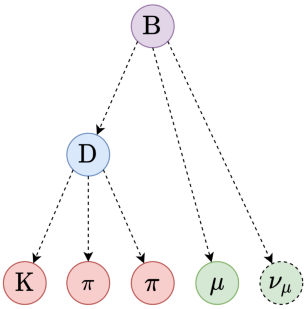


Figure 4.10: Decay tree of the B^0 meson. The B (purple) decays into a μ and ν_μ (green) plus a D meson (blue). The D meson then decays into a K and two π (red).

To better visualize the decay chain, one can observe figure 4.10 where the final state particles in the last row are divided in two colors: red for particles that come from an intermediate decay, i.e. D decay, and green for particles directly obtained from the B meson decay. In section 3.2 it has defined the LCA (lowest common ancestor). The LCA of the two pions and the kaon is one, meaning the D meson is their common ancestor, while the LCA of pairs including the muon (either μ-π or μ-K) is two, with the B meson as their common ancestor.
It should be noted that neutrinos cannot be detected, so there is no information available about them in the simulation.

The first step of the truth matching algorithm is the identification of the interesting particles in every event based on both the ID and the fact that they come from a heavy hadron. This allows only the desired particles to be selected and reduces the computational time required for the algorithm to run. For each event in the set, the algorithm first searches for a pair consisting of a kaon and a pion. If found, there must be a row in the Relation file that contains both their indices. Once a pair is found, it is then checked to see if it satisfies the conditions to be part of the B decay chain, i.e., the K and the π come from a heavy hadron, the LCA is one, and its ID corresponds to the D meson.
If all conditions are satisfied, the algorithm continues in the same manner to identify the second K-π pair and the π-π pair and checks if they meet the previously mentioned conditions. At this point, the K-π-π triplets have been found (shown in red in fig. 4.10), leading to the reconstruction of the D meson (shown in blue in fig. 4.10).
Now, the complete reconstruction of the B decay chain needs to be achieved. To accomplish this, the algorithm continues by taking the triples and checking if each of the three particles forms a pair with a muon (shown in green in fig. 4.10). It must be paid attention to the changed conditions, now the LCA should be two while its ID should correspond to the B meson. If every pair (μ-K and the two μ-π) satisfies all the requirements the cluster is complete and a b hadron decay tree has been found.

An example of the results of the process described above are shown in fig-

ure 4.11.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Event_Number	606	616	1064	2648	3466	3902	4262	4299	4397	6387	7845	9100	9505	10864	11078	11581
B_Index	860	378	691	420	137	351	884	390	220	1126	1576	1656	429	167	100	488
B_Hadron	B0	anti-B0	B0	B0	B0	B0	anti-B0	anti-B0	B0	anti-B0	anti-B0	B0	anti-B0	anti-B0	anti-B0	B0
D_Meson	D-	D+	D-	D-	D-	D-	D+	D+	D-	D+	D+	D-	D+	D+	D+	D-
K_Index	1360	689	872	913	297	603	1637	773	413	1530	2516	2504	813	370	247	653
Pi1_Index	1361	690	873	914	298	604	1638	774	414	1531	2517	2505	814	371	248	654
Pi2_Index	1362	691	874	915	299	605	1639	775	415	1532	2518	2506	815	372	249	655
Mu_Index	1275	675	863	789	290	594	1486	763	407	1525	2496	2507	804	362	215	629

Figure 4.11: Example of the truth matching algorithm output. Each column represents a B decay tree which is uniquely identified by the event number, the B index and daughter particles indexes.

The truth matching algorithm has been developed for every decay in table 4.2 and it has been evaluated on both the Fast and Full simulations.
In both cases it has been considered both inclusive and exclusive simulations.
An overview of the results obtained is shown in table 4.5.

	Fast Incl.	**Full Incl.**	**Full Excl.**	**Fast Excl.**
$B^+ \to \bar{D}^0[K^+\pi^-]e^+\nu_e$	123	19	-	-
$B^+ \to \bar{D}^0[K^+\pi^-]\mu^+\nu_\mu$	97	13	-	-
$B^0 \to D^-[K^+\pi^-\pi^-]e^+\nu_e$	40	6	-	-
$B^0 \to D^-[K^+\pi^-\pi^-]\mu^+\nu_\mu$	31	3	-	-
$B^0 \to K^+\pi^-$	14	7	359	4720
$B_s^0 \to \mu^+\mu^-$	0	0	498	-
$\Lambda_b^0 \to \Lambda_c^+[p\ K^-\pi^+]\pi^-$	3	7	342	1735
$B^0 \to K_s^0[\pi^+\pi^-]\pi^+\pi^-$	0	0	105	-
$B^0 \to K_0^*[K\pi]\mu^+\mu^-$	0	0	1162	4672
$B^0 \to K_0^*[K\pi]e^+e^-$	0	0	833	-
$B_s^0 \to D_s^-[K^-K^+\pi^-]\pi^+$	-	-	-	3937
$B^0 \to D^-[K^+\pi^-\pi^-]D^+[K^-\pi^+\pi^+]$	-	-	-	3372
$B^+ \to K^+K^-\pi^+$	-	-	-	4648
$B_s^0 \to J/\psi[\mu^+\mu^-]\ \phi[K^+K^-]$	-	-	-	4357

Table 4.5: Number of decays found using different types of simulation: PYTHIA (49399 events), Full Inclusive (10066 events) and various Full and Fast exclusive simulations.

4.2.1 Invariant Mass Reconstruction

The output of the truth matching algorithm, shown in figure 4.11 is then processed as input of a second mini-algorithm, which uses this table to select the info of the relevant particles from the data files, and uses the information to fill two histograms for the invariant masses of the D meson and the B hadron:

- The invariant mass of D meson, computed by combining the four-momentum of the K and the two π,
- The invariant mass of the B hadron, computed by combining the four-momentum of the K, the two π and the μ.

The squared invariant mass for a system of particles is a Lorentz-invariant quantity, and is given by:

$$p^{\mu}p_{\mu} = \left(\sum_{i=1}^{n} E_i\right) - \left(\sum_{i=1}^{n} \mathbf{p}_i\right) \tag{4.5}$$

where p^{μ} is the four-momentum, E is the energy, $\mathbf{p}$ is the three momentum. In a particle decay $a \to 1 + 2$, the invariant mass of the decay products is equal to the mass of the decaying particle:

$$(p_1 + p_2)^{\mu}(p_1 + p_2)_{\mu} = p_a^{\mu}\, p_{a\mu} = m_a^2 \tag{4.6}$$

Figure 4.12 shows the reconstruction of the invariant masses of the D (x axis), whose real value is $1869.62 \pm 0.20\ Mev/c^2$ [19], and of the B (y axis), whose real value is $5279.34 \pm 0.12\ Mev/c^2$ [19], considering the $B^0 \to D^{\pm}[K\pi\pi]\mu(e)\nu$ and the $B^{\pm} \to D^0[K\pi]e\nu$ decays.

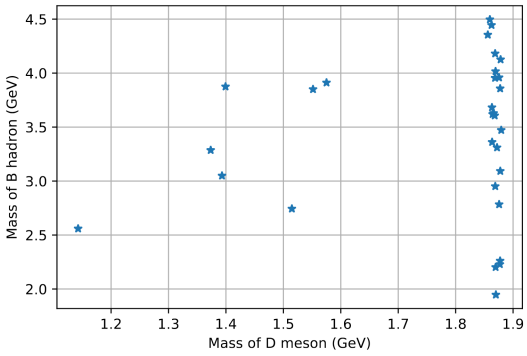


Figure 4.12: Invariant Mass of the $B^0 \to D^+[K\pi\pi]e^-\nu$ decay using a PYTHIA sample with 49399 events.

The invariant mass of the D meson in well-estimated while the B meson one it is mostly under estimated. The main reason is the missed neutrino in

the decay, but a part of the defect in mass could be due to the radiation of photons in different parts of the decay, which are not considered and carry away part of the momentum/energy.

Expanding the study to the other decay modes of interest, several invariant mass distributions are obtained for the full exclusive decays in figure 4.13 and in the fast exclusive decays in figure 4.14.

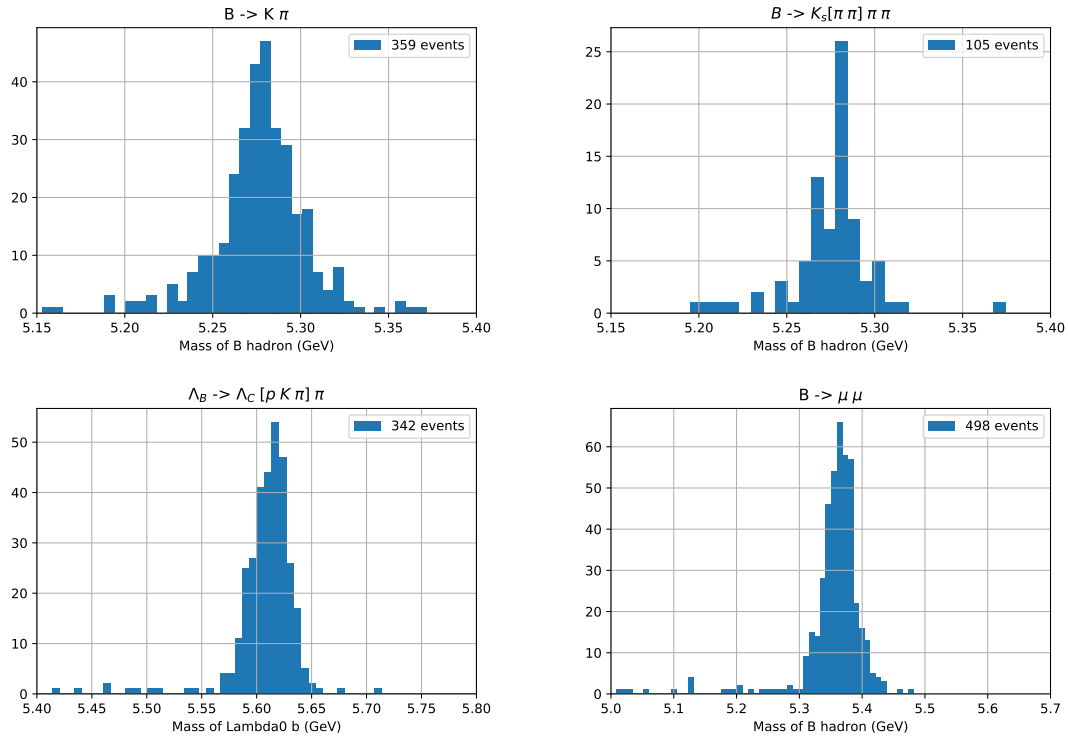


Figure 4.13: Invariant Mass reconstruction of the B meson mass considering $B^0 \to K^+\pi^-$ (top left), $B_s^0 \to \mu\mu$ (bottom right) and $B^0 \to K_s^0[\pi\pi]\pi\pi$ (top right) decays and of the Λ_b considering the $\Lambda_b^0 \to \Lambda_c[pK\pi]\pi$ (bottom left) decay. Full Exclusive simulation samples are used.

The invariant masses of the B^0 meson, the Λ_b^0 baryon ($5619.60 \pm 0.17\ MeV/c^2$) and the B_s^0 ($5366.88 \pm 0.14\ MeV/c^2$), in the full exclusive samples, are almost perfectly reconstructed. Considering the fast exclusive samples, since they have a grater number of events, all the mother particles invariant masses are very well-peaked around the correct mass value.

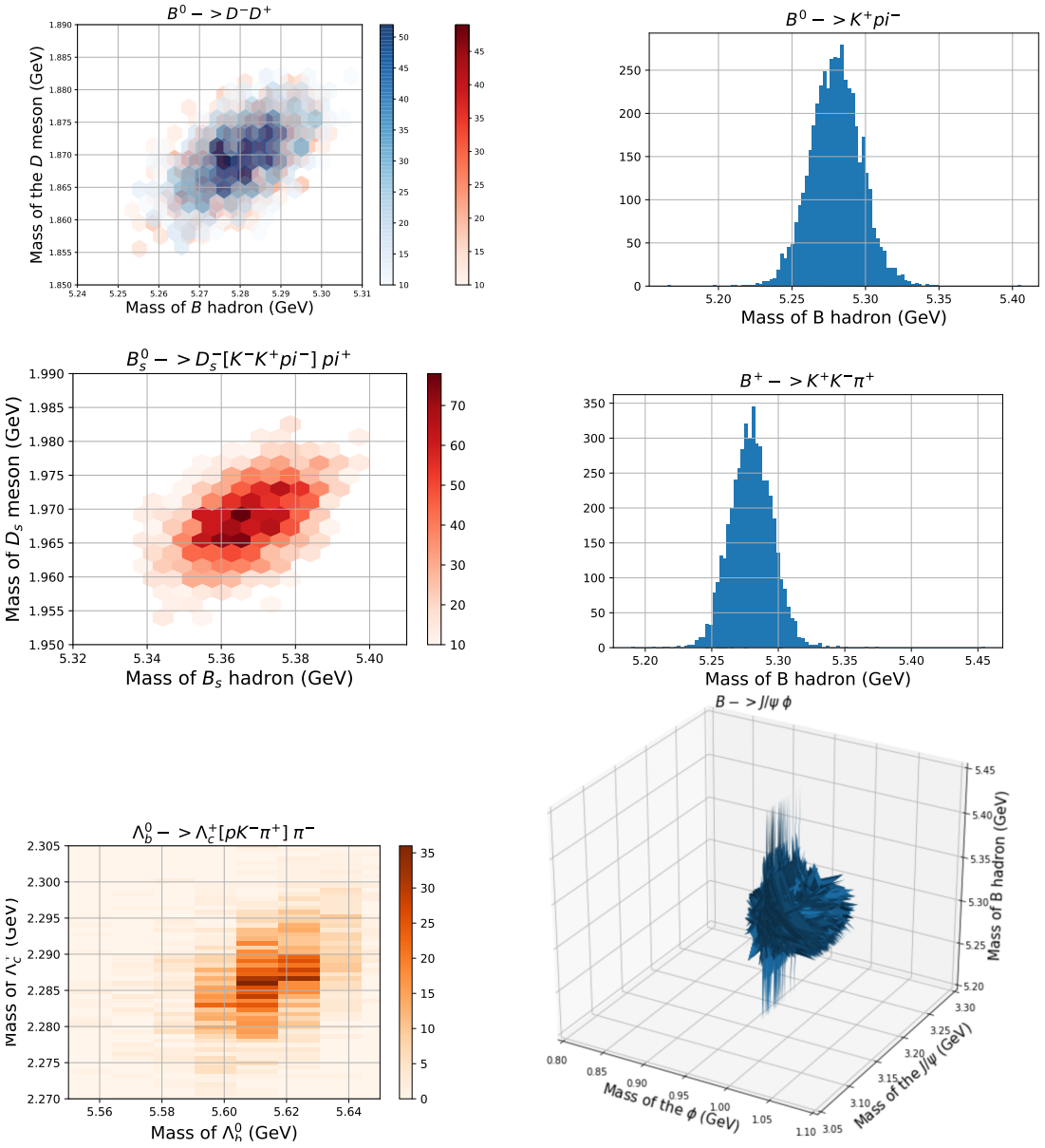


Figure 4.14: Invariant Mass reconstruction of the B meson mass considering $B^0 \to D^+D^-$ (top left), $B^0 \to K^+\pi^-$ (top right), $B_s^0 \to D_s^-[K^-K^+\pi^-]\pi^+$ (central left), $B^+ \to K^+K^-\pi^+\pi$ (central right), $\Lambda_b^0 \to \Lambda_c^+[pK^-\pi^+]$ (bottom left) and $B^0 \to J/\psi\,\phi$ (bottom right) decays.
Fast Exclusive simulation samples are used.

4.3 Edge Pre-filtering

As previously defined in chapter 3, in the DFEI algorithm each event is translated into a graph data structure where the nodes represent the particles and the edges represent the relations between each possible pair of particles.
Edge Pre-filtering refers to a preliminary cut on the edges which will be implemented in the structure of the DFEI algorithm (section 3.2).
The goal of the edge pre-filtering is to find a cut that removes a large fraction of background edges, defined as those connecting particles that do not come from the same b hadron, while keeping a large fraction of signal edges, which are those connecting particles from the same b-hadron. To elaborate further on the second point, an optimal scenario would involve having at least one edge connecting each signal particle to another signal particle originating from the same b hadron. This would enable the information to be conveyed efficiently through the graph.
So instead of looking at the fraction of signal edges, one can compute the fraction of "signal" particles that would remain fully disconnected if one applies a given cut on the edges.

The previous cut on the edges used for the PYTHIA based simulation killed $\sim 60\%$ of the background edges, while leaving only $\sim 3\%$ of the signal particles fully disconnected from other signal particles from the same b-hadron. The objective is to identify an alternative edge cut for the full simulation that performs better than the previous cut used in the PYTHIA based simulation. The previous cut was a combination of transverse distance and opening angle of the particles.

The potentially interesting variables, which can be found in the *Particles* and *Relations files* in section 3.3, would be:

* *DOCA*: Distance Of Closest Approach between the two particles
* *TrDist*: Transverse Distance between the two particles
* *Theta*: opening angle between the two particles' three-momentum.
* *Same PV*: boolean, 0 if the two particles don't come from the same b hadron, 1 otherwise.

    In particle physics, primary vertices are the locations where the colliding protons or nuclei first interact with each other. However, because there are often multiple interactions happening at the same time, it can be challenging to determine which particles came from which primary vertex. The association of primary vertices refers to

the process of assigning each particle in an event to its most likely primary vertex of origin.

The first step of this analysis was to have a look at the variables to roughly estimate some possible thresholds which could give the best separation between signal and background. This is done for three out of four variables since the *Same PV* is boolean and it should be true for signal and false for background. The association of primary vertices is not perfect, so this will have a certain efficiency.
Taking the boolean variable from the *Relations File* considered true when a pair of particles come from the same heavy hadron, it is classified as signal every pair of particles for which *FromSameHeavyHadron*= 1 while the background is where *FromSameHeavyHadron*= 0.
This classification is done for every pair of particles per event.

Afterwards, the number of selected signal edges, killed background edges and fully disconnected signal particles, are computed in the following way:

- *Selected Signal Edges*: when a pair of particles coming from the same b-hadron decay overcome the threshold
- *Killed Background Edges*: when a pair of particles that don't come from the same b-hadron decay do not overcome the threshold
- *Fully Disconnected Signal Particles*: when a particle coming from a b-hadron decay becomes fully disconnected from all the other particles that from the same decay chain, after a cut on all the edges has been applied.

Initially, 1D cuts were applied by considering one variable at a time. The results of these cuts with different thresholds are displayed in Figure 4.15 for the Fast simulation and Figure 4.16 for the Full simulation.

Since the *SamePV* boolean variable has only a given threshold, to compare the various cuts, it has been chosen the Fully Disconnected Signal Particles of the *SamePV* as the reference and then the percentage of Killed Background Edge of each cut has been evaluated. In table 4.6, the results of the 1D cuts on each variable are given for both fast inclusive $b\bar{b}$ and full inclusive $b\bar{b}$, with 6153 and 10066 events, respectively.

After examining the results presented in Table 4.6, it was found that the *Same PV* cut both in the Full and Fast Simulations yielded the best performance, achieving a percentage of Killed Background Edges of approximately 77%. Therefore, the next step was to perform 2D cuts, always considering the *Same PV* in addition to one of the other variables. The variables taken

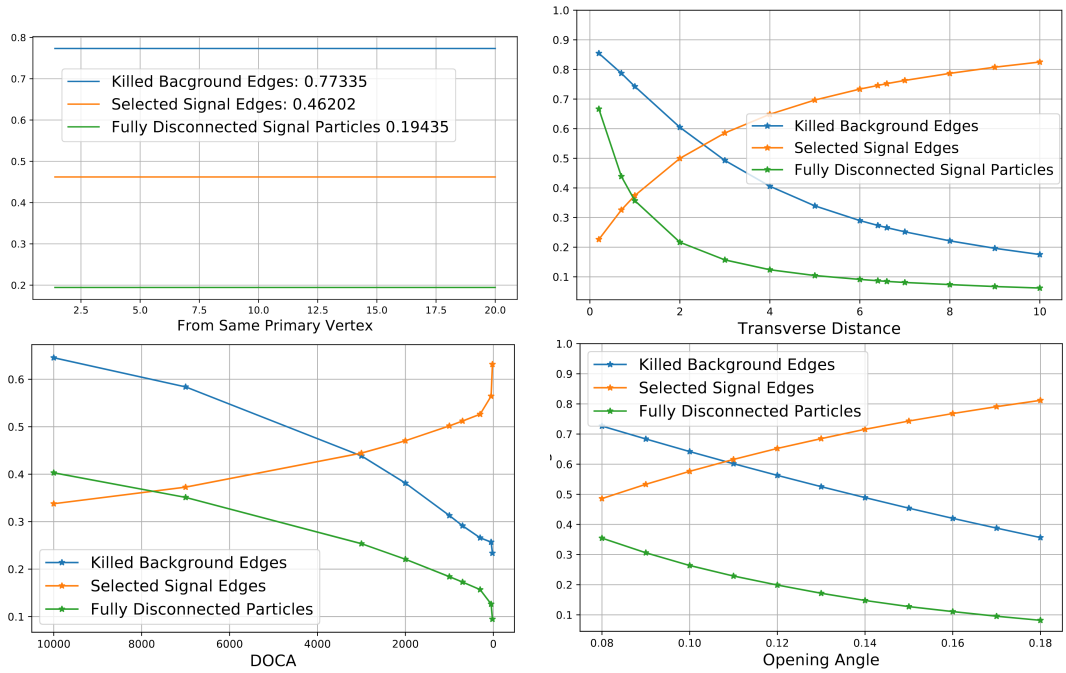


Figure 4.15: 1D cuts of the fast inclusive simulation with 6153 events. From Same Primary Vertex (top left) Transverse Distance (top right), DOCA (bottom left), Opening Angle (bottom right).

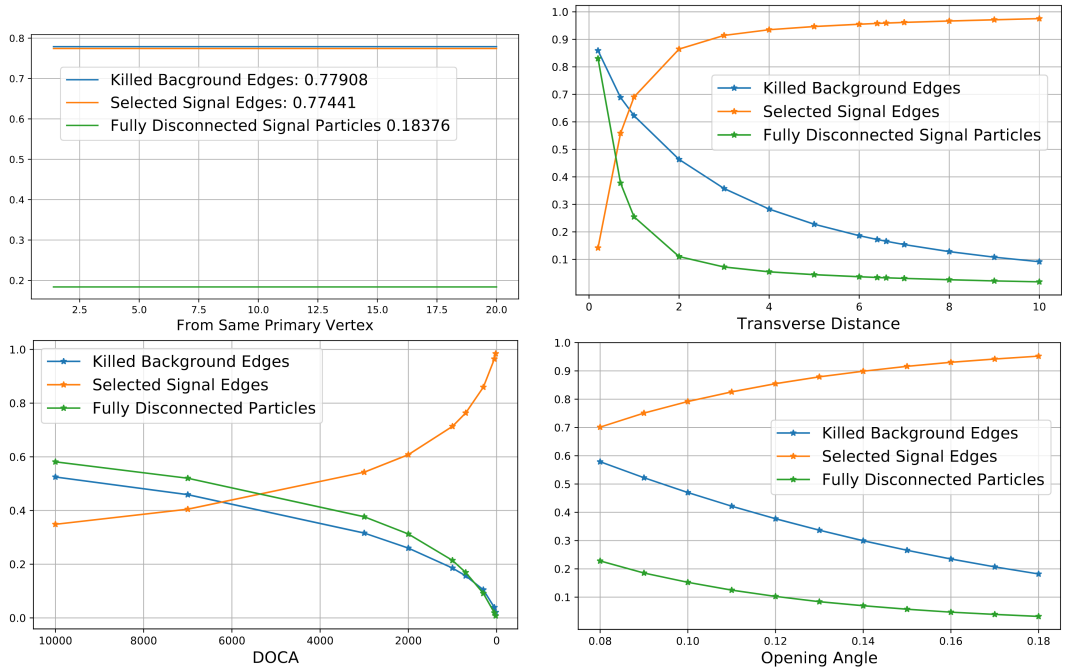


Figure 4.16: 1D cuts of the full inclusive simulation with 10066 events. From Same Primary Vertex (top left) Transverse Distance (top right), DOCA (bottom left), Opening Angle (bottom right).

	Cut	FDSP (%)	KBE (%)	Cut Value
PYTHIA	*Same PV*	19.4	77.3	—
	TrDist	20.9	59.7	2 mm
	DOCA	21.2	38.4	$2 \cdot 10^2$
	Theta	20.3	58.0	0.12 rad
Full inclusive	*Same PV*	18.4	77.9	—
	TrDist	19.8	48.2	1.7 mm
	DOCA	18.1	17.4	$2.2 \cdot 10^4$ mm
	Theta	19.2	52.3	0.09 rad

Table 4.6: 1D Cuts both for fast (6153 events) and full (10066 events) simulations where FDSP are the number of Fully Disconnected Signal Particles while KBE are the number of Killed Background Edges.

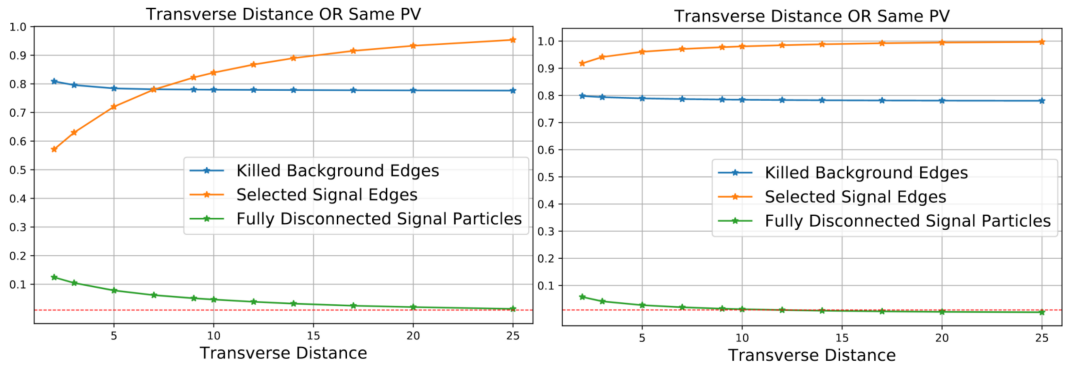


Figure 4.17: 2D cut of transverse distance OR *SamePV* of the fast simulation (left) and the full simulation (right).

into account are only the transverse distance and the opening angle since the distance of closest approach (DOCA) had bad performance in the 1D cut, as it it shown in table 4.6.
The threshold for the 2D cut was determined by the Same PV association or by a threshold of one of the other variables. Various thresholds were tested for each 2D cut.

The results of the 2D cuts for the transverse distance and the opening angle are shown in figures 4.17 and 4.18 for both the fast and full simulations. In these figures, the red dotted line indicates where the percentage reaches 1%.

In order to compare the results, the performance of the Killed Background Edges has been evaluated at a number of Fully Disconnected Particles close to 1% and the results are presented in table 4.7.

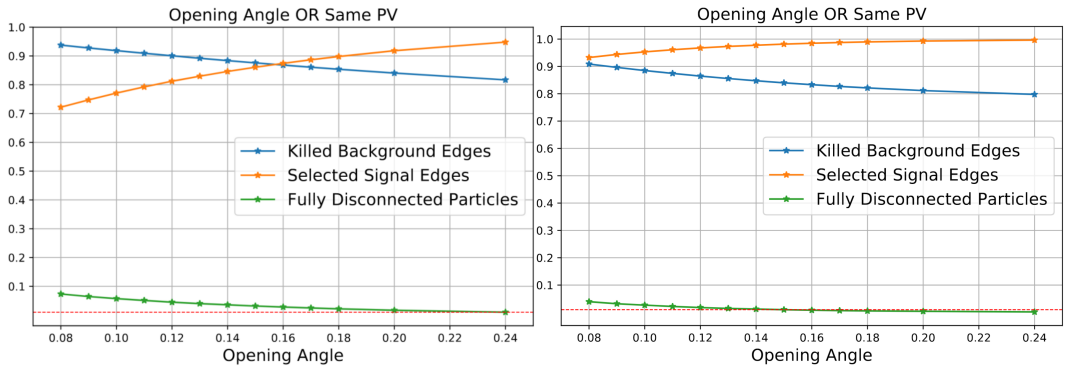


Figure 4.18: 2D cut of opening angle OR *SamePV* of the fast simulation (left) and the full simulation (right).

	Cut	**FDSP** (%)	**KBE** (%)	**Cut Value**
PYTHIA	*TrDist or Same PV*	1.0	77.8	25 mm
	Theta or Same PV	1.0	81.5	0.24 rad
Full inclusive	*TrDist or Same PV*	1.0	78.8	10 mm
	Theta or Same PV	1.0	84.6	0.14 rad

Table 4.7: 2D Cuts both for fast (6153 events) and full (10066 events) simulations where FDSP are the number of Fully Disconnected Signal Particles and KBE are the number of Killed Background Edges.

The performance of the 2D cuts is significantly better than the 1D cuts. The best 2D cut is the *Theta* OR *SamePV* cut, which has been chosen as the edge pre-filtering cut for both the fast and full simulations.

Chapter 5

Fast Simulation Performance

This chapter focuses on the performance evaluation of the fast simulation. For performance evaluation, the DFEI output is compared with true values obtained from Monte Carlo simulations, using the perfect reconstruction efficiency as one of the metrics.

The performance evaluation was done both per signal and per event, and an analysis was done to differentiate the reconstruction of a particular b-hadron decay tree based on the final-state particle keys and lowest common ancestors (LCAs). Four variables were defined: Perfect Signal Reconstruction, Quasi Signal Reconstruction, Not-Isolated Signal Reconstruction, and Partial Signal Reconstruction.

All the quantities mentioned above have been computed for Pythia-based $b\bar{b}$ inclusive samples of 50K events and for various Pythia-based exclusive samples.

The edges preselection discussed in section 4.3 is now the nominal one applied also to the Pythia samples (so the nominal dfei was also retrained).

5.1 Performance classification

This section details the methodology employed to evaluate the performance of the DFEI algorithm. The aim is to compare the output of DFEI with the true values obtained from Monte Carlo simulations

After training the algorithm on the fast simulation, it was tested on a Pythia-based simulation with Run 3 conditions comprising of 49,399 events.

To assess the effectiveness of the DFEI algorithm, various metrics can be used. One of the commonly used metrics is the perfect signal reconstruction

efficiency, which is defined as the ratio of the number of perfectly identified b hadron decays to the total number of b hadron decays in the simulation sample.

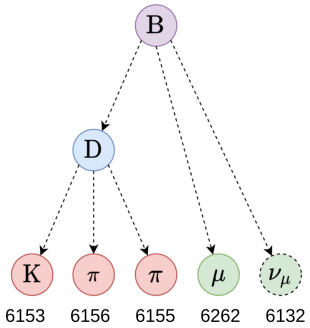


Figure 5.1: Decay tree of the B^0 meson. The numbers at the bottom indicate the indexes of the final state particles

The output of DFEI per event is a Python dictionary that stores the predicted decay tree reconstruction of b hadrons. The dictionary contains information about the keys (or indexes) of the final state particles and their lowest common ancestor (LCA), which represents a unique way to distinguish a specific b hadron decay. An example of a decay tree in the DFEI output dictionary, called *reco_cluster_dict*, is shown in Figure 5.1 and given by:

$$\{6153 : \{\text{'}node\_keys\text{'} : [6153, 6155, 6156, 6262], \text{'}LCA\text{'} : [1, 1, 2, 1, 2, 2]\}\} \quad (5.1)$$

	K	π	π	μ
K	0	1	1	2
π	1	0	1	2
π	1	1	0	2
μ	2	2	2	0

Figure 5.2: LCA matrix for the $B^0 \rightarrow D^-[K^+\pi^-\pi^-]\mu^+\nu_\mu$

The dictionary key is the lowest index (e.g., 6153), the '*node_keys*' are the sorted indexes of the final state particles, and the '*LCA*' are the ordered lowest common ancestors between pairs of particles (e.g., the first $LCA = 1$ is the LCA between the kaon 6153 and the pion 6155). It is important to note that the neutrino is not detected and therefore is not considered in the output. The LCA matrix of this decay, is shown in figure 5.2.

To evaluate the performance of the DFEI algorithm, the *reco_cluster_dict* is compared with the dictionary of real values, called *true_cluster_dict*, which has the same structure as the reconstructed dictionary. This comparison is performed per event, and the B hadron decay trees are referred to as "signal". By comparing the B decay trees reconstructed by DFEI with the true ones obtained from Monte Carlo simulations, it is possible to evaluate the efficiency of finding a perfectly reconstructed B chain. This occurs when both the '*node_keys*' and the '*LCA*' match perfectly. The evaluation is carried out both per signal and per event, with the latter occurring when every signal in the event is perfectly reconstructed and all the background particles are removed.

The evaluation results are presented in terms of two boolean variables: *Perfect Event Reconstruction* (PER) and *Perfect Signal Reconstruction* (PSR). PER contains entries equal to the number of events in the sample, while PSR contains one entry for each b hadron in the sample, resulting in a total number of entries nearly 1.5 times the number of events. The percentage of PER and PSR for the Pythia-based simulation sample of 49,399 events are presented in Table 5.1.

Perfect Event Reconstruction:	2.14 % ± 0.07%
Perfect Signal Reconstruction:	4.55% ± 0.08%

Table 5.1: Percentage of perfectly reconstructed events and signals in the fast simulation with 49399 events.

Given the achieved efficiency of 4.5% in perfectly reconstructed signals, it is of interest to explore where the DFEI algorithm falls short in accurately reconstructing b decay trees. In order to do so, a further analysis was conducted to distinguish different ways in which a particular B decay tree is reconstructed based on the final state particle keys and lowest common ancestors (LCAs).

Focusing solely on signal reconstruction, four distinct classifications were established:

- **Perfect Signal Reconstruction**: The particle keys and LCAs of the reconstructed signal perfectly match the true values.
- **Quasi Signal Reconstruction**: The quasi-signal reconstruction pertains to decays that have the same particles in the final state but has a different decay tree structure (LCA) with respect to the target decay.
- **Not-Isolated Signal Reconstruction**: The Not-Isolated signal reconstruction pertains to decays in which all the signal particles are contained in the reconstructed cluster but, in the same reco cluster, it also contains other (background) particles.
- **Partial Signal Reconstruction**: Any other cases that do not fall into the above categories. This is the only type of reconstruction in which some of the signal-decay particles have been missed. So the only one that would actually be problematic for trigger purposes, where one would want to make sure that the relevant particles are retained.

Each of these classifications is a boolean variable and is only true for one category while being false for the other categories. It is not possible for more than one of them to be true at the same time.

Perfect Signal Reconstruction:	4.5% ± 0.1%
Quasi Signal Reconstruction	5.9% ± 0.1%
Not-Isolated Signal Reconstruction	76.0% ± 0.2%
Partial Signal Reconstruction	13.4% ± 0.1%

Table 5.2: Percentage of perfect, quasi, not-isolated and partial signal reconstruction in the fast simulation with 49399 events.

To have a better understanding of the mentioned performance classification, a plot of every possible reconstruction made by DFEI was created, specifically, the *truth_cluste_dict* and the *reco_cluster_dict* were compared. Considering the $B_s^0 \rightarrow D_s^-[K^-K^+\pi^-]\pi^+$, an example of PSR is shown in figure 5.3, of Quasi Signal Reconstruction in figure 5.4, of Not-Isolated Signal Reconstruction in figure 5.5 and of Partial Signal Reconstruction in figure 5.6.
In these figures it can be also seen how precisely DFEI correctly selects the signal particles both on x-z and y-z axis in the VELO region.

Table 5.2 shows the classification of signals reconstruction giving also the percentage of perfect, quasi, not-isolated and partial signal reconstruction in the fast simulation with 49399 events.

Regarding not-isolated signal reconstruction, which involves reconstructing a decay in which in addition to the signal particles there are also other background particles, it was also evaluated the number of signal particles reconstructed by DFEI. Figure 5.7 shows a histogram illustrating the distribution of the number of signal particles in the not-isolated case.

The distribution of the number of signal particles not-isolated reconstructed by DFEI, as shown in Figure 5.7, exhibits a Gaussian shape with a long tail. The mean and standard deviation of the distribution are 10.4 ± 4.1, indicating that, on average, a substantial number of final state particles from a b hadron decay tree are reconstructed by DFEI in the Not-Isolated case.

To further compare the performance of DFEI, it was evaluated the number of signal particles that were only not-isolated reconstructed by the algorithm, i.e., cases where DFEI identified all the signal particles plus some background ones. To assess this, it is compared the number of not-isolated reconstructed particles with the correct number of signal particles that should have been reconstructed. The results are shown in Figure 5.8, which also includes the number of quasi and perfectly reconstructed particles for comparison.

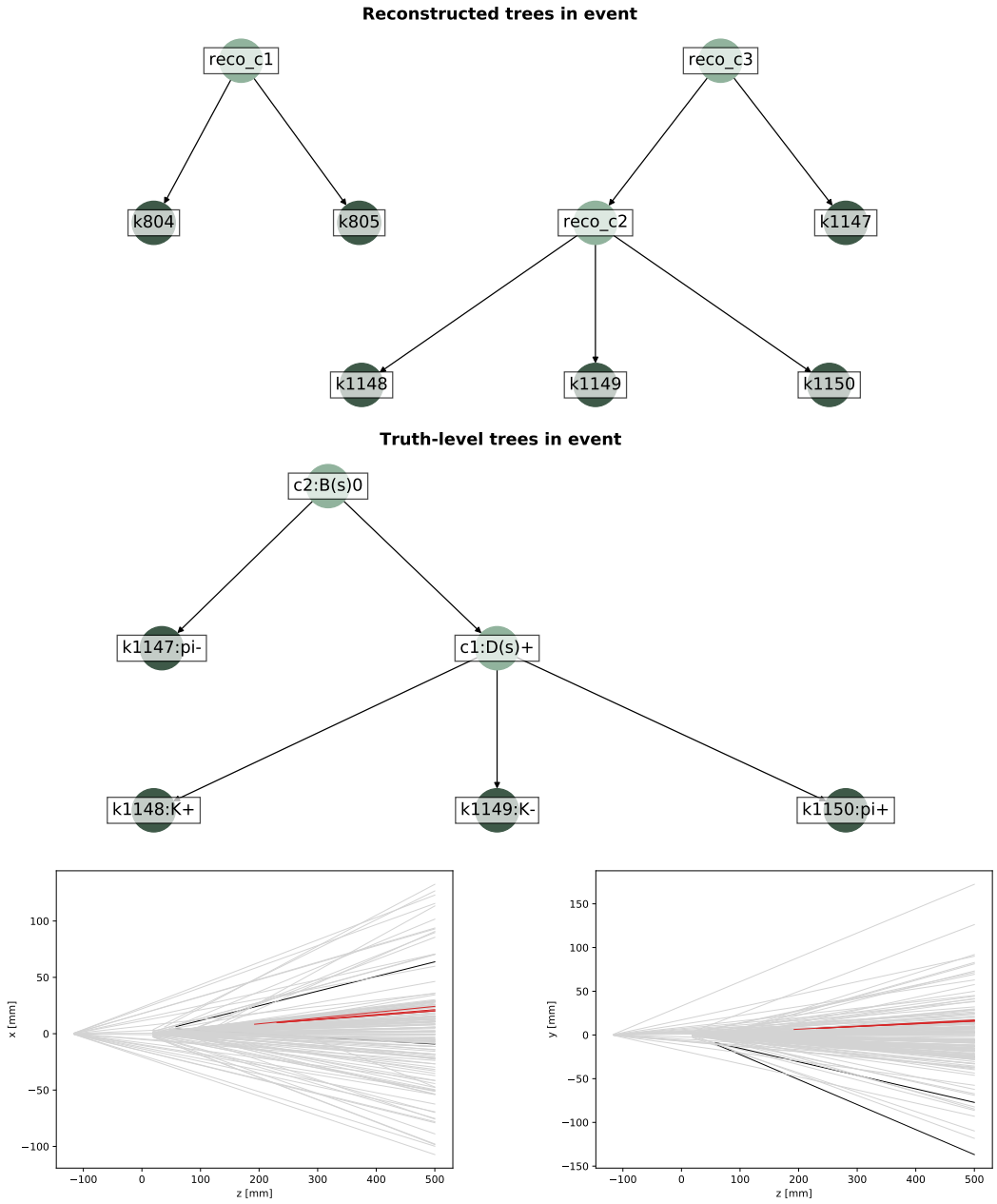


Figure 5.3: Example of Perfect Signal Reconstruction of DFEI considering the $B_s^0 \to D_s^-[K^-K^+\pi^-]\pi^+$ decay. On the top are shown the true (down) and reconstructed (up) signals in the event while on the bottom the selected signal particle (red), the selected background particles (black) and the not-selected background particles(gray) both on the x-z axis (left) and y-z axis (right).

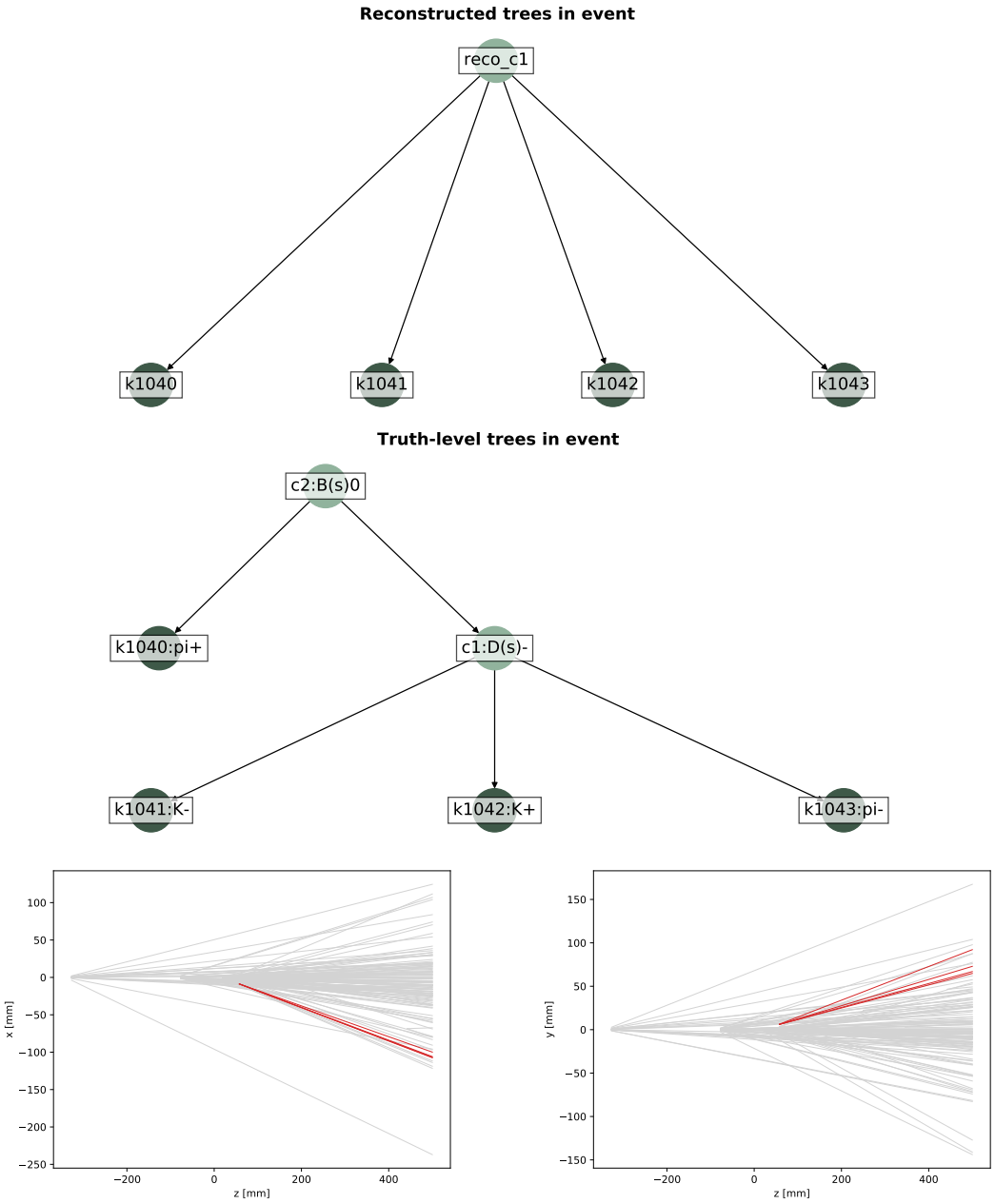


Figure 5.4: Example of Quasi Signal Reconstruction of DFEI considering the $B_s^0 \to D_s^-[K^-K^+\pi^-]\pi^+$ decay. On the top are shown the true (down) and reconstructed (up) signals in the event while on the bottom the selected signal particle (red), the selected background particles (black) and the not-selected background particles(gray) both on the x-z axis (left) and y-z axis (right).

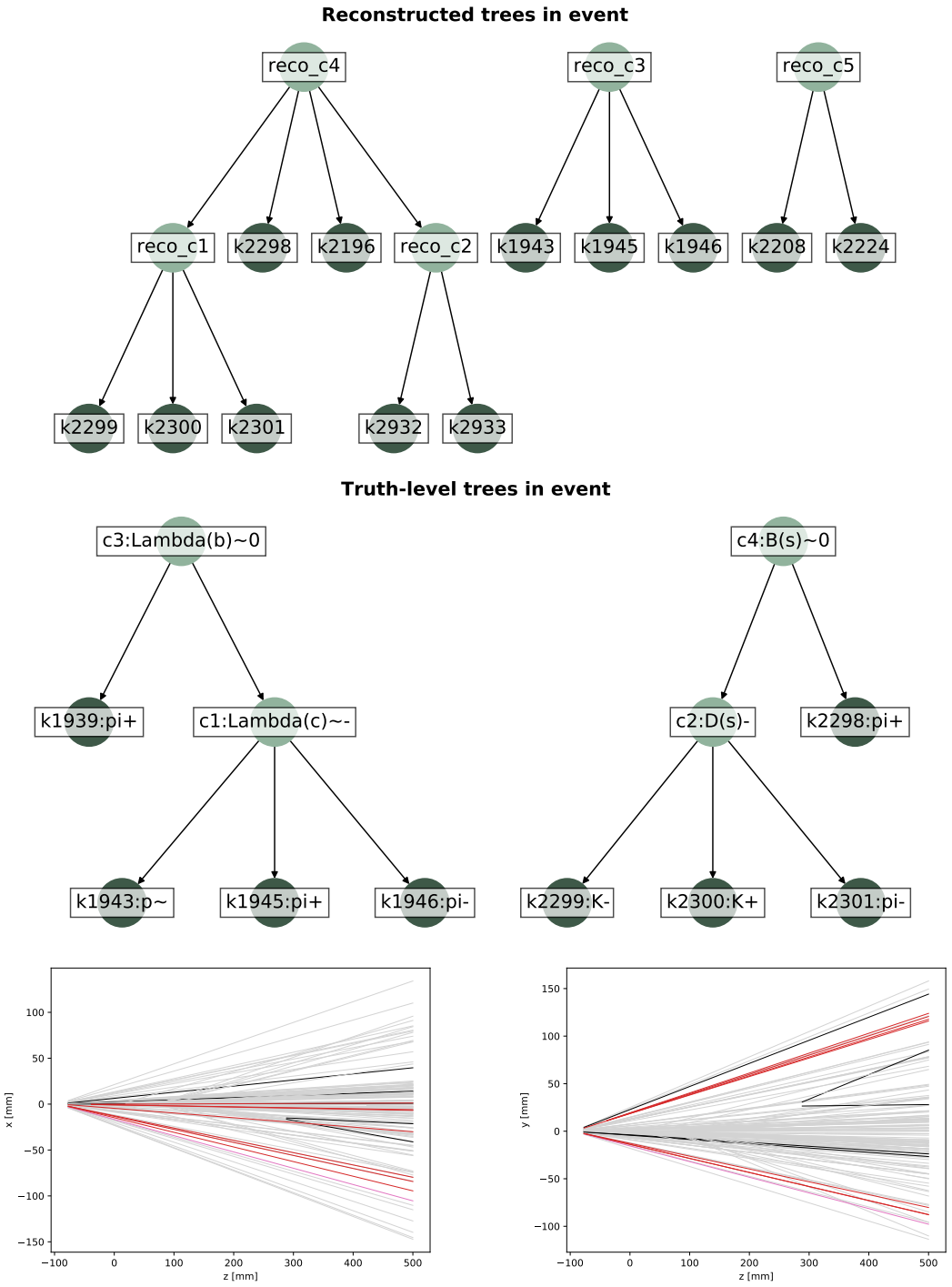


Figure 5.5: Example of Not-Isolated Signal Reconstruction of DFEI considering the $B_s^0 \to D_s^-[K^-K^+\pi^-]\pi^+$ decay. On the top are shown the true (down) and reconstructed (up) signals in the event while on the bottom there are: selected signal particles (red), not-selected signal particles (pink), selected background particles (black), not-selected background particles (gray). Here the selected background particles correspond to the wrongly reconstructed signal both on the x-z axis (left) and y-z axis (right).

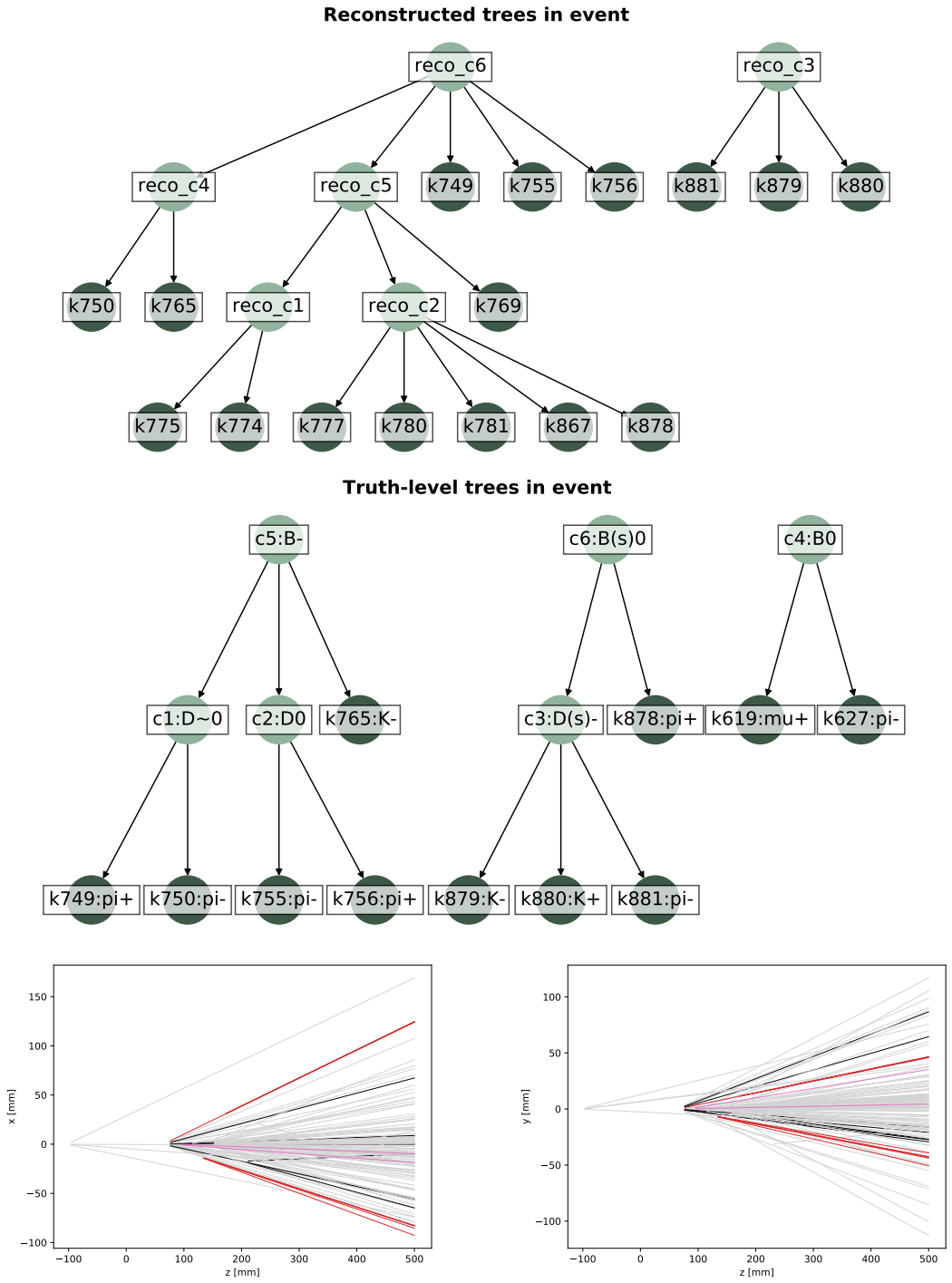


Figure 5.6: Example of Partial Signal Reconstruction of DFEI considering the $B_s^0 \to D_s^-[K^-K^+\pi^-]\pi^+$ decay. On the top are shown the true (down) and reconstructed (up) signals in the event while on the bottom there are: selected signal particles (red), not-selected signal particles (pink), selected background particles (black), not-selected background particles (gray). Here the selected background particles correspond to the wrongly reconstructed signal both on the x-z axis (left) and y-z axis (right).

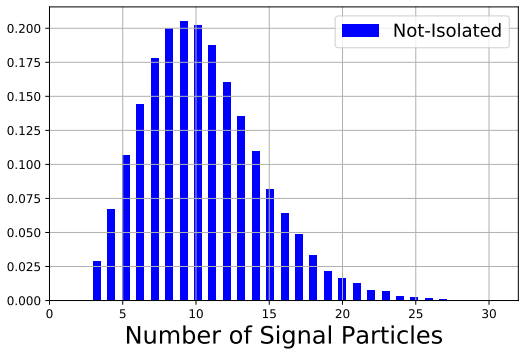


Figure 5.7: Number of signal particles reconstructed by DFEI in the Not-Isolated Signal Reconstruction in the fast simulation with 49399 events.

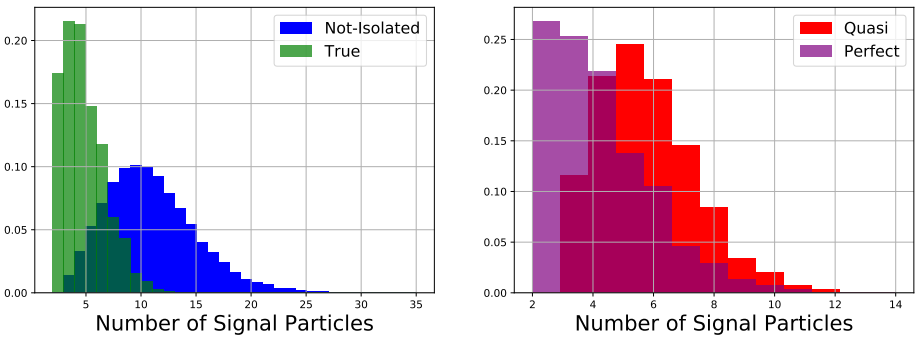


Figure 5.8: On the left is plotted the number of signal particles not-isolated reconstructed by DFEI (blue) compared with the correct number of signal particles (green). On the right is plotted the number of signal particles both quasi (red) and perfectly (purple) reconstructed by DFEI. It was used a fast simulation sample with 49399 events.

5.2 Performance on the inclusive $b\bar{b}$ sample

After evaluating the overall performance of DFEI on all the signals, the next step was to analyze the performance of DFEI on specific decay trees.

To evaluate the performance of DFEI on specific B decay trees, the Truth Matching algorithm explained in section 4.2 was used to identify the true decays in the inclusive sample of 50 000 events. By selecting only one channel at a time, the algorithm allowed for an evaluation of DFEI's performance independently per B decay.

A classification was done in terms of Perfect, Quasi, Not-Isolated and Partial reconstruction, which was restricted to the specific decay. To ensure that only signals with the correct number of visible final state particles were selected, it was required that the number of signal particles was at least equal to the expected number of particles in the specific decay. This requirement assures that only the correct signals are taken into account. The results of this analysis are shown in table 5.3.

	Events	**PSR** (%)	**Quasi** (%)	**Not-Isolated** (%)	**Partial** (%)
$B^+ \to \bar{D}^0[K^+\pi^-]\mu^+\nu_\mu$	97	3.6 ± 1.7	5.4 ± 2.1	91.1 ± 3.0	0.0 ± 0.0
$B^+ \to \bar{D}^0[K^+\pi^-]e^+\nu_e$	123	6.0 ± 2.1	3.8 ± 1.6	90.2 ± 2.6	0.0 ± 0.0
$B^0 \to D^-[K^+\pi^-\pi^-]\mu^+\nu_\mu$	31	3.6 ± 3.5	3.6 ± 3.5	92.9 ± 4.9	0.0 ± 0.0
$B^0 \to D^-[K^+\pi^-\pi^-]e^+\nu_e$	40	7.5 ± 4.2	5.0 ± 3.4	87.5 ± 5.2	0.0 ± 0.0
$B^0 \to K^+\pi^-$	14	4.5 ± 4.4	0.0 ± 0.0	95.4 ± 4.4	0.0 ± 0.0

Table 5.3: Percentage of perfect, quasi, Not-Isolated and partial signal reconstruction in the fast simulation with 49399 events.

Table 5.3 shows that the majority of the specific B decay signals are classified as not-isolated, with few perfect and quasi reconstructed decays. In addition, the errors associated to the few perfectly and quasi reconstructed signals are huge giving no clue of what would be an estimation number of perfect and quasi signals.
It should be notice that the partial signal reconstruction is zero in each decay, meaning that zero signal particles are lost in the DFEI reconstruction.

The performance evaluation of specific b decay trees includes the reconstruction of the invariant mass of the mother b hadron. Since the number of events which are perfectly reconstructed are either few with huge errors or zero, the mass reconstruction makes no sense in this case.

5.3 Performance on exclusive samples

After evaluating the overall performance of DFEI on the inclusive $b\bar{b}$ sample consisting of 50k events, which revealed the limited number of specific b decay tree signals for conducting a comprehensive analysis, exclusive Pythia-based samples were utilized. These samples contained 5000 events each.

Similar to the approach taken for the inclusive $b\bar{b}$ sample, the percentages of perfect, quasi, not-isolated, and partial signal reconstruction were computed and presented in table 5.4.

	PSR (%)	**Quasi** (%)	**Not-Isolated** (%)	**Partial** (%)
$B^0 \to K_0^*[K\pi]\mu^+\mu^-$	35.8 ± 0.7	19.2 ± 0.6	44.9 ± 0.7	0.0 ± 0.0
$B^0 \to K^+\pi^-$	38.0 ± 0.7	0.0 ± 0.0	54.7 ± 0.7	7.2 ± 0.4
$B_s^0 \to D_s^-[K^-K^+\pi^-]\ \pi^+$	32.8 ± 0.7	7.1 ± 0.4	53.7 ± 0.8	6.4 ± 0.4
$B^+ \to K^+K^-\pi^+$	35.7 ± 0.7	10.2 ± 0.4	46.4 ± 0.7	7.7 ± 0.4
$\Lambda_b^0 \to \Lambda_c^+[pK^-\pi^+]\ \pi^-$	21.7 ± 1.0	8.9 ± 0.7	36.8 ± 1.2	32.6 ± 1.1
$B_s^0 \to J/\psi[\mu^+\mu^-]\ \phi[K^+K^-]$	26.9 ± 0.6	20.5 ± 0.5	52.5 ± 0.6	0.0 ± 0.0

Table 5.4: Percentage of perfect, quasi, not-isolated and partial signal reconstruction considering various exclusive decays of the fast simulation.

The percentage of PSR in the exclusive Pythia-based samples is significantly higher compared to the specific decays observed in the inclusive $b\bar{b}$ sample. This is due to two factors: in the previous sample, with too few events one could not evaluate a meaningful percentage, but also in the inclusive $b\bar{b}$ sample there might have been also topologies more difficult to reconstruct than these specific decay modes.
Notably, given the error, the four types of classification numbers can give a good estimation of how DFEI reconstructs a decay. The percentage of perfect signal reconstruction exceeds 30% in most decays, leading to an average PSR of $31.8\% \pm 5.7\%$. The ^0_b decay has the lower percentage of PSR that it might be due to the fact that Λ_c has a much shorter lifetime than D mesons.

In the exclusive decay analysis, the percentages of not-isolated signal reconstruction still surpass those of perfect and quasi reconstruction. However, this percentage is notably smaller compared to the $b\bar{b}$ inclusive case. In the majority of the decays, the percentage of not-isolated reconstruction consistently exceeds that of partial reconstruction, indicating a higher accuracy of the DFEI algorithm in reconstructing the decay tree. This enhanced efficiency is also reflected in the increasing percentages of perfect and quasi signal reconstruction.
This results give a more precise estimation of the performance of the DFEI

reconstruction.

It should be noticed the cases of the $B^0 \to K_0^* \mu^+ \mu^-$ and the $B_s^0 \to J/\psi\ \phi$ decays in which the partial signal reconstruction is outstandingly zero. This is due to a pretty well displaced vertex with 4 tracks coming from the same point so the chances for DFEI to miss some of the particles are very rare, making the 0% of partial cases believable. In contrast, considering the $B^0 \to K^+ \pi^-$ it is much easier to miss a cluster of just two particles, which can be confused with clusters from the rest of the event.

As well as for the inclusive $b\bar{b}$ sample, the extra particles in the not-isolated case were studied. Figure 5.9 shows the number of extra particle reconstructed in the not-isolated case for most of the exclusive decays of the fast simulation.

The histogram distribution shown in figure 5.9 exhibits an evident exponential decay. This is in line with the expected behavior of the DFEI algorithm, which is more likely to reconstruct clusters in the Not-Isolated case with few additional particles besides the signal ones.

In the exclusive sample, the invariant mass of the perfectly reconstructed signals was also reconstructed. Ideally, the invariant mass of the mother particle should be almost perfectly reconstructed. The invariant mass reconstruction for each sample is illustrated in figure 5.10. For decays featuring intermediate particles, the invariant mass of these particles was also reconstructed.
Regarding the $B^0 \to K_0^*[K\pi]\mu^+\mu^-$ and the $B_s^0 \to J/\psi\ \phi$ decays, since the K_0^*, J/ψ and ϕ have a extremely short lifetime (of the order of picoseconds), the DFEI algorithm does not reconstruct them. All the daughter particles appear to have the same LCA, which is the b hadron. Therefore only the mother B mass has been reconstructed.
As expected, the invariant mass reconstruction peaks at the mass of the mother particle and/or at the masses of the intermediate particles.

In the case of the $B \to K^* \mu\mu$ decay, it has been compared the number of background particles in the not-isolated signals with respect to the total distribution of background particles coming from the same primary vertex ($FromSamePV = 1$) per event. This was done in order to evaluate how many background particles DFEI manages to remove from a PV.
Then, these two quantities are further compared with the total number of particles per event. The latter was computed considering only stable charged particles inside the detector's geometric acceptance.

Figure 5.11 shows the distributions of background particles per event (blue), coming from the same PV (orange) and in the not-isolated cluster (green).

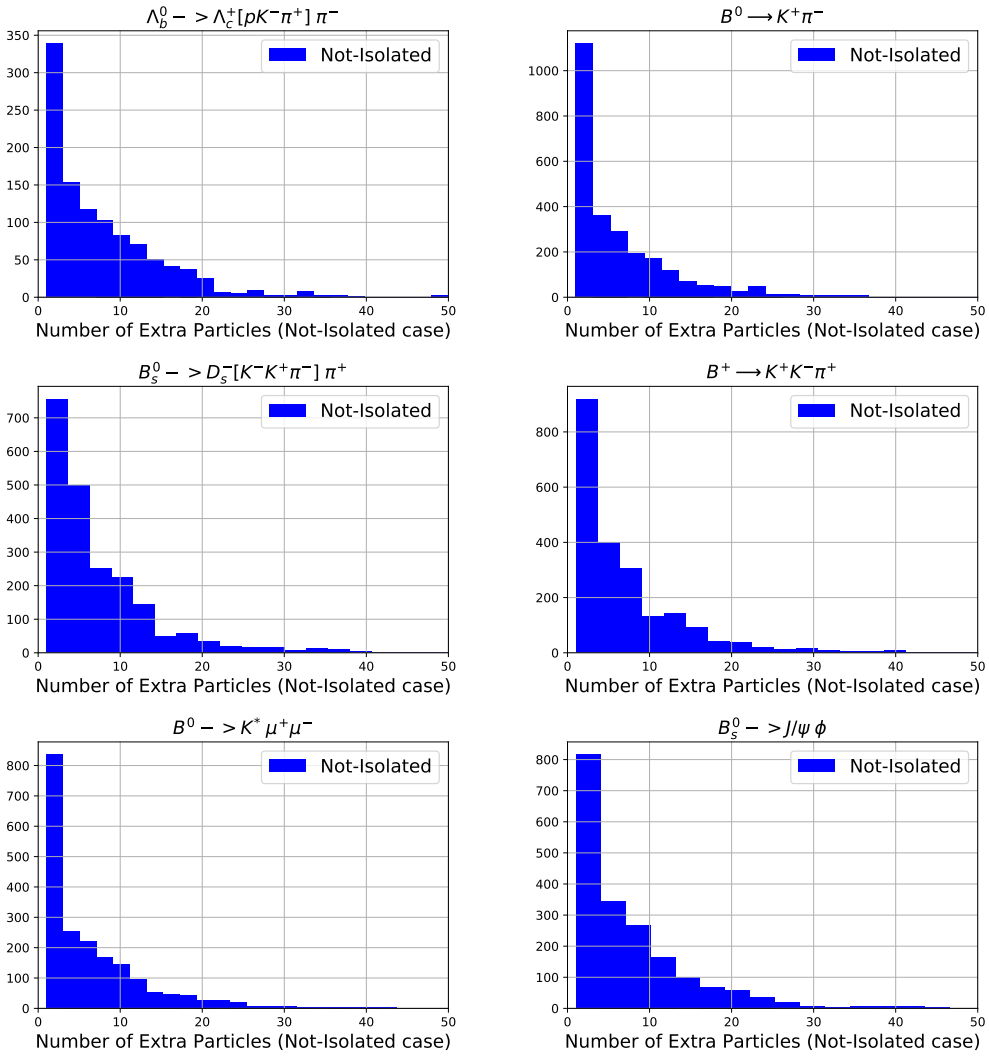


Figure 5.9: Number of extra signal particles reconstructed by DFEI in the Not-Isolated Signal Reconstruction for different exclusive fast simulation samples.

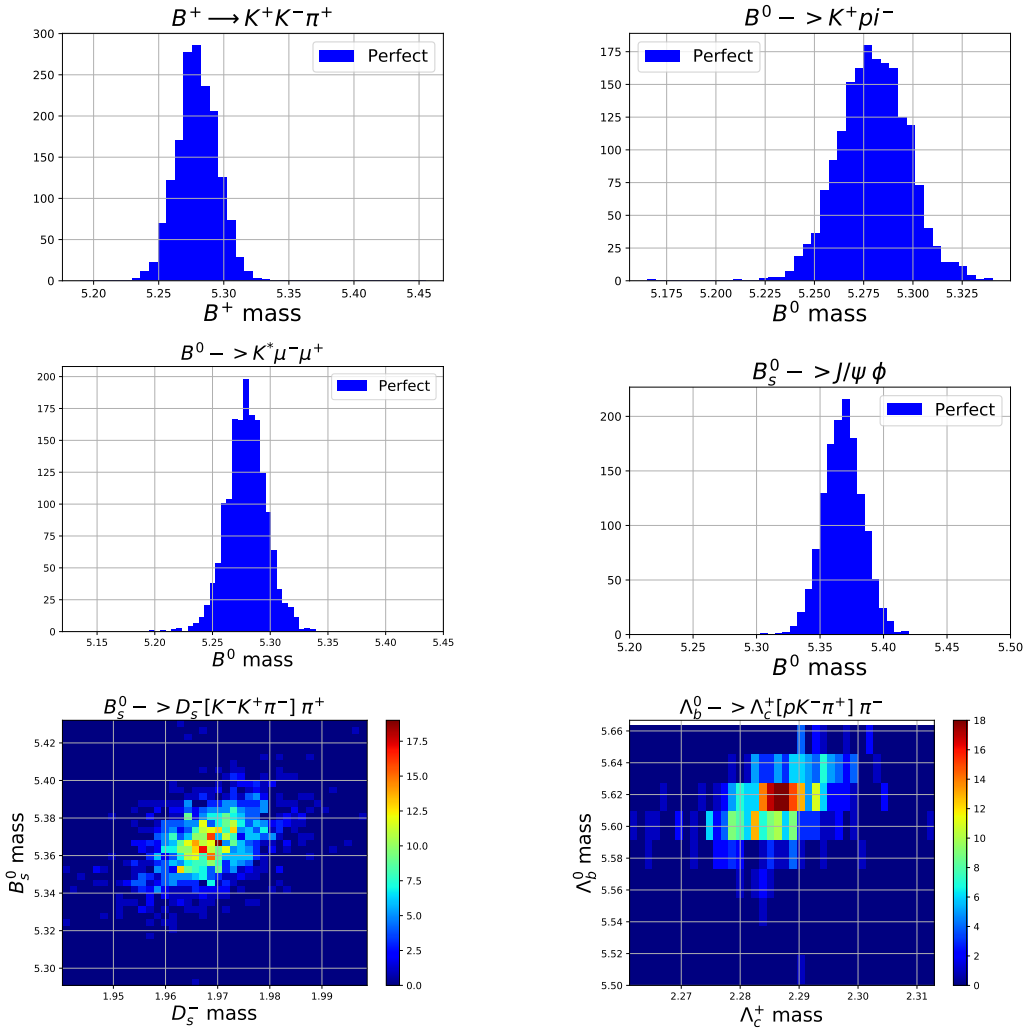


Figure 5.10: Invariant Mass reconstruction for perfectly reconstructed signals in the fast exclusive decays. In the 1D histograms, the mass of the mother b hadron was reconstructed, while in the 2D histograms, in addition to the reconstruction of the b hadron mass, the mass/masses of the intermediate particles were also reconstructed.

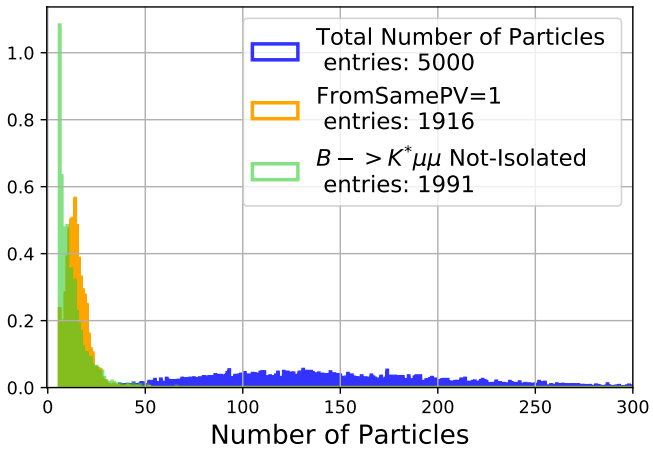


Figure 5.11: Distributions of background particles per event (blue), coming from the same PV (orange) and in the not-isolated cluster (green). Regarding the background particles from the same PV, it was considered only events with at least a $B^0 \to K^*\mu^+\mu^-$ decay removing the cases in which the background particles were zero.

From figure 5.11 starting from an average number of background particles per event of $\simeq 147$ and considering only particles associated to the same PV, one reduces to $\simeq 15$. DFEI has an average number of background particles, which are inside the not-isolated cluster, of $\simeq 11$. DFEI clearly selects less background particles with respect to the ones that come from the same PV.

Chapter 6

Training of the Full Simulation

This chapter focuses on the training of LCA inference module of the full simulation.
The chapter describes the re-optimisation and re-training of the DFEI algorithm on the full $b\bar{b}$ inclusive simulation.
The process of training a model involves determining optimal values for weights and biases using labeled examples. In supervised learning, a machine learning algorithm examines numerous examples and works to minimize the loss function in order to construct an effective model.

To eventually use the DFEI algorithm in the LHCb experiment, it needs to be trained and checked on simulated data which is as close as possible to real data. For this reason, the algorithm as been completely retrained in a full simulation sample, instead of the fast simulation one. This involves retraining the node-pruning, edge-pruning and LCA modules, and re-adjusting all the output thresholds for maximal performance in the new dataset.
The edges preselection discussed in section 4.3 is now the nominal one applied both to the fast and full simulation samples.

This chapter will detail the process of training the full simulation on inclusive $b\bar{b}$ samples to provide a comprehensive evaluation of performance for this type of simulation.
It also explains the use of ROC curves and AUC (area under the curve) as metrics for evaluating the performance of a classification model in extracting a signal from a much larger data set.
Different trainings were performed using various loss functions, such as the cross-entropy loss and focal loss.

6.1 ROC curves and AUC

Extracting a signal from a much larger background is a common challenge in most of the LHC measurements. In some cases, there may be a single feature powerful enough to distinguish the signal from the background.

When treating signal extraction as a classification problem, there is often an imbalance in the number of samples belonging to the signal class versus the background class. Using accuracy as a metric can be misleading since it does not provide information about the signal itself. For instance, a model that predicts every sample to be background would have high accuracy in the presence of significant data imbalance, but it would not be useful in identifying the signal. Thus, accuracy is not a recommended metric in such cases. The ROC curve is a better metric since it provides information about both true and false positive rates.

The ROC curve is a graph that shows the performance of a classification model at all classification thresholds, providing information about true and false positive rates across a range of thresholds [10]. It describes the false positive rate as a function of the true positive rate. The true positive rate measures how many true signal events have been identified as signal, while the false positive rate measures how many true background events have been identified as background.

Another useful single number metric is the AUC, or area under the ROC curve. The AUC measures the two-dimensional area underneath the entire ROC curve, from (0,0) to (1,1). It provides a single number that summarizes the overall performance of the classifier across all possible thresholds. A perfect classifier would have an AUC of 1, while a random classifier would have an AUC of 0.5. The closer the AUC is to 1, the better the performance of the classifier. A higher AUC indicates better model performance in distinguishing between classes of signals and background.

In summary, the ROC curve and AUC are useful tools for evaluating classification model performance in the presence of data imbalance. Figure 6.1 shows a typical ROC curve with true positive rate as a function of false positive rate.

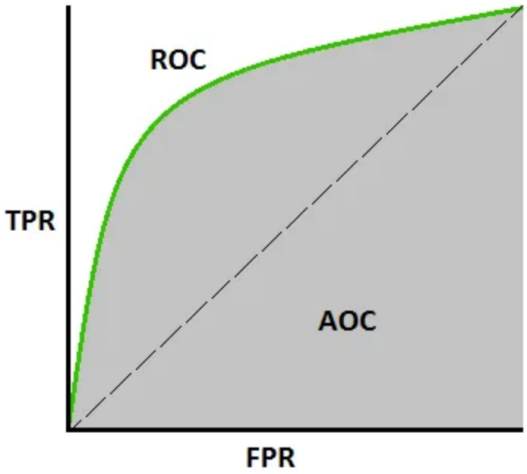


Figure 6.1: ROC curve and AUC (or AOC). The plot shows the true positive rate (TPR) as a function of the false positive rate (FPR).

The data sample comprises instances of two classes, signal (S) and background (B), with a total number of data points given by $N_{tot} = S_{tot} + B_{tot}$. The classes can be further divided into selected and rejected such that:

$$S_{tot} = S_{selected} + S_{rejected} \quad and \quad B_{tot} = B_{selected} + B_{rejected} \tag{6.1}$$

Discrete binary classifiers assign each instance to one of the two classes: classified as signal and selected ($N_{selected} = S_{selected} + B_{selected}$) or classified as background and rejected ($N_{rejected} = S_{rejected} + B_{rejected}$).

The quantities of interest are the true positive rate (TPR) and the false positive rate (FPR), which can be defined as:

$$TPR = \frac{TP}{TP + FN} \quad \text{and} \quad FPR = \frac{FP}{FP + TN} \tag{6.2}$$

Here, TP denotes the number of true positives (i.e., $S_{selected}$), FN denotes the number of false negatives (i.e., $S_{rejected}$), FP denotes the number of false positives (i.e., $B_{selected}$), and TN denotes the number of true negatives (i.e., $B_{rejected}$).

The ROC curves obtained after training the node pruning and edge pruning modules provide the necessary thresholds to be applied before training the LCA inference module.
To compute the ROC curves, the TPR and the FPR are required, where the former corresponds to the signal efficiency ϵ_s and the latter to the background efficiency ϵ_b:

$$\epsilon_s = \frac{S_{selected}}{S_{tot}} \quad \text{and} \quad \epsilon_b = \frac{B_{selected}}{B_{tot}} \tag{6.3}$$

Instead of the background efficiency it has been used the background rejection power, which is the true negative rate, . These quantities are related as follows:

$$\text{Background rejection} = 1 - \epsilon_b = TNR = 1 - FPR \tag{6.4}$$

The node pruning module's training yields an ROC curve, which allows us to identify an appropriate threshold that needs to be applied to the data for further analysis.
The ROC curve in figure 6.2 shows the performance of the node pruning module in terms of background rejection power signal versus the signal efficiency. As the threshold for selecting the signal class is increased, the true positive rate (signal efficiency) increases, while the true negative rate (background rejection power) decreases. The ROC curve illustrates the trade-off between these two quantities as the threshold is varied.

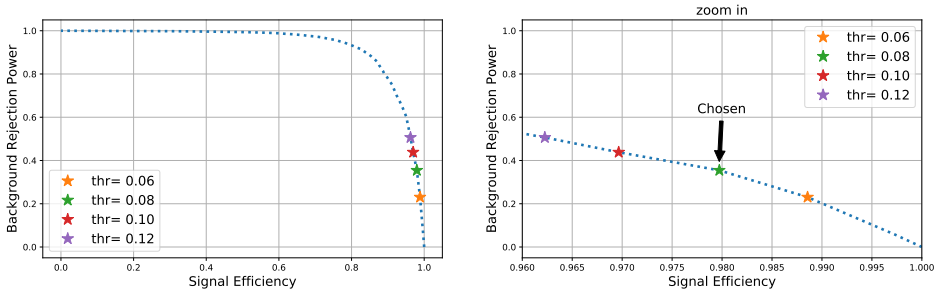


Figure 6.2: ROC curve of the node pruning module (left). On the x axis there is the signal efficiency while on the y axis the is the background rejection power. A zoom in the interested range of thresholds (right) shows the chosen one, which is 0.08.

The node pruning module is trained to identify a threshold that is applied to the data for further analysis. A threshold of 0.08 is chosen such that the signal efficiency is at least 97%, and the AUC of the node pruning is 0.93.

Similarly, the edge pruning module is trained using the threshold identified by the node pruning module, and the resulting ROC curve is shown in Figure 6.3, which describes the background rejection power as a function of the signal efficiency.

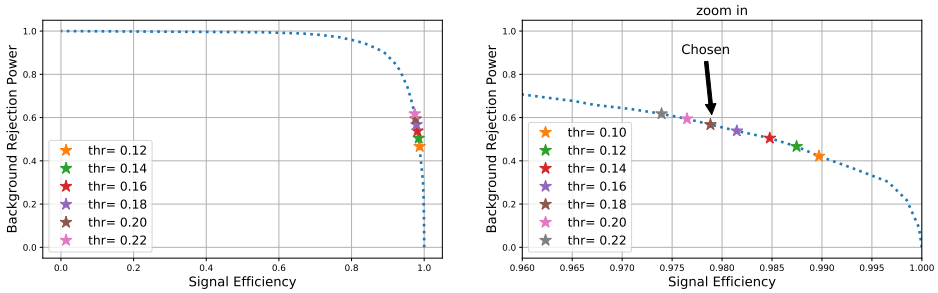


Figure 6.3: ROC curve of the edge pruning module (left). On the x axis there is the signal efficiency while on the y axis the is the background rejection power. A zoom in the interested range of thresholds (right) shows the chosen one, which is 0.18.

As stated earlier for the node pruning, a signal efficiency of at least 97% is desired, and the selected threshold for edge pruning is 0.18. The AUC for edge pruning is 0.96, indicating that the model's ability to distinguish between signal and background classes is excellent. These selected thresholds for both node and edge pruning will be employed in the LCA module's training.

6.2 Training with various Loss Functions

The LCA module of the full simulation was trained using the node and edge pruning thresholds. A learning rate of $5 \cdot 10^{-4}$ with $50,000$ iterations was used. In this training, the LCA module classified pairs of particles into four possible outcomes, namely zero, one, two, or three, depending on the value of the Lowest Common Ancestor (LCA) between the particles. The frequency of each LCA value was not the same, with LCA=0 being the most common, followed by LCA=2, LCA=1, and LCA=3 being the least common. Therefore, the class imbalance was addressed by selecting a proper loss function.

Regarding the loss function, in contrast to the training of the fast simulation, in which the Softmax Cross Entropy was used, in the full simulation training various loss functions have been tried.
The addition of hyperparameters to the loss function creates a relatively larger penalty for misclassifying an example. To start, it was considered the softmax cross entropy (CE) loss function:

$$CE = -\sum_{i=1}^{N} y_i \log p_i \quad where \quad p_i = \sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{6.5}$$

where y_i is the label/ground truth, which can take two values 1 or 0, p_i is the predicted probability (softmax) which varies between 0 to 1, N is the total

number of classes and x_i is the element of the input vector and it can take any real value.
The Balanced Cross-Entropy (BCE) loss adds a weighting factor to each class, represented by $\alpha \in [0, 1]$. α could be the inverse class frequency or a hyper-parameter determined by cross-validation. The alpha parameter replaces the actual label term in the Cross-Entropy equation as follows:

$$BCE = -\sum_{i=1}^{N} \alpha_i \log p_i \tag{6.6}$$

While this loss function addresses the issue of class imbalance, it cannot distinguish between hard and easy examples. The problems is solved using focal loss [11].
Focal loss focuses on the examples that the model gets wrong rather than the ones that it can confidently predict, ensuring that predictions on hard examples improve over time rather than becoming overly confident with easy ones. Focal loss (FL) achieves this through something called Down Weighting. Down weighting is a technique that reduces the influence of easy examples on the loss function, resulting in more attention being paid to hard examples. This technique can be implemented by adding a modulating factor to the Cross-Entropy loss:

$$FL = -\sum_{i=1}^{N} (1 - p_i)^{\gamma} \log p_i \tag{6.7}$$

where γ is the focusing parameter to be tuned using cross-validation. When $\gamma = 0$, it reduces to the cross-entropy function. The focusing parameter, $\gamma \geq 1$, will re-scale the modulating factor such that the easy examples are down-weighted more than the hard ones, reducing their impact on the loss function.

It was also used an α-balanced variant of the focal loss that inherits the characteristics of both the weighing factor α and the focusing parameter γ, yielding slightly better accuracy than the non-balanced form:

$$BFL = -\sum_{i=1}^{N} \alpha_i (1 - p_i)^{\gamma} \log p_i \tag{6.8}$$

Classification models are typically evaluated using accuracy, which measures how well the model performs across all classes. It is calculated as the ratio of correct predictions to the total number of predictions. While minimizing the loss function is important, maximizing accuracy is equally crucial, as it indicates the model's ability to make correct predictions. However, when different classes have varying rates of occurrence, accuracy may not be the

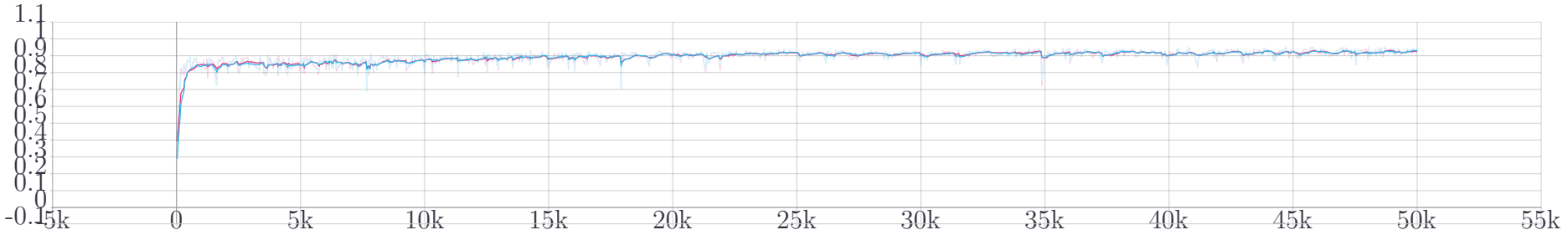
most suitable metric.

In the LCA module, different LCAs occur at different rates, so in addition to overall accuracy, accuracy was computed separately for each true LCA class. It was used, in addition to the balanced cross-entropy, the balanced focal loss function, which performed better than the non-balanced one.

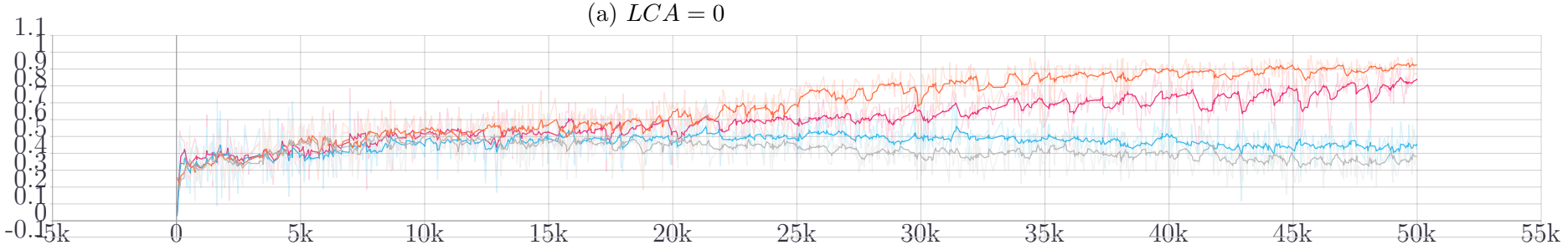
Figures 6.4 and 6.5 show the accuracy for each LCA classification, the loss function, and overall accuracy. The accuracy for $LCA = 0$ is good with both loss functions, likely due to the high frequency of $LCA = 0$ samples. However, for $LCA = 1$ and $LCA = 2$, the train and test accuracies diverge, with the focal loss function showing less divergence than the cross-entropy loss. The general divergence of $LCA = 2$ is better than $LCA = 1$ because the former occurs more often with respect to the latter. $LCA = 3$ is the least common LCA, and the algorithm performs poorly, with the loss functions for the test sample increasing instead of decreasing.

Surprisingly, the total accuracy of the train and test samples increases, despite the poor performance for $LCA = 1, 2, 3$. This is because the LCA=0 (background) cases are much more frequent than the $LCA = 1, 2, 3$ (signal), so the total accuracy is more influenced by the former.

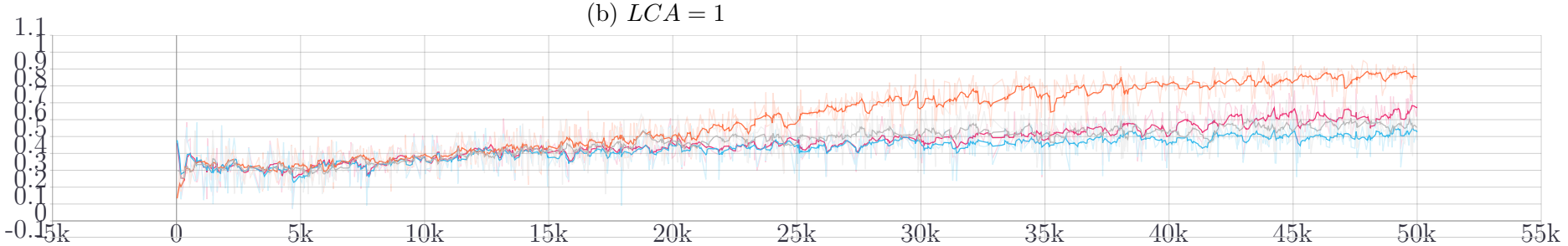
While the focal loss performs better than the cross-entropy loss, the training was not successful due to the scarcity of data for $LCA = 1, 2, 3$. Obtaining more data is the next step to properly train the DFEI algorithm on the full simulation. Therefore, it does not make sense to evaluate the performance of the model on the full simulation at this time.



(a) $LCA = 0$

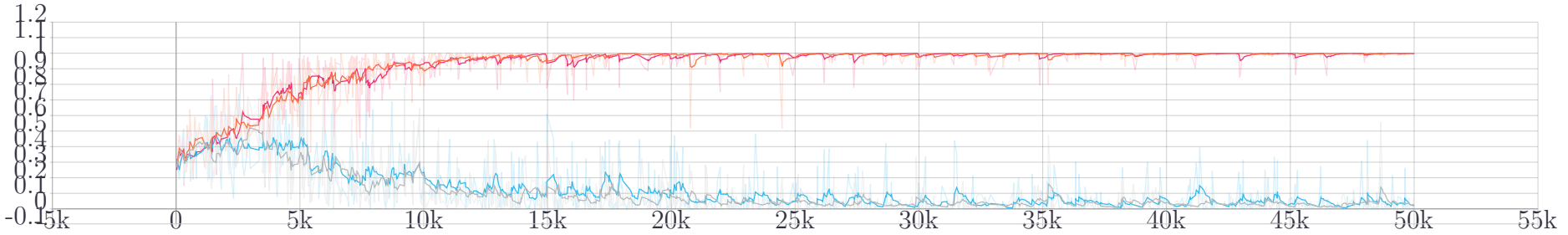


(b) $LCA = 1$

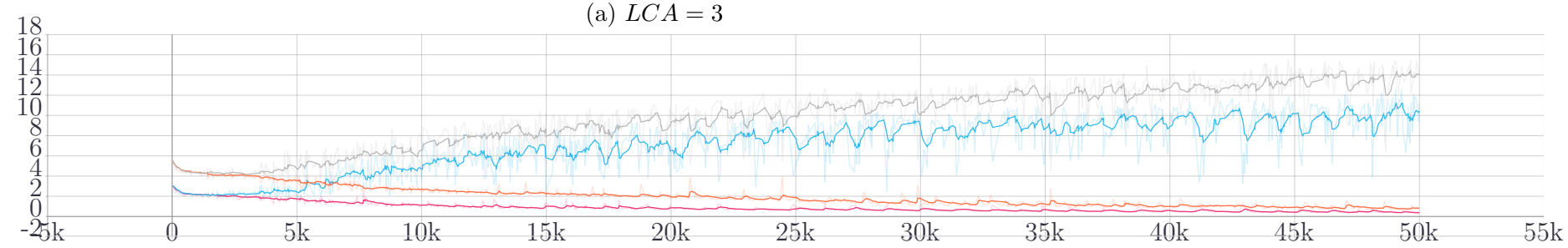


(c) $LCA = 2$

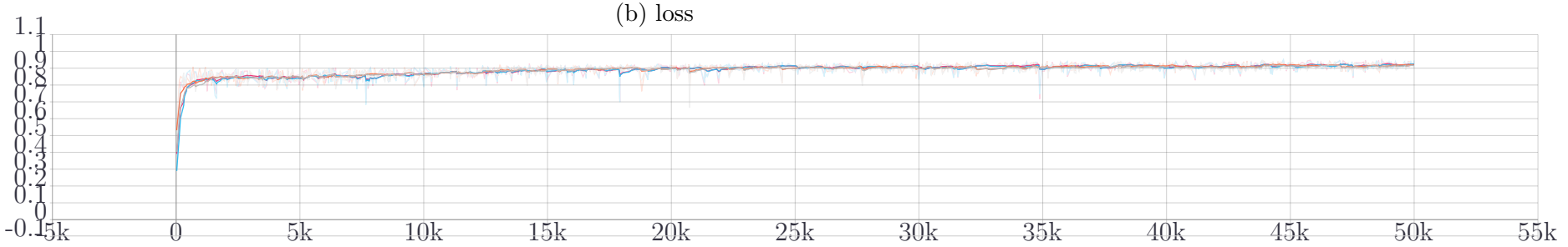
Figure 6.4: Accuracy for $LCA = 0$ (top), $LCA = 1$ (middle), $LCA = 2$ (bottom) both for the cross-entropy, with the Train sample (orange) and the Test sample (grey), and the balanced focal loss function with the Train sample (pink) and the Test sample (light blue).



(a) $LCA = 3$



(b) loss



(c) accuracy

Figure 6.5: $LCA = 3$ accuracy (top), loss function (middle) and total accuracy (bottom) both for the cross-entropy, with the Train sample (orange) and the Test sample (grey), and the balanced focal loss function with the Train sample (pink) and the Test sample (light blue).

Conclusion

This thesis has made significant contributions to the development of a novel machine-learning-based algorithm aimed to improve the future LHCb trigger system. The LHCb experiment has completed Upgrade I and is preparing for Upgrade II in a decade's time, which will increase the instantaneous luminosity by five and ten times, respectively. Such an increase in event complexity will pose unprecedented challenges to the online-trigger system, for which a solution needs to be found.
The thesis describes the construction of a Deep Full Event Interpretation (DFEI) algorithm, base on graph neural networks, that process in real time the final-state particles of each event, identifying which of them come from the decay of a beauty or charm heavy hadron and reconstructing the hierarchical decay chain through which they were produced.

The thesis expands the work by using both the PYTHIA8-based framework, which had already been utilized, and the Geant4-based full LHCb simulation for the first time. The development of a truth matching algorithm, which allows to select the interesting decay, was an important step of the work since it has been used in most of the subsequent sections. The further invariant mass reconstruction of the mother and intermediate particles assured the effectiveness of the truth-matching algorithm.

The present study describes the design and optimisation of the edge preselection step in DFEI, which is a crucial aspect of the study as it helps to eliminate a substantial number of edges connecting particles that do not come from the same B hadron while preserving almost all signal particles' connections. After investigating the effect on different variables, the best selection is obtained with a 2D cut of opening angle between particles' three-momentum or the same primary vertex association. Given the killed background edges, which are 81.5% for the fast simulation and of 84.6% for the full one, at 1% of fully disconnected signal particles, this pre selection incremented the accuracy with respect to the previous edge pre filtering cut. The following study based on determining how many particles are retained after applying the cut on the edges, clearly showed the efficiency of this edge pre selection.

The first detailed study of the DFEI performance on the Pythia-based simulation were performed. The performance evaluation is done both per signal and per event and the DFEI output is compared with true values obtained from Monte Carlo simulations, using the perfect-reconstruction efficiency as one of the metrics. After a general study on the inclusive $b\bar{b}$ sample, the first detailed study of DFEI performance focused on exclusive decay modes was implemented. While for the inclusive Pythia sample the percentage of perfect signal reconstruction is $\sim 4.5\%$, regarding the exclusive decay modes, the percentage increases, on average, to nearly $\sim 31.8\% \pm 5.7\%$.

The thesis re-optimises and re-trains the DFEI algorithm on the full inclusive simulation. The ROC curves study after the training of the node pruning and edge pruning modules gives the necessary thresholds to apply before starting the training of the LCA inference module. Various loss functions, such as cross-entropy loss and focal loss, were used for different trainings.
Even tough the accuracy curves show how the focal loss performs better than the cross-entropy, the training was not successful due to the scarcity of data for LCA = 1, 2, 3. Obtaining more data is the next step to properly train the DFEI algorithm on the full simulation.

Appendix A

Efficiency of pre selection cut

An assessment was conducted on the effectiveness of the chosen cut, which involves the angle between particles' three momentum or their association with the same primary vertex. The evaluation was performed by considering particles from the same heavy hadron as the signal, while the remaining particles were treated as background. The evaluation included determining the number of signal particles that were fully disconnected and the number of background edges that were eliminated.

Another way to assess efficiency is to focus on the particles themselves, rather than the edges. This involves determining how many particles are retained after applying the cut on the edges. A particle is retained if it is connected to at least one other particle through an edge. The signal comprises pairs of particles originating from the same primary vertex of the b hadron ($FromSamePV = 1$), while the remaining particles are considered background.

An algorithm was developed to evaluate the efficiency of the selected cut. The algorithm pre-selects edges based on the cut condition:

$$theta\_reco < thr \quad OR \quad FromSameAssociatedPV\_reco == 1$$

where:

- $theta\_reco < thr$ indicates that the opening angle between particles in the pair is less than $thr = 0.24$ radians for the fast simulation and $thr = 0.14$ radians for the full one.
- $FromSameAssociatedPV\_reco == 1$ implies that the pair shares the same primary vertex.

It is important to note that this algorithm is limited to cases where there are only two b hadrons. This is because the purpose of this paragraph is to

compare the efficiency of the edge pre-filtering cut mentioned earlier with the pre-selection cuts efficiency of long tracks in the study conducted by [5].

After applying the edge cut, the number of particles belonging to the same primary vertex that were saved by the cut was computed. These particles are known as the Saved Signal. A particle is considered saved if it is connected to at least one other particle by an edge. If all edges of a particle are removed by the cut, then that particle is considered lost, and the signal associated with it is lost as well. Therefore an high percentage of saved signal particles is expected.

The second variable of importance is the fraction of all other particles (not originating from the same primary vertex) that are removed by the edge cut. These particles are known as Lost Background. Similar to before, a particle is considered removed if the cut eliminates all of its edges. In this case, the focus is on background particles, and a higher percentage of killed background particles is desirable.

The percentage of Saved Signal and Lost Background particles have been computed for two simulations, fast inclusive $b\bar{b}$ and full inclusive $b\bar{b}$, with 6153 and 10066 events, respectively.

Simulation	**Saved Signal** (%)	**Lost Background** (%)
PYTHIA	97.5 ± 0.2	85.8 ± 0.15
Full Inclusive	97.6 ± 0.2	88.1 ± 0.15

Table A.1: Efficiency of the pre selection cut on particles. The signal particles saved by the cut on the edges are the "Saved Signal" while the background particles removed by the cut are the "Lost Background".

Table A.1 presents the results, where it can be observed that both simulations have a Saved Signal Particles percentage greater than 97%, while the percentage of Lost Background Particles is over 85%. In a previous study, it was shown that a cut can save up to 96% of the particles that belong to the same primary vertex of the B [5].

In conclusion, the evaluation of particle efficiency using two simulations demonstrates high performance in saving signal particles and losing background particles. These results provide valuable information for the per selection needed in the development of the DFEI project.

Appendix B

Algorithms

Listing B.1: Truth Matching

```
for theEvent in range(len(set(ExpandedEventNumberParticles))):
    FindEvent= np.where(ExpandedEventNumberRelations==theEvent)
    Part1=FirstParticleIndex[FindEvent]
    Part2=SecondParticleIndex[FindEvent]
    NewK=IndexK[np.where(EventK==theEvent)] #Same for Pi, Mu and B
    for j in range(len(NewPi)):
        vet=list(range(j+1,len(NewPi)))
        for i in range(len(NewK)):
            cond1= (Part1==NewK[i]) & (Part2==NewPi[j])
            cond2= (Part2==NewK[i]) & (Part1==NewPi[j])
            PKPi1=lenPart+np.where(cond1 | cond2)[0]
            for l in vet:
                if len(PKPi1!=0):
                    cond1_1= (Part1==NewK[i]) & (Part2==NewPi[l])
                    cond2_2= (Part2==NewK[i]) & (Part1==NewPi[l])
                    PPi1Pi2=lenPart+np.where(cond1_1 | cond2_2)[0]
                    #same for KPi2
                    if (len(PKPi2)!=0) & (len(PPi1Pi2)!=0):
                        # abs(411) is the ID of D (+ or -)
                        KPi1=((FromSameHeavyHadron[PKPi1]==1) &
                            (LCA[PKPi1]==1)) & (abs(LCA_ids[PKPi1])==411)
                            #same for KPi2 and Pi1Pi2
                        if (KPi1 & KPi2 & Pi1Pi2):
                            for m in range(len(NewMu)):
                                condKMu_1= (Part1==NewK[i]) & (Part2==NewMu[m])
                                condKMu_2=(Part2==NewK[i]) & (Part1==NewMu[m])
                                PKMu=lenPart+np.where( condKMu_1 | condKMu_2 )[0]
                                #same for Pi1Mu and Pi2Mu
                                if ((len(PKMu)!=0)&(len(PPi1Mu)!=0)&(len(PPi2Mu)!=0)):
                                    KMu=((FromSameHeavyHadron[PKMu]==1) &
                                        (LCA[PKMu]==2)) & (abs(LCA_ids[PKMu])==511)
                                        #same for Pi1Mu and Pi2Mu
                                    if (KMu & Pi1Mu & Pi2Mu):
                                        pairKMuPiPi+=1 #DECAY FOUND
    lenPart+=len(Part1)
```

Listing B.2: Invariant Mass Reconstruction

```
p_Dx, p_Dy, p_Dz  = [],[],[]
mass_D,mass_B=[],[]
for i in range(len(new_findPi1)):
    p_Pi1 = ROOT.TLorentzVector()
    p_Pi1.SetXYZM(px_reco[findPi1[i]], py_reco[findPi1[i]], pz_reco[findPi1[i]], PiMass)
```

```
    p_Pi2 = ROOT.TLorentzVector()
    p_Pi2.SetXYZM(px_reco[findPi2[i]], py_reco[findPi2[i]], pz_reco[findPi2[i]], PiMass)

    p_K = ROOT.TLorentzVector()
    p_K.SetXYZM(px_reco[findK[i]], py_reco[findK[i]], pz_reco[findK[i]],KMass)

    mass_D.append((p_Pi1 + p_K + p_Pi2).M())
    p_Dx.append(px_reco[findPi1[i]]+px_reco[findK[i]]+px_reco[findPi2[i]])
    p_Dy.append(py_reco[findPi1[i]]+py_reco[findK[i]]+py_reco[findPi2[i]])
    p_Dz.append(pz_reco[findPi1[i]]+pz_reco[findK[i]]+pz_reco[findPi2[i]])

for j in range(len(mass_D)):
    p_D = ROOT.TLorentzVector()
    p_D.SetXYZM(p_Dx[j], p_Dy[j], p_Dz[j], mass_D[j])

    p_Mu = ROOT.TLorentzVector()
    p_Mu.SetXYZM(px_reco[findMu[j]], py_reco[findMu[j]], pz_reco[findMu[j]], MuMass**2)

    mass_B.append((p_Mu + p_D).M())
```

Listing B.3: Edge prefiltering: *Theta* OR *SamePV*

```
thr_trDist=[2,3,5,7,9,10,12,14,17,20,25]#thresholds
killed_bkg_edges,selected_Signal_edges_trDist=[],[]
disconnected_Vet=[]

FirstIndex_1=FirstParticleIndex[FromSameHeavyHadron == 1]
SecondIndex_1=SecondParticleIndex[FromSameHeavyHadron == 1]
EventRelations_1=ExpandedEventNumberRelations[FromSameHeavyHadron == 1]
trdist_reco_1=trdist_reco[FromSameHeavyHadron == 1]
SamePV_reco_1=FromSameAssociatedPV_reco[FromSameHeavyHadron == 1]

killed_bkg_edges_trDist,selected_Signal_edges_trDist,disconnected_Vet=[],[],[]
DBkg = np.sum(FromSameHeavyHadron == 0)
DSignal = np.sum(FromSameHeavyHadron == 1)

for thr in thr_trDist: #different thresholds
    selected_Signal_edges = np.sum(((trdist_reco<thr)| (FromSameAssociatedPV_reco==1))
                                            & (FromSameHeavyHadron == 1)) / DSignal
    killed_bkg_edges = np.sum(((trdist_reco>thr)| (FromSameAssociatedPV_reco==0))
                                              & (FromSameHeavyHadron == 0)) / DBkg
    unique_events = np.unique(EventRelations_1)
    part1,part2,disconnected=[],[],[]
    disconn=0
    for evt in unique_events:
        part1=FirstIndex_1[(EventRelations_1 == evt)  &
                           ((trdist_reco_1<thr) | (SamePV_reco_1==1))]
        part2=SecondIndex_1[(EventRelations_1 == evt) &
                           ((trdist_reco_1<thr) | (SamePV_reco_1==1))]

        for i in range(len(FirstIndex_1[EventRelations_1==evt])):
            if (len(np.where(part1 == FirstIndex_1[EventRelations_1==evt][i])[0])==0) |
               (len(np.where(part2 == SecondIndex_1[EventRelations_1 == evt][i])[0])==0):
                disconn=disconn+1
    killed_bkg_edges_trDist.append(killed_bkg_edges) #Killed Backgound Edge
    selected_Signal_edges_trDist.append(selected_Signal_edges) #Selected Signal Edges
    disconnected_Vet.append(disconn / DSignal) #Fully Disconnected Signal Particles
```

Listing B.4: Edge prefiltering on Particles

```
LostSignalParticle,LostBKGParticle=0,0
thrFast=0.24
thrFull=0.14
```

```
#-------------------------------------------signal-----------------------------------------
FirstIndex_1=FirstParticleIndex[FromSamePV == 1]
SecondIndex_1=SecondParticleIndex[FromSamePV == 1]
EventRelations_1=ExpandedEventNumberRelations[FromSamePV == 1]
theta_reco_1=theta_reco[FromSamePV == 1]
SamePV_reco_1=FromSameAssociatedPV_reco[FromSamePV == 1]
denominator1=[]
for evt in np.unique(EventParticles):
    if len(np.where(ParticleType[EventParticles==evt] ==2)[0])==2:#two b hadrons

        theParticles=ExpandedParticleIndex[EventParticles==evt]
        #applying the cut opening_angle+FromSameAssociatedPV
        part1=FirstIndex_1[(EventRelations_1 == evt)  &
                           ((theta_reco_1<thr) | (SamePV_reco_1==1))]
        part2=SecondIndex_1[(EventRelations_1 == evt) &
                            ((theta_reco_1<thr) | (SamePV_reco_1==1))]
        denominator1.append(len(theParticles))
        for index in theParticles:
        #check if there is a particle that remains completly disconnected (no edges)
            if (len(np.where(part1==index)[0])==0) and (len(np.where(part2==index)[0])==0):
                LostSignalParticle=LostSignalParticle+1
#saved signal particles
SAVEDPARTICLES=1 - LostSignalParticle/np.sum(denominator1)
#-------------------------------------------background---------------------------------------
FirstIndex_0=FirstParticleIndex[FromSamePV == 0]
SecondIndex_0=SecondParticleIndex[FromSamePV == 0]
EventRelations_0=ExpandedEventNumberRelations[FromSamePV == 0]
theta_reco_0=theta_reco[FromSamePV == 0]
SamePV_reco_0=FromSameAssociatedPV_reco[FromSamePV == 0]

denominator=[]
for evt in np.unique(EventParticles):
    if evt % 350 == 0: print(int(evt/n * 100), '%', end=' ')
    if len(np.where(ParticleType[EventParticles==evt] ==2)[0]) ==2:#two b hadrons
        theParticles=ExpandedParticleIndex[EventParticles==evt]
        #removed particles with the cut opening_angle+FromSameAssociatedPV
        part1=FirstIndex_0[(EventRelations_0 == evt)  &
                                ((theta_reco_0>thr) | (SamePV_reco_0==0))]
        part2=SecondIndex_0[(EventRelations_0 == evt) &
                                ((theta_reco_0>thr) | (SamePV_reco_0==0))]
        tot=FirstIndex_0[EventRelations_0 == evt]

        part1=[pos for pos in tot if pos not in part1]
        part2=[pos for pos in tot if pos not in part2]
        denominator.append(len(theParticles))
        for index in theParticles:
        #check if there is a particle that remains completly disconnected (no edges)
            if ((len(np.where(part1==index)[0])==0) and (len(np.where(part2==index)[0])==0)):
                LostBKGParticle=LostBKGParticle+1
#lost background particles
LOST_BACKGROUND_PARTICLES=LostBKGParticle/np.sum(denominator)
```

Listing B.5: Event and Signal Classification

```
def Reconstruction_classification(PER_df,Event_Number):#,truth_cluster_dict):

    # Reco-level decay chain, for filtered final state particles
    input_graph_tuple  = utils_tf.data_dicts_to_graphs_tuple([env.get_graph_LCA()])
    reco_LCA, time_LCA_reconstruction = eval_model_LCA(input_graph_tuple)
    reco_LCA = reco_LCA[0]
    particle_keys = env.get_particle_keys() #list of particles' index per event

    #dictionary
```

```
reco_cluster_dict, reco_num_clusters_per_order, _=
                                reconstruct_decay(reco_LCA, particle_keys, 0)

# Truth-level decay chain, for unfiltered signal-only final state particles
env.set_use_only_signal_particles(1)
truth_graph = env.get_truth_graph_LCA()
truth_LCA = pd.DataFrame(np.column_stack((truth_graph["senders"],
               truth_graph["receivers"])), columns = ['senders', 'receivers'])
truth_LCA['LCA_dec'] = np.reshape(np.argmax(np.reshape(truth_graph["edges"],
                          (truth_graph["edges"].shape[0], 4)), axis=-1),(-1,))
truth_LCA['LCA_mistag'] = 0.
truth_LCA = truth_LCA[truth_LCA['senders'] < truth_LCA['receivers']]
#print(particle_name, env.get_truth_TopoLCA_ids())
truth_LCA['LCA_id_label'] = list(map(particle_name,env.get_truth_TopoLCA_ids()))
truth_LCA['TrueFullChainLCA'] = env.get_truth_LCA()
signal_particle_keys = env.get_particle_keys()
signal_particle_ids = list(map(particle_name, env.get_truth_particle_ids()))
env.set_use_only_signal_particles(0)
truth_cluster_dict, truth_num_clusters_per_order, max_full_chain_depth_in_event=
   reconstruct_decay(truth_LCA, signal_particle_keys, 0, signal_particle_ids, 1)

perfect_event_reconstruction = 0
TheRecoDict={}
if truth_cluster_dict!={}:
    for tc_firstkey in truth_cluster_dict.keys():

        if (reco_cluster_dict=={}):
            PER_df = PER_df.append({'EventNumber': env.current_event_number,
                    'NumSignalParticles': 0,
                    'The_signal_particles':
                          len(truth_cluster_dict[tc_firstkey]['node_keys']),
                    'True_key' : tc_firstkey,
                    'PerfectSignalReconstruction': 0,
                    'PartialSignalReconstruction': 0,
                    'CombinatorialSignalReconstruction': 1,
                    'QuasiSignalReconstruction': 0},
                    ignore_index=True)
            break

        perfect_signal_reconstruction = 0
        quasi_signal_reconstruction = 0
        partial_signal_reconstruction = 0
        combinatorial_signal_reconstruction = 0

        if tc_firstkey in reco_cluster_dict.keys():
            all_nodes_ok = (truth_cluster_dict[tc_firstkey]['node_keys']
                            == reco_cluster_dict[tc_firstkey]['node_keys'])
            all_LCA_values_ok = (truth_cluster_dict[tc_firstkey]['LCA_values']
                            == reco_cluster_dict[tc_firstkey]['LCA_values'])
        else:
            all_nodes_ok = 0
            all_LCA_values_ok = 0

        number_of_signal_particles=-1

        if all_nodes_ok:
            number_of_signal_particles=
                             len(reco_cluster_dict[tc_firstkey]['node_keys'])
            if all_LCA_values_ok:
                perfect_signal_reconstruction = 1
            else:
                quasi_signal_reconstruction = 1

        if perfect_signal_reconstruction==0 and quasi_signal_reconstruction==0:
```

```
                # Look for non-isolated signals
                reco_key = -1
                all_particles_reconstructed = 0
                for rc_firstkey in reco_cluster_dict.keys():
                    if tc_firstkey in reco_cluster_dict[rc_firstkey]['node_keys']:
                        all_particles_reconstructed = 1
                        for true_particle in
                                    truth_cluster_dict[tc_firstkey]['node_keys']:
                            if not true_particle in
                                     reco_cluster_dict[rc_firstkey]['node_keys']:
                                all_particles_reconstructed = 0
                                break
                        reco_key = rc_firstkey
                        break

                if all_particles_reconstructed:
                    number_of_signal_particles=
                                   len(reco_cluster_dict[reco_key]['node_keys'])
                    partial_signal_reconstruction = 1
                else:
                    combinatorial_signal_reconstruction = 1

            TheRecoDict.update(reco_cluster_dict)
            perfect_event_reconstruction *= perfect_signal_reconstruction
            PER_df = PER_df.append({'EventNumber': env.current_event_number,
                        'NumSignalParticles': number_of_signal_particles,
                        'The_signal_particles':
                                 len(truth_cluster_dict[tc_firstkey]['node_keys']),
                        'True_key' : tc_firstkey,
                        'PerfectEventReconstruction':perfect_event_reconstruction,
                        'PerfectSignalReconstruction':perfect_signal_reconstruction,
                        'PartialSignalReconstruction':partial_signal_reconstruction,
                        'CombinatorialSignalReconstruction':
                                                combinatorial_signal_reconstruction,
                        'QuasiSignalReconstruction':quasi_signal_reconstruction},
                        ignore_index=True)
    return PER_df, TheRecoDict


def evaluate_classification(first_event_number, number_of_events):
    PER_df= pd.DataFrame(columns=['EventNumber', 'NumSignalParticles',
                     'The_signal_particles','True_key','PerfectEventReconstruction',
                        'PerfectSignalReconstruction','PartialSignalReconstruction',
                    'CombinatorialSignalReconstruction','QuasiSignalReconstruction'])
    PER_df = PER_df.astype({'EventNumber':np.int32,'NumSignalParticles':np.int32,
                            'The_signal_particles':np.int32,'True_key':np.int32,
                            'PerfectEventReconstruction':np.int32,
                            'PerfectSignalReconstruction':np.int32,
                            'PartialSignalReconstruction': np.int32,
                            'CombinatorialSignalReconstruction' : np.int32,
                            'QuasiSignalReconstruction' : np.int32 })

    env.current_event_number = first_event_number-1
    k=0
    a={}
    while (env.current_event_number+1) < (first_event_number + number_of_events):
        print('Processing␣event', env.current_event_number+1)
        PER_df,TheRecoDict= Reconstruction_classification(PER_df,env.current_event_number+1)
        b=[]
        if len(list(TheRecoDict.values()))!=0:
            for k in range(len(list(TheRecoDict.values()))-1):
                b.append(list(TheRecoDict.values())[k]['node_keys'])
            a.update({env.current_event_number+1 : b})

    w = csv.writer(open('TheRecoDict'+'_DECAY_'+
```

```
                    str(first_event_number + number_of_events -1)+'.csv', "w"))
    for key, val in a.items():
        w.writerow([key, list([i for i in val])])

    file_signal = uproot3.recreate(
  'PYTHIATheNew_DECAY_SignalClassification_PerformanceMetrics_allbkgtracks_Signal_eval_check_'
          +str(first_event_number)+'_'+str(first_event_number + number_of_events -1)+'.root')
    new_branches_signal = list(PER_df.columns)
    new_branch_types_signal = PER_df.dtypes
    file_signal['ReconstructionEvaluation']=
                    uproot3.newtree(dict(zip(new_branches_signal,new_branch_types_signal)))
    file_signal['ReconstructionEvaluation'].extend(
                                      {k: PER_df[k].values for k in new_branches_signal})
    file_signal.close()
```

Listing B.6: Training of the Full Simulation

```
def create_loss(name, target, outputs):
    if (name == "filtering"):
        class_weights = tf.math.reciprocal(tf.compat.v2.reduce_sum(target.nodes, axis=0))
        weights = tf.compat.v2.reduce_sum(class_weights * target.nodes, axis=1)
        unweighted_losses =
          tf.compat.v2.nn.softmax_cross_entropy_with_logits(target.nodes, outputs[-1].nodes)
    elif (name == "coarsening"):
        class_weights = tf.math.reciprocal(tf.compat.v2.reduce_sum(target.edges, axis=0))
        weights = tf.compat.v2.reduce_sum(class_weights * target.edges, axis=1)
        unweighted_losses =
          tf.compat.v2.nn.softmax_cross_entropy_with_logits(target.edges, outputs[-1].edges)
    elif (name == "LCA") and LOSS=CrossEntropy: #softmax cross entropy-------------------------
        class_weights = tf.math.reciprocal(tf.compat.v2.reduce_sum(target.edges, axis=0) + one)
        weights = tf.compat.v2.reduce_sum(class_weights * target.edges, axis=1)
        unweighted_losses =
          tf.compat.v2.nn.softmax_cross_entropy_with_logits(target.edges, outputs[-1].edges)
        table=pd.DataFrame(target.edges, outputs[-1].edges)
    weighted_losses = unweighted_losses * weights
    loss = tf.reduce_sum(weighted_losses)
    elif (name == "LCA") and LOSS=FocalLoss: #focal loss---------------------------------------
        gamma=int(2)
        alpha_t=0.25
        class_weights = tf.math.reciprocal(tf.compat.v2.reduce_sum(target.edges, axis=0) + one)
        weights_old = tf.compat.v2.reduce_sum(class_weights * target.edges, axis=1)
        unweighted_losses =
            tf.compat.v2.nn.softmax_cross_entropy_with_logits(target.edges, outputs[-1].edges)
        p=tf.math.exp(-unweighted_losses)
        weighted_losses= alpha_t* (1-p)**gamma *unweighted_losses
    loss = tf.reduce_sum(weighted_losses* alpha_t)
    return loss

#----Hyperparameters
batch_size = 128
learning_rate = 5e-4
num_training_iterations = 4000
print("INFO: Number of recurrent passes: ", num_recurrent_passes_LCA)
print("INFO: Using batch size: ", batch_size)
print("INFO: Using learning rate: ", learning_rate)

#optimizer = snt.optimizers.RMSProp(learning_rate)
optimizer = snt.optimizers.Adam(learning_rate)
model = models_LCA.EncodeProcessDecode(edge_output_size=4)
checkpoint = tf.train.Checkpoint(module=model)
checkpoint.restore(tf.train.latest_checkpoint("./LCA_model"))

current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'logs/gradient_tape/' + current_time + '/train'
test_log_dir = 'logs/gradient_tape/' + current_time + '/test'
```

```
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
test_summary_writer = tf.summary.create_file_writer(test_log_dir)

#Training loop
last_iteration = 0
logged_iterations = []
losses_tr = []
corrects_tr = []
losses_ge = []
corrects_ge = []

#Some example data that resembles the tensors that will be fed into update_step():
example_input_data, example_target_data = create_data("LCA", batch_size)[:2]

#Get the input signature for that function by obtaining the specs
input_signature = [
    utils_tf.specs_from_graphs_tuple(example_input_data),
    utils_tf.specs_from_graphs_tuple(example_target_data)
]

#Compile the update function using the input signature for speedy code
compiled_update_step =
        tf.function(update_step_LCA, experimental_relax_shapes=True)

log_every_seconds = 5
print("# (iteration number), T (elapsed seconds), "
  "Ltr (training loss), Lge (test/generalization loss), "
   "Acctr (training compute_LCA_edge_accuracy),
    "Accge (test/generalization compute_LCA_edge_accuracy), "
    "Acctr_LCA0, Acctr_LCA1, Acctr_LCA2, Acctr_LCA3")

start_time = time.time()
last_log_time = start_time

for iteration in range(last_iteration, num_training_iterations):
    last_iteratation = iteration
    inputs_tr, targets_tr = create_data("LCA", batch_size)[:2]

    outputs_tr, loss_tr =
      compiled_update_step("LCA", model, optimizer, inputs_tr, targets_tr)

    the_time = time.time()
    elapsed_since_last_log = the_time - last_log_time
    if elapsed_since_last_log > log_every_seconds:
        last_log_time = the_time
        elapsed = time.time() - start_time

        inputs_ge, targets_ge = create_data("LCA", batch_size)[:2]
        outputs_ge = model(inputs_ge, num_recurrent_passes_LCA)
        loss_ge = create_loss("LCA", targets_ge, outputs_ge)

        correct_tr, correct_tr_0LCA, correct_tr_1LCA,
        correct_tr_2LCA, correct_tr_3LCA = compute_LCA_edge_accuracy(
            utils_tf.nest_to_numpy(inputs_tr),
            utils_tf.nest_to_numpy(targets_tr),
            utils_tf.nest_to_numpy(outputs_tr[-1]))
        correct_ge, correct_ge_0LCA, correct_ge_1LCA,
        correct_ge_2LCA, correct_ge_3LCA = compute_LCA_edge_accuracy(
            utils_tf.nest_to_numpy(inputs_ge),
            utils_tf.nest_to_numpy(targets_ge),
            utils_tf.nest_to_numpy(outputs_ge[-1]))

        losses_tr.append(loss_tr.numpy())
        losses_ge.append(loss_ge.numpy())
        logged_iterations.append(iteration)
```

```
corrects_tr.append(correct_tr)
corrects_ge.append(correct_ge)
```

Listings

List of Figures

List of Tables

Bibliography

[1] Peter W. Battaglia and Hamrick et. all. Relational inductive biases, deep learning, and graph networks. *arXiv*, abs/1806.01261, 2018.

[2] Rene Brun and Fons Rademakers. Root - an object oriented data analysis framework. *Nucl. Inst. & Meth. in Phys. Res. A*, 389:81–86, 1997. Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996.

[3] CERN. Standard model - the building blocks of matter. `https://home.cern/science/physics/standard-model`, 2021.

[4] The LHCb Collaboration. The LHCb detector at the LHC. *Journal of Instrumentation*, 3(08):S08005, 2008.

[5] The LHCb Collaboration. Flavour tagging in run 3. Presentation at the RTA Meeting, 2021. Available at: `https://indico.cern.ch/event/1253657/contributions/5271087/attachments/2595657/4480555/Flavour20Tagging20in20Run20320-20RTA20meeting-1.pdf`.

[6] Barbosa-Marinho et. all. *LHCb muon system: Technical Design Report*. Technical design report. LHCb. CERN, Geneva, 2001.

[7] IBM. Neural networks. *IBM Journal of Research and Development*, 2021. `https://www.ibm.com/topics/neural-networks`.

[8] Hasso Plattner Institute. Future soc lab (service-oriented computing). `https://hpi.de/en/research/future-soc-lab.html`, 2007.

[9] T. Keck and F. Abudinén et. all. The full event interpretation. *Computing and Software for Big Science*, 3(1), feb 2019.

[10] Christopher W. Murphy. Class imbalance techniques for high energy physics. *SciPost Physics*, 7(6), dec 2019.

[11] Roshan Nayak. Focal loss: A better alternative for cross entropy. *Towards Data Science*, Apr 202.

[12] J. Garcia Pardiñas. Towards a deep full event interpretation algorithm for the lhcb experiment. https://indico.cern.ch/event/1078970/contributions/4833296/attachments/2443229/4186053/IMLWorkshop_2022_DFEI_JulianGarciaPardinas.pdf, 2021.

[13] J. Garcia Pardiñas. Graph neural networks for a deep-learning based full event interpretation (dfei) at the lhcb trigger. file:///home/computer/Downloads/DFEI_ML4Jets_v0.pdf, 2022.

[14] Renato Quagliani. Novel real-time alignment and calibration of lhcb detector for run ii and tracking for the upgrade. *Journal of Physics: Conference Series*, 762:012046, 10 2016.

[15] Alexander Radovic, Mike Williams, David Rousseau, Michael Kagan, Daniele Bonacorsi, Alexander Himmel, Adam Aurisano, Kazuhiro Terao, and Taritree Wongjirad. Machine learning at the energy and intensity frontiers of particle physics. *Nature (London)*, 560(7716), 8 2018.

[16] Quast G. et. all Reuter L., Ferber T. Full event interpretation using graph neural networks. Master's thesis, Karlsruhe Institute of Technology, 2 2022.

[17] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, jan 2021.

[18] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[19] R. L. Workman and Others. Review of Particle Physics. *PTEP*, 2022:083C01, 2022.

Acknowledgements

As I approach the completion of my Master's degree, I would like to express my gratitude to those who have supported me along the way.

First and foremost, I am deeply thankful to my supervisor, Prof. Marta Calvi, and my co-supervisor, Dr. Julián García Pardiñas, for their invaluable guidance and tutelage throughout the development of my thesis. Their expertise and availability have played a crucial role in the success of my work, and I am fortunate to have had the opportunity to learn from them.

I would like to extend my sincere thanks to the DFEI group who welcomed me into their project and provided valuable feedback during the development of my thesis.

Of course, none of this would have been possible without the continuous encouragement of my family. I am grateful to my parents, my grandmother, and my siblings, whose love and support have sustained me through this challenging journey.
I must also acknowledge the unwavering support of my boyfriend, Joele, whose insights, patience, and love have been a source of inspiration and comfort throughout these months.

Finally, I am grateful to all of my friends, from those who have been with me since primary school, to the now-expanded high school group, to my bachelor's classmates and my study hall companions, and to the wonderful people I have met during my Master's studies. Your support and encouragement have been instrumental in helping me reach this milestone.

Thank you all for being part of this journey with me.