UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di
Scienze Matematiche
Fisiche e Naturali
Corso di Laurea Magistrale
in Scienze Fisiche e Astrofisiche

Tecniche di simulazione parametrica con reti neurali profonde e loro implementazione per l'esperimento LHCb al CERN e le sue future evoluzioni

Techniques for parametric simulation with deep neural networks and implementation for the LHCb experiment at CERN and its future upgrades

Relatore:
Dott. Lucio Anderlini

Correlatore:
Prof. Piergiulio Lenzi

Candidato:
Matteo Barbetti

Anno Accademico: 2018/2019

A mia madre e al suo prezioso sostegno.
A mio padre e alla sua travolgente forza d'animo.
A mia sorella e alla sua inesauribile dolcezza.

Contents

Introduction

The present knowledge of elementary particles and their interactions is collected within a successfully theory named *Standard Model*, which continues to predict the majority of the experimental results obtained to date. However, despite its clear success, the Standard Model is not a complete theory because of the existence of several questions still unanswered: dark matter, neutrino masses or abundance of matter over antimatter are just few examples. It is therefore necessary to keep testing the theory to search evidences of phenomena beyond the Standard Model which can eventually hint the path to follow for *New Physics*. To this end, being able to increase the currently reached precision by the particle experiments and to provide significant terms of comparison with the theoretical expectation is crucial. In this scenario the Large Hadron Collider (LHC), the world's largest and most powerful particle accelerator, plays a key role, facing the technological challenges to move forward in understanding the Universe and its laws.

The LHCb experiment is one of the four detectors along the accelerator ring and is dedicated to the study of heavy flavour physics at the LHC. Its primary goal is to look for indirect evidence of New Physics in CP-violation and in rare decays of b- and c-hadrons. Due to its field of interest, the LHCb detector has a peculiar geometry totally different from the one of the other LHC experiments, designed to maximize the angular acceptance of high-energy $b\bar{b}$ pairs. Moreover, in order to complete its physics program, LHCb is able to reconstruct with extreme precision the decay vertices, to measure accurately track momenta and to provide a robust particle identification system. All these features ensure to identify heavy hadron decays with high efficiency, and to measure rare processes exploiting the flexibility of a trigger system capable to cope effectively to the harsh environment produced by a hadronic collider.

To improve its capabilities in search of New Physics, the LHCb is currently being upgraded, exploiting the stop of data taking for the Long Shutdown 2 (2018-2021). The LHCb Upgrade detector will operate through LHC Runs 3 and 4, and will be able to take full advantage from the increasing of instantaneous luminosity by a factor five and from a purely-software trigger able to improve the selection efficiency by at least a factor two. This will result in data samples roughly increased by an order of magnitude, and in the possibility of reaching *unprecedented* physics accuracy. However, the huge datasets alone are not enough to achieve the physics goal, which requires accurate studies of the background contributions on the data, and of the selection efficiency of the trigger and the particle identification systems. To this end, it is necessary to produce simulation samples that has to be at least of the same size as the collected data. Consequently, making the most of the upgraded experiment requires of improving significantly the simulation production.

The full software trigger designed for the LHCb Upgrade detector will allow the simu-

lation to consume almost the totality of the computing resources available to the experiment. Nevertheless, the *traditional* simulation system is already now incapable to sustain the analysis needs of the physics groups, and this kind of scenario will only get worse. The traditional simulation can be split into two main independent phases. The first phase consists of the event generation from proton-proton collisions to the decay of particles in the channels of interest for the physics program. Instead, the second phase consists in the tracking through the LHCb detector of the particles produced by the generator phase, simulating all the radiation-matter interactions occurring within the detector. Due to the full reproduction of these interactions, the second phase determines high CPU-consuming computations which are the real bottleneck, in terms of time performance, of the entire simulation process. This strategy is named *full* simulation.
The *full* simulation has already saturated the computing resources available and is not able to provide all analyses with the simulated samples necessary. As a consequence, the analyses that need large samples have uncertainty, in many cases already dominant, due to the statistical uncertainity on the simulated sampled. In order to avoid that this scenario can occur also for the LHCb Upgrade detector, faster simulation options must be adopted.
Renouncing to simulate the radiation-matter interactions one can obtain the maximum speedup of the simulation system. Similar strategies are called *ultra-fast* simulation and consist of reproducing the high-level response of the detectors, mapping it with parametric or non-parametric functions. Among non-parametric solutions, methods based on Machine Learning algorithms and, in particular, based on *Generative Adversarial Networks* (GAN) have proved to be very promising.

The GAN systems are a powerful class of *generative models* based on simultaneous training of two neural networks. The first neural network, named *generator* G, outputs synthetic data trying to reproduce the probability distributions of the elements contained in a reference sample. Instead, the second neural network, named *discriminator* D, given elements both from the reference sample and the generated one as inputs, outputs the probability that the input belongs to the reference sample with respect to the generated one. The goal is that the discriminator distinguishes the origin of the two samples, and simultaneously the training procedure for the generator is to hinder the discriminator task. This framework corresponds to a *minimax two-player game*. In the space of arbitrary functions G and D, a unique solution exists, with the generator recovering the reference data distribution and the discriminator output equal to $1/2$ everywhere.
Throughout recent years, GAN systems have demonstrated to be a backbone for Computer Vision, proving their capacity in reproducing highly faithful and diverse probability distributions thanks to models learned directly from data. Therefore, it is not surprising that models trained thanks of the minimax procedure are able to reproduce accurately the detector responses and, in particular, to map basic information about track kinematics and about event multiplicity into the correctly distributions of different high-level particle identification variables. The idea is to exploit the competition between the generator and discriminator in order to update their respective parameters (training process), and to adopt the first neural network as a *generative model* capable to parameterize the responses of the particle identification system of LHCb. Similar models can then be implemented within the simulation framework of the LHCb experiment providing a powerful ultra-fast and fairly accurate solution to produce the huge simulated samples necessary for the analyses demands.

A large part of my thesis work has covered the development and implementation of generative models for the particle identification system of LHCb based on GANs. Despite GANs have proved to have remarkable capabilities in learning probability distributions directly from data, their training is often *hard* and being able to reproduce the *entire* reference space is not always guaranteed. In order to maximize the correct convergence of the generative models trained, I have explored more deeply the theory behind adversarial systems implementing *state-of-the-art* algorithms capable to accomplish effectively the target set. In order to tune and select the best model for each subdetector of the particle identification system, I have developed a statistical method based on *robust multivariate classifiers* to test the quality of the generated samples. This statistical method has been used to choose the best strategies for data preparation in order to complete successfully the training process, and to assess the quality reached by different learning algorithms proposed in the literature. The built models are able to output *good-looking* distributions for the high-level particle identification variables of LHCb, allowing to reproduce a large amount of diverse function shapes starting from track kinematics parameters and a measure of the detector occupancy.

The trained models can be exploited within a simulation framework in order to offer an efficient *ultra-fast* solution to produce large simulated samples. Then, during my thesis work I have participated to the development of a new simulation framework designed to take full advantage of the most modern software for evaluating large computational graphs, such as complex neural networks. In particular, the integration of the trained models within this simulation framework consists of a personal contribution. The framework developed has proved to be able to produce faithful simulated samples starting from the generation phase, and to provide detector parameterization competitive with what obtained from the *full* simulation.

Chapter 1 reports an overview about the Standard Model and an introduction about reasons behind the necessity of producing large simulated samples, taking as an examples the needs of the LHCb experiment, presented in detail in Section 1.2. In Chapter 2, the limitations of a full simulation approach are treated and some faster solutions are discussed. Chapter 3 is deeply dedicated to the Machine Learning paradigm, giving particular attention to generative models. In particular, the theory behind the Generative Adversarial Networks and their training problems are discussed in Section 3.3. Chapter 4 reports details about the implementation of GAN models for the particle identification system of LHCb. The statistical method mentioned above is described in Section 4.2.2, while Section 4.4 shows the resulting distributions obtained at the end of the training process. Finally, Chapter 5 discusses this new simulation framework, paying particularly attention to its technical implementation: its impressive results are reported in Section 5.4.

Chapter 1

Scientific Background and LHCb

1.1 The Standard Model and Beyond: an Overview

Since the 1930s, physicists witnessed a proliferation of particles thanks to an increasing number of studies and experiments investigating the fundamental structure of matter. The amount of predictive theories and new discoveries led to develop, in the 1970s, a theory gathering the best knowledge of these particles and their interactions: the *Standard Model* (SM). The SM is a Quantum Field Theory (QFT) paradigm describing three of the four known fundamental forces: the electromagnetic, weak and strong interactions are included, unlike the gravitational force. The QFT treats particles as excited states (or *quanta*) of their underlying fields, and describes interactions between particles using the Lagrangian formalism.
Elementary particles are the building blocks of matter. Having half-integer spin, they are distributed according to the Fermi-Dirac statistics and, for this, they are named *fermions*. The SM divides elementary particles into two families: *quarks* and *leptons*. Quarks can appear in six different *flavours*, three positively charged (*up*, *charm* and *top*), and three negatively charged (*down*, *strange* and *bottom*). Even leptons have six flavours, but this time the negatively charged triplet (*electron*, *muon* and *tau*) is paired with three neutral particles called *neutrinos* (ν_e, ν_μ and ν_τ). Every type of particle f has an associated *antiparticle* $\bar{f}$ with the same mass but with opposite electric charge. Charge values both of particles and antiparticles are reported in Table 1.1.
Within the SM, interactions can be described mathematically as quantum fields. The strong interaction, responsible for quark bound states, is carried by a particle called *gluon*.

Family	Flavour			Charge	Spin
Quarks	u ($\bar{u}$)	c ($\bar{c}$)	t ($\bar{t}$)	$+2/3$ ($-2/3$)	1/2
	d ($\bar{d}$)	s ($\bar{s}$)	b ($\bar{b}$)	$-1/3$ ($+1/3$)	
Leptons	e^- (e^+)	μ^- (μ^+)	τ^- (τ^+)	-1 ($+1$)	1/2
	ν_e ($\bar{\nu}_e$)	ν_μ ($\bar{\nu}_\mu$)	ν_τ ($\bar{\nu}_\tau$)	0	

Table 1.1: Classification of matter (and antimatter). The charge values are reported in units of elementary charge e.

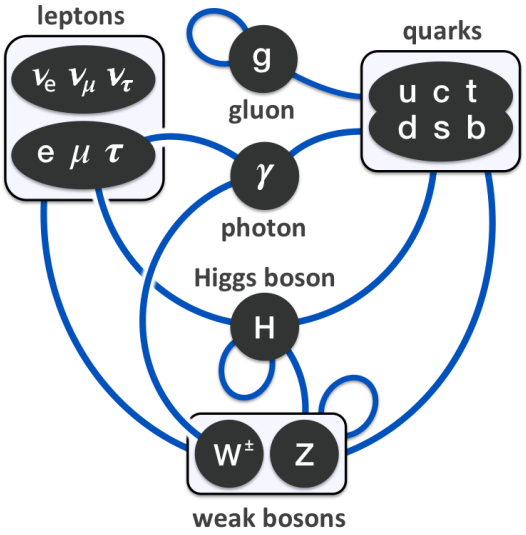


Figure 1.1: Summary of interactions between particles described by the Standard Model. Reproduced from `https://en.wikipedia.org/wiki/Standard_Model`.

Electromagnetism underlies interaction between charged particles, while the weak force is responsible for radioactive decay phenomena. The electromagnetic and weak forces can be unified into the electroweak interaction, that has four exited states: two neutral (*photon* and Z^0) and two charged ($W^{\pm}$). All these particles have integer spin ($s = 1$ to be exact) and, following the Bose-Einstein statistics, are named *bosons*.

In the SM formalism, leptons do not interact through the strong force, and are able to interact only thanks to the electroweak one: obviously neutrinos, being electrically neutral, can only interact through the weak force. On the other hand, quarks are the only particles of the SM to experience all fundamental interactions. In fact, quarks are characterized not only by an electric charge but also by a color charge, allowing them to undergo strong force. A schematic representation of elementary particles and their interactions is reported in Figure 1.1.
The SM is a *gauge theory*, which means there are degrees of freedom in the mathematical formalism that do not correspond to changes in the physical state. The gauge group[1] of the SM is $SU(3) \otimes SU(2) \otimes U(1)$, where the symmetry term $SU(3)$ refers to strong interaction. The latter is responsible for *color confinement*, the phenomenon for which color-charged particles (such as quarks and gluons) cannot be isolated, and therefore cannot be directly observed in normal conditions. Consequently, we can find only quark bound states, like *mesons* ($q\bar{q}$ states) and *hadrons* (qqq states).

Another important feature of the SM is that weak and mass eigenstates of quarks do not coincide. This results into the need of defining a mixing matrix to describe the weak interaction of the mass eigenstates[2]: it is named *Cabibbo–Kobayashi–Maskawa matrix*, or

[1] A full description of this topic is beyond the scope of this thesis.

[2] In the Lagrangian formalism the Hamiltonian describing the interaction is non-diagonal. The quark mixing matrix, in this sense, represents a change of basis, and that's why the CKM matrix must be unitary.

simply *CKM matrix*:

$$V_{CKM} = \begin{pmatrix} V_{ud} & V_{us} & V_{ub} \\ V_{cd} & V_{cs} & V_{cb} \\ V_{td} & V_{ts} & V_{tb} \end{pmatrix} \tag{1.1}$$

with $V_{pq} \in \mathbb{C}$ such that

$$\sum_i V_{ij} V_{ik}^* = \delta_{jk} \qquad \text{and} \qquad \sum_j V_{ij} V_{kj}^* = \delta_{ik} \tag{1.2}$$

The non-null phase between different complex elements of V_{CKM} is responsible for all CP-violating phenomena in flavor-changing processes of the SM. Studying the quark mixing matrix is therefore crucial to understand the processes behind the abundance of matter over antimatter that we observe in the present-day Universe. Moreover, the absence of theoretical uncertainties in the unitarity condition (1.2) provides us a perfect laboratory to check the validity of the SM. In this sense, observing a violation of the unitarity condition would unequivocally allow to conclude that the theory is not complete, eventually hinting the path to follow for New Physics (NP).

Recent decades have witnessed more and more studies, both theoretical and experimental, about V_{CKM} elements measure. Difficulties in producing a sufficient heavy quarks sample have historically delayed this kind of studies. Nowadays, however, measuring V_{CKM} parameters is fruitfully carried on by B-Factories and hadron colliders. Precise measurements of CKM matrix have already been done for b-sector, for which it is possible to rewrite the condition (1.2) in terms of *unitarity triangles*, that are non-degenerate only if the CP symmetry is violated. The (bd) triangle is shown in Figure 1.2. The sum of the unitarity triangle angles concurrently measured, $\alpha + \beta + \gamma = (180 \pm 7)^\circ$, is consistent with the SM expectation [1].
Also the *charmed* mesons ($q\bar{q}$ states made of c-quark coupled with a lighter one) offer a decay channel to study CP-violation. And only recently, the LHCb collaboration at CERN has observed, for the first time, this matter-antimatter asymmetry in D^0 meson [2], confirming the extremely small contribution of $D^0 - \bar{D}^0$ mixing expected by the SM.

Heavy flavour physics allows indirect search for NP also studying *lepton universality*, a property of the SM ensuring that charged leptons (e, μ and τ) interact in the same way with other particles. As a result, the different lepton types should be produced with the same probability in particle decays, once mass differences are taken into account. To

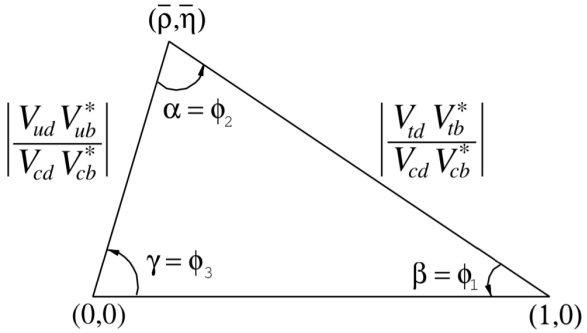


Figure 1.2: Sketch of the unitarity triangle [1].

enhance possible effects of NP, however, it is important to choose processes that are highly suppressed in the SM, in order to make differences, otherwise invisible, significant. Processes lead by *Flavour-Changing Neutral Currents* (FCNC), for example, are allowed in the Standard Model but tremendously suppressed by simple, symmetry-justified motivations. Studying FCNC processes involving leptons, such as the two different semileptonic decays as $B \to \ell^- \ell^+ X$, with $\ell = e, \mu, \tau$ and X representing a generic portion of the final state, is a powerful technique to test the validity of lepton universality or, in the case of violation, to highlight effects beyond the SM [3, 4].

Despite its clear success, the SM is still far from being a complete theory. While there are signs of NP, none of them sharply points to a specific extension of the theory, nor tells us by which kind of experiments NP will be discovered [5]. This forces us to increase the statistical power of our experiments, reaching the precision necessary for indirect search of NP. In this scenario, the largest Particle Physics experiments, like the Large Hadron Collider, play a key role, facing the technological challenges to move forward in understanding the Universe and its laws.

1.2 The LHCb Experiment

The *Large Hadron Collider* (LHC) is the world's largest and most powerful particle accelerator. Fired up for the first time on 2008, it is the latest addition to CERN's accelerator complex. The LHC consists of a 27-kilometre ring of *superconducting magnets* and *radiofrequency cavities*, capable to boost the energy of the particles along the way. Inside the accelerator two tubes are kept at *ultra-high vacuum*, allowing two high-energy proton beams[3] to travel in opposite directions at close to the speed of light. Protons inside the beams are arranged in *bunches* spaced of 25 ns, and are made to collide at four locations around the accelerator ring, corresponding to the positions of four particle detectors: ALICE, ATLAS, CMS and LHCb.

[3] In appropriate configurations, it is also possible using high-energy ion beams.

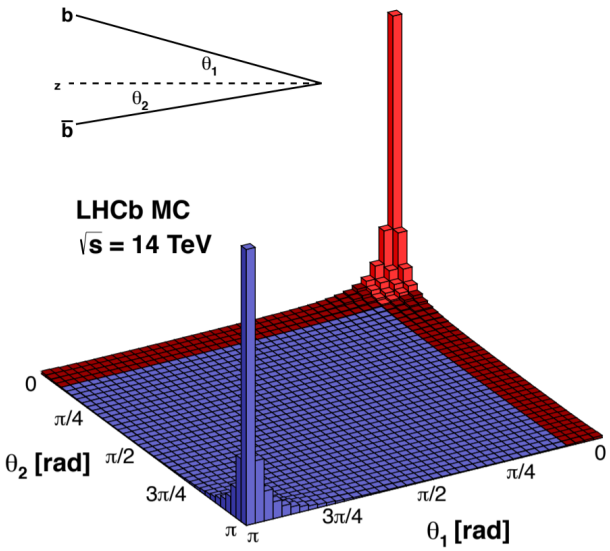


Figure 1.3: Production angles of high-energy $b\bar{b}$ pairs at LHC.

The *Large Hadron Collider beauty* (LHCb) experiment is dedicated to heavy flavour physics at the LHC. Its primary goal is to look for indirect evidence of NP in CP-violation and in rare decays of b- and c-sectors [6].
The large production cross section of $b\bar{b}$ pairs at the energy reachable in high-energy proton collisions ($\sigma_{b\bar{b}} \sim 500$ μb at 14 TeV) makes the LHC the most copious source of B mesons in the world. To exploit this large number of b-hadrons, the LHCb detector has a geometry totally different from the one of the other LHC experiments, designed to detect mainly forward particles. As shown in Figure 1.3, high-energy $b\bar{b}$ pairs are predominantly produced with angles lower than 300 mrad, and this justifies construction choices that allows LHCb to have a geometrical acceptance in *pseudorapidity*[4] within the range $2 < \eta < 5$.
The distinctive lifetime of heavy hadrons ($\sim 10^{-12}$ s) can be effectively exploited to discriminate them, but to do that, an excellent vertex resolution is crucial. The correct measure of particles momentum and invariant-mass is necessary to reduce combinatorial background, while the identification of protons, kaons and pions is fundamental in order to cleanly reconstruct final states in b- and c-sectors. In addition to these requirements, the LHCb detector is provided with a flexible trigger capable to cope with the harsh hadronic environment, resulting in high efficiency to study decay channels of interest [6].

1.2.1 The LHCb detector

As seen above, the LHCb detector is a single-arm spectrometer with a forward angular coverage approximately the range $10 \div 300$ mrad. The layout of the spectrometer is shown in Figure 1.4, where the right-handed coordinate system adopted has the z-axis along the beam, and the y-axis along the vertical.
LHCb sub-detectors can be conceptually divided into the *tracking system* and the *particle identification system*. To these are added the *trigger system*, key component in the measurement process, responsible for the event selection and data rate (or data *bandwidth*) reduction.

Tracking System

The tracking system is made of the VELO and four planar tracking stations, one in front of the spectrometer magnet and three behind it.

The *VErtex LOcator* (VELO) is the LHCb component with the highest resolution. Its goal is to provide precise measurements of track coordinates close to the interaction region, which are used to identify the primary and secondary vertices of b and c-hadron decays. The VELO consists of a series of silicon modules, each measuring the r and ϕ coordinates, arranged along the beam direction and perpendicular to its line. Two planes are located upstream of the VELO sensors to select the direction of particles origin: they are called the *pile-up veto system*. Another 25 modules provide the experiment to the resolution necessary for analysis [6].

Four planar tracking station follow the VELO: the *Tracker Turicensis* (TT) upstream of the dipole magnet and T1-T3 downstream of the magnet. As the VELO, also the TT station uses *silicon microstrip* detectors. In T1-T3 instead, silicon microstrips are used

[4]Recalling the pseudorapidity definition $\eta = -\log[\tan(\theta/2)]$, where θ represents the angle between particle momentum and beam axis, the range $2 < \eta < 5$ corresponds approximately to $10 < \theta < 300$ mrad.

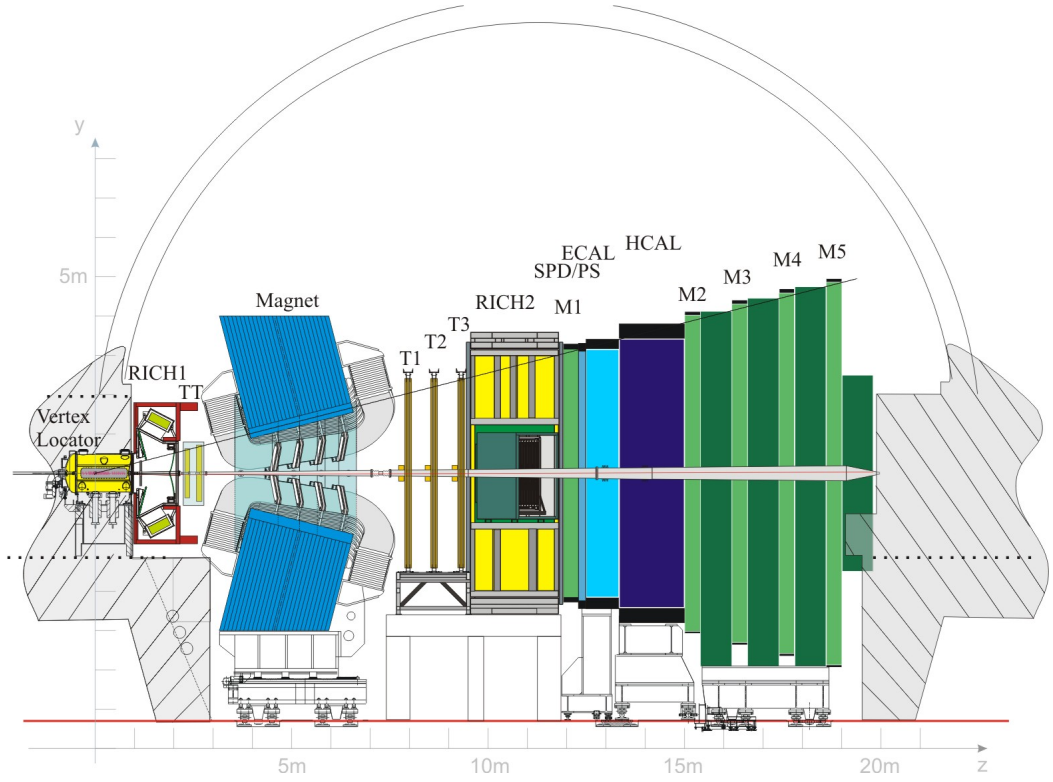


Figure 1.4: View of the LHCb detector.

only in the region close to the beam pipe, whereas *straw-tubes* are employed in the outer region of the stations. Each plane provides a precise measure of particle path coordinates, allowing to reconstruct its momentum with good resolution.

Finally, we have the spectrometer magnet. Properly speaking, the dipole magnet used by the LHCb experiment does not belong to the tracking system. Despite this, it is important pointing out that its magnetic field of 1.1 T allows to measure the momentum of charged particles. In addition, the periodical inversion of the magnetic field can be used to remove any systematic uncertainties due to the asymmetry of the detector from the measures.

Particle Identification System

The particle identification system consists of two RICH detectors, the calorimeter system and five tracking stations for muon identification.

LHCb, unlike the other *general-purpose* experiments of the LHC, is equipped with two *Ring Imaging CHerenkov* (RICH) detectors. These, based on Cherenkov radiation[5], provide the experiment with the capability to effectively distinguish kaons, protons and lighter particles (electrons, muons and pions), all abundantly produced in b- and c-hadrons decays [6].

[5] A charged particle passing through a dielectric medium at a speed greater than the phase velocity of light in that medium, emits an electromagnetic radiation called *Cherenkov radiation*. The radiation emitted angle θ_C is inversely proportional to the particle velocity, following the relation $\cos\theta_C = 1/n\beta$ where n is the refractive index of the medium and $\beta = v_p/c$.

An electromagnetic calorimeter (ECAL) and a hadron calorimeter (HCAL), paired with finely segmented scintillators (SPD/PS), form the calorimeter system that performs several functions. One is to select hadron, electron and photon candidates with a quickly energy measure for the first trigger level (L0). The calorimeter system is also able to identify electrons, photons and hadrons as well as to measure their energies and positions. In addition, it is designed to distinguish π^0 and prompt photons with good accuracy, providing useful input to the flavour tagging algorithms necessary for the physics program [6].

Muons are the only particles to survive the calorimeter system since they are deeply penetrating. Their identification is of primary importance to study CP-violation events and rare b-decays that may reveal NP beyond the SM. For this reason, the muon system provides fast information for the high-p_T muon trigger at the earliest level (L0), as well as muon identification for the High-Level Trigger (HLT) and offline analysis [6].
The muon system consist of five tracking station (M1-M5). Station M1, equipped with *Electron Multiplier* (GEM) detectors in the central region and with *Multi-Wire Proportional Chambers* (MWPC) elsewhere, is placed in front of the calorimeters and is used to improve the p_T measurement in the trigger. Instead, stations M2 to M5, equipped with MWPC detectors, are placed downstream the calorimeters and are interleaved with 80 cm-thick iron absorbers. This ensures that secondary particles produced in the muon interaction with the chambers do not propagate the following stations, mimicking the behaviour of a muon [6].

Trigger

The LHCb experiment operates at an average luminosity of 2×10^{32} cm^{-2} s^{-1}, two orders of magnitude lower than the maximum design luminosity of the LHC. This construction choice ensures to detectors and electronics a slower aging due to radiation damage. Furthermore, the fact that the number of interactions per bunch crossing is dominated by single events, facilitates the triggering and reconstruction task because of the low channel occupancy. Due to the LHC bunch structure and low luminosity, the crossing frequency with interactions visible[6] by the spectrometer is about 10 MHz. This values must be reduced by the trigger to about 12 kHz, at which rate the events are written to storage for further *offline* analysis [6]. This reduction is achieved in two trigger levels, as shown in Figure 1.5: the *Level-0* (L0) and the *High-Level Trigger* (HLT).

The L0 trigger is implemented using custom made electronics, operating synchronously (*hardware trigger*) with the 40 MHz bunch crossing frequency, while the HLT is executed asynchronously (*software trigger*) on a processor farm, using commercially available equipment. The response of both the hardware and the software trigger define the *online* event selection: L0 makes decisions based on information from the calorimeter and muon systems, while the HLT applies a full event reconstruction.
At the LHCb luminosity, the bunch crossings with visible pp interactions contains a rate of about 100 kHz of $b\bar{b}$ pairs. However, only about 15% of these events includes at least one B meson with all its decay products contained in the spectrometer acceptance. The trigger system must be therefore capable to select events of interest efficiently, in order to allow studying rare decays, with typical branching ratios smaller than 10^{-3}, in *offline* analysis. This results in an optimized system designed to achieve the highest efficiency

[6] An interaction is defined to be visible if it produces at least two charged particles with sufficient hits in the VELO and T1–T3 to allow them to be reconstructible.

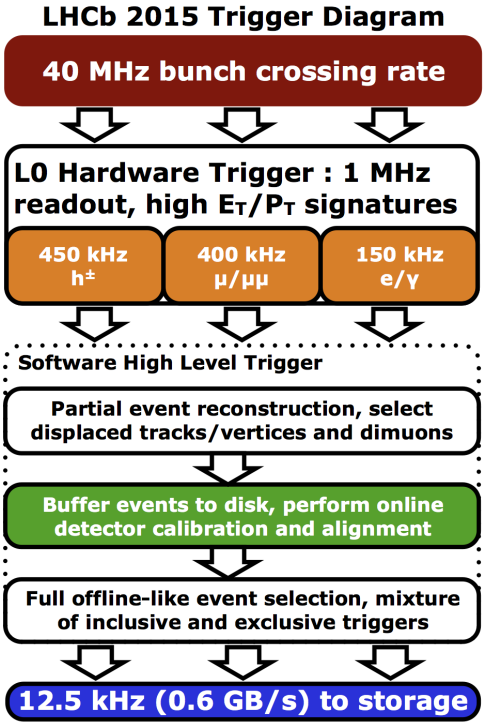


Figure 1.5: Representation of the Run 2 trigger scheme.

for the *offline* analysis, while rejecting uninteresting background events as strongly as possible [6].

1.2.2 Particle Identification Variables

The responses of RICH, calorimeter, and muon systems, or their combinations, associated to each track in the reconstruction process are named for brevity PID variables. The algorithms behind each variable are very different, but everyone consists of computing a likelihood ratio between particle hypotheses for each reconstructed track [7].

The primary role of the RICH system is the identification of charged hadrons (π, K and p), to which is added a contribution to the charged leptons identification (e and μ) complementing the information from calorimeter and muon systems [8]. In order to determine the particle species for each track, the Cherenkov angle θ_C is combined with the track momentum measured by the tracking system. In doing so, each track can be associated with an *assumed* emission point[7], allowing to build the likelihood $\mathcal{L}_{\text{RICH}}$ for each particle hypotheses. Furthermore, operating in a high occupancy environment, the $\mathcal{L}$-algorithm needs that all tracks in the event and in both RICH detectors must be considered simultaneously, to optimize the computing efficiency [9].
Since the most abundant particles in pp collisions are pions, the likelihood maximization algorithm starts by assuming all particles are pions. Then, $\mathcal{L}$ is recomputed iterating over all the tracks in the event, and changing their mass hypothesis to e, μ, π, K and p whilst leaving all other hypotheses unchanged. At the end of iteration, each track is associated

[7]Since the exact emission point of each photon is unknown (it can be anywhere along the particle trajectory through the radiator), the mid-point of the trajectory in the radiator is taken.

with the mass hypothesis that maximizes the likelihood [9]. Typically, the ratio between the computed hypothesis and pion one is reported as RICH-PID variable: it is called *Differential Log-Likelihood* (DLL) or, in this case, `RichDLL`.

The main role of the calorimeter system, in terms of particle identification, is to provide for the recognition of photons, electrons and π^0 candidates. Studying the presence or absence of tracks in front of the energy deposits allows to distinguish charged from neutral particles, while the clusters shape can be used to discriminate between photons and π^0 [8]. Photon identification is performed by the difference in log-likelihood between the photon and the background hypotheses, taking into account the possibility that photons convert when interacting with the detector material upstream of the calorimeter. Two independent estimators are therefore built to establish the photon hypothesis: one for non-converted candidates, based on distribution from PS and ECAL responses, and the other for converted ones, using the information from electron-positron tracks. Furthermore, the misidentification between photons and π^0 is minimized, employing a *feedforward neural networks* classifier[8] to evaluate the difference between the distribution of the expected energy deposit for the two cases [8].
Electron identification is performed using the response from PS, ECAL and HCAL. The procedure to combine these different sources of information is based on signal and background likelihood distributions constructed for each sub-detector [8]:

$$\Delta \log \mathcal{L}_{\mathrm{CALO}}(e-h) = \Delta \log \mathcal{L}_{\mathrm{PS}}(e-h) + \Delta \log \mathcal{L}_{\mathrm{ECAL}}(e-h) + \Delta \log \mathcal{L}_{\mathrm{HCAL}}(e-h) \quad (1.3)$$

where $\Delta \log \mathcal{L}$ denotes the differential log-likelihood, and $\mathcal{L}(e-h)$ represents the likelihood ratio between electron and hadron hypotheses.

The muon system provides information for the selection of high-p_T muons at the trigger level and for the offline muon identification. Here, computing the likelihood follows from performing a loose binary selection of muon candidates, named `isMuon`. The latter provides high efficiency and is based on the penetration of muons through the calorimeters and iron filters. The response of `isMuon` depends on the number of stations where a hit is found within a *field of interest* (FOI) defined around the track extrapolation. Clearly, the number of stations required to have a muon signal is a function of track momentum p, as shown in Table 1.2. This strategy allows to reduce the misidentification probability of hadrons to the percent level [10].
The muon system computes the likelihood only for particles that have passed the `isMuon` check. Defining D^2 as the average squared distance significance of the hits in the muon

[8]Feedforward neural networks are defined in Section 3.2.1.

Momentum range	Muon stations
3 GeV/c < p < 6 GeV/c	M2 and M3
6 GeV/c < p < 10 GeV/c	M2 and M3 and (M4 or M5)
p > 10 GeV/c	M2 and M3 and M4 and M5

Table 1.2: Muon stations required to trigger the `isMuon` binary decision as a function of momentum range.

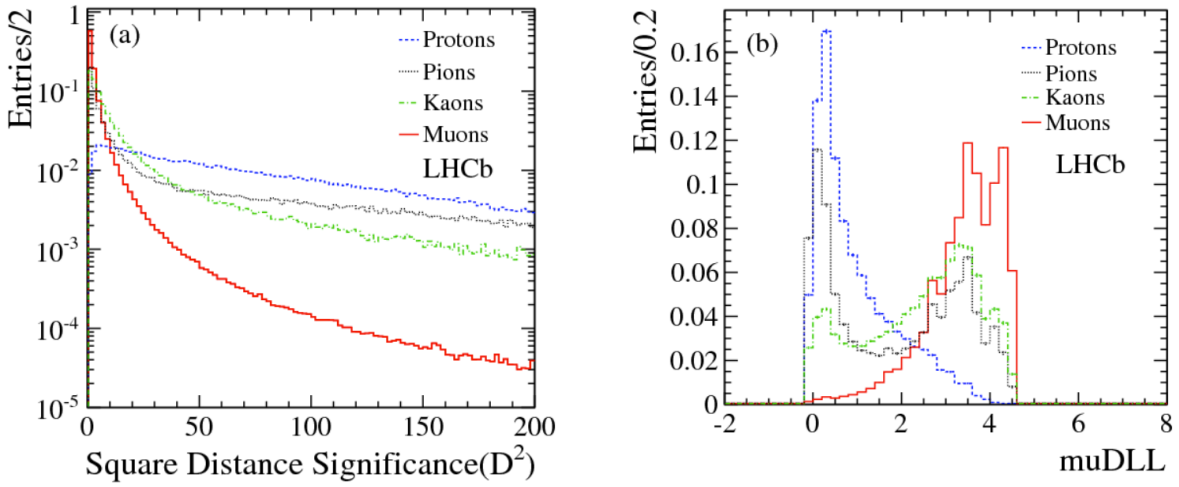
chambers with respect to the linear extrapolation of the tracks from the tracking system, then the likelihood corresponds to the cumulative probability distributions of D^2. The average distance significance is defined as follows:

$$D^2 = \frac{1}{N}\sum_i \left[\left(\frac{x^{(i)}_{\text{closest}} - x^{(i)}_{\text{track}}}{pad^{(i)}_x} \right)^2 + \left(\frac{y^{(i)}_{\text{closest}} - y^{(i)}_{\text{track}}}{pad^{(i)}_y} \right)^2 \right] \tag{1.4}$$

where the index i runs over the the total number of stations (denoted by N) containing hits within the FOI, $(x, y)_{\text{closest}}$ are the coordinates of the closest hit to the extrapolated track point $(x, y)_{\text{track}}$ of each station and $pad_{x,y}$ correspond to one half of the pad sizes along directions perpendicular to the beam.
The distribution of D^2 is reported in Figure 1.6a, for different particle hypotheses: as shown in red, true muons tend to have a much narrower distribution (close to zero) than the other particles, incorrectly selected by the `isMuon` requirement. The likelihood for the muon hypothesis `MuonMuLL` is defined as the cumulative of the red distribution in Figure 1.6a. Instead, the likelihood for the non-muon hypothesis `MuonBkgLL` is calibrated with the D^2 distribution for protons (represented in blue). Indeed, the other charged hadrons (pions and kaons) selected by `isMuon` are characterized by a D^2 distribution with a narrowed component around zero similar to true muons, superposed to another component more similar to distribution for protons. Typically, the logarithm of the ratio between `MuonMuLL` and `MuonBkgLL` is used as discriminating variable and is called muDLL. Its distribution, for different particle hypotheses, is reported in Figure 1.6b [10].

Combined Particle Identification Variables

The PID information obtained separately from RICH, calorimeter, and muon system can be combined to improve a single set of more powerful variables: two different approaches



(a) (b)

Figure 1.6: In (a) it is shown the average square distance significance distributions for muons, protons, pions and kaons as obtained from data, while in (b) it is represented their corresponding muDLL distributions. Plots reported from Ref. [10].

are used. The first method is still based on likelihood computation, and simply corresponds to the linear combination of the information produced by each sub-system. This results in the *Combined Differential Log-Likelihood* (CombDLL), usually defined with respect to the pion hypothesis, and denoted with $\Delta \log \mathcal{L}_{comb}(X - \pi)$, where X represents either the electron, muon, kaon or proton mass hypothesis [8].
The second approach exploits advanced Machine Learning algorithms to define a classifier which combines the likelihoods ratios defined above with the information from the tracking system. The classifier with the widest application in this category, named PIDANN, was developed using *feedforward neural networks* with a single hidden layer activated through a sigmoid function[9]. In doing so, the network outputs are normalized between 0 and 1, reason why PIDANN is often referred to as `ProbNN`. The neural network is trained minimizing the likelihood of the Bernoulli Cross-Entropy with Stochastic Gradient Descent as implemented in the TMVA package [11]. The variables combined to produce the `ProbNN` classifier are listed in Table 1.3.

1.2.3 Particle Identification Performance

LHCb physics program needs an extensively usage of PID information both for trigger and for offline analysis. Being able to measure PID performance and to assess the systematic uncertainties on the selection efficiency due to requirements on PID variables becomes therefore crucially important. The multitude of decay studied and the difficulties in simulating perfectly PID (RICH, calorimeters and muon system) responses pushed the LHCb Collaboration towards the development of dedicated techniques to measure the PID efficiencies. These strategies are based on *calibration samples*, pure samples of electrons, muons, pions, kaons and protons collected through more than twenty exclusive trigger selections [7].

The assumption underlying the usage of calibration samples is that the distribution of PID variables is independent of the selection strategy. That's why one uses exclusive trigger channels: simply avoiding explicit requirements on the PID variables is not sufficient. In fact, the hardware trigger relies on information from calorimeter and muon systems to reduce data frequency up to 1 MHz, while first steps of the HLT include dedicated algorithms to identify high-p_T muons and muon pairs. Thus, the algorithms behind the calibration samples selection must impose requirements in the previous trigger layers, avoiding that a pre-selection introduces biases in PID variables [7].
For each of the five most common charged particle interacting within the spectrometer (e, μ, π, K and p), we choose one or more decay modes with low-multiplicity and large branching ratio. The choice aims to ensure the statistics and purity necessary to the calibration samples, which prefers, for this reason, decay channels only composed of charged particles. Table 1.4 shows an overview of the used modes. Referring to such decays, the calibration samples are collected primarily from the high momentum samples, while the low momentum ones are included to extend the kinematic coverage as much as possible.
The trigger algorithms that select decays reported in Table 1.4, cannot rely on PID variables, or can use such information only for particles not used to measure the performance. It is the case of the *tag-and-probe* strategy. Considering the $J/\psi \to \mu^+\mu^-$ decay as an example, the tag-and-probe strategy allows to define a list of well-identified *tag* muons

[9]The meaning of all this information will be clarified in Section 3.2.1

Tracking
Total momentum
Transverse momentum
Quality of the track fit
Number of clusters associated to the track
Neural network response trained to reject ghost tracks [12]
Quality of the fit matching track segments upstream and downstream of the magnet
RICH detectors
Geometrical acceptance of the three radiators, depending on the track direction
Kinematical acceptance due to Cherenkov threshold for muons and kaons
Likelihood of the electron, muon, kaon, and proton hypotheses relative to the pion
Likelihood ratio of the below-threshold and pion hypotheses
Electromagnetic calorimeter
Likelihood ratio of the electron and hadron hypotheses
Likelihood ratio of the muon and hadron hypotheses
Matching of the track with the clusters in the *preshower* detector
Likelihood ratio of the electron and pion hypotheses, after recovery of the Bremsstrahlung photons
Hadronic calorimeter
Likelihood ratio of the electron and hadron hypotheses
Likelihood ratio of the muon and hadron hypotheses
Muon system
Geometrical acceptance
Loose binary requirement already available in the hardware trigger
Likelihood of the muon hypothesis
Likelihood of the non-muon hypothesis
Number of clusters associated to at least another tracks

Table 1.3: Input variables of the `ProbNN` classifier for the various subsystems of the LHCb detector. Table reproduced from Ref. [7].

Species	Low momentum	High momentum
$e^{\pm}$	$B^+ \to (J/\psi \to e^+e^-)K^+$	$B^+ \to (J/\psi \to e^+e^-)K^+$
$\mu^{\pm}$	$B^+ \to (J/\psi \to \mu^+\mu^-)K^+$	$J/\psi \to \mu^+\mu^-$
$\pi^{\pm}$	$K^0_s \to \pi^+\pi^-$	$D^{*+} \to (D^0 \to K^-\pi^+)\pi^+$
$K^{\pm}$	$D^+_s \to (\phi \to K^+K^-)\pi^+$	$D^{*+} \to (D^0 \to K^-\pi^+)\pi^+$
$p, \bar{p}$	$\Lambda^0 \to p\pi^-$	$\Lambda^0 \to p\pi^-$; $\Lambda^+_c \to pK^-\pi^+$

Table 1.4: Decay modes used to select calibration samples.

of a certain charge using PID information, in order to obtain a list of *probe* tracks with opposite charge selected avoiding any requirement associated to any particle identification detector. The purity of the sample can be improved adding kinematic constraints, such as the invariant-mass of muon pairs consisting with the J/ψ mass. The residual background that cannot be rejected with an efficient selection strategy, can be statistically subtracted using the ${}_s\mathcal{P}lot$ technique [13], as briefly described in Section 4.2.1.

The response of the PID detectors to a traversing particle depends on the kinematics of the particle, the occupancy of the detectors, and experimental conditions such as alignments, temperature, or gas pressure. Then, one may assume that the response of the PID system is fully parameterized by a set of known variables, such as the track momentum p, the track pseudorapidity η, and the total number of reconstructed tracks `nTracks`. By partitioning the sample with sufficient granularity in these parameterizing variables, the PDF of the PID response does not show significant variation within each subset. This is like saying that the efficiency of a selection requirement on that variable is constant within each subset, property that ensures studying its performance with *binned* approach [7].
Collected the calibration samples, we can measure the performance of a specific selection requirement on PID variables. Considering the trivial case of events that come from the calibration sample (same decay modes), the average efficiency is simply given by the fraction of events passing the PID requirement once subtracted the background contribution. Instead, to study the PID performance on a sample other than the calibration sample (different decay channels), it is necessary to compute per-subset efficiencies. Denoting the sample under investigation as *reference sample*, the parameterizing variables in the calibration sample can be weighted to match those in the reference sample. Therefore, defining the per-subset weights as follows

$$w_i = \frac{R_i}{C_i} \times \frac{C}{R} \tag{1.5}$$

with R_i (C_i) the number of reference (calibration) tracks in the i-subset and R (C) the total number of reference (calibration) tracks in the sample, then the average efficiency is

$$\bar{\varepsilon} = \frac{\sum_i \varepsilon_i w_i C_i}{\sum_i w_i C_i} \tag{1.6}$$

where ε_i is the per-subset efficiency. In this sense, the computation of the PID efficiency can be thought of as the reweighting of the calibration sample to match the reference. Typically, the reference sample consists of simulated data to provide the kinematics of the signal tracks under consideration. Nonetheless, when the signal in data can be reliably separated from the other species in the sample, the reference sample can also be made of real, acquired data [7].

1.2.4 Offline Computing and Computing Model

Even if most of the reconstruction is performed in real time during the data acquisition, at the highest level of the trigger, the analysis of the data requires a powerful *offline* computing infrastructure obtained with a network of computing centers around the world connected within the *LHC Computing Grid* (LCG).
The main activities performed with the *offline* computing resources are:

- The simulation of the response of the detector at specific physics processes happening during or immediately after the *pp* collision;
- The re-reconstruction of the acquired data to modify parameters or refine algorithms, defining the *offline* event reconstruction procedure for specific studies that require a dedicated treatment of the raw data;
- The combination of reconstructed elements (such as tracks or energy clusters in the calorimeter consistent with neutral objects) to match them to decays of particles with a lifetime too short to be directly detectable.

Today, the simulation is by far the most expensive task performed with *offline* computing resources and is treated in detail in Chapter 2. The re-reconstruction and the combinatorial tasks have different needs in terms of treatment of the data flows, which are part of the experiment design and are therefore discussed here.
To allow the re-reconstruction of the event, the whole information on the detector response has to be stored on disk, while to study combinations of reconstructed elements, only the ones of potential interest are actually needed. These two different strategies are reflected into two separate *trigger streams* with different *persistency* options: the `Full` stream and the `Turbo` stream. An additional stream, named `TurCal` stream, combines the output of the *online* reconstruction algorithms with the raw data necessary to re-reconstruct the events and has been specifically designed for detector studies and calibration tasks. A schematic representation of the trigger and the related data formats is reported in Figure 1.7.

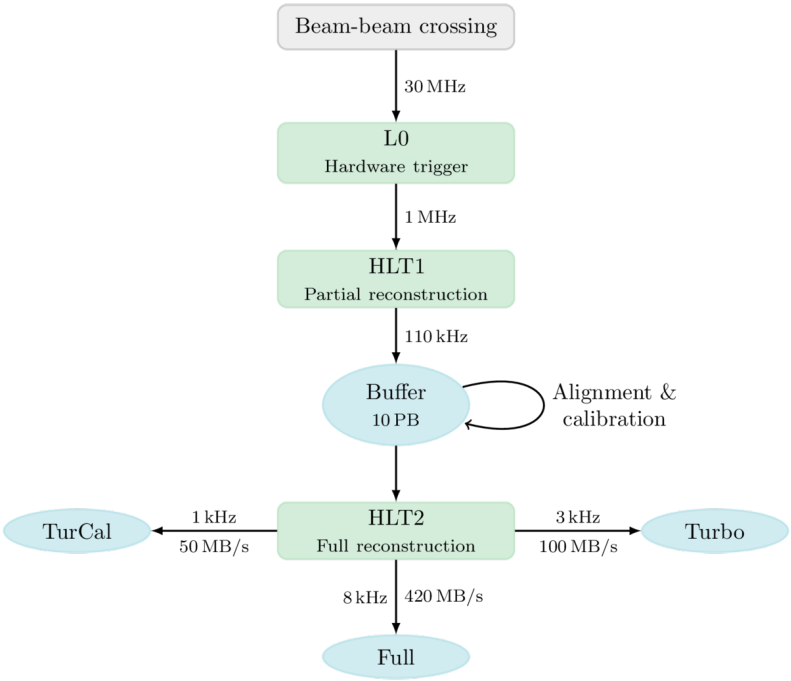


Figure 1.7: An overview of the LHCb trigger scheme in Run 2. The green boxes represent trigger stages, the blue ellipses represent storage units, and arrows between elements represent data flow, labelled with approximate flow rates. Reproduced from Ref. [14].

Trigger selections (also called *trigger lines*) writing to `Turbo` stream are intended for analyses of very large samples where only the information related to the candidates and associated reconstructed objects is needed. Trigger lines that are part of the `Turbo` stream produce a decay candidate which is stored for *offline* analysis, along with a large number of detector-related variables; instead the raw detector data is not kept [15]. Considering this data format, it is therefore evident that the calibration samples must provide the PID information as computed *online* in order to asses the efficiency of selection requirements.

Trigger lines that are part of the `Full` stream are intended for precision measurements and searches for which the `Turbo` approach is not applied. While the software trigger fully reconstructs candidates, those are not saved. If the trigger decision is affirmative, the raw detector data is saved together with summary information on the trigger decision, including the CombDLL and `isMuon` variables, for each particle involved in the trigger decision. The track and decay candidates are reproduced in a further *offline* reconstruction step that accesses the raw detector data. Using this data format aims to accomplish the most-complicated and highest-profile analyses, which typically need to squeeze as much as possible the systematic uncertainties related to the event selection. Therefore, the events selected as part of the calibration samples must include the raw data, allowing dedicated studies of the PID performance down to the detector level [7, 15].

The PID variables play a key role for data analysis, allowing to increase the signal purity of a sample, reducing the processing time devoted to reconstruct background events, and helping in fitting into the data storage constraints. To exploit the PID information, measuring the performance of their selection requirements is equally important, and so it is collecting the calibration samples. This motivates the design of a stream dedicated to the selection of the calibration samples, as described above [15]. After the *online* full event reconstruction, events in which decay candidates useful for calibration are identified and selected in real time, are stored including both the trigger candidates themselves and raw detector data. The two output formats are processed independently for each event, to obtain both decay candidates propagated from the trigger and decay candidates reconstructed *offline* from the raw detector data, which must then be matched [7].

1.2.5 The LHCb upgrades

The Upgrade I of the LHCb experiment is currently in commissioning, exploiting the stop of data taking for the Long Shutdown 2 (2018-2021). At the moment, a new detector is being installed, with more than 90% of the active detector channels replaced. The LHCb Upgrade detector will operate through LHC Runs 3 and 4, driven by a new paradigm for the trigger system providing a much higher efficiency [16].
Starting from Run 3, the instantaneous luminosity will increase by a factor five, from 4×10^{32} to 2×10^{33} cm^{-2} s^{-1}. The upgrade aims to facilitate recording data in this new framework, allowing to collect a dataset of at least 50 fb^{-1} in less than ten years. To this end, the trigger system has been redesigned, maximizing its event output rate.

One of the main limitation of the experiment is that the collision rate must be reduced to the readout rate of about 1 MHz within a fixed latency. This reduction is achieved by the hardware trigger L0 using information from the calorimeter and muon systems. It is at this level that the largest inefficiencies in the entire trigger chain occurs, especially for purely hadronic decays. Therefore, the LHCb Upgrade detector implements a trigger-less readout system, removing the bottleneck of L0 [17].

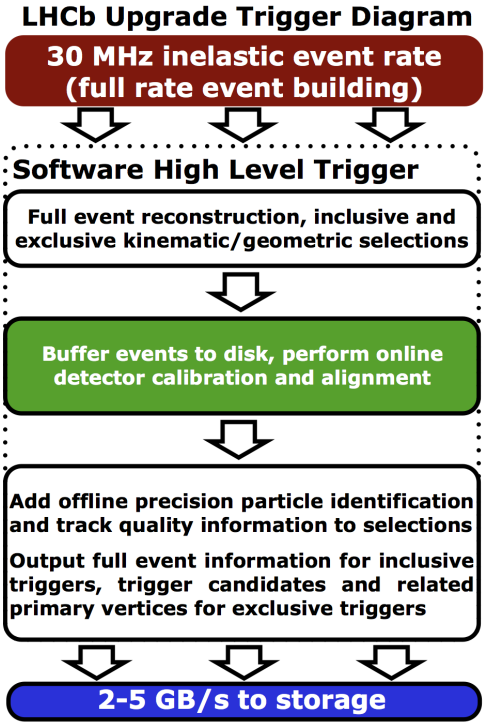


Figure 1.8: Representation of the Run 3 trigger scheme.

The new paradigm adopted leaves to a full software trigger reconstruction and selection of the events. This is possible using the Event Filter Farm (EFF) which allows to sustain triggering capabilities for rates up to the inelastic pp collision rate of 30 MHz. In architecture, the trigger is similar to the one used for Runs 1 and 2, and is composed of two main blocks: the *Low-Level Trigger* (LLT) and the *High-Level Trigger* (HLT). The LLT is essentially the previous hardware trigger modified to run within the new readout architecture. The HLT is again divided into two parts, HLT1 and HLT2, which are executed sequentially. The former performs synchronously a first full track reconstruction, while the latter processes high-quality calibration and alignments to finalize asynchronously event reconstruction [16, 17]. Figure 1.8 reports the new trigger scheme adopted for the LHCb Upgrade detector.

Since calibrations and alignments are performed on EFF, there will be no need to re-reconstruct events *offline*. Therefore, the *offline* data processing will be limited to streaming[10] and the storage costs for recorded data will be essentially driven by the trigger output rate. Consequently, the vast majority of *offline* CPU work will be dedicated to the production of simulated events [18].

The LHCb experiment will be upgraded again during the Long Shutdown 3 (2025) to Upgrade Ib, and during the Long Shutdown 4 (2030) to Upgrade II. The LHCb Upgrade II program aims to make full use of the capabilities of a forward acceptance detector during the *High Luminosity LHC* (HL-LHC) operational period. After the Upgrade II, the LHCb experiment will operate at instantaneous luminosities of up to 2×10^{34} cm^{-2} s^{-1}, and will accumulate a data sample corresponding to a minimum of 300 fb^{-1}, as shown

[10]Considering the higher purity of selected data, any *offline* data processing will be employed with limited additional selections.

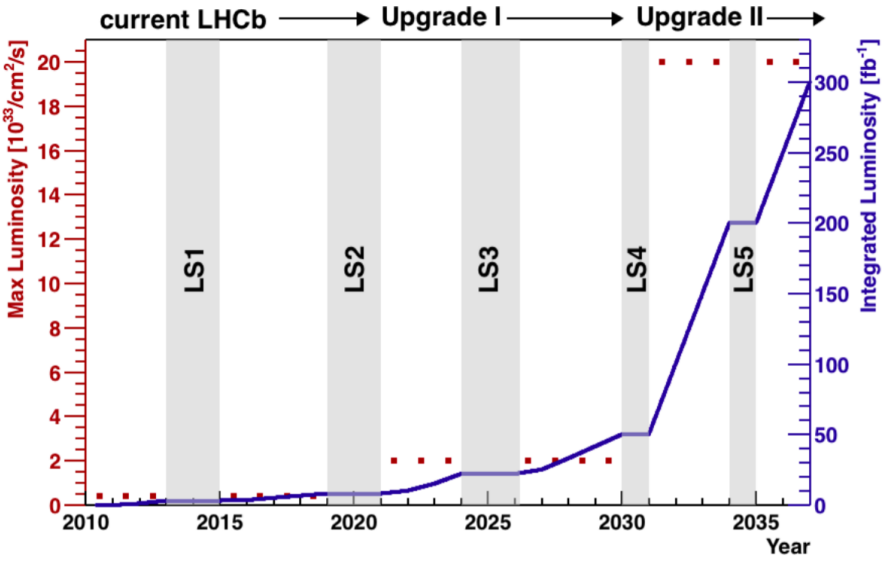


Figure 1.9: Luminosity projections for the original LHCb, Upgrade I, and Upgrade II experiments as a function of time [19].

in Figure 1.9. The HL-LHC will increase significantly the number of interactions per bunch crossing, and will force the LHCb Collaboration to develop and use cutting-edge technologies in order to face unprecedented radiation and occupancy conditions [19].

Chapter 2

The Simulation Paradigm

2.1 Computing Requirements for Upgrade

The experiment upgrade currently in commissioning will provide LHCb with the necessary improvements to increase the instantaneous luminosity by a factor five, while the purity reached by a full software trigger will allow to improve the selection efficiency by at least a factor two. This new experimental framework will demand a comparable upgrade of the computing resources to face the increasing of the event throughput to offline storage by at least an order of magnitude. In addition, we can expect that also data samples will increase roughly by at least a factor ten, imposing strict requirements to the simulation processing in order to achieve the highest possible physics accuracy.

The data processing will follow the same strategy adopted for the logic workflow of Run 2, ending each of the trigger lines to one of the three `Turbo`, `Full` or `TurCal` streams, then stored offline. The majority of triggered events, about 70% of the total, will be sent to the `Turbo` stream, where only high-level information is saved to offline, while the raw data is discarded: this is the case of events selected by exclusive trigger lines, such as for charmed decays. More inclusive trigger lines as well as calibration ones (the remaining 30%) will be saved in the `Full` and `TurCal` streams, where the entire event is persisted. Since calibration adjustments are already applied at trigger level, the offline reconstruction step needed for calibration could be skipped, saving CPU resources. Consequently, any offline data reprocessing in addition to reconstruction is limited to data filtering, which is known in LHCb as *stripping*. In the stripping process, many selection algorithms (*stripping lines*) are executed, in order to save only the interesting parts of the event and to increase the signal purity. The CPU needs for the offline computing resources will be therefore dominated by Monte Carlo (MC) simulation, as clearly shown in Figure 2.1 [16, 20].

Simulated samples play a key role in designing, building and commissioning the LHCb Upgrade detector and the data processing applications. They are also fundamental for physics analysis, contributing to the precision of physics measurements. In this sense, the simulation requirements by *Physics Working Groups* (PWG) are strongly analysis-dependent and it is not easy to quantify them. In general one needs more simulated events for the signal of interest than those collected, hence it is not surprising that LHCb, until 2017, produced, reconstructed, recorded and analysed tens of billions of simulated events. Figure 2.2 reports the total number of simulated events produced until 2017 compared

with the integrated luminosity achieved by LHCb during Runs 1 and 2 [18].
Consequently to what said above, the number of events to be simulated for Run 3 will need to increase in order to face the increasing of the integrated luminosity by at least a factor ten. Furthermore, it should be pointed out that, for some analyses with large samples, the amount of MC events that can be produced is already insufficient, and the uncertainty due to the limited amount of simulated samples is, in many cases, already dominant. This scenario has started becoming predominant in Run 2, where the distributed computing resources were about 80% of the total available to LHCb. In Run 3, with the reconstruction being done in real time, the distributed computing resources will be used essentially only by simulation and user analysis. Nevertheless, the upgraded computing budget will not be able to support MC demands, as demonstrated by the green bars always above the pledge line in Figure 2.1. The solution adopted for Run 3 in order to cope the computing requests is to optimize the simulation process, mixing the standard *full* simulations with faster options. By referring again to Figure 2.1, one may be convinced how the usage of fast and ultra-fast simulations allows to make the analysis requirements sustainable with respect to the computing resources available. The LHCb Collaboration pursues the ideal goal of simulating all the events needed for the different analyses by adopting time performance improvement strategies which does not degrade unacceptably physics accuracy [18].

2.2 Full and Fast Simulation

A common challenge of the most of physics analyses performed at LHCb is the need for a deep understanding and precise modelling of the effects of the detector response on the physics parameters of interest. This response is driven by resolution effects that degrade the true distribution of a certain quantity and by inefficiencies that are introduced by either an imperfect reconstruction in the detector or an incorrect event selection. The result is an unknown number of true events to be found within a set of reconstructed events. Obtaining an unbiased estimation of the physics parameters of interest requires therefore the correction of these effects. This is why one generates MC events and simulates the detector response: the goal is to study the evolution of the statistical distributions describing the collision event from the physics model through the reconstruction and selection algorithms.

In order to generate simulated events, two applications are used in LHCb: Gauss, in charge of the event generation and particles tracking in the detector, and Boole, in charge of the simulation of the detector response. The samples that output from Boole are not dissimilar to true data and are injected to the trigger lines to take into account also the selection effects. A schematic representation of this chain is shown in Figure 2.3. The simulation software is developed within the Gaudi software framework that has been developed to allow running all LHCb applications in a multi-threaded environment. This results in a significant reduction of the memory footprint, thereby opening the possibility to use resources, such as *High Performance Computing* (HPC) farms and many-core architectures, where the memory per logical core is smaller than that of the grid computing nodes currently used by LHCb [20].
From here on, we will dwell mostly on the Gauss application, which represents the component with the hardest computing requirements, and which has posed the majority

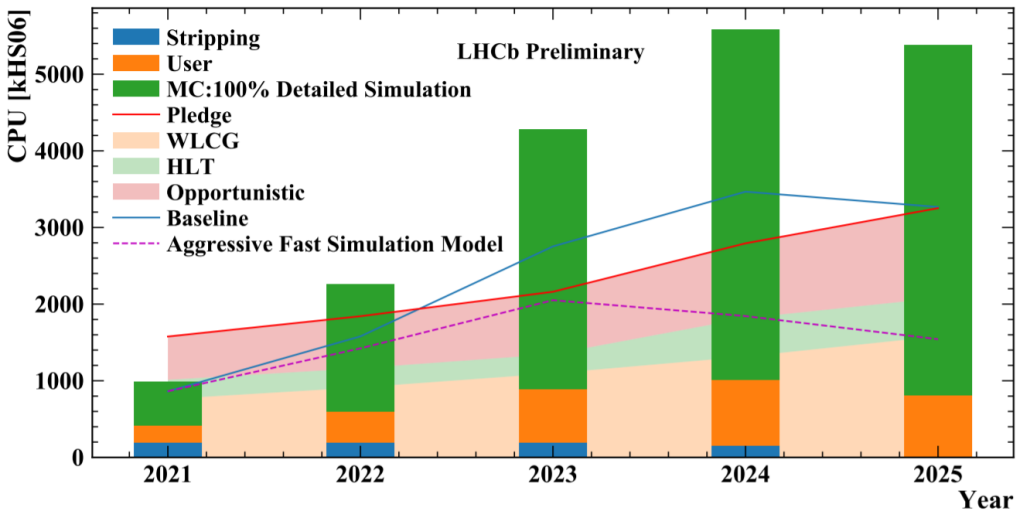
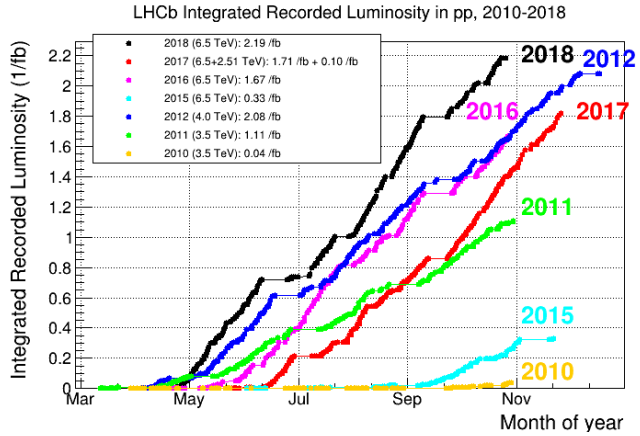
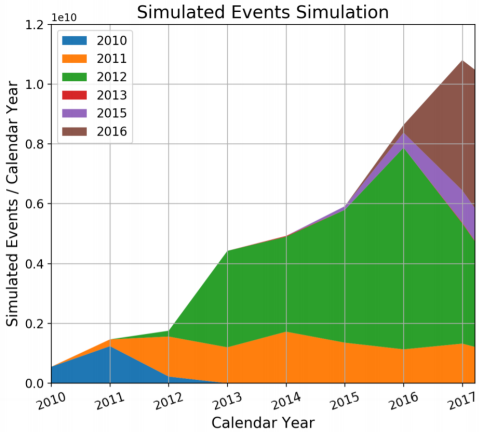


Figure 2.1: Estimated CPU usage for the LHCb experiment from 2021 through 2025. The bars give the total CPU usage expected, composed of stripping jobs (blue), all estimated user jobs (orange) and simulation production (green). The green bars here assume only fully detailed simulation to be run. CPU usage is also considered under different scenarios: the baseline scenario (light blue line) uses a 40%/40%/20% split between full, fast and ultra-fast simulations, while the aggressive scenario (dashed purple line) increases to 30%/50%/20% the fast simulation usage. These two trends are compared to the computing resources available, represented by the pledge (red line). Reproduced from LHCB-FIGURE-2019-01.



(a) *LHCb integrated luminosity of Runs 1 and 2.*



(b) *Number of simulated events for a given data-taking year produced every year.*

Figure 2.2: Comparison between the integrated luminosity recorded by LHCb during Runs 1 and 2 (a), and the amount of simulated samples produced for each data-taking year during the same period (b), as reported in Ref. [16]. Notice the relation between the luminosity reached every year and the corresponding colored area in the simulation report.

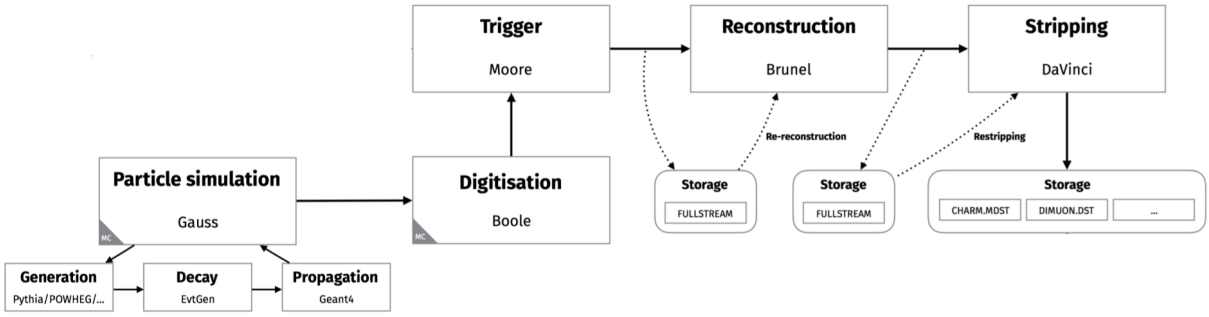


Figure 2.3: The LHCb event simulation flow [16].

of challenges for Upgrade I. Gauss is structured in two main independent phases (event generation and detector simulation) that can be run together or separately. Normally they are run in sequence as a single job since the first phase is much faster than the second one [18].

The first phase consists of the event generation from pp collisions to the decay of particles in the channels of interest for the LHCb physics program. The generation phase itself is structured as an algorithm that delegates specific tasks to dedicated tools implementing well defined interfaces. For example, the Pythia program [21, 22] is exploited to simulate the evolution of pp collisions until their complex multi-hadronic final states, while the EvtGen package [23] is specialized in performing heavy flavour decays which take into account CP-violation processes.

The second phase of Gauss consists in the tracking through the LHCb detector of the particles produced by the generator phase. The simulation of the physics processes, which the particles undergo when travelling through the experimental setup, is currently delegated to the Geant4 toolkit [24–26]. Latter is able to simulate radiation-matter interactions effectively, reproducing electromagnetic, hadronic and optical processes. This results in high CPU-consuming computations which are the real bottleneck, in terms of time performance, of the entire simulation process.

The detector response simulation as product of all the particle-matter interactions is called *full simulation*. As shown above (Figure 2.1), a similar approach is not sustainable for the computing resources available, and this is why *fast simulation* has been introduced. Since Geant4-based phase consumes more than 90% of the event simulation time, a faster option is represented by renouncing to the *full* simulation of the detector, preferring to reproduce the experimental setup, and hence the radiation-matter interactions, only partially. In the *TrackerOnly* method, for example, only the simulation of the tracking detectors is performed, while the *RICHless* method performs a simulation where the two RICH detectors are switched off. In the following it is discussed the *ReDecay* technique, where the simulation speed-up is achieved by reusing the non signal part of a fully-reconstructed event multiple times [16].

2.2.1 The ReDecay Approach

Studying decays of heavy particles to exclusive final states where individual children are reconstructed for each particle, all long lived particles in the event can be split into two distinct groups: the particles that participate in the signal process, and all remaining

ones hereinafter referred as the rest of the event (ROE). These cases are typically characterized by signal process composed by a few particles, while most particles are part of the ROE. Therefore, the majority of the computing time per event is spent on simulating particles that are never explicitly looked at: the opposite scenario to what we want. Ideally, most of the computing resources should be spent to simulate the signal decays themselves. Nevertheless, we cannot simply renounce the ROE simulation because it would result in a much lower occupancy of the detector, and consequently in a significant mis-modelling of the detector response, with underestimated resolution effects and overestimated reconstruction efficiencies [27].

ReDecay approach mitigates these problems simulating only the signal process and reusing the ROE multiple times instead of generating a new one for every event. Therefore, given a fixed ROE, the signal decays are reproduced multiple times starting from identical values for the origin-vertex position and kinematics. While the starting point of the signal particle is fixed in order to preserve the correlations with the ROE, the decay time and thus decay vertex as well as the final state particle kinematics are different [27]. Algorithm 1 describes schematically the simulation process of ReDecay.

Algorithm 1 ReDecay simulation process. Typically N_{ReDecay} is of the order of 100.

Require: Exclusive heavy flavour decay
Require: Number of iterations N_{ReDecay} per fixed ROE
1: **while** The size of the simulation sample not reached **do**
2: Generate *full* MC event including the signal decay
3: Save origin-vertex position and momentum of the signal particle (before the generated particles are passed through the detector simulation)
4: Remove signal particle and its decay products from the event (before the generated particles are passed through the detector simulation)
5: Simulate the remaining ROE as usual (passing through the detector simulation)
6: Keep the entire output (information on the true particles and the energy deposits in the detector)
7: **for** N_{ReDecay} iterations **do**
8: Generate and simulate signal decay using the saved information on origin-vertex position and momentum
9: Merge the persisted ROE and the signal decay
10: Write out the combined sample to disk as a *full* event
11: **end for**
12: **end while**
13: **return** Simulation sample

It is important to note that, given a new signal decay, latter and its ROE are digitized simultaneously (merge-step in line 9). This is done to ensure that the energy deposits produced by the signal and ROE particles can interfere, as occurs in the standard method to simulate events. On the contrary, different complete events, for example obtained combining the same ROE with different signal decays, are digitized independently [27].

Summarizing, in ReDecay approach hadrons are decayed independently and the quasi-stable tracks are propagated through the detector individually. The approximated process allows to reproduce efficiencies and resolutions identical to those found in *full* simulation.

In addiction, with increasing N_{ReDecay}, more and more computing time is spent to simulate the detector response for the signal particle and its decay products, contrary to the *full* simulation scenario. However, achieving this goal is paid with a correlation between events stemming from the same origin and, consequently, having identical kinematics. The correlation results in an increasing of the statistical uncertainties related to the simulation sample. Taking into account this contribution is not easy, but can be done comparing the ReDecay-based simulation sample with a pseudo-sample obtained from it[1] (for additional detail refer to Ref. [27]).

ReDecay approach has proved to be appropriate for some analysis, especially in *c*-sector. It is currently used in production with a CPU gain of a factor between 10 and 20 depending on the multiplicity of the decay of interest [18].

2.3 Ultra-Fast Simulation

The LHCb Upgrade detector will require larger simulation samples, increasing their size of at least one order of magnitude. As already stated, the computing resources are not able to produce them in time for the analysis needs, and it is therefore necessary to simplify the simulation process from *pp* collisions to the detector response. The GEANT4-based step is the hardest of the entire chain, requiring the *full* computation of radiation-matter interactions within the experimental setup. It immediately follows that faster solutions can be obtained renouncing to radiation-matter interactions partially: for example, one can disable specific processes (*e.g.* Cherenkov effect in RICH detectors), simplify the geometry (*e.g.* removing sub-detectors) or re-use the underlying event (such as in ReDecay approach). All these cases are examples of *fast* simulations.

However, fast simulations alone are not sufficient to respond to the computing time speedup necessary for Run 3. To this end, the LHCb Collaboration has started to develop alternative methods with the aim of modelling the radiation-matter processes of the detector thanks to parametric or non-parametric techniques (or *regressors*). In other words, the effect of the detector and of the reconstruction algorithms is encoded in parametric formulas or non-parametric predictions. Latters provide an estimate of the reconstructed quantities including a random component capable to simulate the experimental errors or contributions from missing input variables. Following a similar approach allows to speed up significantly the simulation process, and it is therefore called *ultra-fast simulation*. Typically, *ultra-fast* options pay their time performances reproducing resolution effects and efficiencies of the detector with lower quality. Despite this, *ultra-fast* strategies are able to provide huge samples intended to all those analysis not requiring high physics accuracy. In the following, some examples of *ultra-fast* simulation are reported: DELPHES and RAPIDSIM as parametric solutions, and FALCON as non-parametric one.

2.3.1 The DELPHES Framework

DELPHES is a fast-simulation framework whose goal is to allow the simulation of a multipurpose detector (such as ATLAS and CMS detectors) for phenomenological studies.

[1] A common approach is the so-called *block bootstrapping* where the sample is divided into blocks. In order to capture the correlations arising in the ReDecay approach, a block is naturally given by all events using the same ROE.

The simulation includes a track propagation system embedded in a magnetic field, electromagnetic and hadron calorimeters, and a muon identification system. The DELPHES simulator allows to reconstruct tracks and calorimeter deposits parameterizing the detectors response. It can also perform particle identification, and is able to produce high-level objects combining information from different detectors. Then, the framework outputs observables such as isolated leptons, missing transverse energy and collection of jets which can be used for dedicated analyses [28].

Detector Response Simulation

The sub-detectors included in DELPHES (tracking, calorimeter and muon systems) are organized concentrically with a cylindrical symmetry around the beam axis. The user may specify the detector active volume, the calorimeter segmentation and the strength of the uniform magnetic field. The first step carried by DELPHES is the propagation of long-lived particles extracted from input files within the magnet-sensitive zone (or *tracking volume*). Charged particles have a user-defined probability to be reconstructed as tracks in the central tracking volume. Then, a smearing on the norm of the transverse momentum vector is applied. Neutral particles instead follows a straight trajectory from the production point to a calorimeter cell [28].

After their propagation in the magnetic field, long-lived particles reach the calorimeters. The ECAL measures the energy of electrons and photons, while the HCAL measures the energy of long-lived charged and neutral hadrons. The geometrical characteristics of both calorimeters can be set in the configuration file. Long-lived particles reaching the calorimeters deposit a fixed fraction of their energy in the corresponding ECAL (f_{ECAL}) and HCAL (f_{HCAL}) cells[2], which are then grouped in a calorimeter tower. Two independent resolution effects ($\sigma_{ECAL}, \sigma_{HCAL}$) are then applied to the total energy deposited on a tower, whose position in the (η, ϕ) plane is smeared in turn [28].

Object Reconstruction

The DELPHES framework includes a roughly emulation of the particle-flow reconstruction philosophy used in ALEPH and CMS to reconstruct and identify all particles individually. The simplified particle-flow algorithm produce two collections of 4-vectors exploiting the information from tracking and calorimeter systems: tracks associated with calorimeter towers form the *particle-flow tracks*, while towers with no tracks are converted into *particle-flow towers*. These objects are then used to produce high-level observables. Indeed, DELPHES is able to perform a realistic simulation, identifying isolated leptons and photons, reconstructing and tagging hadronic jets, and quantifying the missing energy [28].

Software implementation

The DELPHES software is a modular framework written in C++ and based on the ROOT analysis framework [29]. The modular system allows the user to configure and schedule modules, add modules, change data flow and alter output information via a configuration file (named *card file* and written in Tcl). Event files coming from external MC generators

[2]During the configuration phase, it is possible to customize the fraction values (f_{ECAL}, f_{HCAL}) for each long-lived particle species.

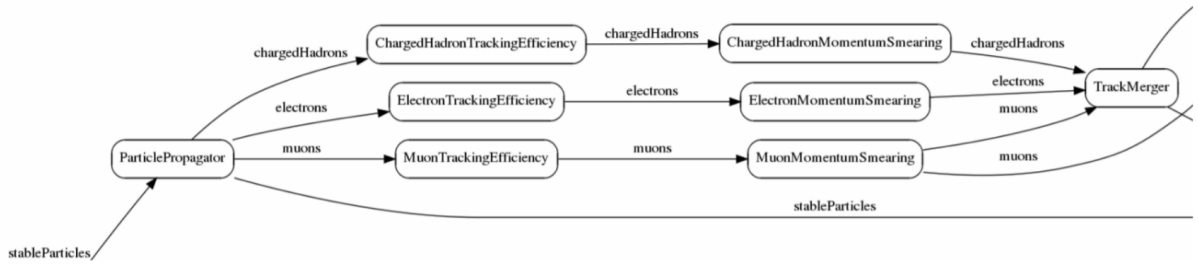


Figure 2.4: Scheme of the DELPHES data flow through the tracking system modules.

can be passed to DELPHES by a `reader`, which converts stable particles into a collection of universal objects. This collection is then processed by a series of modules that transforms the passing data, performing efficiency correction or adding smearing effects. Figure 2.4 shows different species of particles passing through the tracking system modules. Finally, reconstructed and high-level objects are stored in a ROOT tree format in order to analyze the simulated events [28].

2.3.2 The RAPIDSIM Application

When a specific decay channel is selected to measure physics parameters, understanding the kinematic properties of the background is as crucial as studying the signal ones. Indeed, final states similar to the signal channel or misidentified particles from detector inaccuracy can degrade the signal contribution in the invariant mass distribution: modelling the background shape becomes therefore fundamental. To this end, one method consists in generating large samples of the decays and pass them through the *full* detector simulation and reconstruction software chain in order to study potential background sources. The disadvantage of this approach is the typically long time required to generate, reconstruct and select these background samples and the mass storage requirements to retain these samples [30].

RAPIDSIM is a lightweight application for the fast simulation of phase space decays of b- and c-hadrons, providing large samples to study signal and background properties. The speed of generation allows analysts to quickly perform preliminary studies. RAPIDSIM supports the generation of a single decay chain with any number of sub-decays, namely it follows a *particle-gun*-like approach. This offers a significant time saving, but requires input histograms from which extracts the kinematic distribution of the generated particles. By default, RAPIDSIM generates all decays flat across the available phase space, however, inputting the specific histograms, it is possible to generate decays according to any distributions (for more details, see Ref. [30]).
RAPIDSIM utilises the ROOT software package [29] and, specifically, the `TGenPhaseSpace` class to perform the fast generation. In addition to generating decays in 4π, it is possible for the user to specify[3] that the decays of interest fall within the geometrical acceptance of the LHCb experiment. Furthermore, RAPIDSIM can accommodate basic user-defined kinematic efficiency effects during generation and misidentification of final-state particles. All of these features are implemented in a generic way to enable them to be configured for any decay chain [30].

[3]The simple design allows users to extend RAPIDSIM to include alternative detector geometries.

2.3.3 The FALCON Simulator

FALCON is a non-parametric simulator whose idea consists in modelling the detector response function as a huge, highly optimized, lookup table that maps objects at the parton or particle level to objects at the reconstruction level. This approach was firstly proposed by the CDF Collaborations with TURBOSIM [31]. In this sense, FALCON simply represents an updated version of TURBOSIM.
The FALCON package comprises two components. The first, called `builder`, abstracts the detector response function from existing fully simulated events and creates a database containing a non-parametric representation of the function. This is done associating the kinematic information at generator level to the reconstructed one as produced by the *full* simulation. Then, the second component, called `simulator`, is able to use this database to simulate the detector response starting from new generated events. To optimize the nearest neighbour search, in order to obtain information at reconstruction level, it is important to choose a smart data structure for the database. FALCON uses a *k-d tree*[4], a binary tree in which every leaf node is a k-dimensional point [32].

2.4 The New Simulation Paradigm

In the early 2000s, when the world's largest and most powerful particle accelerator has been going to be fired up, the scientific community had to face the production and processing of an unprecedented amount of data. It was before the *Big Data* era, and commercial technologies capable to cope such requests did not yet exist. Therefore, the CERN community launched the *Worldwide LHC Computing Grid* (WLCG), a global collaboration of computer centres with the purpose of providing the resources necessary to store, distribute and analyze the 15 PB of data generated every year by LHC.
Since 1999, in fact, it rapidly became clear that the required computing power was far beyond the funding capacity available to the CERN experiments. On the other hand, most of the laboratories and universities collaborating on the LHC had access to national or regional computing facilities. These were integrated into a single LHC computing service in 2002, setting up the WLCG. Then, the computing strategies of CERN can be described as prolonged storing of data produced by the accelerator followed by later processing spread over time, exploiting the computing resources provided by the WLCG. This scenario is changing driven by the technological development of last decades.

The global spread of the internet and the increasing digitization of the society have allowed to collect huge data samples. This has led the industry to invest significant capitals, boosting the distributed computing development. Today, in fact, companies as Google or Facebook produce and process data samples orders of magnitude larger than LHC does. This has resulted into a proliferation of open-source software resources to develop and implement optimized computing farms and software for distributed computing.
Also the scientific community benefits from this new paradigm. The decrease in the price of storing and processing data (Figure 2.5) can be used to develop technological solutions in order to analyze larger and larger physics samples. Referring to LHCb, this kind of studies has led to implement a trigger capable to perform the reconstruction process in

[4]The search operation on k-d trees require, in the average case, time proportional to the logarithm of the total number of tree entries n, namely $\mathcal{O}(\log n)$, but $\mathcal{O}(n)$ time in the worst case.

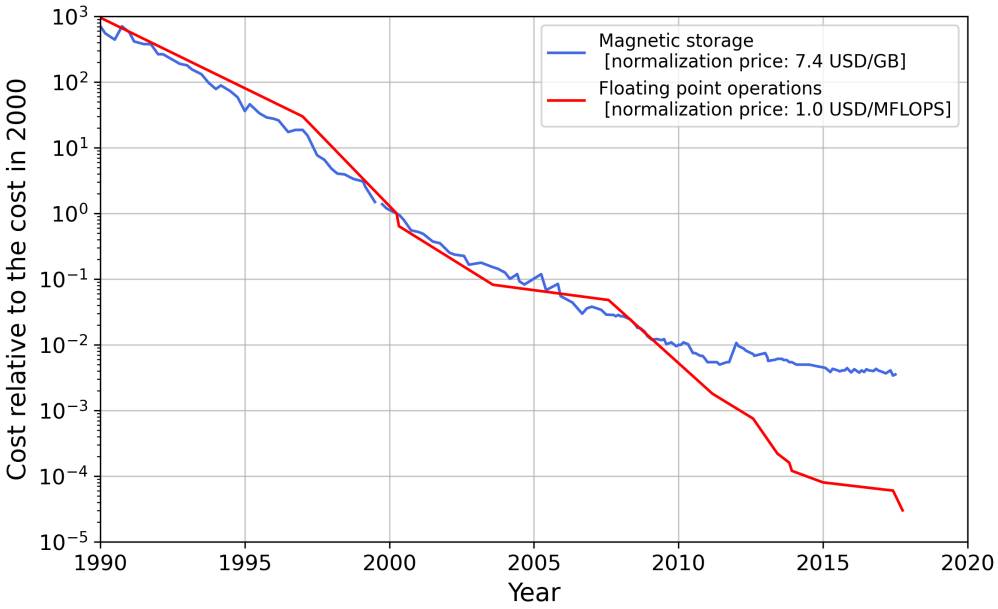


Figure 2.5: Historical average cost for storing (blue line) and processing (red line) data. The information is taken from `https://hblok.net/blog/storage/` and `https://aiimpacts.org/wikipedia-history-of-gflops-costs/` respectively.

real time. This witnesses a changing of the previous paradigm: processing larger amount of data as soon as possible becomes progressively more convenient than storing them, as shown by the different price trends in Figure 2.5.

Repeating what said above, the upgraded trigger will allow to spend most of the computing resources for the simulation process. Nevertheless, these are not sufficient to produce simulated samples large enough for physics analyses, using *full* methods. This leads to the introduction of *fast* and *ultra-fast* approaches.

The ultra-fast simulations represent a very promising solution to optimize the samples production. They have excellent time performance that pays with a lower physics accuracy. However, it is important to point out that this approach produce variables at reconstruction level, allowing to perform quickly statistical analyses.

This thesis work consists in proposing an ultra-fast non-parametric solution for the simulation framework of the LHCb experiment. The detector response at reconstruction level is obtained using *Generative Adversarial Networks* (see Section 3.3), which allow to reproduce faithfully the distributions of the PID variables defined in Section 1.2.2.

Chapter 3

Deep Generative Models

3.1 Machine Learning: an Overview

Machine Learning is the science of programming computers so they can *learn* from data. In this sense we can talk about science of learning, a field shared with Statistics, Data Mining and Artificial Intelligence. A more formal definition follows:

Definition (Tom Mitchell, 1997)**.** A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Recent decades have witnessed Machine Learning field rapidly growing, until it has become a crucial technology outside the narrow circle of experts, capable to conquer the industry [33]. First success commercial applications of Machine Learning date back to the 1990s, when obtaining good results was often regarded as being more an art rather than a technology. The point-break was that the amount of skill required to make these algorithms work reduces as the amount of training data increases. So, from one side, the recent progress in Machine Learning has been driven by the increasing digitization of society and amount of data recorded: the age of *Big Data*. On the other hand, computational cost reduction has had a key role allowing the increase of model size over time thanks to the availability of faster CPUs and thanks to the advent of general purpose GPUs [34].

Many Machine Learning algorithms focus on function approximation problems, where the task T is embodied in a parameterized function f_θ. The experience E consists of a sample of known input-output pairs (x, y) from which extracts $f : x \to y$. In this context[1], the learning problem corresponds to improving the performance P defined by the *loss function* (or *cost function*) $L(y, f(x))$. Training the model f_θ means finding values for θ that optimize the performance metric [33]:

$$\theta^* = \arg\min_{\theta \in \Theta} \mathbb{E}_{(x,y)\sim p}[L(y, f_\theta(x))] \tag{3.1}$$

where Θ is the parameter space, p is the joint probability distribution for (x, y) and $\mathbb{E}_{(x,y)\sim p}[L(y, f_\theta(x))]$ denotes the expected value of the loss function with respect to p.

[1]The scenario described goes by the name of *supervised learning* because of the presence of outcome variables to guide the learning process.

In general, most of Machine Learning algorithms aim to make predictions: this means to be able to generalize examples never seen before, starting from a training data. However, obtaining a good performance measure, like the loss function, only on the training set is insufficient, and this because the true goal is to perform well on new instances [35]. In order to measure the ability of the prediction to generalize to new samples, it is customary to reserve part of the labeled dataset, usable for training, for the validation of the algorithm. This sample, not used for during the training phase, is named *test sample*.

3.1.1 Types of Learning Problems

Finding the solution to a learning problem depends both on the training set and on the specific task required. Many different types of Machine Learning problems exist and it is useful to classify them with the following criteria:

- The level of human supervision required for training (categories include supervised, unsupervised, semi-supervised, and reinforcement learning);
- The ability of the systems to learn incrementally from a stream of incoming data (online versus batch learning);
- The learning strategy, for example there are systems based on comparing new data point to known data points, or instead on detecting patterns in the training data and building a predictive model (instance-based versus model-based learning).

These criteria are not exclusive and can be combined according to the chosen strategy [35].

Supervised/Unsupervised Learning

In *supervised learning*, as the previous example, the training data consists of input-output pairs, with which the algorithm builds a model predicting the desired solutions (outputs), called *labels*. The input set is made up of different *features* containing the information to predict a qualitative output (for *classification*), or a quantitative one (for *regression*).

The training set of *unsupervised learning* problems is unlabeled, hence the algorithm extracts information only from the feature space.

When the training data consists of unlabeled instances and labeled ones, we talk about *semi-supervised learning*. The solution to this kind of learning problems is usually a combination of unsupervised and supervised algorithm.

Finally, *reinforcement Learning* is a different learning system in which an *agent* has to choose the best strategy based on a reward-penalty scheme.

Batch and Online Learning

In *batch learning*, the system is incapable of learning incrementally from data, and instead it can only be trained over all the available instances. The computational cost of this strategy is generally high and forces to pursue it offline: it is referred to as *offline learning*.

A complementary strategy is the *online learning*, where the system can improve adding new data instances sequentially, either individually or in small groups called *mini-batches*.

Instance-Based Versus Model-Based Learning

When the system learns the examples by heart and makes predictions based on a similarity measure with learned examples, we talk about *instance-based learning*. If instead, during the training step, the system builds a model that uses to generalize new instances, then it is a *model-based learning* problem.

3.1.2 Issues on Learning Process

For the way in which we have defined the learning strategies, it is obvious that the achievement of a specific task depends on both the quality of dataset and on the algorithm chosen.

Thinking of an instance-based learning system, it is self-evident the key role of the data sample, but also for the model-based ones the makeup of training set is crucial [36]. The presence of noisy instances or of irrelevant features can prevent from accomplishing the task, and having non-representative training data makes any kind of generalization impossible. It is therefore evident that *data preprocessing* step is as important as the training one, since the preprocessing step is responsible for cleaning up of the data sample.

From the other side, there is the role played by the model chosen that can be too complex or too simple with respect to the available training set. In the first case, the noisiness of the data and its low-dimensionality ensures that the model performs well on the training set, but it does not generalize well: in Machine Learning it is called *overfitting*. A way out for this drawback is *regularization*[2], a technique to make the model simpler constraining its degrees of freedom. Obviously, *underfitting* is the opposite of overfitting, and it occurs when the model is too simple to learn the underlying structure of the data [35].

3.1.3 Choosing the Best Algorithm

As shown above, solving a learning problem corresponds to find the solution (or the solution set) of an optimization problem, like the one defined in (3.1). This means building a model over all the training data, capable to generalize to new cases. It is therefore necessary for our purpose defining a metric to monitor how well the model performs, and it is done splitting the data sample into two sets: the *training set* and the *test set*. The error rate on new cases is called *generalization error*, and we can get an estimate of this error by evaluating the trained model on the test set [35].

Let us reflect briefly on the meaning of a model: we can describe it as a simplified version of the observations. The simplifications are intended to remove noisy contributions to the features that are unlikely to generalize to new instances. Doing so expects to make *assumptions*, like the one of approximating y with a linear model: $f_\theta(x) = \theta_0 + \theta_1 x$.
The role played by the assumptions in the choice of the best algorithm is crucial. Good evidence of this can be found in a famous 1996 paper [37], where David Wolpert demonstrated that, for any two learning algorithms A and B there are just as many situations (appropriately weighted) in which algorithm A is superior to algorithm B as vice versa.

[2]The amount of regularization to apply during learning can be controlled by a *hyperparameter*, namely a parameter of the learning algorithm (not of the model).

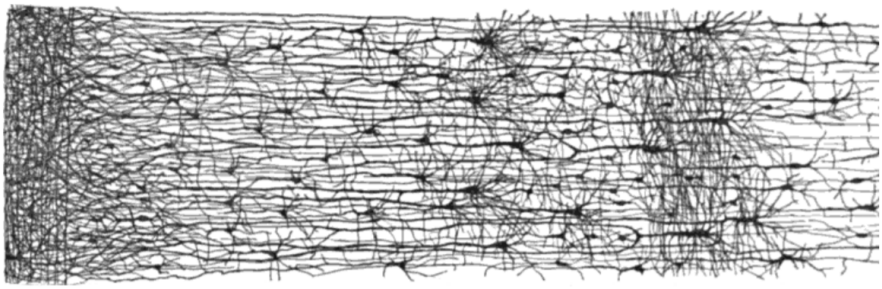


Figure 3.1: Drawing of a multiple layers in a biological neural network (human cortex) reproduced from `https://en.wikipedia.org/wiki/Cerebral_cortex`.

This is called the *No Free Lunch* (NFL) theorem, and it ensures that *a priori* better-working models simply do not exist: the best way to know for sure which model performs better is to evaluate them all [35].

3.2 Deep Learning

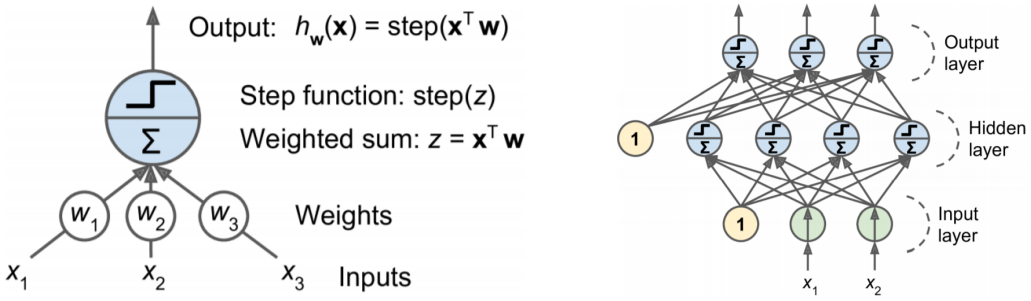
Deep Learning is a branch of Machine Learning that achieves power, scalability and flexibility by representing the world as a nested hierarchy of concepts. This corresponds to defining each concept starting with simpler ones, and to compute more abstract representations from less abstract ones. The study of models that involve a structured hierarchy of the learned concepts with respect to *standard* Machine Learning is what qualifies it as "deep" [34].

Deep Learning studies date back to 1940s, although one can think to tag it as a new technology. This common mistake is due to the fact that it was relatively unpopular for several years, and to the fact that it has gone through many different names as a result of a rebranding strategy.
Since its inception, researchers have seen in Deep Learning a reliable approach to *Artificial Intelligence*. Unsurprisingly, therefore, some of the earliest learning algorithm we recognize today were inspired by biological learning system, like our brains (Figure 3.1). As a proof of this, in 1943 the neurophysiologist Warren McCulloch and the mathematician Walter Pitts published a landmark paper [38], in which they proposed a simplified computational model based on animal neurons. This was the first *Artificial Neural Network* (ANN) architecture, and since then many other architectures have been invented [35].

3.2.1 Perceptron and Multilayer Perceptron

The *Perceptron* [39] is the simplest ANN architecture, where input-output values are numbers unlike the binary elements of McCulloch-Pitts proposal. Perceptron algorithm is in fact based on a slightly different artificial neuron called *threshold logic unit* (TLU) and represented schematically in Figure 3.2a. The TLU computes a weighted sum of its inputs ($z = w_1\, x_1 + w_2\, x_2 + \cdots + w_n\, x_n = \boldsymbol{x}^T\, \boldsymbol{w}$)[3], then applies a *step function* to that sum and outputs the result: $h_{\boldsymbol{w}}(\boldsymbol{x}) = \text{step(z)}$. The step function is added to emulate the

[3]Here the inputs $\{x_i\}_{i=1,\ldots,n}$ denote the features used to build a predictive model.



(a) *Threshold logic unit with three inputs, and explicit computation expressions.*

(b) *Multilayer Perceptron architecture with two inputs, one hidden layer, and three outputs.*

Figure 3.2: Schematic representation of Perceptron and Multilayer Perceptron, reproduced from Ref. [35].

behaviour of a biological neuron, that fires electrical impulses when it receives a sufficient amount of chemical signals. The most common step functions used for Perceptrons are Heaviside step function and sign function.
A Perceptron is simply composed of a single layer of TLUs, with each TLU connected to all the inputs. When all the neurons in a layer are connected to every neuron in the previous one, the layer is named a *fully connected layer*, or *dense layer*. The Perceptron inputs are fed to special neurons called *input neurons*, which output whatever input they are fed. Together they form the *input layer*. Generally, to this layer it is added an extra bias feature ($x_0 = 1$): latter is represented by a special type of neuron called *bias neuron*, which output 1 all the time. At this point, training a TLU corresponds to find the set of weights $\boldsymbol{w} = (w_0, w_1, \ldots, w_n)$ that optimizes the loss function [35].

The Perceptron algorithm can be used for simple linear binary classification, but it is incapable of solving some trivial problems, like the XOR classification one. To overcome these limitations, we can stack multiple Perceptrons obtaining a new ANN architecture called *Multilayer Perceptron* (MLP), or *feedforward neural network* (FNN).
The goal of a FNN is to approximate some function f^*: in a regression problem, for example, an input $\boldsymbol{x}$ is mapped to an output $\boldsymbol{y}$ through the relation $\boldsymbol{y} = f^*(\boldsymbol{x})$. Therefore, a FNN defines a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$, where the parameters $\boldsymbol{\theta}$ are computed by the optimization problem (3.1). These models are called "feedforward" because information flows through the function being evaluated from $\boldsymbol{x}$, the *input layer*, through the intermediate computations used to define f, and finally to the output $\boldsymbol{y}$, the *output layer*. On the other side, FNNs are typically represented by composing together many different function: for example, $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$ describes a map three-layer deep [34]. The internal layer, namely the ones different from input-output layers, are called *hidden layer*. A schematic representation of a FNN is given in Figure 3.2b.

Activation functions

In order for FNNs to solve non-linear problems, it is necessary to map the linear combination of each layer inputs in a new representation, as shown in Figure 3.2a for a single TLU by h function: that map is called *activation function*.

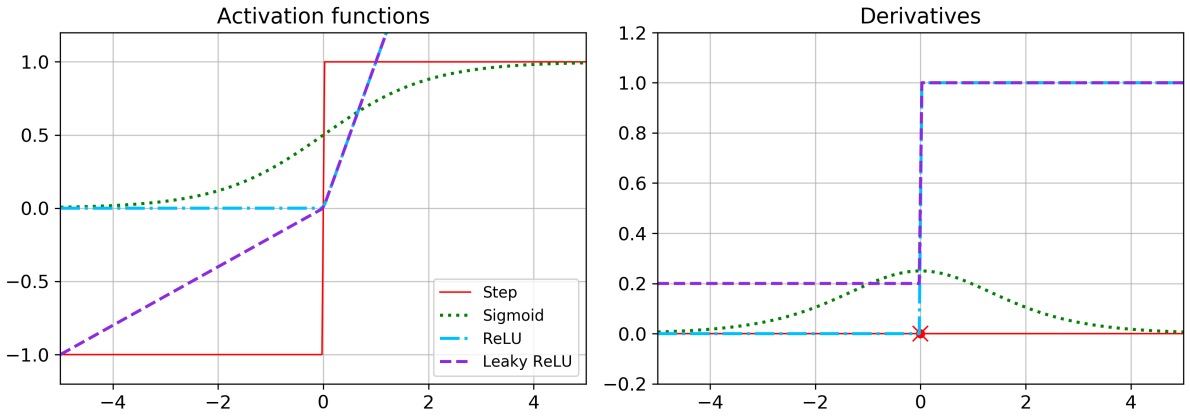


Figure 3.3: Activation functions and their derivatives. The step function shown is $\text{sign}(z)$, and the Leaky ReLU reported has $\alpha = 0.2$.

To ensure that FNN training succeeds, the step function of the original Perceptron must be replaced by a different one with non-zero gradient almost everywhere. This is a crucial characteristic to exploit gradient-based method in computing the optimization problem solution, as we will see in Section 3.2.2. Some of the most widely used activation functions follow:

- The *logistic function* (or *sigmoid*): $\sigma(z) = 1 \,/\, (1 + \exp(-z))$;
- The *Rectified Linear Unit function*: $f(z) = \max(0, z)$;
- The *Leaky Rectified Linear Unit function*: $f(z; \alpha) = \max(\alpha z, z)$.

All the activation functions are reported in Figure 3.3 together with their derivatives.

3.2.2 Training Deep Models

As mentioned above, training a neural network corresponds to find the parameters $\boldsymbol{\theta}$ that minimize a specific loss function. Considering a regression problem, for example, we want to solve an optimization problem like the one reported in (3.1), and to find a map $f(\boldsymbol{x}; \boldsymbol{\theta})$ capable of reliable predictions.
Let's try now exploiting this equation to compute indeed the best parameters. Obviously, we don't know the joint probability p for $(\boldsymbol{x}, \boldsymbol{y})$, so we can't compute directly the expected value $\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim p}[L(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{\theta}))]$. The only way is to use the instances from training set $\mathcal{T}$ to obtain an estimator for the average value of the loss function:

$$\hat{\boldsymbol{\theta}}^* = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{m} L(\boldsymbol{y}_i, f(\boldsymbol{x}_i; \boldsymbol{\theta})) = \arg\min_{\boldsymbol{\theta}\in\Theta} J(\boldsymbol{\theta}) \tag{3.2}$$

where Θ is the parameter space, $\boldsymbol{x}_i$ is the i-th input[4] of $\mathcal{T}$, $\boldsymbol{y}_i$ is the i-output of $\mathcal{T}$, and m is the total number of instances. The function $J(\boldsymbol{\theta})$ is called *empirical risk*, and it is the real subject of our optimization problem.

[4]Here the set $\{\boldsymbol{x}_i\}_{i=1,\dots,m}$ denotes all the inputs in training data, while with i-th index fixed we can access the n-features $\{x_{ij}\}_{j=1,\dots,n}$ (note the difference between bold elements and normal ones).

The set of optimization problems that can be solved analytically is relatively small and certainly does not include non-trivial neural networks. Approximate methods, such as Gradient Descent, are therefore preferred as baseline solution for training neural networks. *Gradient Descent* (GD) is a generic optimization algorithm capable of finding optimal solution to a wide range of problems. The algorithm consists of updating the parameters according to the local gradient of $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3.3}$$

where η weights gradient contributions to parameters update. It is an important training hyperparameter called *learning rate*.
The step (3.3), to be exact, is the parameters update of *Batch Gradient Descent* algorithm, where the gradient is computed over all the instances. It is a powerful tool in optimum search, but it suffers from slowness for large training sets. An extreme solution is *Stochastic Gradient Distance* (SGD), which updates the parameters $\boldsymbol{\theta}$ computing a single instance gradient randomly chosen:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}) \tag{3.4}$$

where $J_i(\boldsymbol{\theta}) = L(\boldsymbol{y}_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}))$. In-between algorithm also exists and it is named *Mini-batch Gradient Descent* that computes the gradients on small random sets of instances called *mini-batches*:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \sum_{i=1}^{\ell} L(\boldsymbol{y}_i, f(\boldsymbol{x}_i; \boldsymbol{\theta})) \tag{3.5}$$

where ℓ is the dimension of mini-batches ($\ell \leq m$).

Training a very large deep neural network may prove to be awfully slow, although one uses a Gradient Descent optimizer. A huge speed boost can come from the deployment of faster optimizer, such as Adam.
Adam, reported in Algorithm 2 [40], combines the ideas of two popular optimization algorithm: *momentum optimization* and *RMSProp*. The first algorithm introduces the *momentum vector* $\boldsymbol{m}$, used to speed up the learning process for directions already found in previous steps (line 6). The second algorithm, instead, keeps track of squared gradients from most recent iterations into the vector $\boldsymbol{s}$ (line 7), so that latter can be used to drop the learning rate faster for steep dimensions than for dimensions with gentler slopes. The parameters update step is reported at line 10.

Back-propagation algorithm

At this point is evident that training neural networks requires computing gradients, often of complicated functions. Therefore, we do not only have to worry about how to solve the optimization problem, but also about how to compute gradients. The *back-propagation* algorithm [41] plays a key role in this sense, allowing the information from the cost $J(\boldsymbol{\theta})$ to then flow backward through the network in order to compute the gradient [34].

Describing neural networks in terms of computational graphs, the back-propagation algorithm simply consists of performing a Jacobian-gradient product for each operation

Algorithm 2 Adam is a widely used algorithm for stochastic optimization. Good default settings for typical Machine Learning problems are $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$. The operations $\odot$ and $\oslash$ indicates the element-wise product and division, while β_1^t and β_2^t are the t-powers of β_1 and β_2 respectively.

Require: Learning rate η
Require: Exponential decay rates $\beta_1, \beta_2 \in [0, 1)$
Require: Empirical risk function $J(\boldsymbol{\theta})$
Require: Initial parameter vector $\boldsymbol{\theta}_0$
$\boldsymbol{m}_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$\boldsymbol{s}_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while $\boldsymbol{\theta}_t$ not converged **do**
  $t \leftarrow t + 1$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} - (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})$
  $\boldsymbol{s}_t \leftarrow \beta_2 \boldsymbol{s}_{t-1} + (1 - \beta_2) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}) \odot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m} / (1 - \beta_1^t)$
  $\hat{\boldsymbol{s}}_t \leftarrow \boldsymbol{s} / (1 - \beta_2^t)$
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} + \eta \, \hat{\boldsymbol{m}}_t \oslash \sqrt{\hat{\boldsymbol{s}}_t + \varepsilon}$
end while
return $\boldsymbol{\theta}_t$ (Resulting parameters)

in the graph, according to the chain rule of calculus. Suppose that $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{y} \in \mathbb{R}^m$, $g : \mathbb{R}^n \to \mathbb{R}^m$, and $f : \mathbb{R}^m \to \mathbb{R}$. If $\boldsymbol{y} = g(\boldsymbol{x})$ and $z = f(\boldsymbol{y})$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{3.6}$$

In vector notation, this may be equivalently written as

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \right)^T \nabla_{\boldsymbol{y}} z \tag{3.7}$$

where $(\partial \boldsymbol{y} / \partial \boldsymbol{x})$ is the $m \times n$ Jacobian matrix of g.
Using equation (3.6), therefore, the algorithm can measure the contribution to $J(\boldsymbol{\theta})$ of each connections from the previous below until the input layer.

3.2.3 High Performance Computing

Deep learning is based on the philosophy of *connectionism*. An individual biological neuron is not intelligent, just like a single feature in a machine learning model, but if we take a large population of these neurons or features acting together then they can exhibit intelligent behaviour. It is the *large* number of neurons to make a difference [34]. Because the size of neural networks is critical, Deep Learning requires high performance hardware and software infrastructure.

Traditionally, neural networks were trained using the CPU of a single machine. Today, this approach is considered insufficient, and use of GPU computing or the CPUs of many computers in a network together are generally preferred.

Graphics Processing Units (GPUs) are specialized hardware components that were originally developed for graphics applications. Spurred by video game industry, graphics processing hardware has been designed to have a high degree of parallelism and high memory bandwidth, at the cost of having a lower clock speed and less branching capability relative to traditional CPUs. These characteristics can be used effectively to train deep models that usually involve large and numerous buffers of parameters, activation values, and gradient values, each of which must be completely updated during every step of training. Moreover, since neural networks can be divided into multiple individual "neuron" that can be processed independently from the other neurons in the same layer, neural networks easily benefit from the parallelism of GPU computing [34].

The popularity of graphics cards for neural network training exploded after the advent of *general purpose* GPUs. However, writing high-performance GPU code remains a big stumbling, and one should prefer use a software library of high-performance operations, such as TensorFlow.
TensorFlow [42] is an end-to-end open source platform for Machine Learning, which provides high-performance operations whether the program runs on CPU or GPU.

3.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a powerful class of generative models based on simultaneous training of two neural networks [43]: a generative model G (*generator*) that produces synthetic data given some noise source, and a discriminative model D (*discriminator*) that distinguishes generator's outputs from true data. The goal is that D optimally discriminates on the origin of the two samples, and simultaneously the training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a *minimax two-player game*.

The generator captures the data distribution p_r starting from the *latent space* $\mathcal{Z}$, where elements $\boldsymbol{z}$ are sampled according to known distribution p_z. Then, the generative model $G(\boldsymbol{z}; \boldsymbol{\theta}_g)$ maps the latent space to the data space, inducing a distribution p_g to generator outputs. The discriminative model $D(\boldsymbol{x}; \boldsymbol{\theta}_d)$ outputs a single scalar, readable as the probability that $\boldsymbol{x}$ came from the data rather than G. Both the models can be represented by FNNs with parameters $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_d$. In this context, the optimization problem corresponds to training D to maximize the probability of correct labelling, and simultaneously training G to minimize $\log(1 - D(G(\boldsymbol{z})))$. Therefore, defining the $V(D, G)$ function as follows

$$V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_r}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z}[\log(1 - D(G(\boldsymbol{z})))] \tag{3.8}$$

the minimax game can be written in this form:

$$\min_G \max_D V(D, G) \tag{3.9}$$

In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $1/2$ everywhere. In the case where G and D are defined by FNNs, the entire system can be trained with back-propagation. Algorithm 3 describes the logic behind the minimax game, using a simple Mini-batch Gradient Descent as optimizer. It follows immediately that latter can be easily replaced

with a faster optimizer. More pedagogical explanation of the approach can be found in Figure 3.4.

Algorithm 3 Mini-batch Gradient Descent training of GANs.

Require: Learning rate η
Require: Number of discriminator updates k per generator iteration
Require: Mini-batch size ℓ
Require: Initial parameter vectors $\boldsymbol{\theta}_d^{(0)}$ and $\boldsymbol{\theta}_g^{(0)}$
$t \leftarrow 0$ (Initialize timestep)
while $\boldsymbol{\theta}_g$ not converged **do**
  $s \leftarrow 0$ (Initialize a second timestep every iteration)
  $\tilde{\boldsymbol{\theta}}_d^{(s)} \leftarrow \boldsymbol{\theta}_d^{(t)}$
  for k steps **do**
    $s \leftarrow s + 1$
    Sample ℓ elements from the latent space, $\{\boldsymbol{z}_i\}_{i=1,\dots,\ell}$
    Sample ℓ elements from the data space, $\{\boldsymbol{x}_i\}_{i=1,\dots,\ell}$
    $g_d^{(s-1)} \leftarrow \nabla_{\boldsymbol{\theta}_d} \frac{1}{\ell} \sum_{i=1}^{\ell} \left[\log D\left(\boldsymbol{x}_i; \tilde{\boldsymbol{\theta}}_d^{(s-1)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}_i; \boldsymbol{\theta}_g^{(t)}\right); \tilde{\boldsymbol{\theta}}_d^{(s-1)}\right)\right)\right]$
    $\tilde{\boldsymbol{\theta}}_d^{(s)} \leftarrow \tilde{\boldsymbol{\theta}}_d^{(s-1)} + \eta\, g_d^{(s-1)}$
  end for
  $t \leftarrow t + 1$
  Sample mini-batch of ℓ elements from the latent space, $\{\boldsymbol{z}_i\}_{i=1,\dots,\ell}$
  $g_g^{(t-1)} \leftarrow \nabla_{\boldsymbol{\theta}_g} \frac{1}{\ell} \sum_{i=1}^{\ell} \log\left(1 - D\left(G\left(\boldsymbol{z}_i; \boldsymbol{\theta}_g^{(t-1)}\right); \tilde{\boldsymbol{\theta}}_d^{(s)}\right)\right)$
  $\boldsymbol{\theta}_g^{(t)} \leftarrow \boldsymbol{\theta}_g^{(t-1)} - \eta\, g_g^{(t-1)}$
  $\boldsymbol{\theta}_d^{(t)} \leftarrow \tilde{\boldsymbol{\theta}}_d^{(s)}$
end while
return $\boldsymbol{\theta}_g^{(t)}$ (Resulting generator parameters)

It can be proved that following Algorithm 3 allows the generator to achieve the desired result, computing the minimax game and correctly mapping the latent space into the data one. Solving the optimization problem (3.9) with respect to D, in fact, one can find the *optimal* discriminator D^* for fixed generator G:

$$D_G^*(\boldsymbol{x}) = \max_D V(D, G) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{3.10}$$

If we substitute expression (3.10) in the loss function (3.8), it follows that

$$V(D^*, G) = -\log(4) + 2 \cdot JS\left(p_r \| p_g\right) \tag{3.11}$$

where $JS\left(p_r \| p_g\right)$ denotes the Jensen–Shannon divergence between the data distribution and the one induced from generating process. Therefore, solving the minimax game (3.9) corresponds to minimize the JS divergence:

$$\min_G \max_D V(D, G) = \min_G V(D^*, G) = \min_G JS\left(p_r \| p_g\right) \tag{3.12}$$

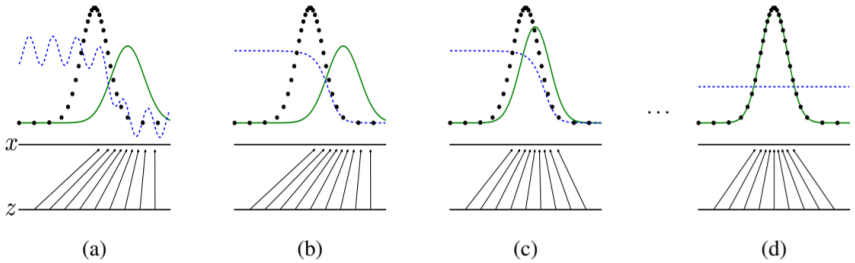


Figure 3.4: GANs are trained by simultaneously updating both of the discriminator D (dashed line in blue) and of the generator G. The goal of D is to distinguish between the data sample (dotted line in black) from the generator one (solid line in green). The lower horizontal lines and the upward arrows represent the latent space $\mathcal{Z}$ and the generator map $G : \boldsymbol{z} \to \boldsymbol{x}$. In (a) an adversarial pair near convergence is shown, where p_g is similar to p_r, and D is a partially accurate classifier. In the inner loop of the algorithm (k steps), D is trained to discriminate samples from data, converging to D^* (b). After an update to G, gradient of D has guided the map G to flow to regions that are more likely to be classified as data (c). After a sufficient number of iterations (d), if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_r$. Here the discriminator is unable to differentiate between the two distributions, and $D(\boldsymbol{x}) = 1/2$. Pedagogical example reproduced from Ref. [43].

Since the JS divergence between two distributions is always non-negative and zero only when they are equal, this proves that $V(D^*, G) = -\log(4)$ is the global minimum of $V(D, G)$ and that the only solution is $p_r = p_g$, the desired result [43].

GANs can be extended to a conditional model [44] if both the generator G and discriminator D are conditioned on some extra information $\boldsymbol{y}$, where latter can be any kind of auxiliary information. One can perform the conditioning by adding new nodes containing $\boldsymbol{y}$ to the input layer. In doing so, the loss function (3.8) can be rewritten as follows:

$$V(D, G) = \mathbb{E}_{\boldsymbol{x}\sim p_r}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))] \tag{3.13}$$

3.3.1 Training GANs and Wasserstein distance

GANs take a radically different approach compared to other generative models not requiring inference or explicit calculation of the data likelihood. Instead, two neural networks are used to solve a minimax game, that corresponds to minimize the JS divergence between true data distribution and the generated one [45].
Even though $JS(p_r \| p_g)$ reaches its minimum for $p_r = p_g$, in practice this result is often irksome because GANs suffer from many issues, particularly during training:

- The generator collapses producing only a single sample or a small family of very similar samples: it is known as *mode collapse* (or *mode dropping*);
- Generator and discriminator oscillate during training, rather than converge to a fixed point;
- If imbalance between the two agents occurs, the system simply stops learning.

These recurrent problems have forced researchers to employ many tricks during the training process [45], but despite this the set of hyperparameters for which latter successes is generally very small.

A complete analysis of the reasons behind the massive instability of GANs training is reported in a paper by Martin Arjovsky and Léon Bottou [46]. Here, they notice that generator updates get consistently worse as the discriminator gets better, although the optimization problem (3.12) follows from achieving the optimal discriminator D^*.
As mentioned above, with trained discriminator the loss function can be at most $-\log(4)+2 \cdot JS(p_r \| p_g)$. However, in practice, if we train D till convergence, its error goes to 0, showing that the JS divergence between p_r and p_g is maxed out[5]. The only way this can happen is if the supports of distributions are disjoint or lie in low dimensional manifolds. These conditions are not so rare. Considering the generator map $G : \mathcal{Z} \to \mathcal{X}$, for example, if the dimensionality of $\mathcal{Z}$ is less than the dimension of $\mathcal{X}$ (as is typically the case), then the support of p_g can only lie in a manifold with measure 0 for $\mathcal{X}$.
Under this assumption it can be proved a *perfect* discriminator $D^\star$ always exists, where $D^\star$ is a discriminator smooth and constant almost everywhere in the supports of p_r and p_g. The fact that the discriminator is constant in both manifolds ensures that we cannot really learn anything from gradient-based method. Therefore, from the existence of $D^\star$ results the *vanishing gradient* of the generator. If we consider a discriminator D approaching to $D^\star$, we get

$$\lim_{\|D-D^\star\|\to 0} \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{z}\sim p_z} \left[\log\left(1 - D\left(G_{\boldsymbol{\theta}}(\boldsymbol{z})\right)\right)\right] = 0 \tag{3.14}$$

This shows that as our discriminator gets better, the gradient of the generator vanishes. In other words, either our updates to the discriminator will be inaccurate, or they will vanish. Thus, expression (3.14) points out that using the loss function (3.8) or leaving the number of D-updates as hyperparameter can make GANs training extremely hard.

There is something we can do to break our gradient problem: adding continuous noise to the inputs both of discriminator and generator (for more clarification, see Ref. [46]). The insertion of noise allows to learn thanks to non-zero gradient of the generator. However, the optimization problem for G is now proportional to the gradient of *noisy* JS divergence:

$$\mathbb{E}_{\boldsymbol{z}\sim p_z,\varepsilon'} \left[\nabla_{\boldsymbol{\theta}} \log\left(1 - D^*_\varepsilon\left(G_{\boldsymbol{\theta}}(\boldsymbol{z}) + \varepsilon'\right)\right)\right] = 2 \cdot \nabla_{\boldsymbol{\theta}} JS\left(p_{r+\varepsilon} \| p_{g+\varepsilon'}\right) \tag{3.15}$$

where $\varepsilon, \varepsilon' \sim \mathcal{N}(0, \sigma^2\,\mathbb{I})$ and $p_{X+\varepsilon}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{y}\sim p_X}\left[p_\varepsilon(\boldsymbol{x}-\boldsymbol{y})\right]$.
This variant of JS divergence measures a similarity between the two noisy distribution and it is no more an intrinsic measure of p_r and p_g distance. All these drawbacks may be exceeded employing a different loss function, such as the Wasserstein distance.

Wasserstein GANs

Before defining new notion of distance, it is useful to open a parenthesis about the kind of loss function we are looking for. First of all, we want a distance measure between two probability distributions (one depending on parameters $\boldsymbol{\theta}$), namely $\rho(p_{\boldsymbol{\theta}}, p_r)$. The most fundamental difference between these distances is their impact on the *convergence*

[5]The global minimum of $V(D, G)$ is a negative value equal to $-\log(4)$.

of sequences of probability distributions. A sequence of distributions $\{p_t\}_{t\in\mathbb{N}}$ converges if and only if there is a distribution p_∞ such that $\rho(p_t, p_\infty)$ tends to zero, a property that depends heavily on the kind of distance ρ chosen [47].
Therefore, in optimizing $\boldsymbol{\theta}$ we would like to define a model inducing distribution $p_{\boldsymbol{\theta}}$ (the generator), that makes the mapping $\boldsymbol{\theta} \mapsto p_{\boldsymbol{\theta}}$ continuous. Here, with continuity we refers to the sequence of distributions property for which the convergence of $\boldsymbol{\theta}_t$ to $\boldsymbol{\theta}$ implies the convergence of $p_{\boldsymbol{\theta}_t}$ to $p_{\boldsymbol{\theta}}$. Concluding the digression, the *Wasserstein distance* is an interesting distance capable to be continuous even if the supports of distributions are disjoint or lie in low dimensional manifolds. The Wasserstein distance is defined as follows:

$$W(p_r, p_g) = \inf_{\gamma\in\Pi(p_r,p_g)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\gamma} \left[\| \boldsymbol{x} - \boldsymbol{y} \| \right] \tag{3.16}$$

where $\Pi(p_r, p_g)$ denotes the set of all joint distributions $\gamma(\boldsymbol{x}, \boldsymbol{y})$ whose marginals are respectively p_r and p_g.
Even if the Wasserstein distance is much weaker than the JS divergence (for more clarification, see Ref. [47]), $W(p_r, p_g)$ is a continuous loss function on $\boldsymbol{\theta}$ under mild assumptions:

1. If $G_{\boldsymbol{\theta}}$ is continuous in $\boldsymbol{\theta}$, so is $W(p_r, p_{\boldsymbol{\theta}})$;
2. If $G_{\boldsymbol{\theta}}$ is locally Lipschitz and satisfies regularity assumption 1, then $W(p_r, p_{\boldsymbol{\theta}})$ is continuous everywhere and differentiable almost everywhere;

Statements 1-2 are false for the Jensen-Shannon divergence $JS(p_r, p_{\boldsymbol{\theta}})$.
The fact that JS divergence provides non-sensitive loss function for supports lying in low dimensional manifolds highlights the importance of using a different distance measure, one capable to learn even in these circumstances.

Training GANs with Wasserstein distance allows to obtain systems more stable and resistant to mode collapse. They are called Wasserstein GANs, or simply WGANs. However, the infimum in (3.16) is computationally intractable, and we need a different formulation. Fortunately, the Kantorovich-Rubinstein duality tells us that

$$W(p_r, p_g) = \sup_{\| f \|_L \leq 1} \left(\mathbb{E}_{\boldsymbol{x}\sim p_r} [f(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim p_g} [f(\boldsymbol{x})] \right) \tag{3.17}$$

where the supremum is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$. Note that if we replace $\| f \|_L \leq 1$ for $\| f \|_L \leq K$ (consider K-Lipschitz for some constant K), then we end up with $K \cdot W(p_r, p_g)$.
We can now rewrite the original minimax game as follows:

$$\min_G \max_{C\in\mathcal{C}} W(C, G) = \min_G \max_{C\in\mathcal{C}} \left(\mathbb{E}_{\boldsymbol{x}\sim p_r} [C_{\boldsymbol{w}}(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p_z} [C_{\boldsymbol{w}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}))] \right) \tag{3.18}$$

where C (*critic*) replaces the discriminator and $\mathcal{C}$ is the set of 1-Lipschitz functions. Again, both C and G can be represented by FNNs with parameters $\boldsymbol{w}$ and $\boldsymbol{\theta}$. Training WGANs can still be described by Algorithm 3 adopting the appropriate changes on loss function. The only significant difference is that we need to ensure that $C \in \mathcal{C}$. This can

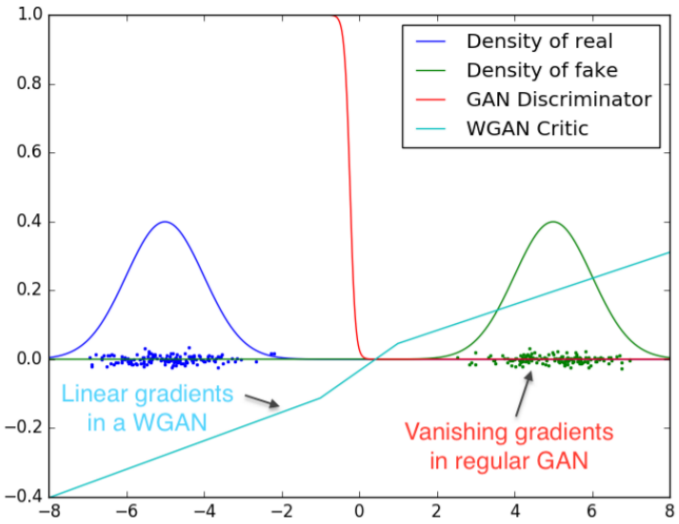


Figure 3.5: Optimal discriminator and critic when learning to distinguish two Gaussians. Here it's clearly shown that the discriminator of original GAN saturates and results in vanishing gradients. On the other hand, WGAN critic is able to provide very clean gradients on all parts of the space. Reproduced from Ref. [47].

be done adding a regularization term to the loss function $W(C, G)$ (for more clarification, see Ref. [48]):

$$W(C,G) = \underbrace{\mathbb{E}_{\boldsymbol{x}_r \sim p_r}[C(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}_g \sim p_g}[C(\boldsymbol{x}_g)]}_{\text{Original critic loss}} + \underbrace{\lambda \cdot \mathbb{E}_{\hat{\boldsymbol{x}} \sim p_{\hat{\boldsymbol{x}}}}[\,(\|\nabla_{\hat{\boldsymbol{x}}} C(\hat{\boldsymbol{x}})\|_2 - 1)^2\,]}_{\text{Gradient penalty}} \tag{3.19}$$

where, for simplicity, $\boldsymbol{x}_g$ denotes $G(\boldsymbol{z})$ with $\boldsymbol{z} \sim p_z$, and $\hat{\boldsymbol{x}}$ results from a linear interpolation between real and generated samples: $\hat{\boldsymbol{x}} = \varepsilon \boldsymbol{x}_r + (1-\varepsilon)\boldsymbol{x}_g$ with ε is uniformly distributed. The introduction of this term is called *gradient penalty* and defines WGAN-GP systems.

Lastly, returning to Wasserstein distance properties, the fact that $W(C, G)$ is continuous everywhere and differentiable almost everywhere means that we can train the critic function till optimality without any problems because it cannot saturate. In Figure 3.5 we represent this behaviour comparing it with the one of original discriminator that, as expected, provides no reliable gradient information.

3.3.2 Biased gradients and Cramér distance

The use of Wasserstein distance with the regularization term reported in (3.19) improves significantly the training process, allowing to increase its stability and to reproduce faithful sample. The great performances of WGAN systems are due in part to a critic function capable of driving effectively the training process, and in part to the fact that the Wasserstein distance is an *ideal divergence*[6].

[6] A divergence that satisfies *scale sensitive* and *sum invariant* conditions is called ideal. Deep studies of these properties are beyond the scope of this thesis (for more details, see Ref. [49]).

Another important requirement for distance measure used in Machine Learning algorithms is to satisfy the *unbiased sample gradients* condition. This property ensure that[7], given a set of random variables $\{x_i\}_{i=1,\dots,\ell}$ distributed according to q and given the empirical distribution $\hat{q}_\ell = \frac{1}{\ell}\sum_{i=1}^{\ell}\delta_{x_i}$ (note that $\hat{q}_\ell$ is a random quantity), the distance from the empirical distribution converges to the distance from real one. In other word, the property we are looking for is that

$$\mathbb{E}_{x\sim q}[\nabla_\theta \rho(\hat{q}_\ell, q_\theta)] = \nabla_\theta \rho(q, q_\theta) \tag{3.20}$$

where ρ is a general distance, and q_θ denotes the usual parameterized distribution. Better said, having unbiased sample gradients allows to rewrite relation (3.1) as (3.2) saying to solve the same optimization problem. Unfortunately, Wasserstein distance does not have unbiased sample gradients.
The most dangerous consequence of not satisfying condition (3.20) is that the minimum of the expected sample loss $\hat{\theta}^*$ is in general *different* from the minimum of the real Wasserstein loss θ^*:

$$\hat{\theta}^* = \arg\min_{\theta\in\Theta} \mathbb{E}[W(\hat{q}_\ell, q_\theta)] \neq \arg\min_{\theta\in\Theta} \mathbb{E}[W(q, q_\theta)] = \theta^* \tag{3.21}$$

where not equal relation is to be considered in general. This affects heavily adversarial models, preventing the generator from widespread generalization if one uses SGD-like methods for optimization (Figure 3.6). Again, the solution lies in changing the loss function.

Cramér GANs

An alternative to the Wasserstein distance is the *Cramér distance* that, used as loss function, shows the same appealing properties of $W(p, q)$, but also providing us with

[7]The absence of bold symbols in the following steps is not a typo: for simplicity, we are considering the unidimensional case.

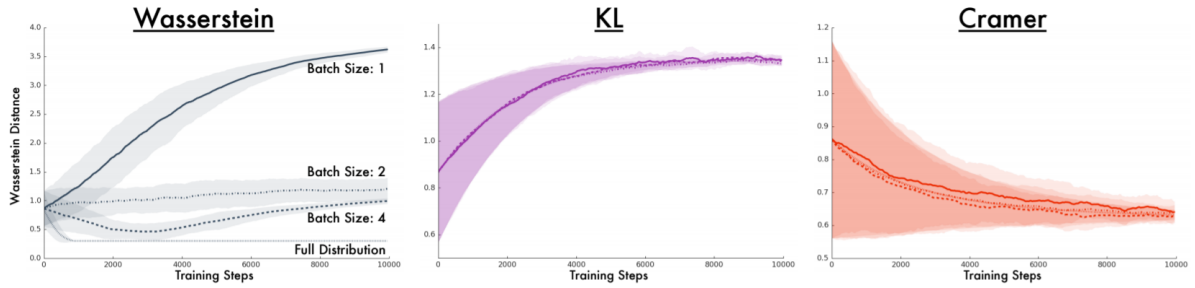


Figure 3.6: Learning curves of three different adversarial models: using Wasserstein distance, Kullback–Leibler divergence and Cramér distance respectively. For the last two, having unbiased sample gradients, the batch size does not prevent from reaching the minimum. Instead, the convergence of Wasserstein distance is heavily conditioned by hyperparameter setting. Plots reported from Ref. [49].

unbiased sample gradients. The Cramér distance is defined as follows:

$$l_n(p,q) = \sup_{f \in \mathcal{F}_m} | \, \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)] \, | \tag{3.22}$$

where $\mathcal{F}_m = \left\{ f : f \text{ is absolutely continuous, } \left\| \frac{df}{dx} \right\|_m \leq 1 \right\}$ and m is the conjugate exponent of n, namely $n^{-1} + m^{-1} = 1$. Noticing that l_n and Wasserstein distance are identical at $n = 1$, it is not surprising that the Cramér distance is an ideal divergence for $1 \leq n \leq \infty$. Requiring also unbiased sample gradients, the range comes down to the only $n = 2$. Choosing the definition (3.22) with $n = 2$ therefore, the Cramér distance gains all the positives of $W(p, q)$ with the addition of unbiased sample gradients.

The *energy distance* $\mathcal{E}$ is a natural extension of the Cramér distance to the multivariate case. Let p and q be probability distributions over $\mathbb{R}^d$ and let $\boldsymbol{x}, \boldsymbol{x}'$ and $\boldsymbol{y}, \boldsymbol{y}'$ be independent random variables distributed according to p and q respectively. For

$$f^*(\boldsymbol{a}) = \mathbb{E}_{\boldsymbol{y}'\sim q}\left[\, \|\boldsymbol{a} - \boldsymbol{y}'\|_2 \, \right] - \mathbb{E}_{\boldsymbol{x}'\sim p}\left[\, \|\boldsymbol{a} - \boldsymbol{x}'\|_2 \, \right] \tag{3.23}$$

the energy distance can be written as

$$\mathcal{E}(p,q) = \mathbb{E}_{\boldsymbol{x}\sim p}[f^*(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{y}\sim q}[f^*(\boldsymbol{y})] \tag{3.24}$$

Starting from expression (3.24), we can define a minimax two-player game to train generative and discriminative models represented by FNNs. Just like for Wasserstein case, the discriminator is renamed critic and it still has the task of distinguishing data origin, but in a transformed space defined by $h : \mathbb{R}^d \to \mathbb{R}^k$. Rewriting the critic function (3.23) as follows

$$f_h^*(\boldsymbol{a}) = \mathbb{E}_{\boldsymbol{x}'_g\sim p_g}\left[\, \|h(\boldsymbol{a}) - h(\boldsymbol{x}'_g)\|_2 \, \right] - \mathbb{E}_{\boldsymbol{x}'_r\sim p_r}\left[\, \|h(\boldsymbol{a}) - h(\boldsymbol{x}'_r)\|_2 \, \right] \tag{3.25}$$

where $\boldsymbol{x}_g$ still denotes $G(\boldsymbol{z})$ with $z \sim p_z$, the minimax game corresponds to

$$\min_G \max_h \mathcal{L}_g = \min_G \max_h \left(\mathbb{E}_{\boldsymbol{x}_r\sim p_r}[f_h^*(\boldsymbol{x}_r)] - \mathbb{E}_{\boldsymbol{x}_g\sim p_g}[f_h^*(\boldsymbol{x}_g)] \right) \tag{3.26}$$

with $f^*(\boldsymbol{a})$ belonging to $\mathcal{F}_2 = \{f : f \text{ is absolutely continuous, } \|\nabla_{\boldsymbol{a}} f^*\|_2 \leq 1\}$. To ensure this condition, it is a good idea adding the gradient penalty term to $\mathcal{L}_g$:

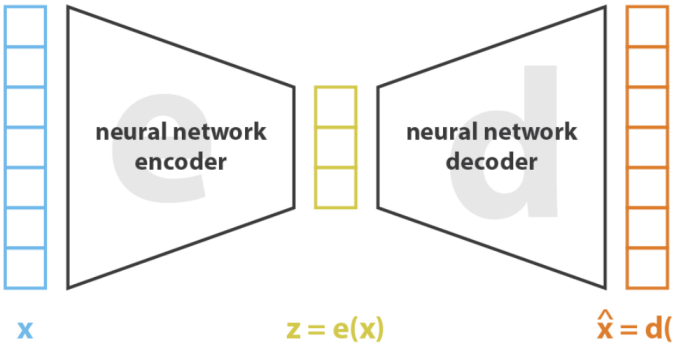
$$\mathcal{L}_{\text{critic}} = -\mathcal{L}_g + \lambda \cdot \mathbb{E}_{\hat{\boldsymbol{x}}\sim p_{\hat{\boldsymbol{x}}}}\left[\, (\|\nabla_{\hat{\boldsymbol{x}}} f_h^*(\hat{\boldsymbol{x}})\|_2 - 1)^2 \, \right] \tag{3.27}$$

where $\hat{\boldsymbol{x}} = \varepsilon \boldsymbol{x}_r + (1 - \varepsilon)\boldsymbol{x}_g$ with $\varepsilon \sim \mathcal{U}(0, 1)$.
Using $\mathcal{L}_g$ and $\mathcal{L}_{\text{critic}}$, we are able to build an adversarial model called Cramér GAN. It is important to note that the minimax game (3.26) is based on the loss maximization with respect to the map h, and not with respect to the critic itself[8] (such as for Wasserstein case). Therefore, the discriminative neural network is responsible for mapping its inputs into a space in which the critic can infer their origin. On the other hand, the generative neural network is driven by critic mistakes, succeeding into widespread generalization.
Algorithm 4 describes the logic behind Cramér GAN systems, using a simple Mini-batch Gradient Descent not to make reading even heavier. This strategy ensures incredible stability during the training process, and provides increased diversity in the generated samples [49].

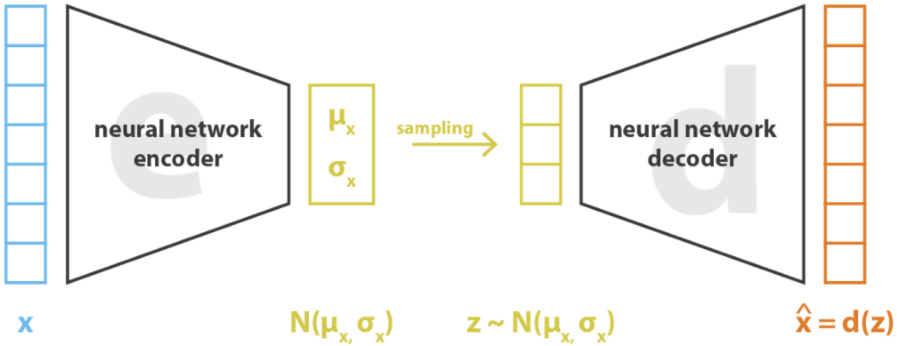
[8]Not touching the critic function, solving the minimax game, is what guarantees that the unbiased sample gradients condition remains fulfilled.

Algorithm 4 Cramér GAN trained by Mini-batch Gradient Descent. The critic function is replaced with an approximation, according to the hypothesis $h(\boldsymbol{x}_r) \approx 0$ for $\boldsymbol{x}_r \sim p_r$.

Require: Learning rate η
Require: Number of critic steps n_{critic} per generator iteration
Require: Mini-batch size ℓ
Require: Gradient penalty coefficient λ
Require: Initial parameter vectors $\boldsymbol{w}^{(0)}$ and $\boldsymbol{\theta}^{(0)}$
$t \leftarrow 0$ (Initialize timestep)
while $\boldsymbol{\theta}$ not converged **do**
  $s \leftarrow 0$ (Initialize a second timestep every iteration)
  $\tilde{\boldsymbol{w}}^{(s)} \leftarrow \boldsymbol{w}^{(t)}$
  for n_{critic} steps **do**
    $s \leftarrow s + 1$
    for $i = 1, \ldots, \ell$ **do**
      Sample $\boldsymbol{x}_r$ from the data space
      Sample $\boldsymbol{z}, \boldsymbol{z}'$ from the latent space
      Sample a random number ε from $\mathcal{U}(0, 1)$
      $\boldsymbol{x}_g \leftarrow G\left(\boldsymbol{z}; \boldsymbol{\theta}^{(t)}\right)$
      $\boldsymbol{x}'_g \leftarrow G\left(\boldsymbol{z}'; \boldsymbol{\theta}^{(t)}\right)$
      $\hat{\boldsymbol{x}} \leftarrow \varepsilon \boldsymbol{x}_r + (1 - \varepsilon)\boldsymbol{x}_g$
      $f_r^* \leftarrow \left\| h\left(\boldsymbol{x}_r; \tilde{\boldsymbol{w}}^{(s-1)}\right) - h\left(\boldsymbol{x}'_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2 - \left\| h\left(\boldsymbol{x}_r; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2$
      $f_g^* \leftarrow \left\| h\left(\boldsymbol{x}_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) - h\left(\boldsymbol{x}'_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2 - \left\| h\left(\boldsymbol{x}_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2$
      $\mathcal{L}_g \leftarrow f_r^* - f_g^*$
      $f_i^* \leftarrow \left\| h\left(\hat{\boldsymbol{x}}; \tilde{\boldsymbol{w}}^{(s-1)}\right) - h\left(\boldsymbol{x}'_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2 - \left\| h\left(\hat{\boldsymbol{x}}; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2$
      $\mathcal{L}_{\text{critic}}^{(i)} \leftarrow -\mathcal{L}_g + \lambda \cdot (\|\nabla_{\hat{\boldsymbol{x}}} f_i^*\|_2 - 1)^2$
    end for
    $\tilde{\boldsymbol{w}}^{(s)} \leftarrow \tilde{\boldsymbol{w}}^{(s-1)} + \eta \cdot \nabla_{\boldsymbol{w}} \left(\frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}_{\text{critic}}^{(i)} \right)$
  end for
  $t \leftarrow t + 1$
  for $i = 1, \ldots, \ell$ **do**
    Sample $\boldsymbol{x}_r$ from the data space
    Sample $\boldsymbol{z}, \boldsymbol{z}'$ from the latent space
    Sample a random number ε from $\mathcal{U}(0, 1)$
    $\boldsymbol{x}_g \leftarrow G\left(\boldsymbol{z}; \boldsymbol{\theta}^{(t-1)}\right)$
    $\boldsymbol{x}'_g \leftarrow G\left(\boldsymbol{z}'; \boldsymbol{\theta}^{(t-1)}\right)$
    $\hat{\boldsymbol{x}} \leftarrow \varepsilon \boldsymbol{x}_r + (1 - \varepsilon)\boldsymbol{x}_g$
    $f_r^* \leftarrow \left\| h\left(\boldsymbol{x}_r; \tilde{\boldsymbol{w}}^{(s)}\right) - h\left(\boldsymbol{x}'_g; \tilde{\boldsymbol{w}}^{(s)}\right) \right\|_2 - \left\| h\left(\boldsymbol{x}_r; \tilde{\boldsymbol{w}}^{(s)}\right) \right\|_2$
    $f_g^* \leftarrow \left\| h\left(\boldsymbol{x}_g; \tilde{\boldsymbol{w}}^{(s)}\right) - h\left(\boldsymbol{x}'_g; \tilde{\boldsymbol{w}}^{(s)}\right) \right\|_2 - \left\| h\left(\boldsymbol{x}_g; \tilde{\boldsymbol{w}}^{(s)}\right) \right\|_2$
    $\mathcal{L}_g^{(i)} \leftarrow f_r^* - f_g^*$
  end for
  $\boldsymbol{w}^{(t)} \leftarrow \tilde{\boldsymbol{w}}^{(s)}$
  $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta \cdot \nabla_{\boldsymbol{\theta}} \left(\frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}_g^{(i)} \right)$
end while
return $\boldsymbol{\theta}^{(t)}$ (Resulting generator parameters)



(a) *Autoencoders scheme.*



(b) *Variational Autoencoders scheme.*

Figure 3.7: Reported from `https://bit.ly/2YXaNku`.

3.4 Variational Autoencoders

In addition to the GAN systems, another deep generative method spreading in recent years is based on *Variational Autoencoders* (VAEs). VAEs represent a generalization of dimensionality reduction methods and, in particular, they improve *autoencoders* allowing latters to generate new data [50, 51].

In Machine Learning, *dimensionality reduction* is the process with which is possible to reduce the number of features that describe some data. This reduction is performed selecting a sub-set of existing features, or extracting a new set of features combining the old ones. A similar approach shows to be necessary in all that situations requiring low dimensional data, such as the data storing. To pursue this, one can define two maps, called *encoder* and *decoder*, whose goal is to remap the input data into itself passing through a space with lower dimensionality, named *latent space*. Suppose that $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{z} \in \mathcal{Z}$ with $\dim(\mathcal{X}) > \dim(\mathcal{Z})$, then the dimensionality reduction problem can be described by the following map sequence

$$\mathcal{X} \xrightarrow{e(\boldsymbol{x})} \mathcal{Z} \xrightarrow{d(\boldsymbol{z})} \mathcal{X} \tag{3.28}$$

where e and d denote the encoder and decoder functions respectively. When the two functions (e, d) are represented by FNNs, we talk about *autoencoders*. Figure 3.7a shows their schematically representation.

The method just described does not allow to generate a new sample distributed ac-

cording to the input space. Therefore, instead of encoding an input as a single point, VAE encodes it as a distribution over the latent space. To generate a sample from the model, the VAE system starts with the extraction of a $\boldsymbol{z}$ sample according to $p_{\text{model}}(\boldsymbol{z})$, usually set to $\mathcal{N}(0, \mathbb{I})$. The sample is then run through a generator network $g(z)$. This induces a distribution $p_{\text{model}}(\boldsymbol{x}; g(\boldsymbol{z})) = p_{\text{model}}(\boldsymbol{x}|\boldsymbol{z})$ from which it is finally possible to sample $\boldsymbol{x}$. In this sense, $g(\boldsymbol{x})$ defines the *decoder network*. During training, however, the approximate inference network (or *encoder network*) $q(\boldsymbol{z}|\boldsymbol{x})$ is used to extract $\boldsymbol{z}$, and is typically define as a Gaussian distribution with mean and covariance functions of $\boldsymbol{x}$: $\mu(\boldsymbol{x})$ and $\sigma(\boldsymbol{x})$ [34]. Training VAEs therefore requires to solve an optimization problem involving three neural networks: $g(\boldsymbol{x})$, $\mu(\boldsymbol{x})$ and $\sigma(\boldsymbol{x})$. A schematic representation of these systems is reported in Figure 3.7b.

This description is not exhaustive to cover entirely the theory behind VAEs, but is aimed to give a brief overview about a second powerful method to generate good-looking synthetic sample. As far as I know, VAE-based models for High Energy Physics application does not exist, then I will not further explore the topic in this thesis work.

3.5 GANs Usage in High Energy Physics

Throughout recent years, GAN systems have demonstrated to be a backbone for Computer Vision, proving ones of the most widely used *generative image models*. This choice results from their capacity in reproducing highly faithful and diverse images thanks to models learned directly from data.
The extreme scalability of Deep Learning based models does not go unnoticed by the High Energy Physics world, which has soon exploited their results for particles study. In this respect, deep generative models offer a perfect solution for *ultra-fast simulation* to replace the CPU consuming Monte Carlo based one. Starting from GANs natural propensity for image generation, we can immediately think of using them to parameterize calorimeter response. The LAGAN [52] and CaloGAN [53] were among the first examples interpreting calorimeter simulation. Since then, many other examples were followed [54–57]. More recently, adversarial models have started to be used for data analysis [58] and for particle identification [59].

Chapter 4

Generative Adversarial Networks for Particle Identification

4.1 Introduction

The increasing of luminosity scheduled for LHCb Upgrade I and Upgrade II has led the Collaboration to develop and implement new simulation methods in order to meet the growing needs of physics analyses. As shown in Chapter 2, the usage of *full* simulation alone is not able to produce large simulated samples in reasonable time because of the high CPU-cost of GEANT4-based methods. To this end, *ultra-fast* options have been introduced, allowing to simulate high-level reconstructed variables without describing detector raw responses, and instead exploiting the outputs of parametric or non-parametric functions. Among non-parametric solutions, methods based on Machine Learning algorithms and, in particular, based on Generative Adversarial Networks are being studied in recent years.

Part of this thesis work has consisted in implementing models based on GANs to reproduce the high-level variables used by LHCb for the particle identification. This idea is to extend the work done with similar approaches to reproduce the likelihood response of the RICH detectors [59] and of the muon system [60].
As described in Section 1.2.2, RICH detectors make use of the Cherenkov effect to identify particles. Computing the *full* simulation of this process is particularly intensive, since it requires a precise modelling of the low-energy secondary electrons, as well as of the light propagation, diffraction and absorption effects. On the other hand, the solution proposed in Ref. [59] allows to bypass an accurate RICH simulation, and provides directly the PID information. This is done training Cramér GAN systems (see Section 3.3.2) to reproduce the likelihood values for different particle species starting from track kinematic parameters (momentum p and pseudorapidity η) and detector occupancy (`nTracks`). This technique allows to obtain high-fidelity distributions for `RichDLL` variables, as shown in Figure 4.1 [59].
Following the same logic, the interesting thesis work presented in Ref. [60] demonstrates that using GANs it is also possible to reproduce good-looking distributions for the likelihood functions outputting from the muon system. Here, the algorithm is more similar to the one proposed by Goodfellow [43], but the original loss function is replaced with the *binary cross-entropy*, while generator and discriminator are injected with noise to stabilize

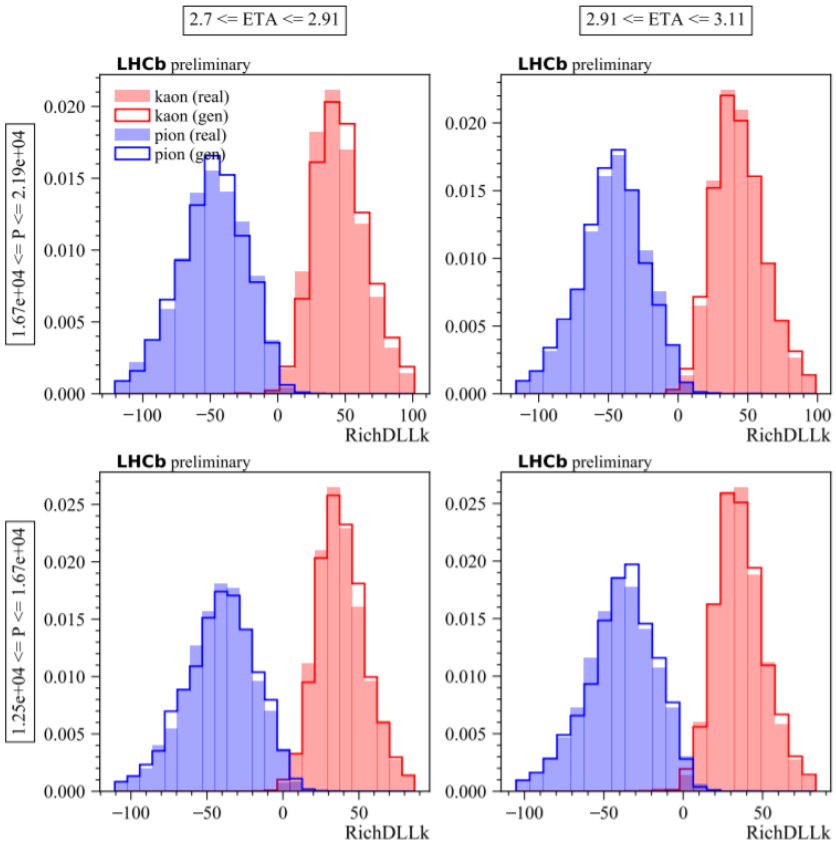


Figure 4.1: Weighted real data and generated distributions of `RichDLLk` for kaon and pion track candidates in bins of pseudorapidity (ETA) and momentum (P) in a well-populated phase space region. Histograms reproduced from Ref. [59].

the training process (as described briefly in Section 3.3.1). Also this technique allows to reproduce successfully the likelihood values distributions of the muon system (`MuonMuLL` and `MuonBkgLL`) for different particle species.

This thesis work aims to combine and extend these two GAN models in order to provide also the combined PID variables, namely CombDLL and `ProbNN`. The starting point will be the same, exploiting only track kinematic parameters and detector occupancy to generalize all the PID variables, while the latent space is responsible for mimicking the missing information.
Detailed discussion of the GAN model adopted is reported in Section 4.2, where a scoring method to assess the goodness of the generator output is also defined. This method is used to tune the hyperparameters, widely discussed in Section 4.3. Finally, Section 4.4 reports the PID variables distributions as obtained from the model described.

4.2 Models Definition

The model proposed follows the path traced by Ref. [59], and is based on the energy distance (3.24) to drive the training of two neural networks through the minimax game: namely it is a Cramér GAN system [49]. Since the response of the PID detectors to a

traversing particle depends on the kinematics of the particle (momentum p and pseudorapidity η) and on the occupancy of the detector (`nTracks`), we expect that also the likelihood functions resulting from the PID system depends on the same parameters. It is therefore necessary to use a *conditional version* of the Cramér GAN introduced in Section 3.3.2.

Suppose that the training set can be expressed as $\mathcal{T} = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ denotes the *conditional space* and $\mathcal{Y}$ represents the space from which we want to extract the elements distribution, namely the *reference space*. Let $\boldsymbol{x}_r, \boldsymbol{x}_g, \boldsymbol{x}'_g \in \mathcal{X}$, $\boldsymbol{y}_r \in \mathcal{Y}$ and $\boldsymbol{z}, \boldsymbol{z}' \in \mathcal{Z}$ (latent space) be independent random variables distributed according to q, p_r and p_z respectively. Let $\boldsymbol{y}_g, \boldsymbol{y}'_g$ be outputs of the generator network G for $\boldsymbol{z}, \boldsymbol{z}'$, which induces a distribution p_g. Then, the energy distance (3.24) can be rewritten as

$$\mathcal{E}(p_r, p_g) = \mathcal{L}_g = \mathbb{E}_{\substack{\boldsymbol{x}_r \sim q \\ \boldsymbol{y}_r \sim p_r}} [f_h^*(\boldsymbol{y}_r|\boldsymbol{x}_r)] - \mathbb{E}_{\substack{\boldsymbol{x}_g \sim q \\ \boldsymbol{y}_g \sim p_g}} [f_h^*(\boldsymbol{y}_g|\boldsymbol{x}_g)] \tag{4.1}$$

where the critic function f_h^* (3.25), reviving the approximate form reported in Algorithm 4, now becomes

$$f_h^*(\boldsymbol{a}|\boldsymbol{b}) = \mathbb{E}_{\substack{\boldsymbol{x}'_g \sim q \\ \boldsymbol{y}'_g \sim p_g}} \left[\|h(\boldsymbol{a}|\boldsymbol{b}) - h(\boldsymbol{y}'_g|\boldsymbol{x}'_g)\|_2 \right] - \|h(\boldsymbol{a}|\boldsymbol{b})\|_2 \tag{4.2}$$

The conditional form of the energy distance (4.1) redefines the minimax game (3.26) and can be used to train the generator G, whose updates are based on $\mathcal{L}_g$. On the other hand, to train the discriminator h it is necessary to add to $\mathcal{L}_g$ the gradient penalty term, as done in (3.27), after having corrected the latter into its conditional form.
Looking at the two relations above, it follows immediately that to perform the conditioning it is sufficient passing the $\boldsymbol{x}$ elements as inputs of the discriminator together with the objects whose origin it has to infer. Again, the goal of h is to map the concatenated feature sets $(\boldsymbol{y}_r|\boldsymbol{x}_r)$ and $(\boldsymbol{y}_g|\boldsymbol{x}_g)$ into a space in which the data sample can be separated from the generated sample. Simultaneously, the generator updates its parameters to get closer the real distribution and hinder discriminator intent. The training process of conditional Cramér GANs is still described by Algorithm 4, after having made the appropriate adjustments. Finally, the scheme of a generic conditional GAN system is reported in Figure 4.2.

4.2.1 Input Sample

Track kinematics parameters and detector occupancy certainly affect the responses of the PID system, but they are not the only one in doing so. On the other hand, the purpose of using Generative Models for modelling the detector response is precisely to take into account, as statistical fluctuations, the contributions of the many variables out of the experimentalist control that may affect the detector response. In this sense, it is important to identify the variables describing the input particle and the event as input, and let all of the other to be either an output of the GAN-based parameterization or part of the noise representing the limitations to the experimental reproducibility (such as uncertainties or mutable conditions).

Conceptually, one can imagine a single Generative Model producing all the outputs given a set of inputs. However, a larger number of variables involved in the model results into a larger number of possible correlations that must be reproduced by the model and

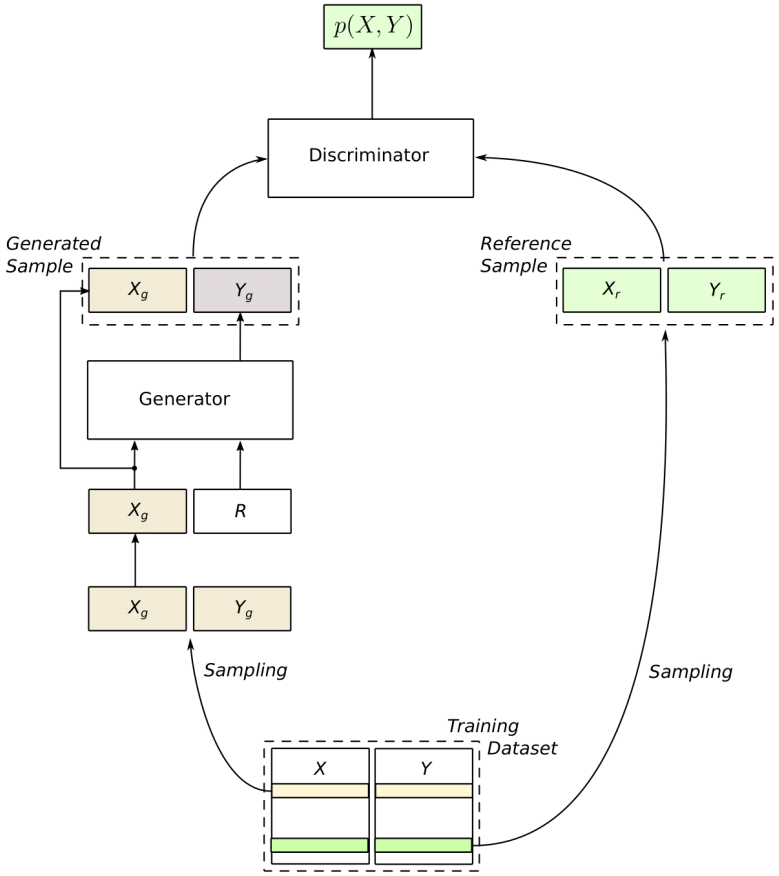


Figure 4.2: Conditional GAN scheme. Here, the discriminator outputs a probability like in the original Goodfellow's proposal. Anyway, the representation remains valid also in case of conditional Cramér GAN, substituting $p(X, Y)$ with the map $h(X, Y)$. Finally, notice that latent space elements are denoted with R instead of Z.

validated at the end of the training procedure. When possible, we found preferable to split the Generative Model into a number of building blocks connecting input and output variables on the basis of physics causality principles. In practice, we preferred to build one model for each subsystem, choosing the proper set of parameters from the conditional space $\mathcal{X}$ and from the reference space $\mathcal{Y}$ for each one. The PID system has been divided into four subsystems and as many models: the RICH and muon systems reimplementing the neural networks discussed in Ref. [59] and Ref. [60], respectively, to which two *Global PID* models are enqueued. The new Generative Models predicting the combined PID variables are differentiated between the case of muon candidates (called `GlobalMuonId`) and the case of all the other long-lived particle species (called `GlobalPID`). Table 4.1 summarizes the input dependencies of each listed model.
An attentive reader will have noticed the total absence of the calorimeter subsystem although both the CombDLL and `ProbNN` variables depend on it. For the `ProbNN` outputs the lack of information is even greater, as clearly shown by the different entries in Table 1.3 and Table 4.1. The strategy adopted here is to avoid using any high-level reproduction of the calorimeters response to obtain all the PID variable distributions. Instead, assuming that the kinematics information and the event multiplicity are able to parameterize effec-

Model	Conditional sample	Reference sample
Rich	Track momentum p	RichDLLe
	Track pseudorapidity η	RichDLLmu
	Detector occupancy nTracks	RichDLLk
	Track charge	RichDLLp
Muon	Track momentum p	MuonMuLL
	Track pseudorapidity η	MuonBkgLL
	Detector occupancy nTracks	
	Track charge	
GlobalMuonId	Track momentum p	PIDmu
	Track pseudorapidity η	ProbNNmu
	Detector occupancy nTracks	
	Track charge	
	RichDLLe	
	RichDLLmu	
	RichDLLk	
	RichDLLp	
	MuonMuLL	
	MuonBkgLL	
GlobalPID	Track momentum p	PIDe
	Track pseudorapidity η	PIDk
	Detector occupancy nTracks	PIDp
	Track charge	ProbNNe
	RichDLLe	ProbNNpi
	RichDLLmu	ProbNNk
	RichDLLk	ProbNNp
	RichDLLp	
	isMuon	
	MuonMuLL	
	MuonBkgLL	

Table 4.1: List of the variables in the training set for each PID model. The CombDLL values for different particle hypotheses are denoted by PID*, where "*" stands for mu (muon), e (electron), k (kaon) and p (proton).

tively the calorimeter system, compensating the missing information is left to the latent space and to the generator capacity of synthesizing the reference space.
Finally, to take into account the various responses of the PID system to the different particle species, each model is trained over a set of four particle candidates (muon, pion, kaon and proton), resulting in as many instances for the same model. Summarizing, we have therefore models for each of the four PID subsystems described and for each of the four particles just listed, summing up to 16 GAN models to train.

All the models are trained using data from several *calibration samples* collected in 2016 [15]. As described in Section 1.2.3, these consist of pure samples of charged tracks

of different particle types that have been selected without the use of information from the PID subsystems. The particles under study are then extracted from the decay channels reported in Table 1.4: namely, muons from $J/\psi \to \mu^+\mu^-$ decays, pions from $D^{*+} \to (D^0 \to K^-\pi^+)\pi^+$ and $K^0_s \to \pi^+\pi^-$ decay channels, kaons from $D^{*+} \to (D^0 \to K^-\pi^+)\pi^+$ and $D^+_s \to (\phi \to K^+K^-)\pi^+$ channels, and protons from $\Lambda^0 \to p\pi^-$ decays[1].
Even if the calibration samples are selected by exclusive trigger lines and even if their purity can be improved adding kinematic constraints, the samples can still have some residual background. What is typically done in order to clean up further these samples is to use the ${}_s\mathcal{P}lot$ technique to subtract statistically all the residual background [13].

Background Subtraction

In the recent years, GANs have proved to be a powerful tool to extract data distributions from reference samples. In is original formulation, the Cramér loss function is unable to distinguish between signal and background events. In absence of a further development in this direction, the Generative Model would predict PID variables of the input particles based on a mixture of the probability distributions of electrons, pions, kaons, protons, and muons, instead of drawing examples from the distribution associated to one single mass hypothesis. The background subtraction is implemented integrating the ${}_s\mathcal{P}lot$ technique within the training procedure of the GAN system by statistically subtracting the background contribution to the computation of the loss function.
As a consequence of GAN loss function, the role of data samples is crucial, and especially their purity with respect to the distributions we want to synthesize. In fact, the generator is not able to recognize an eventual background component in order to avoid its reproduction, mitigating our purposes. It is therefore necessary to incorporate the ${}_s\mathcal{P}lot$ technique inside the training process of our GAN systems, preventing the generator network from mapping the latent space into the *background space*.

Considering a data sample populated by several sources of events, the ${}_s\mathcal{P}lot$ technique is a statistical tool capable to unfold the contributions of these different sources to the distribution of a given variable. This is achievable under the assumption that all the events are characterized by a set of variables which can be split into two components. The first component is a set of variables for which the distribution of all the sources of events are known: these variables are collectively referred to as a (unique) *discriminating* variable. The second component instead is composed of a set of variables whose distributions are either truly unknown or considered as such: these variables are collectively referred to as a (unique) *control* variable [13].
The ${}_s\mathcal{P}lot$ tool allows to reconstruct the distributions for the control variable, independently for each of the various sources of events, without making use of any *a priori* knowledge on this variable. This is done exploiting the information available from the discriminating variable to define a set of weights named ${}_s\mathcal{W}eights$. These can be used to infer the behavior of the individual sources of events with respect to the control variable, allowing to assess on average the distributions of interest [13].

Consider the dataset $\mathcal{D}$ as defined by a discriminating variable x and a control variable y, and composed as the mixture of two components, say *signal* and *background*. Signal

[1]For each of the processes listed, both the process itself and its charge conjugate are implied.

and background are described by different probability density functions, namely

$$\begin{cases} f_{\text{sig}}(x,y) & \text{for the signal} \\ f_{\text{bkg}}(x,y) & \text{for the background} \end{cases} \tag{4.3}$$

Assuming that the two joint probability density functions can be expressed as product of the PDFs in x and y ($f_{\text{sig}}(x,y) = f^x_{\text{sig}}(x) f^y_{\text{sig}}(y)$ and analogue expression for background), the ${}_s\mathcal{P}$lot technique allows to infer the distribution f^y_{sig} and f^y_{bkg} known the distributions f^x_{sig} and f^x_{bkg} and the relative yield of signal and background entries. The distributions f^y_{sig} can be approximated by the histogram of $\mathcal{D}$ with respect to the variable y obtained assigning to each entry a weight w, named ${}_s\mathcal{W}eights$, computed with the algorithm described in Ref. [13]. The result is then generalized to the computation of the average of the generic expression $\mathcal{F}$:

$$\mathbb{E}_{y \sim f^y_{\text{sig}}}[\mathcal{F}] = \frac{\sum_i w^{(i)} \mathcal{F}\left(y^{(i)}\right)}{\sum_i w^{(i)}} \tag{4.4}$$

where i runs over the entries of the dataset $\mathcal{D}$. A special case of this technique, named ${}_s\mathcal{F}it$ is obtained when $\mathcal{F}$ is a log-likelihood maximized in an unbinned maximum-likelihood fit [61].

Getting back to neural networks, we can follow the logic just described to correct the background contribution during the GAN training process. Consider, for simplicity, only the generator loss $\mathcal{L}_g(\boldsymbol{y}_r, \boldsymbol{y}_g, \boldsymbol{y}'_g)$ (4.1) ignoring the gradient penalty term so far. If $\boldsymbol{y}_r, \boldsymbol{y}_g, \boldsymbol{y}'_g$ were conditioned by the same variable $\boldsymbol{x}$, then rewriting the generator parameters update as follows would be enough to subtract the background:

$$\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta \cdot \nabla_{\boldsymbol{\theta}} \left(\frac{\sum_{i=1}^{\ell} w^{(i)} \mathcal{L}_g^{(i)}}{\sum_{i=1}^{\ell} w^{(i)}} \right) \tag{4.5}$$

where η is the learning rate and ℓ denotes the mini-batch size. However, the loss entries $\boldsymbol{y}_r, \boldsymbol{y}_g, \boldsymbol{y}'_g$ are conditioned by $\boldsymbol{x}_r, \boldsymbol{x}_g, \boldsymbol{x}'_g$ respectively, each of which is resampled every time from the training set. Therefore, it is necessary to assess the background contribution for each condition, using as many ${}_s\mathcal{W}eights$. It follows that

$$\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta \cdot \nabla_{\boldsymbol{\theta}} \left(\frac{\sum_{i=1}^{\ell} w_r^{(i)} w_g^{(i)} w'^{(i)}_g \mathcal{L}_g^{(i)}}{\sum_{i=1}^{\ell} w_r^{(i)} w_g^{(i)} w'^{(i)}_g} \right) \tag{4.6}$$

where w_r, w_g, w'_g are the weights extracted for the events of $\boldsymbol{x}_r, \boldsymbol{x}_g, \boldsymbol{x}'_g$ respectively. For the discriminator loss, the one with the gradient penalty term, we can repeat a similar discussion since $\mathcal{L}_{\text{critic}}(\boldsymbol{y}_r, \boldsymbol{y}_g, \boldsymbol{y}'_g)$ is still conditioned by $\boldsymbol{x}_r, \boldsymbol{x}_g, \boldsymbol{x}'_g$.

Then, fixing the update steps of the Algorithm 4 according to the form (4.6), it is possible to train our GAN systems with the calibration samples, being able to subtract all the background contributions. To this end, the ${}_s\mathcal{W}eights$ of the calibration samples are added to the training sets reported in Table 4.1.

4.2.2 Model Scoring

As discussed above, the training process of a GAN system is driven by a minimax game carried out by two neural networks. On one side, there is the discriminator whose aim is to

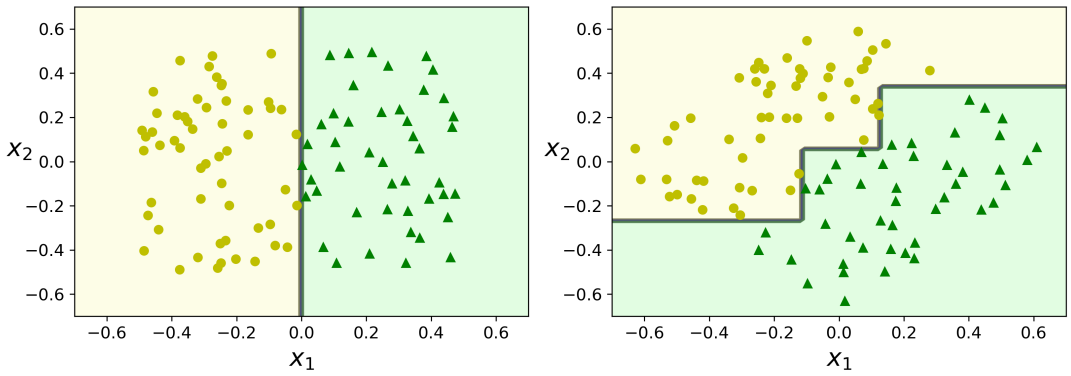


Figure 4.3: Representation of a classification problem faced with Decision Tree algorithm. The feature space is 2-dimensional and composed by x_1 and x_2. While in the left plot the split is easy, on the right the decision boundary looks unnecessarily convoluted, although the dataset is simply rotated by 45°. Reproduced from Ref. [35].

infer the origin of its inputs, distinguishing them. On the other side, there is the generator which updates its parameters in order to deceive the discriminator. In the original Goodfellow's paper [43], the Generative Model is described as a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is compared to the police, trying to detect the counterfeit currency. In this framework, the training process can be seen as a competition that allows both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles.
Following this example, the discriminator is no longer able to assess objectively the quality of the synthetic sample since the generator is trained to cloud its judgement. Therefore, if we want to test the good-looking of the generated distributions, the only way is to introduce a third player independent from the other two. Its goal is to assess the generator work without that latter can affect it. To this end, we have chosen one of the most powerful Machine Learning algorithm: a *Boosted Decision Tree* (BDT) classifier.

In order to describe the BDT classifier, it is necessary to introduce the *Decision Tree* algorithm. The latter is a versatile instrument that can be used to perform either classification or regression tasks. The basic idea is to dissect the *feature space* into multiple subsets in order to minimize the presence of impurities[2]. This sorted sequence of selection cuts can be then exploited to generalize new data input, following a tree structure where the last leaf determines the belonging class or the corresponding value inferred. This strategy is versatile, easy to interpret and, despite its simplicity, surprisingly powerful. However, it should be pointed out that Decision Trees perform predictions based on orthogonal boundaries (all splits are perpendicular to an axis), which makes them sensitive to training set rotation, as clearly shown in Figure 4.3. More generally, the main issue with Decision Trees is that their outputs are significantly affected by small variations in the training data [35].
A possible solution to this limitation is to group together the outputs of a set of Decision Trees, each of which trained over a different portion of the training sample. Such an

[2]While speaking of impurity within classification problems is straightforward, it could be tricky referring to regression problems. In latter case we can think to define an algorithm that splits the sample in order to separate outputs with values significantly different.

ensemble of Decision Trees is called a *Random Forest* and represents one of the most powerful Machine Learning algorithm available today. To improve further this method one can train the Decision Trees sequentially, each trying to correct its predecessor. This strategy is called *hypothesis boosting* and allows to obtain a predictor even more robust named *Boosted Decision Tree* (BDT).

BDT Predictions as Score

BDTs can be used effectively to assess the quality of the synthetic sample produced by the generator. The idea, originally discussed in Ref. [62], is to train a binary BDT classifier using a *labeled* dataset composed of a sampling from the reference dataset and another one from the generated data. Each of the entries for both the origins is concatenated to the corresponding elements from the conditional space (see Table 4.1) in order to provide BDTs with the same information of the other two player (discriminator and generator). The trained classifier is able to output the probability that a certain input belongs to the reference space p_{ref} or to the generator one p_{gen}[3]: they are called *class probabilities*. Frozen its parameters at the end of the training, the BDT performance produces a specific distribution for the class probabilities computed. Generally, the performances, and hence the distributions produced, are different over different data samples. Therefore, one is able to assess the quality of the synthetic sample measuring the differences between the p_{ref} distribution induced from the BDT performance over the reference dataset and the one on the generated dataset.

The BDT classifier has been implemented using the `GradientBoostingClassifier` class made available by the Scikit-Learn software package [63]. As the class name suggests, the boosting algorithm behind the strategy chosen is *Gradient Boosting* [64], where the sequentially improving of the Decision Trees is performed in order to reduce the *residual errors* made by the previous predictor.
There are 500 Decision Trees to make of this BDT a robust classifier suitable for our purposes. The power of this learning strategy can result in a heavy overfitting of the model over the training data. To avoid this, it is necessary to pursue some regularization strategies, such as restricting the maximum depth of the Decision Tree (by the `max_depth` hyperparameter) or setting the maximum number of leaf nodes (by the `max_leaf_nodes` hyperparameter). From the algorithm just described we are able to derive a score to judge the generator work.

After having trained this BDT over a sample of 500000 labeled data, we can obtain the distributions of p_{ref} over the reference space and over the generated one. The trained classifier is the most serious judge of the quality of the agreement between the real and the synthetic dataset: if the distribution of the response of the BDT is the same when evaluated on reference and generated samples, then the Generative Model is ideal. Deviations from perfection can be measured by using the *Kolmogorov–Smirnov test* which provides a score test base on the maximum absolute difference between two cumulative distribution functions:

$$D_{KS} = \max_{p_{\text{ref}} \in [0,1]} |R(p_{\text{ref}}) - G(p_{\text{ref}})| \tag{4.7}$$

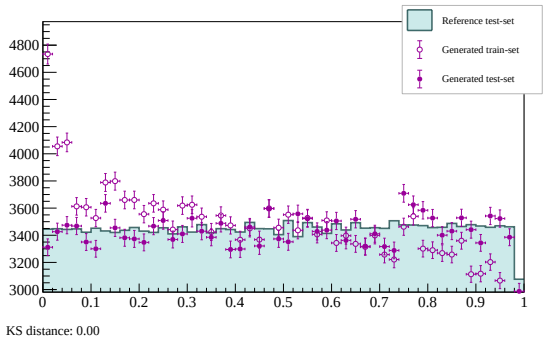
[3]Despite the notation is similar to the one adopted for p_r and p_g of the GAN loss function, p_{ref} and p_{gen} have a meaning totally different and represent output parameters of the BDT classifier.

where R is the cumulative function of the distribution produced from the reference sample, while G is the one from the generated sample. The score test D_{KS} is called *Kolmogorov–Smirnov distance*. In order to assess cleanly the Generative Model score, the KS distance is computed over a set of reference and generated entries not used for training the algorithm.
Figure 4.4 shows the p_{ref} distributions for three different cases. To emphasize the differences, p_{ref} is *flattened* over the reference sample. The KS distance between the two distributions is reported below each plot. The first case, Figure 4.4a, reports the distributions produced by a BDT trained over two randomly-selected sub-samples of the reference dataset, mimicking a perfect generator performance: as expected, $D_{KS} = 0$. On the contrary, Figure 4.4c shows what is obtained by a Generative Model failed. To ensure this outcome, both discriminator and generator have been built with a single layer and trained without updates, resulting in two networks filled randomly. Finally, in Figure 4.4b it is reported a typical plot obtained during the hyperparameters study of one of the GAN systems. We notice that as long as the score of a trained generator is significantly different from zero, there is no concern about the over-training of the neural network.

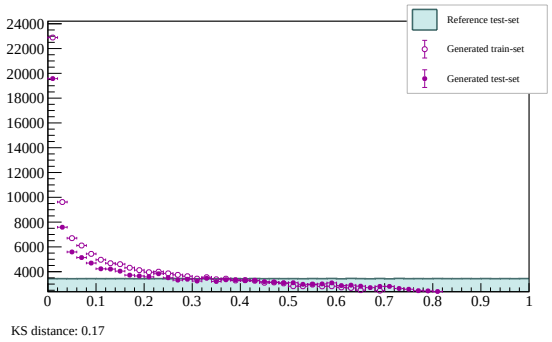
The scoring method provided by the BDT classifier plays a key role for the hyperparameters optimization described in Section 4.3. In fact, although both discriminator and generator are trained to minimize the corresponding loss function, obtaining loss values close to zero does not imply necessary good-looking distributions as output of the generator. In this sense, introducing a third player to the minimax game helps in finding the best model representation. Therefore, what is typically done to make the scoring method even more robust is to train ten independent BDTs and then to take their combination as result. Every KS distances shown in the following is obtained according to the method just described.

4.2.3 Data Preprocessing

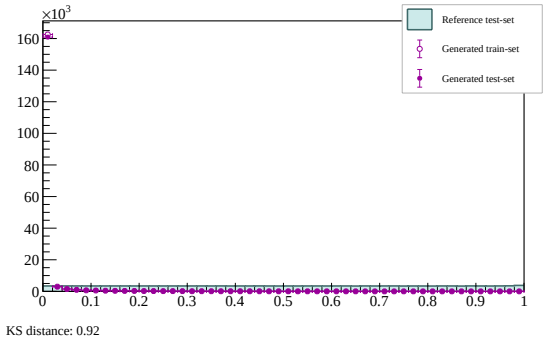
So far, we have discussed about the necessity of rewriting the Cramér GAN (3.26) in a conditional form in order to take into account the dependencies of the PID detector responses. Furthermore, we have faced with contaminated data samples and we have proposed a modification of the training process, step (4.6), in order to perform a statistical subtraction of the background, following the ${}_s\mathcal{P}lot$ technique. Finally, we have described a scoring method to assess the quality of the generator outputs. To this end, as just seen, it is necessary to introduce a BDT classifier, whose outcomes can be exploited to derive a score, the KS distance (4.7), for the Generative Model: a value closer to zero indicates a better generator.
Before diving into the training process and hyperparameters tuning, it is worth discussing the data *preprocessing*. Indeed, it should be noticed that the training set of all the GAN models is composed of different variables (see Table 4.1), each of which characterized by a specific probability distribution and by a different range. The neural networks are powerful algorithm to perform the majority of the tasks, but their inherent complexity makes the training difficult, especially when the input features take values significantly different. In fact, this can result into a trained network that makes predictions based on a subset of the feature space, discarding the features for their values and not for their information. To avoid this, it is necessary to *preprocess* the input features applying *invertible* or *bijective* transformations by preserving the whole information, but remapping their values into a



(a) *Expected outcome of a perfect Generative Model.*



(b) *Typical outcome of our Generative Models.*



(c) *Expected outcome of a Generative Model failed.*

Figure 4.4: Results of a binary BDT classifier composed of 500 Decision Trees with `max_depth = 5` and `max_leaf_nodes = 14`. Both the training set and the test one are labeled, weighted (with ${}_s\mathcal{W}eights$) and made of 500000 entries. The plots represent the BDT outcomes for three different scenarios: in the first case (a) a perfect generator is mimicked, in the second one (b) a typical result is reported, while in the last plot the outcome of a failed model is shown. In all the cases the distribution of the class probability p_{ref} over the reference test set is reported *flat* by construction. Finally, the p_{ref} distributions both over the generated train set and over the generated test set are shown in order to check an eventual overfitting of the BDT.

predefined range of values. In the particular case of Generative Models, the requirement of being *invertible*, normally motivated by generic information-preservation consideration, is more strict because the inverse transformation has to be applied to the generated dataset in order to predict physical quantities possibly associated to a specific unit of measurement.

Several data preprocessing methods exist, but in the following we will discuss about two of them in particular: a linear transformation named *standard scaling* and implemented within a first layer of the generator architecture named `NaiveLinearLayer`, and another one more sophisticated performed by the `FastQuantileLayer`. As the class names suggest, the data preprocessing phase can be performed by the first layer of the generator network. This layer is special in that its parameters are ignored by the training process, and hence they remain fixed at the values set during a special round before the start of the training.

The `NaiveLinearLayer` performs a linear transformation that requires to compute the mean and the variance of all the features during the preprocessing round. Then, taken a new sample from the training set, the input variables $\boldsymbol{x}$ are transformed according to

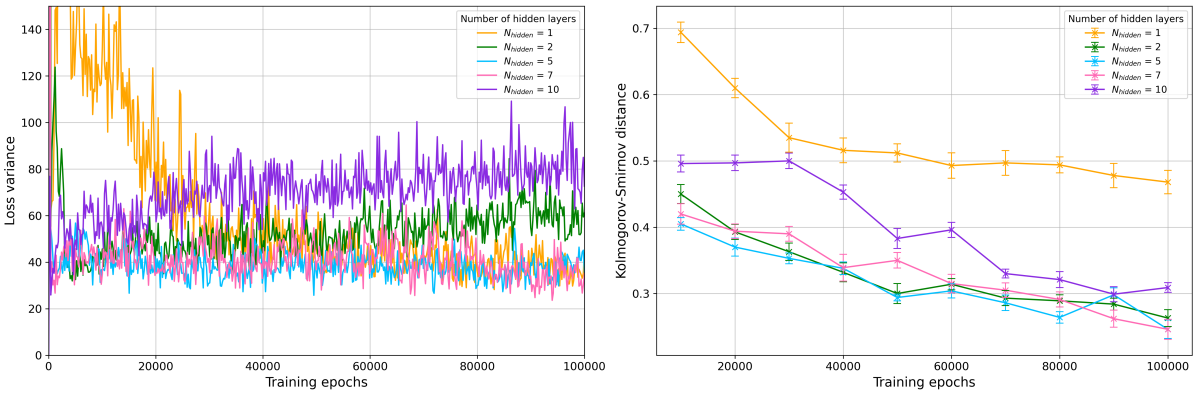
$$\mathcal{F}(\boldsymbol{x}) = \frac{\boldsymbol{x} - \boldsymbol{x}_0}{\boldsymbol{\sigma}_0} \tag{4.8}$$

where $\boldsymbol{x}_0$ and $\boldsymbol{\sigma}_0$ are the mean and the standard deviation computed during the preprocessing round. The transformation (4.8) leaves the features distributions unchanged, while roughly providing the transformed $\boldsymbol{x}$ with null mean and unitary variance.

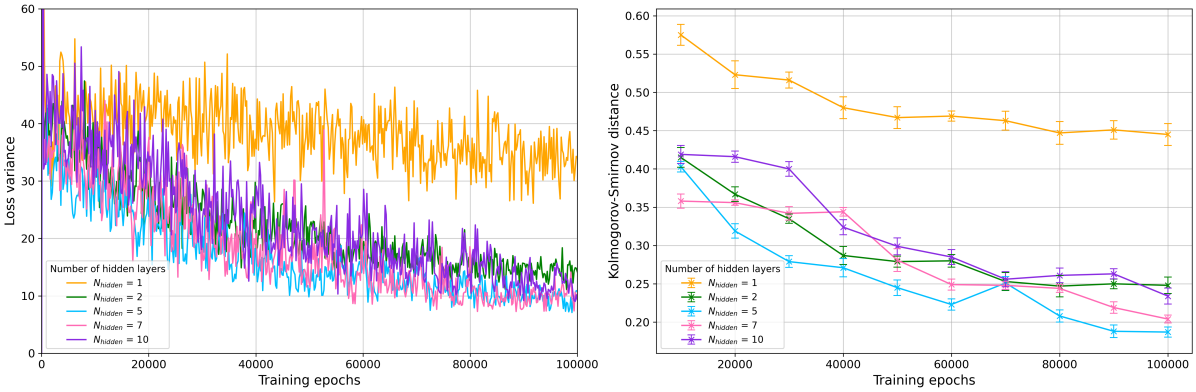
The `FastQuantileLayer` performs a more sophisticated transformation similar to the Scikit-Learn's `QuantileTransformer` method [63]. This method transforms the features to follow a uniform or a normal distribution. During the preprocessing round, the cumulative distribution function of each feature is estimated and used to map the original values to a uniform distribution. The obtained values are then mapped to the desired output distribution using the associated quantile function. Therefore, unlike the previous case, this transformation is non-linear, property from which follows the distortion of linear correlations between variables measured at the same scale. Nevertheless, this strategy allows to render the variables measured at different scales more directly comparable and to reduce the impact of outliers. This results into a robust preprocessing scheme.

Both `NaiveLinearLayer` and `FastQuantileLayer` are custom layers written with Keras [65], a widespread TensorFlow's API. In addition, both the methods provides an inverse transformation which can be used to remap the generator outputs into the original reference space[4]. Figure 4.5 represent the scores obtained from various Generative Models (with different depth), with training data preprocessed through the two methods described above. The two preprocessing techniques do not result into score trends significantly different, while the variance of the loss function highlights the better treatment of outliers obtained with the quantile-based transformer. This property together with slightly better results has led us to prefer the `FastQuantileLayer` as preprocessing strategy, selecting the normal distribution as output of the layer.

[4]To be exact, the GAN systems see only transformed data since the training process follows from the preprocessing one. Therefore, in order to reproduce the reference space, the generator outputs have to be remapped back inverting the transformation.



(a) *Training results after a linear-based data preprocessing.*



(b) *Training results after a quantile-based data preprocessing.*

Figure 4.5: Representation of the training outcomes related to neural networks (both discriminator and generator) with depth N_{hidden}. The plots report the variance of the loss functions $\mathcal{L}_g$ (left) and the KS distance (right) as a function of the training epochs. In (a) data preprocessing is performed by the `NaiveLinearLayer`, while (b) shows the results obtained following the `FastQuantileLayer` strategy.

4.3 Network Architectures and Tuning

The discriminator and generator are represented by neural networks constructed with a set of fully connected layers of 128 neurons. Each of the hidden layers is activated through a Leaky ReLU function (see 3.2.1), while the total depth of the networks N_{hidden} depends on which PID subsystem response they aim to describe: for example, the high-level reconstructed variables of the RICH detectors are reproduced effectively by a 5-layer deep map ($N_{\text{hidden}} = 5$). Following the path traced by Ref. [59], the latent space $\mathcal{Z}$ feeding the generator is obtained adding 64 input noise neurons distributed according to $\mathcal{N}(0, \mathbb{I})$. Recalling that, in the Cramér framework, the role of the discriminator is played by the map $h : \mathbb{R}^d \to \mathbb{R}^k$, we have to define also the k-size of the output space. Again, following Ref. [59], we have chosen to end the discriminator network with 256 neurons. Finally, the output layers of both discriminator and generator do not use activation functions.

Given the networks architecture, the discriminator and generator can be trained following the Algorithm 4, where both the loss functions $\mathcal{L}_g$ and $\mathcal{L}_{\text{critic}}$ are replaced by their

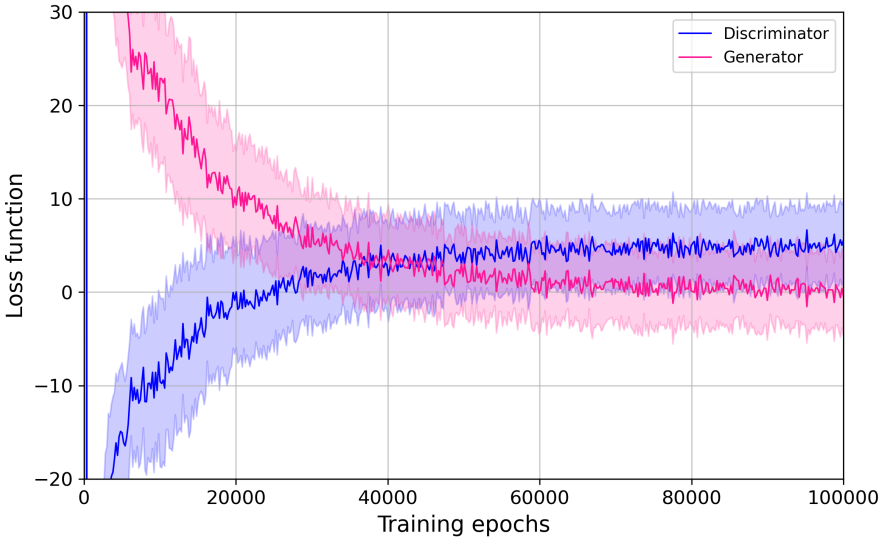


Figure 4.6: Discriminator and generator learning curves.

conditional form in order to accurately reproduce the PID subsystems with respect to track kinematics parameters and detector occupancy. Since it is a GAN algorithm, the training process is driven by a minimax game which can be computed by gradient-based methods, such as Adam [40]. To be exact, latter is an improvement of the Mini-batch Gradient Descent method, and therefore it solves the optimization problem computing the gradient over subsets of the training set, the mini-batches. The size of these mini-batches is called *batch-size* and represents a crucial element of the training process. Then, following the Algorithm 4, mini-batches are continuously sampled from the calibration samples [15], where the residual background is statistically subtracted using the ${}_s\mathcal{P}lot$ technique. Finally, to further stabilize the training process, input data are preprocessed using a quantile-based transformer.
These strategies allow to produce good-looking distributions for the high-level PID variables of interest (see Table 4.1) already from 100000 *epochs*. Looking at the Algorithm 4 once again, it follows immediately that the number of epochs coincides with the amount of updates for the generator parameters. The discriminator ones, instead, are updated n_{critic} times per epoch: in our models, $n_{\text{critic}} = 5$. Figure 4.6 shows a typical learning curve resulting from the steps just described.

4.3.1 Hyperparameters Tuning

Now that we have defined the baseline of our model and of our training process, we can exploit the scoring method provided by the BDT classifier to select the best combination of hyperparameters in order to improve the starting model. Two highly important hyperparameters are the *learning rate* η and the *batch-size*. While the first one is arguably the most important hyperparameter, the other one can have a significant impact on the model performance and on the training time [35]. In addition, these two hyperparameters are heavily related, hence finding the best combination is not particularly straightforward.
Finding a good learning rate is very important. In fact the convergence or divergence of

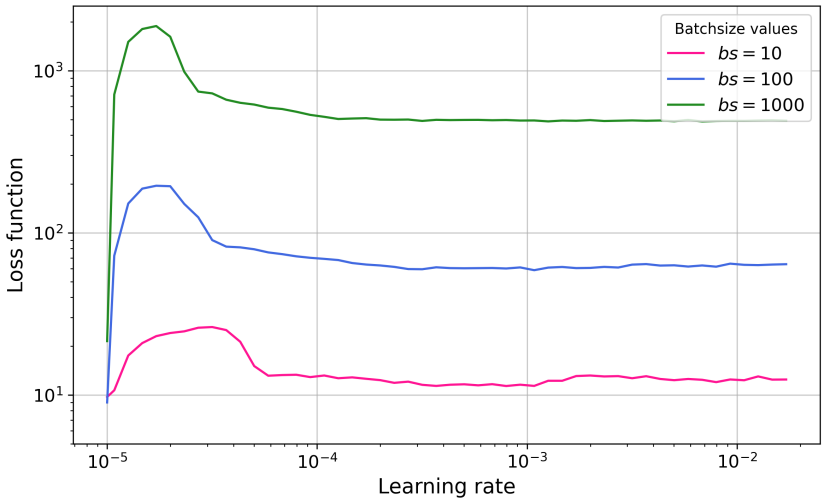


Figure 4.7: Learning rate preliminary study.

the training process depend mostly on its value. Therefore, in order to find a good starting point, one can train the model under study for a few hundred iterations requiring that the learning rate varies by several orders of magnitude. In doing so, the loss function will show trend changes according to the corresponding learning rate value. In this scenario, reported in Figure 4.7, the loss function, after an initial rising, drops until the model is no more able to learn (flat line). A good starting point for the learning rate is the value previous to the flat line. Referring to the Figure 4.7, for the model with $bs = 10$ one can set $\eta_0 = 3 \times 10^{-4}$, for $bs = 100$ follows $\eta_0 = 2 \times 10^{-4}$, while $bs = 1000$ results in $\eta_0 = 1 \times 10^{-4}$.

The values listed above are good learning rates for the first steps of the training process. However, to avoid oscillations around the optimum due to the high value chosen, one can reduce progressively the learning rate with the passing of epochs: this strategy is named *learning scheduling*. Figure 4.8 reports an example of *exponential scheduling*, where the initial learning rate value η_0 is reduced exponentially according to

$$\eta_t \leftarrow \alpha_{\text{drop}}\, \eta_{t-1} \tag{4.9}$$

This update step is performed every certain number of epochs whose number is defined through another hyperparameter. Figure 4.8 shows how the generator performance change for different exponential scheduling strategies.

Another interesting study is about different combinations of the hyperparameters mentioned. In fact, we expect models trained with smaller batch-sizes to arrive to an end of the training process faster, but paying with a lesser number of entries taken into account and then quality of the generated sample. On the other hand, models trained with large batch-size are significantly slower but provide good results thanks to the greater amount of data processed. This comparison is reported in Figure 4.9 where the expected behaviours are shown. In addition, this figures offer two interesting items for discussion.
The first one is related to the existing connection between batch-size and loss variance: larger size implies larger variance. This is probably due to an increase in the probability of selecting an outlier providing unexpectedly large contributions to the gradients. As

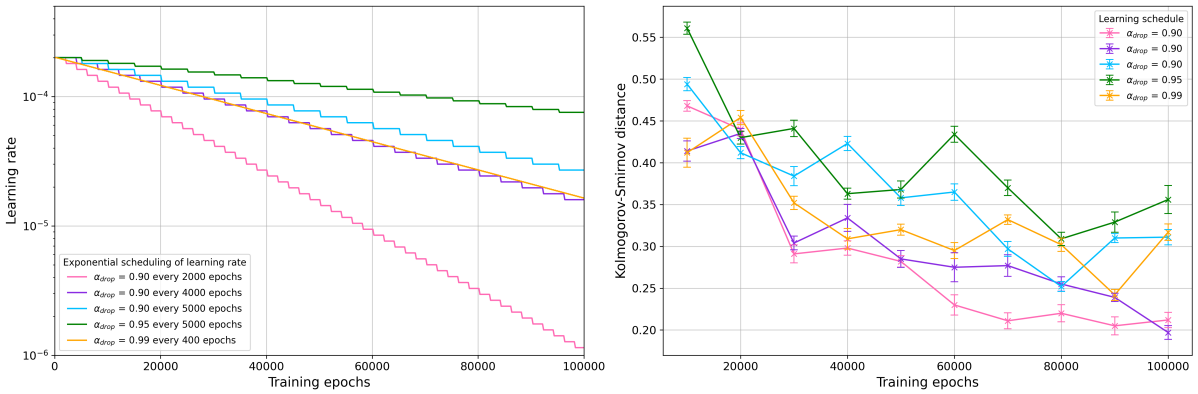


Figure 4.8: Studying the effects of exponential scheduling over a model with $bs = 100$ and $\eta_0 = 2 \times 10^{-4}$. On the left, the learning rate trends are reported for different values of α_{drop} and for various updates frequency. On the right, the corresponding KS distances are shown as a function of the training epochs.

a matter of fact, the optimal learning rate derived from the study depicted in Figure 4.7 *decreases* at increasing batch-size. This behaviour is unusual since larger batch-size usually result into more reliable estimates for the gradients allowing for larger learning rates, but can be understood in the context of an adversarial training. Nonetheless, a larger learning rate and a smaller variance do not result into a better-performing generator, as made evident by the comparison of the scores obtained with different combinations of batch size and learning rate in Figure 4.9.
Another interesting property we can talk about looking at the Figure 4.9 is the relation between the loss function value and the Generative Model performance. As said, the training process of both the discriminator and generator is driven by a minimax game where, for what concern the generator, the parameters are updated to minimize the loss function $\mathcal{L}_g$ (reported on the left plot of Figure 4.9). Nevertheless, reaching a loss function value close to zero does not mean uniquely producing a Generative Model with good performance. This is pointed out by the results shown in Figure 4.9, especially looking at the first steps of the training process. In fact, after 20000 parameters updates, both the models with $bs = 10$ and $bs = 100$ seem to be close to zero, while the one with $bs = 1000$ is away from the target. However, looking at the KS distances (reported on the right plot of Figure 4.9) for the same epoch, one may be convinced that the best performance is achieved by the generator trained with the larger batch-size. What said above highlights the importance of having a third independent player to assess the results the minimax game: the BDT classifier, in this context, plays a key role allowing to overcome the clouded competition between discriminator and generator.

4.3.2 Merging Loss Functions

As said at the beginning of this chapter, both the algorithms and the models just described follow from Ref. [59] and Ref. [60]. The usage of Cramér GAN systems derives directly from Ref. [59], to which we have referred in order to reproduce the high-level variables of the RICH detectors. However, the good work of Ref. [60] have proved that also GAN systems based on *cross-entropy* can reproduce accurately the PID variables. In this sense,

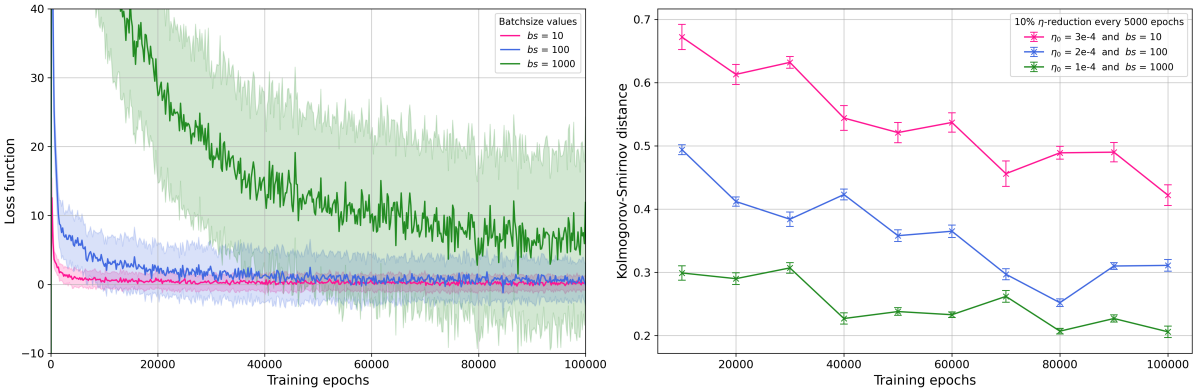


Figure 4.9: Study of the different outcomes for three combinations of learning rate and batch-size. On left plot, the loss function values are reported with the corresponding variance. On the right, the KS distances are shown for ten different points of the training process.

we can exploit the scoring method developed in Section 4.2.2 to assess the quality of the generated samples obtained from the Cramér loss $\mathcal{L}_{\text{cramer}}$, from the cross-entropy loss $\mathcal{L}_{\text{bce}}$ or from a mixture of both.

Before showing the results, we should define the cross-entropy. The cross-entropy can be formulated as a *distance* measure between two distribution, and hence it can be used to train Machine Learning algorithms. The cross-entropy of the distribution q relative to a distribution p over a given set is defined as follows:

$$H(p, q) = -\mathbb{E}_{x\sim p}[\log q(x)] = -\sum_{x\in\mathcal{X}} p(x) \log q(x) \tag{4.10}$$

The cross-entropy is widely used also in statistics for its relation with the maximum likelihood estimation[5]. In Machine Learning it is widely used for classification problems. Therefore, we can replace the Goodfellow's original loss function [43] with a binary cross-entropy, letting the discriminator output be the probability $\hat{p}$ that the input comes from the reference space:

$$\mathcal{L}_{\text{bce}} = -\sum_i \left[p_i \log\left(\hat{p}_i(\boldsymbol{x}_i)\right) + (1 - p_i) \log\left(1 - \hat{p}_i(\boldsymbol{x}_i)\right)\right] \tag{4.11}$$

where the index i runs over the mini-batch entries, p_i denotes a binary label which is 0 if the i-th entry comes from the generated sample, while it is 1 if the i-th entry comes from the reference sample. This loss function can be included in Algorithm 3 and used to train both discriminator and generator. The former updates its parameters to maximize the correct identification of inputs origin, while the latter, again, modifies its parameters in order to deceive the discriminator.
Following what done in the energy distance (4.1), generalizing the cross-entropy loss is straightforward. Furthermore, to allow the training over the calibration samples, also the

[5]It is straightforward prove that the cross-entropy coincides with the log-likelihood of a Bernoulli distribution.

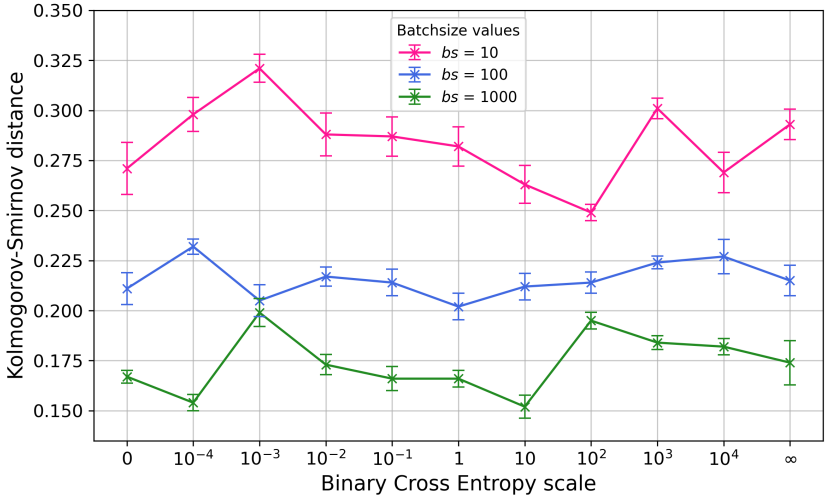


Figure 4.10: Comparison between models trained with a combination of the energy distance and the binary cross-entropy, weighted by the hyperparameter ξ, the binary cross-entropy scale. The results are reported for three different batch-size values ($bs = 10, 100, 1000$). It should be noticed that the x-axis is not in scale and include the special symbol ∞. The latter denote simply a GAN system based only on a cross-entropy loss.

(4.11) have to be modified with in order to subtract the residual background. Then we have

$$\mathcal{L}_{\text{bce}} = -\sum_i w_i \left[p_i \log\left(\hat{p}_i(\boldsymbol{x}_i)\right) + (1 - p_i) \log\left(1 - \hat{p}_i(\boldsymbol{x}_i)\right) \right] \tag{4.12}$$

where w_i denotes the i-th ${}_s\mathcal{W}eight$.

At this point, we can merge the Cramér loss function and the BCE one to test the performance of each strategy. Weighting the two contributions with a new hyperparameter, we are able to derive models trained using loss function dominated by either the first strategy or the other one. The resulting loss function follows:

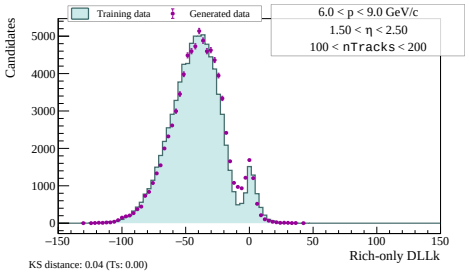
$$\mathcal{L} = \mathcal{L}_{\text{cramer}} + \xi\, \mathcal{L}_{\text{bce}} \tag{4.13}$$

where ξ is a hyperparameter named *binary cross-entropy scale*.
Figure 4.10 shows the KS distances resulting from models trained with various batch-size values ($bs = 10, 100, 1000$) and different choices for the hyperparameter ξ. Although the theory behind the Cramér distance provides the corresponding loss function with useful properties to face the *mode collapse* problem and the *vanishing gradient* of the generator, the outcomes reported in Figure 4.10 does not shown a significantly trend change over the tested range for ξ. The training process dominated by the cross-entropy ($\xi \to \infty$) are not always able to converge because of the vanishing gradient, but when successful the results achieved does not show significant deterioration. On the other hand, the energy distance greatly improves the success rate of the training, but no significant trend of the generator score on ξ is observed.

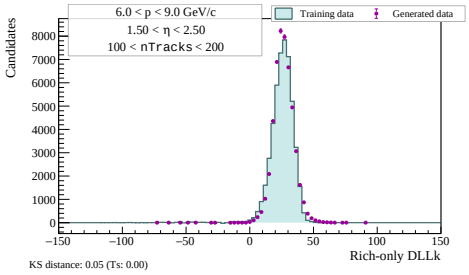
The conclusion of this study is that using the Cramér loss function to optimize the generator in an adversarial scheme leads to a more stable training procedure, but does not correspond to a significant improvement in the quality of the trained generator. Furthermore, we observe that a whatever tiny contribution to the loss function from the energy distance, cures the instabilities of the training procedure based on the binary cross-entropy, which no longer needs noise injection to prevent vanishing gradients.

4.4 Model Validation

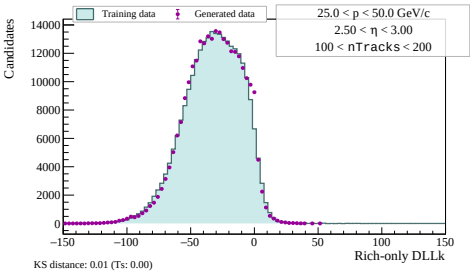
While the score described above is an effective and synthetic assessment of the quality of the generator, it is important to evaluate how the distribution of the physical quantities obtained through the generator are similar to those in the reference sample. The algorithm described and validated above has been used successfully to train generators predicting the response of the RICH and MUON systems, together with the Global Particle ID and Global Muon ID variables.
We report in the following some example of the good agreement obtained between the generated and the reference samples. Possible generalization error of the generator will be investigated in the next chapter and are not expected to be related to an over-training of the algorithm, but rather to differences between different decay modes not encoded in the input variables of the generator. For this reason, the same datasets used for training and listed in Table 4.1 are used for validation. In the following some good-looking examples are reported. Figure 4.11 shows the outcome of the `Rich` model for the four particle species (pion, kaon, muon and proton) in reproducing the DLL reconstructed from the detectors for the kaon hypothesis. Figure 4.12 reports the output of the `Muon` model for muon and proton track candidates. Finally, Figures 4.13 and 4.14 show an example of what mapped out from the `GlobalMuonId` and `GlobalPID` models respectively. All the distributions are reported in bins of momentum p, pseudorapidity η and detector occupancy `nTracks`. The kinematic ranges are chosen to emphasize the variability of the shapes of the distributions parametrized by the neural network, and represent only a small part of the available plots.



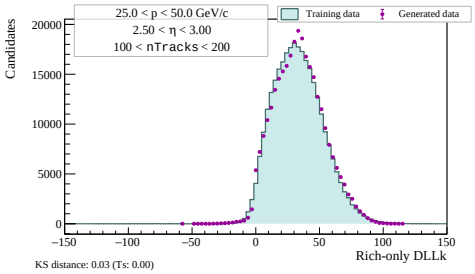
(a) *Pion track candidates.*



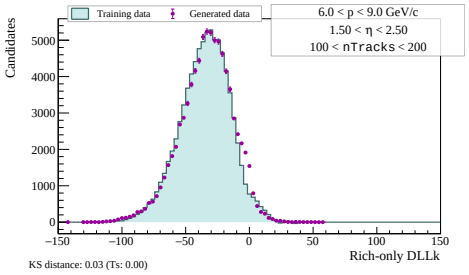
(b) *Kaon track candidates.*



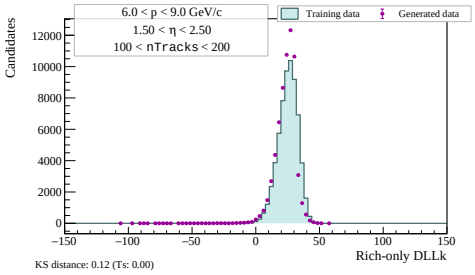
(c) *Pion track candidates.*



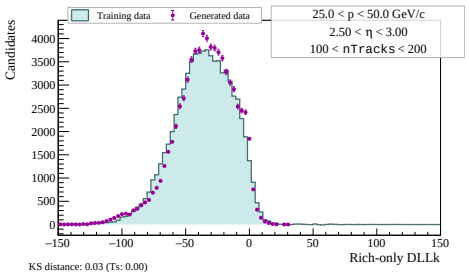
(d) *Kaon track candidates.*



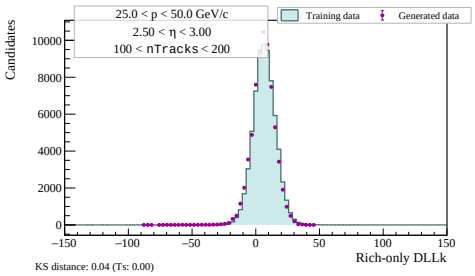
(e) *Muon track candidates.*



(f) *Proton track candidates.*

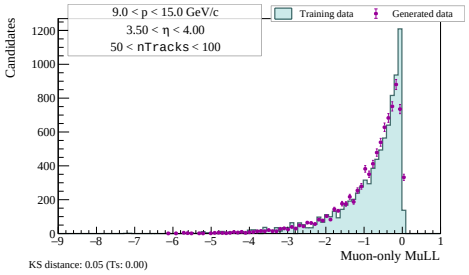


(g) *Muon track candidates.*

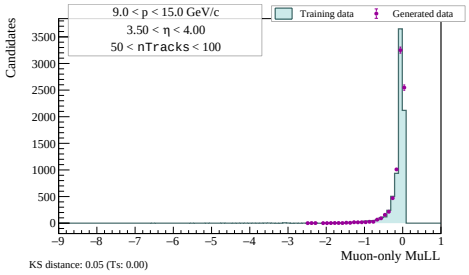


(h) *Proton track candidates.*

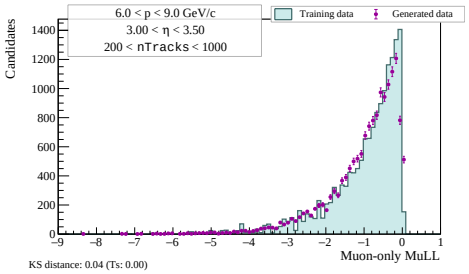
Figure 4.11: Weighted real data and generated distributions of `RichDLLk` for pion, kaon, muon and proton track candidates in bins of momentum p, pseudorapidity η and detector occupancy `nTracks`.



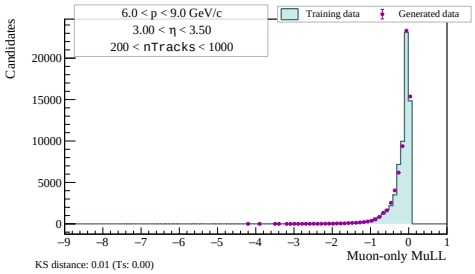
(a) *Muon track candidates.*



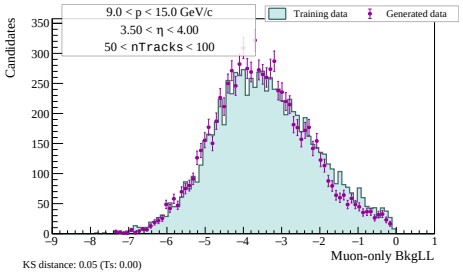
(b) *Proton track candidates.*



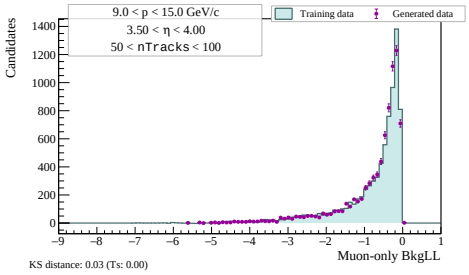
(c) *Muon track candidates.*



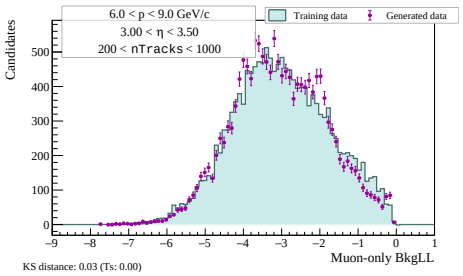
(d) *Proton track candidates.*



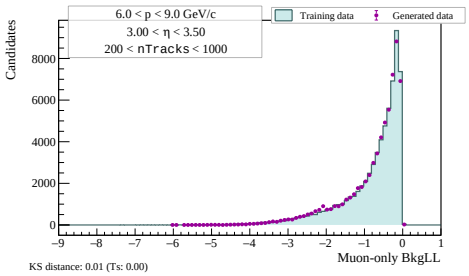
(e) *Muon track candidates.*



(f) *Proton track candidates.*

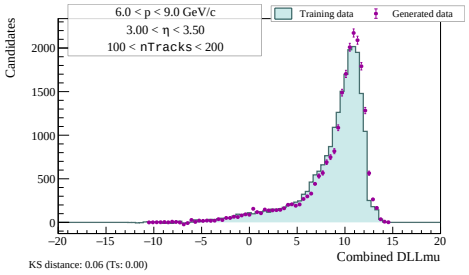


(g) *Muon track candidates.*

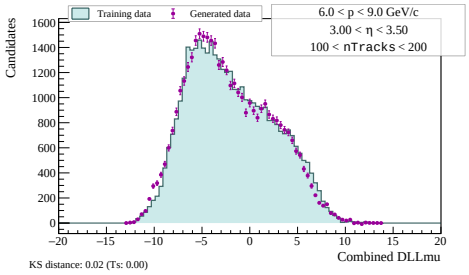


(h) *Proton track candidates.*

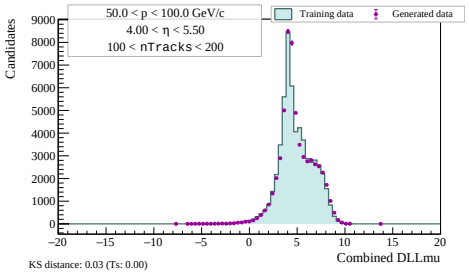
Figure 4.12: Weighted real data and generated distributions of `MuonMuLL` and `MuonBkgLL` for muon and proton track candidates in bins of momentum p, pseudorapidity η and detector occupancy `nTracks`.



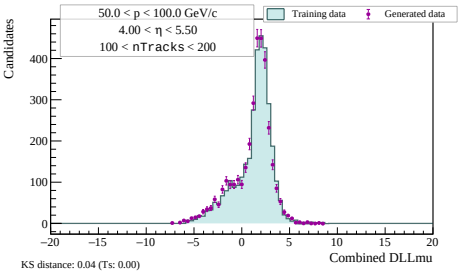
(a) *Muon track candidates.*



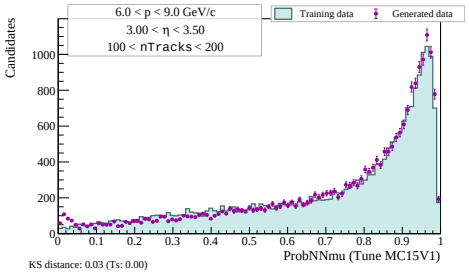
(b) *Proton track candidates.*



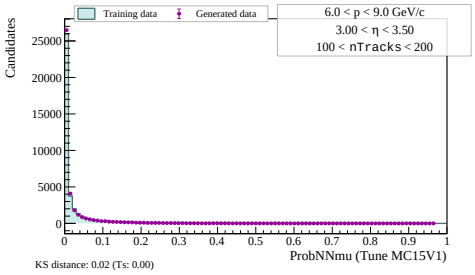
(c) *Muon track candidates.*



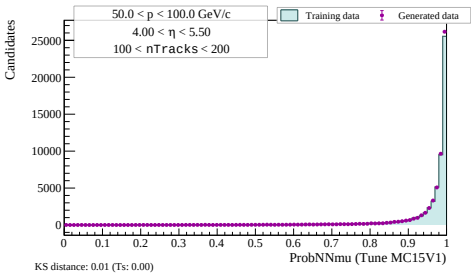
(d) *Proton track candidates.*



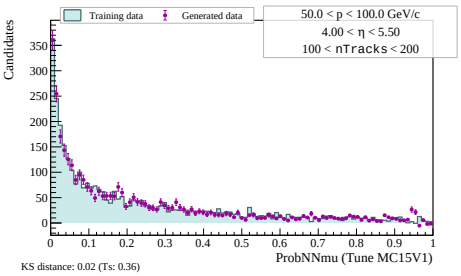
(e) *Muon track candidates.*



(f) *Proton track candidates.*

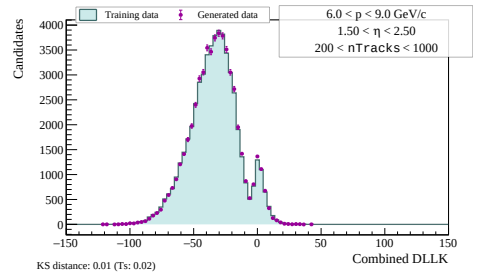


(g) *Muon track candidates.*

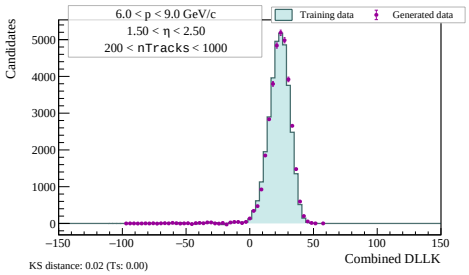


(h) *Proton track candidates.*

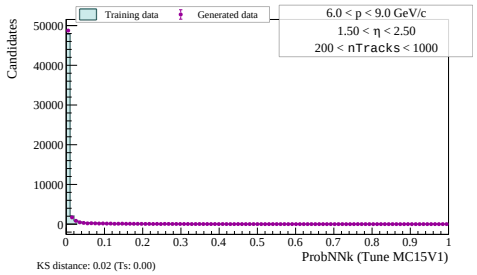
Figure 4.13: Weighted real data and generated distributions of `PIDmu` and `ProbNNmu` for muon and proton track candidates in bins of momentum p, pseudorapidity η and detector occupancy `nTracks`.



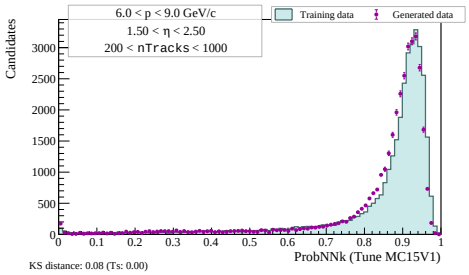
(a) *Pion track candidates.*



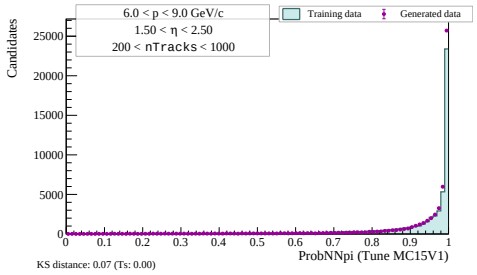
(b) *Kaon track candidates.*



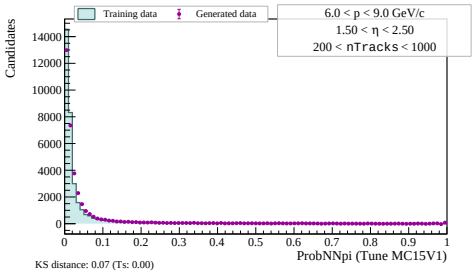
(c) *Pion track candidates.*



(d) *Kaon track candidates.*



(e) *Pion track candidates.*



(f) *Kaon track candidates.*

Figure 4.14: Weighted real data and generated distributions of `PIDk`, `ProbNNk` and `ProbNNpi` for pion and kaon track candidates in bins of momentum p, pseudorapidity η and detector occupancy `nTracks`.

Chapter 5

Integration of GAN models within the LHCb Simulation

5.1 Introduction

The upgrades journey undertaken by LHCb will lead to an increase of the luminosity and to an improvement of the selection efficiency, all in order to reach unprecedented physics accuracy. To this end, it is crucial to succeed in collecting larger and larger datasets, but also designing trigger lines capable to select highly pure sample will play a key role. Both the tasks will prove to be great challenges, especially starting from Run 5 when the increased amount of pileups[1] will make the hadronic environment even harsher. Providing trigger solutions against the contamination of the data samples and defining strategies aimed to take full advantage of experiment upgrades become therefore mandatory. All these challenges can be faced exploiting an increasing variety of simulated decay modes, whose role will become more and more critical to achieve the desired diversity in the physics program of the experiment.
As discussed in Chapter 2, according to the predictions on the CPU needs for the future offline computing system, the pledged resources will be barely sufficient to comply with the data analysis requirements, leaving little or no room to the simulation intended for the design and refinement of the selection strategy at trigger level [20]. In this context, developing and implementing faster simulation solutions becomes mandatory.

In recent years, the community of theoretical physicists has been demanding with increasing enthusiasm mathematical models to parameterize the ability of the experiments to reconstruct physics processes and, in ultimate analysis, to be sensitive to new models of physics beyond the Standard Model.
Ultra-fast simulation has been proving a viable solution to overcome the computing resources constraints and a precious tool for theorists. Among various solutions, generative models provided by GAN training seem promising, allowing good-looking distributions reproduction, as shown in Chapter 4. The resulting distributions do not need hand tuned parameterization, but are drawn directly from a competition between two neural networks, within a *minimax game*. As seen in Section 3.3, the optimal configuration can be achieved when both discriminator and generator are not able to improve their performance exploiting the information of the counterpart. This condition is known as *Nash*

[1] Number of events occurring per bunch crossing.

equilibrium and allows, in principle, to obtain a perfect generative model ($D_{KS} = 0$).

In the previous chapter, it has been shown that GANs can be trained over calibration samples and that can exploit effectively particle kinematics and event multiplicity to reproduce faithfully the distributions of high-level reconstructed variables. Then, we can think to define a simulation framework that, starting from a set of variables at generator-level, provides a set of parameterizations to map these ones into reconstruction-level quantities. After having reconstructed the tracks, they can be passed sequentially to the various models in order to obtain high-level variables, for instance related to Particle Identification. This strategy would allow to produce huge simulated samples at low CPU-cost. Furthermore, these samples could be used to design new trigger lines, exploiting the rapidity with which it is possible to reproduce several large samples, testing the many selection strategies upon the refinement of some efficiency or resolution effect.

During the first months and years of operations of a new or newly upgraded experiment, it is indeed reasonable to expect that parts of the description of the detector in the simulation, or some of the reconstruction algorithms, will be refined to achieve optimal performance on the real data. Full simulation can be used to describe the effect of the modifications in terms of efficiency and resolutions. These can then be propagated via a fast simulation to the of hundreds to thousands physics processes an experiment is intended to be sensitive at, in order to assess the effect of the modifications on the efficiency and purity of the selected signal candidates.
It is worth noticing that for such a quick turn-over, the accuracy on the selection efficiency is not the top priority as it is not in general for the trigger studies. A simulation framework like the one just described can become a precious tool to reduce computing requirements for simulation, ensuring to carry on this kind of studies. An example of simulation framework that follows the structures just described is *Lamarr*.

5.1.1 The Lamarr Framework

Historically, the first attempt of the LHCb Collaboration to provide an ultra-fast simulation framework was based on an integration between Delphes and Gauss. The main idea was to provide parameterizations of the LHCb experiment within Delphes cards for theorists to access them, while integrating Delphes within Gauss, the full simulation software of LHCb, to replace *parts* of the full simulation with the corresponding parameterization implemented within a Delphes card.

The brilliant idea, however, was found of difficult implementation for several reasons: Delphes was developed bearing in mind experiments with a 4π geometry, with subsequent layers of different detectors enclosed one within the other, and with a cylindrical symmetry. Besides, Delphes was developed to take as input the simulated events from generators that are typically used for *high-*p_{T} physics, mainly focusing on jets, rather on the generators used to simulate the physics of heavy quarks. While Delphes standalone can easily take as an input the files produced by these special generators, once integrated within Gauss to ensure the consistency of the configuration of the experiment and of the LHC, the difficulties begun to arise. As a matter of fact, the LHCb Collaboration developed several *ad-hoc* parameterizations for Delphes, rewriting most of the parameterization-related part while paying a high cost in terms of development and CPU resources to make the Delphes framework consistent with Gauss. The natural conclusion of the story has been that the parameterizations produced for Delphes were

directly fit into GAUSS creating a new DELPHES-independent application producing the same output with a cost reduced by 40% to 70% depending on the decay mode and simulation configuration. The new application, named Lamarr, is today the official ultra-fast simulation framework in LHCb.
The building blocks composing Lamarr are:

- *Input preprocessing*, the output of the generators simulating the physics processes in their version embedded in GAUSS are acquired and preprocessed into STL containers needed to compute the reconstruction-level quantities;
- *Event statistics*, the total number of tracks that would be reconstructed in that event in a full simulation is randomized, possibly using the actual number of generated particles, to provide a measure of the detector occupancy affecting the efficiency and the resolution of the next steps;
- *Tracking efficiency*, some of the generated particles are promoted to reconstructed objects according to a look-up table defining the reconstruction efficiency based on the kinematic variables of the particle;
- *Smearing*, the momentum of the particle is modified adding a random contribution representing the error introduced in the measurement of the trajectory of the particle within the spectrometer;
- *RICH*, the differential log-likelihood that would be obtained running the RICH reconstruction algorithm are predicted using a deep neural network trained as described in Chapter 4.
- *isMuon*, a simple FNN is run to predict the efficiency of the `isMuon` boolean criterion based on the kinematics of each particle and of the average occupancy of the detector;
- *MUON*, the differential log-likelihood of the hypothesis that a particle is a muon versus the hypothesis it is not are predicted using a deep generative model as described in Chapter 4.
- *Global Particle Identification*, the information simulated for the RICH and for the MUON system are combined with the kinematics of the particle and the tracking information to formulate predictions on the variables used for particle identification. While in the real reconstruction procedure these variables are obtained with multivariate procedure (likelihood combination or FNNs), in the fast simulation they are randomized according to a generator trained as discussed in Chapter 4.

The L0 trigger, selecting events according to the presence of high-p_T particles, introduces an efficiency term at pre-reconstruction level, whose result is to prevent some lower momentum decay-candidate to be reconstructed. Representing this contribution within Lamarr is currently not possible, reason why the only way is to *emulate* the latter applying reasonable requirements on the reconstructed momentum.
The integration of the generative models with Lamarr relies on an interface named `GaudiTensorFlow` that binds the C APIs of TensorFlow to any GAUDI application, in this case GAUSS. `GaudiTensorFlow` provides the whole set of operations that can be run

in TensorFlow, but introduces an important *overhead* that will be discussed in the next section.

At the end of the described steps, Lamarr produce an output including the majority of the variables and related uncertainty that are necessary at analysis level, stored on disk in the exact same format as the datasets acquired in pp collisions. These output files are therefore processed with the LHCb analysis software (named *Bender*) to combine the "reconstructed" particles into their decayed mother particles, produce `nTuples` and draw plots.

Figure 5.1 reports the invariant mass of $\Lambda_c^+ \to pK^-\pi^+$ candidates as obtained in real data and through the Lamarr simulation. The Λ_c^+ candidates are selected by requiring a reconstructed trajectory of the Λ_c^+ inconsistent with the primary vertex, but consistent with a secondary vertex producing, together with the Λ_c^+, a muon of opposite charge. The selection criteria are therefore tuned to select mainly Λ_c^+ produced in semileptonic decays $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$.

It is worth to highlight that this decay mode was not used as part of the training dataset of any of the algorithm involved in Lamarr, the acceptable agreement of the resolutions displayed in Figure 5.1 has been obtained modelling the detector response upon a generic simulated sample which is not specific to Λ_b^0 or Λ_c^+ in anything. While the resolution is acceptably reproduced, the shape of the resolution function is not, and this is due to imperfections in the parameterization caused by a training sample with suboptimal kinematic coverage, and will be improved in the future releases of Lamarr.

The decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ will be used for the validation of all the steps of Lamarr because it is a non-trivial decay sensitive to several aspects of the simulation, in particular:

- it is a semileptonic decay with a non-trivial decay dynamic that must be modeled effectively with decay form factors;
- it includes most of the charged stable particles that need parameterizations, bad modelling of the Particle Identification properties of each family would result in bad agreement of the reconstructed Λ_c^+ and Λ_b^0 kinematic distributions;
- it is selected among the *calibration samples* with special care not to bias the detector response on the proton, but, as mentioned above, was never included in any training procedure to obtain parameterizations because of the limited statistics.

Figure 5.2 reports the dependency on the momentum of the efficiency on the selected Λ_c^+ candidates of three different requirements on the particle identification of the proton. The variable used to defined the criterion is the `ProbNN`, obtained with as output of a simple FNN. The agreement between the efficiency dependency obtained from calibration data and from the ultra-fast simulation is an evidence of the generalization capabilities of the neural network discussed in Chapter 4, that modeled the detector response to protons using examples of $\Lambda^0 \to p\pi^-$ decays, *only*.

The limitations of Lamarr

Lamarr represents a promising option to implement ultra-fast simulation within the Gauss framework, with the opportunity of choosing which parts of the full simulation

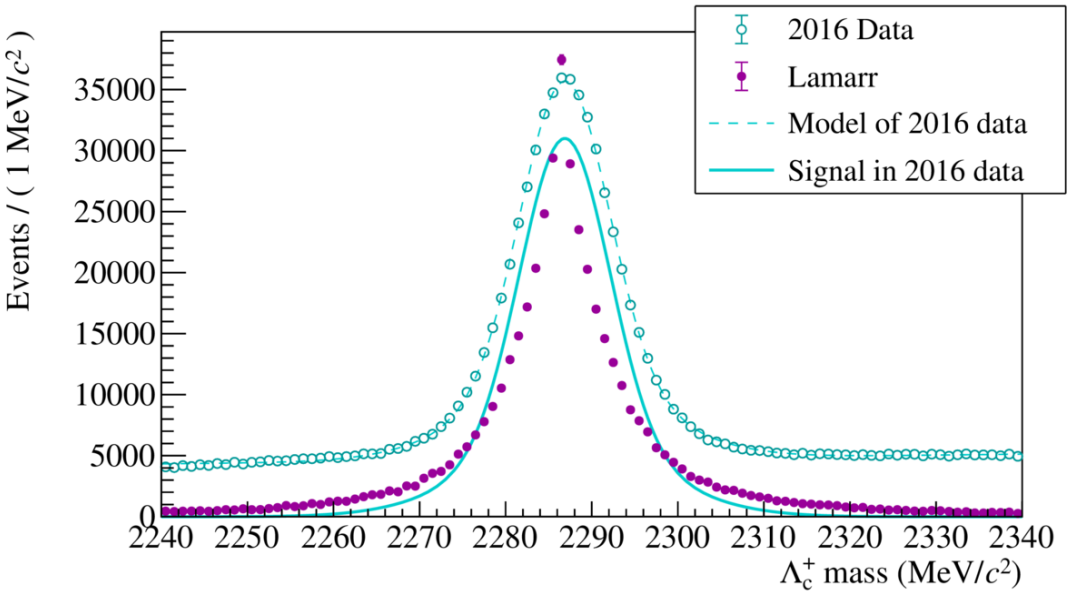


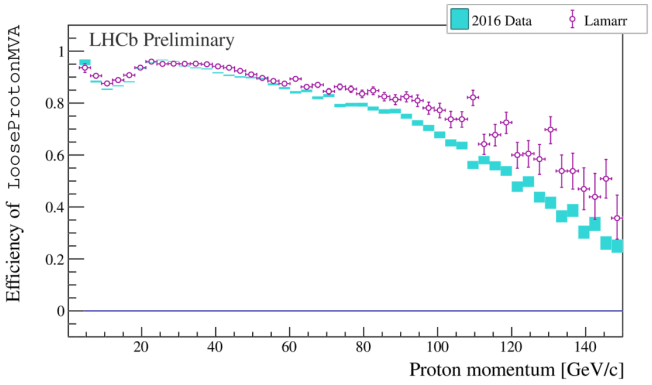
Figure 5.1: Performance plots of the Lamarr Prototype reporting the invariant mass distribution of reconstructed and simulated Λ_c^+ baryons produced in the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ and decays as $\Lambda_c^+ \to pK^-\pi^+$. The data is fitted with a model composed of a double Gaussian function for the signal and a second-order polynomial for the combinatorial background. The shape of the signal peak is compared to the distribution obtained from simulation. Reproduced from LHCB-FIGURE-2019-017.

should be replaced by a parameterization. However, this possibility comes with a few drawbacks discussed in the following.

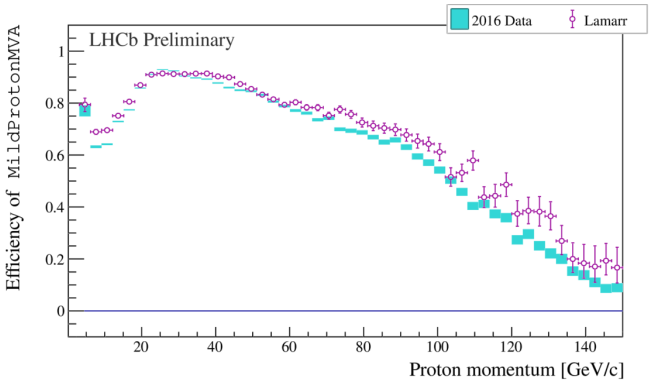
The most obvious limitation of Lamarr is its usability for a community which has *no access* to the LHCb software stack and data, within the computing framework provided by CERN. The opportunity for theorists to model the response of the LHCb detector in order to check for the sensitivity to models of physics beyond the Standard Model, which was originally provided by the DELPHES cards integrated within GAUDI, is *lost* when moving to Lamarr.

From the perspective of the LHCb Collaboration, it should be noticed that the development cycle of a Lamarr parameterization follows substantially the same development cycle of the whole simulation software, and hence some steps requires the Collaboration approval. Since imperfections in the parameterization get only evident with large statistics, and since large statistics samples are only available with released software, converging towards an optimal parameterization can become laborious.

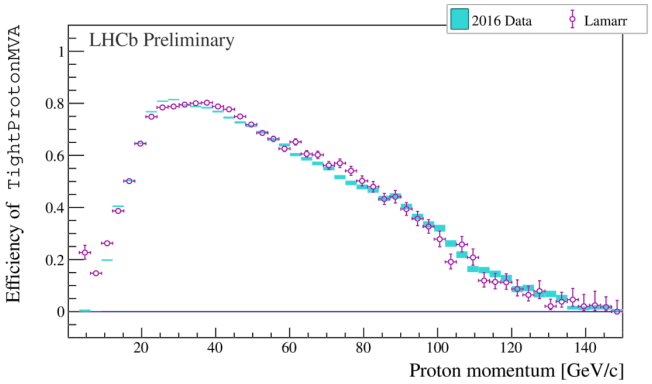
Lastly, Lamarr is not efficient in terms of CPU usage because it relies on GAUDI which is designed for *event-grained* parallelism, with events that run independently on each other and possibly in parallel processes on the same machine. In other words, a set of algorithms and routines involved in the data processing are loaded once per event, resulting into a continuous access to disk memory and hence a tremendous waste of CPU cycles. The waste is major when the libraries containing the algorithms are large, as it is the case for complex neural networks developed with frameworks like TensorFlow [42] or PyTorch [66].



(a) *Efficiency plot for* `ProbNNp > 0.6`.



(b) *Efficiency plot for* `ProbNNp > 0.8`.



(c) *Efficiency plot for* `ProbNNp > 0.95`.

Figure 5.2: Performance plots of the Lamarr Prototype reporting the efficiency of three different requirements on the output of a single FNN trained to identify protons (namely `ProbNNp`) as evaluated on protons tagged through the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to pK^-\pi^+$. The efficiency is compared, in bins of the proton momentum, for a dataset selected without introducing bias on the Particle Identification of the proton (cyan shaded area), and an ultra-fast simulation sample where the PID variables are modeled through a Generative Adversarial Network trained using protons from $\Lambda^0 \to p\pi^-$ decays, only (purple markers). Reproduced from LHCB-FIGURE-2019-017.

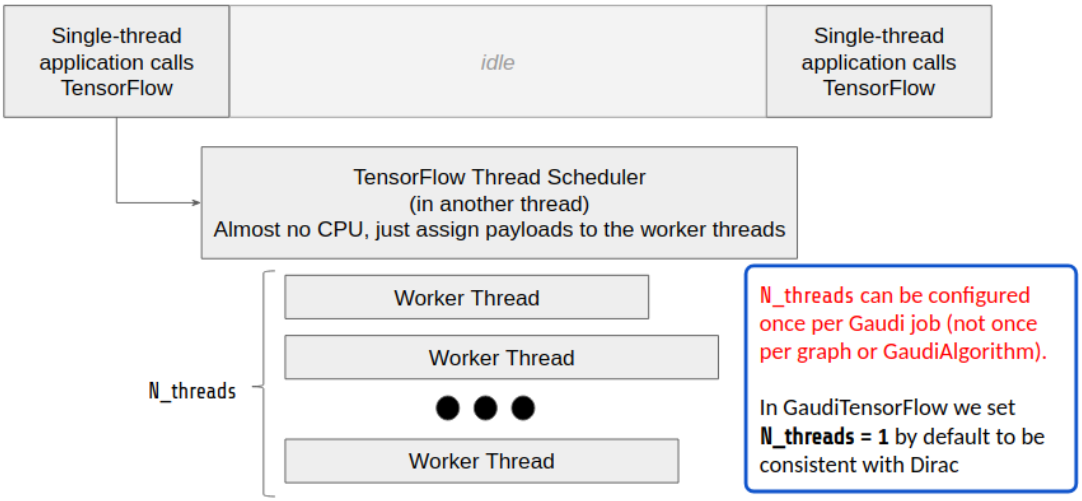


Figure 5.3: Simplified and schematic representation of the interplay between the GAUDI and TensorFlow scheduler when using `GaudiTensorFlow`.

This kind of frameworks builds computational graphs designed to be evaluated efficiently *on batches* of variables saturating the fast-access RAM memory. However, the *event-grained* approach forces them to run on tiny batches representing the information on the particles identified as potential signal in a single event, resulting into a computing resources waste even worst. While the *event-grained* approach is perfectly justified for large events, with hundreds to thousands of tracks and neutral objects, the overhead when processing the tiny events produced in ultra-fast simulation is unacceptably large. Indeed, ultra-fast simulation parameterize or model the effect of the detector occupancy as a direct deterioration of the reconstructed physical quantities, making the simulation of the particle which are not part of the signal unnecessary.
The events obtained through the fast simulation contain therefore up to some ten tracks and particles, pushing the framework towards a limit it was not designed for. The overhead become even more important when relying on `GaudiTensorFlow` to cross the border between GAUDI and TensorFlow. Indeed, the two frameworks exploit different thread schedulers for different purposes. The thread scheduling cannot be simply disabled because of the internal architecture of TensorFlow, which defines a manager thread and several working threads. Since the resource provisioning of the LHC Computing Grid expects each GAUDI job to use at most one core, the number of parallel working threads has been fixed to one. Still there are three different threads, one waiting for the other. A schematic representation of the multi-threading structure of `GaudiTensorFlow` is represented in Figure 5.3. Passing the control of the execution from one scheduler to another and from one thread to another introduces a small overhead of few milliseconds, which is perfectly acceptable to run a large neural network on a large number of tracks in a single event, but become an unacceptable bottleneck when evaluating a neural network on less than ten tracks. For the same reason, relying on hardware accelerator is very difficult within GAUDI: the overhead introduced in the communication with the GPU or the FPGA is dominating over the time spent by the accelerator to perform the computation.

Similar considerations apply to the very first steps of the software trigger decision, where the algorithms are sufficiently simple that they can run on hardware accelerators

(GPUs or FPGA) but not within an *event-grained* framework. Indeed, using the latter would result in the overhead of the communication between the CPU and the accelerator much larger than the effective computing time.

Despite the difference between the two types of overhead described so far, both of them are due to the *event-grained* strategy adopted for the reconstruction process, as well as for the trigger algorithms and the simulation production. Therefore, a solution to overcome these limitations is offered by the migration from the single-event paradigm to *batch-grained* frameworks, allowing to process sets of many events in one go.
An approach following this batch-based paradigm for the first stage of the LHCb trigger has been proposed in Ref. [67]. The framework, named Allen, has been designed to combine a C++ infrastructure consistent with Gaudi with a *batch-grained* CUDA framework [68] processing hundreds of events in one call to a device function. This GPU-based implementation is able to process the 40 Tbit/s data rate of the upgraded LHCb detector and to cover the majority of the physics program of interest, using an analogous reconstruction and selection sequence as in Run 2.
For the offline analysis, where the analysts wish to operate within the wide and powerful Python ecosystem (possibly involving the usage of hardware accelerators) we propose *mambah*, a manager of memory chunks represented within Pandas dictionaries [69] with a complete set of functions to access the particles and the vertices composing the decay trees in the event.

In this thesis I contributed to develop a new framework in pure Python to perform data processing and in particular ultra-fast simulation with a *batch-grained parallelism* approach. The new framework, named `mambah`, shares the parameterizations used in Lamarr, offering a tool for a *quick* development-cycle of new parameterizations that can be translated for Lamarr and officially released for future reference. Furthermore, `mambah` will provide a tool for theorists to investigate the sensitivity of LHCb to new phenomena, and will allow to exploit hardware accelerators for the most expensive parts relying on the kernels developed for TensorFlow/CUDA. Moreover, differently from Delphes, `mambah` is designed to arrange the physics events implementing the same data structures used by Gaudi as discussed widely in the next section. This structural feature is the key component which allows the new framework to *emulate* Lamarr in a lighter and faster environment, thus enabling to develop new parameterizations which can then be easily shared between the two frameworks.

5.2 Mambah: a Memory Manager for Batch-Analyses

`mambah` is a generic framework designed to process particle-physics events in Python: the recursive acronym stands for *Mambah (properly) Arranges Memory for Batch-Analysis in High-energy physics*. In general terms, `mambah` is a composed of an `EventStore` which represents the events in relational databases, and a `MambahAlgorithm` class that allows to schedule operations on the `EventStore`. The implementation of the `EventStore` with several Pandas' dataframes allows to query the database per columns, caching large arrays of homogeneous data that can be processed fully exploiting the vectorization capabilities of the hardware across several events.
To make it clearer with an example, consider the computation of the transverse momentum of a track. In the LHCb-version of Gaudi it is computed by obtaining for each track the

x and y components of the momentum stored in the `Track` class, and then computing the squared sum. In the LHCb-Upgrade version of GAUDI, the x and y components of the momenta of all the tracks in a *single event* are obtained in one go from the memory, resulting in two homogeneous arrays on which the CPU can compute the squared sum with an efficiency increased by several factors with respect to the non-vectorized version. In `mambah`, the x and y components are obtained from the `EventStore` in one go for all the events *in one batch*, which can be several millions of tracks, and then processed with some fast C-backed operation to compute the squared sum. While there is no gain for such a simple computation, the approach allows to gain order of magnitudes in the CPU cost for evaluating neural networks or other algorithms described by TensorFlow (or competitors) computing graphs.

The `mambah` framework is designed in pure Python with an effort to avoid mandatory dependencies on packages that cannot be installed in virtual environments, while allowing integration with algorithms developed in C++ through cppyy or Cython bindings. In addition, to avoid directly dependencies on ROOT and other popular frameworks as GAUDI, the conversion of the `mambah` event structure to and from those frameworks is demanded to other applications, relying on the universal SQL data format to exchange the data. An example of such a converter application for the LHCb software libraries and more generally for GAUDI is the `Ficino` application.
The aim of `mambah` is to provide and manage user friendly data structures for High Energy Physics applications. Its objects are designed to exploit the batch-based optimization for the parallel computing offered by framework like Scikit-Learn, Keras, TensorFlow or PyTorch. In addition, the `mambah` package allows to apply advanced algorithms to physics objects, providing sequentially pipelines able to efficiently compute and store variables then used in subsequent algorithms.

5.2.1 Event Description and Low-Level Access

The `mambah` representation of physics events is handled by the `EventStore` class, which collects data per batches. Each batch provides information over events, particles and vertices in the form of *relational databases*, namely labeled tables whose content allows referring to each other. Technically, the relational database is implemented by the Pandas' `DataFrame` class [69], a data structure containing labeled axes (rows and columns) which provides several smart methods to access to data with fast C implementations.

The `mambah` batch can be defined as a container of Pandas dataframes, where each one has an index filled incrementally and a column reporting the batch identification: this is needed to ensure the unambiguity of the database rows even when the analysis needs to merge several batches into a unique dataset. To these basic attributes several other ones can be added in order to report the specific information of the various database.
The default setting of `mambah` manage the physics data dividing it into five major categories, each one described by as many dataframes, described in detail in the next section: `events`, `protoparts`, `vertices`, `mcParticles` and `mcVertices`. The number of dataframes used to describe the data could be expanded in the future to include additional objects that are too different from events, particles and vertices. However, it should be pointed out that a greater amount of dataframes would result into a greater *code duplication* within the `mambah` framework and its dependencies. On the contrary, adding more attributes to any dataframe is straightforward, as well as accessing and filling them.

Batch 1				
events	protoparts	vertices	mcParticles	mcVertices
_index	_index	_index	_index	_index
batch	batch	batch	batch	batch
nTracks	event	event	event	event
produced	prodVertex	mother	prodVertex	mother
...	decayVtx	daugh1	decayVtx	decayVtx
	truth	daugh2	reconstr	daugh2
	...	...	...	...
Batch 2				
events	protoparts	vertices	mcParticles	mcVertices
_index	_index	_index	_index	_index
batch	batch	batch	batch	batch
nTracks	event	event	event	event
produced	prodVertex	mother	prodVertex	mother
...	decayVertex	daugh1	decayVertex	daugh1
	truth	daugh2	reconstr	daugh2
	...	...	...	...

Table 5.1: `EventStore` example made of two batches.

A schematic representation of the `EventStore` is reported in Table 5.1. Then, we can better understand the meaning of relational database looking at this example: the `mcParticles` dataframe has a column named `prodVertex` holding the index of the vertex that has produced that particle and that is stored in the `vertices` dataframe. The meaning of the attributes reported in Table 5.1 is listed below:

- `event`, origin event index from `events` dataframe;
- `prodVertex`, production vertex index from `vertices` dataframe;
- `decayVertex`, decay vertex index from `vertices` dataframe;
- `truth`, true particle index from `mcParticles` dataframe;
- `reconstr`, reconstructed particle index from `protoparts` dataframe;
- `mother`, mother particle index from either `mcParticles` or `protoparts`;
- `daugh*`, daughter particle index from either `mcParticles` or `protoparts`.

It is therefore possible to retrieve information at various level, moving through the corresponding dataframe. This data structure enhance the homogeneity of data when they are represented in the fast-access cache memory of the CPU, allows to select only the relevant input information for hardware accelerated algorithms, reducing the overhead of transferring large memory chunks, while retaining the possibility of splitting the batch in single events when this is made necessary in the analysis workflow.

MCParticles, ProtoParticles and Particles

The `mambah`'s `EventStore` class allows three different representations for particles called MCParticles, ProtoParticles and Particles.

The *MCParticles* (also indicated as true particles) are the particles generated by the Monte Carlo simulation and stored in the `mcParticles` dataframe. Together with the relational content reported in Table 5.1, the database provides also particle information (such as mass, lifetime, spin) extracted directly from the PDG database [1], as well as kinematic parameters as obtained from the generator.

The simulated particles can be passed through some reconstruction algorithm capable to parameterize the detector response. The reconstructed objects are *ProtoParticles*, namely particle candidates whose tracks information is corrected by efficiency and resolution effects with respect to what reported in the `mcParticles` dataframe. The ProtoParticles are stored in the `protoparts` dataframe.

One or more ProtoParticles can be interpreted as different particle hypotheses to compute, for example, the invariant mass of the mother particle of a specific decay channel. Considering the reconstructed decays $D \to K\pi$ and $D \to \pi\pi$, an entry extracted from the `protoparts` dataframe can be used as first child for both of them. Then the kaon and pion appearing as first child share the same ProtoParticle, but they are two different *Particles*. Similarly, the two reconstructed D are two different particles.

The Particle Alias System

The `EventStore` structure, based on Pandas dataframes, allows to select any particle by index. Nevertheless, it is often useful to identify groups of particles by name: for example, one may need to identify all the pions from a D decay. To this end, `mambah` provides `EventStore` with the `makeAlias` method aimed to group together particles under a unique *alias*.
Similarly, when using `mambah` for simulation, it may be necessary to modify some properties for a set of particles, such as the mass or the lifetime, before simulating the decay of those or to those particles. For this reason, the `makeAlias` function also enables to override some of the particle properties with customized, user-defined values or ranges. This is done returning a `ParticleAlias` object, class inherited from `scikit-hep/Particle` [70] which provides an effective interface to the PDG database [1].

The `ParticleAlias` class is therefore able to inspect the nature of the particle (such as its spin and internal quantum numbers) directly from the PDG, while the user can customize the mass and lifetime setting a single value or as a ranges within which values are randomized at run-time. The randomization is intended to *mask* some parameter whose exact value is unknown to the analyst, such as the mass or the lifetime of a never-observed particle. If the lifetime is sufficiently short, the particle is considered a *resonance* and its mass is randomized following a relativistic Breit-Wigner distribution. In case the mass of the particle is defined within a range, then the mass is generated according to distribution obtained convolving the Breit-Wigner with a flat distribution in a user-specified range.

After having created an alias, this is shared across all batches and are stored in a table with four columns:

- a unique, automatically incremented index;
- a new name for the alias;
- a `ParticleAlias` object;
- a numerical hash of the new name that fills an `id` column of the `mcParticles` dataframe.

5.2.2 Database Management and Variables Access

`mambah` is a memory manager intended for High Energy Physics consumption. It is represented by Pandas dictionaries which allow to select, modify and create data portion exploiting the `DataFrame` methods. However, pursuing this strategy would impose non-trivial Pandas expertise, discouraging a widespread package adoption. To this end, `mambah` provides user with several functions and routines aimed to manage easily the databases and to access the variables.

Low-Level Access to Variables

Variables within the batch can be accessed directly using the `DataFrame` methods. However, handling selections and join operation between dataframes to reconstruct the graph of events, particles and vertices requires some non-trivial Pandas expertise. To ease these operations, `mambah` objects are usually accessed through the `ObjectHandler` class, a thin layer keeping track of the database it refers to (the type of object), and an array of indices representing a certain set of objects in that database.

`ObjectHandler` can be obtained through the `EventStore` by selecting lines according to their index, through a boolean selection mask, exploiting the Pandas query function, or, in case of particles, selecting elements by their name. Furthermore, this thin layer allows to create and initialize new instances, retrieving the indices of the interested dataframe and appending sequentially the additional data.
`ObjectHandler` objects are particularly useful to follow the graph of a decay tree through particles and vertices. This is achieved using different helper functions on the kind of `get<Some>Handler` which, taking as input a database and an attribute, allows to select the corresponding rows in the dataframe identified by `<Some>`.

Therefore, the `ObjectHandler` class and its methods provide a set of user-friendly tools to select and modify portion of specific dataframe within the batch, as well as to create and initialize new instances. In addition, the class allows to retrieve information from different sources, either particles or vertices, performing join operations at low-level.

High-Level Access with Functors

Low-level access to variables is powerful, but sometimes it is useful to define simple relations or requirements that can behave as functions without writing a dedicated tool to hold the low-level processing of those variables. These *composable* relations between

variables of objects (such as the particle momentum given its components) are named *Functors*, and are described in the following.

A Functor is a special object that behaves like a function when called on an `ObjectHandler`, but that can be combined with other Functors *before* it is evaluated. For example, applying the Functor `PZ` to a `ObjectHandler` of particles, it returns an array with the longitudinal momenta of those particles. Evaluating instead the combination of Functors $\sqrt{\texttt{PZ}^2 + \texttt{PT}^2}$ with `PT` transverse momentum on the same `ObjectHandler`, one can directly obtain an array containing the particles momentum[2]. Functors provide therefore a highly customizable access to the variables stored in the dataframes.
Another interesting property of Functors is that they can be used to effectively translate strings, imported for instance from configuration files, into selection criteria or parameterizations for some feature of the detector. For example, the string `''(PX**2 + PY**2)**0.5 > 500''` can be converted in a valid Python function through the built-in `eval` function of Python. In some sense, it is the string itself that can be evaluated on an `ObjectHandler` to return an array of boolean values, one per particle, indicating whether the particle has passed the selection criterion or not.

The data extraction from `mambah` objects is made even easier providing a special class of Functor, named *Adapter Functor*, that allows to jump from a dataframe to another within the same batch. For example, to extract the position of the decay vertex of a set of particles, one needs to jump from the database associated to the `ObjectHandler` of particles to the one of vertices, and then executes some vertex Functor in order to obtain the x, y and z coordinates of the position. This can be easily done using Adapter Functors that hide the jump operation. In the case just described, one can apply the `DV()` Functor to an `ObjectHandler` of particles. The `DV()` Functor reads the `decayVertex` column of the particle database and use it as the index of a new `ObjectHandler` pointing to the vertex database. The newly created `ObjectHandler` (database of vertices) can be then used to evaluate the vertex Functors of interest: `VX`, `VY` and `VZ` in this case[3].

`mambah` provides therefore a set of useful functions and routines, which allow the user to extract and combine variables in order to easily prepare datasets for advanced studies.

Database Cleaning Function

The `mcParticles` and `mcVertices` dataframes describe the decay trees generated by the Monte Carlo simulation. In order to avoid reconstructing events which are partially outside of the detector acceptance and which have therefore no chance to appear in the final dataset, it is customary to introduce loose acceptance requirements on the simulated particles. Rejecting particles that are outside of the acceptance may result into events with *broken links* between particles and vertices.
To this end, `mambah` allows to define special filters that remove from the databases not interesting elements. However, it can happen that the rows eliminated were related to various other contents, resulting in a set of dead links. For example, such row can represent a root mother like D^{*+} in the decay $D^{*+} \to (D^0 \to K^-\pi^+)\pi^+$, as well as a deeper leaf particle as it is the case of the kaon. Either way, removing any row causes a lack of information

[2]Speaking in a Pythonic way, the object `[(PZ**2 + PT**2)**0.5] (parts)`, where `parts` is an `ObjectHandler` of particles, represents the momentum array of `parts`.

[3]Speaking in a Pythonic way, the object `[DV (VX)] (parts)`, where `parts` is an `ObjectHandler` of particles, represents an array containing the x-coordinate of the decay vertex of `parts`.

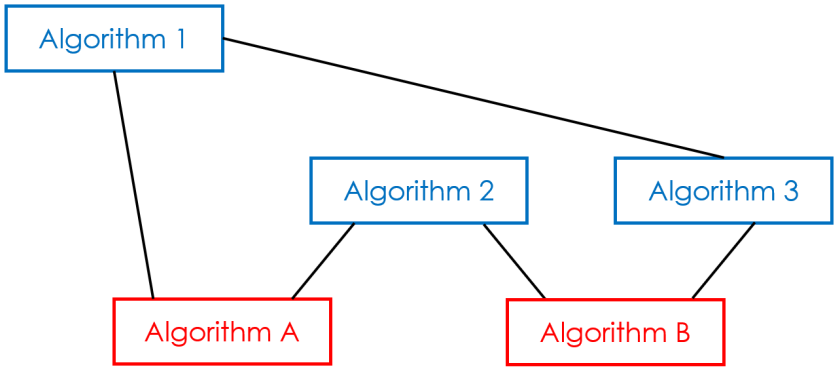


Figure 5.4: Example of dependency graph.

within the entire decay tree, and one may want to clean up all the dataframes from elements belonging to this corrupted event. This can be done using the `cleanDeadMcLinks` method within a batch of `EventStore`.

The first step performed by the `cleanDeadMcLinks` function consist of listing the indices of events with at least one dead link. Obtained this list, it is also possible to derive the indices of particles and vertices whose event is corrupted. Then, the methods offered by `DataFrame` allow to use these three sets of indices to clean up correctly the corresponding databases.

5.2.3 Algorithms in `mambah`

The `mambah` package is designed to support that the data processing elements can be connected in series, with the output of one element that becomes the input of the next one. A similar computing strategy goes by the name of *data pipeline*, which can be seen as a sequence of algorithms. `mambah` provides user with an interface to create algorithms and integrate them easily within more completed execution sequences (or pipelines): it is the `MambahAlgorithm` class.

`MambahAlgorithm` objects are declared with a name, whose uniqueness is verified by the framework by design. This is done to ensure an unambiguous identification of the algorithm once more than one algorithm is collected within a pipeline. Sometimes it may be useful to specify that an algorithm needs the outputs of another algorithm in order to produce its result, i.e. it exists a *dependency* between the two algorithms. As long as the sequencing of algorithms is single-threaded, writing all the algorithm within a single pipeline is the best option. However, in some circumstances, it is not practical to define a pipeline, such as when providing a library of algorithms that can be used independently but share some input.
Let's consider, for instance, the example reported in Figure 5.4. We want to provide a library containing two algorithms that can run independently, named Algorithm A and Algorithm B. Now, Algorithm A needs the output produced by two other algorithms the client of our library will not care about, namely Algorithm 1 and Algorithm 2. Algorithm B will also use the output produced by Algorithm 2, plus some output produced by Algorithm 3, that uses as input the output of Algorithm 1 in turn. Then, two possible strategies exist:

- Define the two following pipeline: [1|2|A] and [1|2|3|B]. If the two pipelines are joint together in a single pipeline, the `mambah` framework avoid running Algorithm 1 and 2 twice, and the second time it just skip it;
- Declare the dependencies of each algorithm and let `mambah` to build the dependency graph.

The two strategies are both perfectly legit. The former is probably easier to debug, while the latter is easier to maintain when the dependency graphs get complicated. `mambah` provides user with high-level methods to implement easily both the strategies.

Tools

In `mambah`, tools are lightweight algorithms that do not require initialization. Tools are often useful to add simple functionalities to `mambah`, typically adding variables or attributes to the databases. Different families of tools are designed to take as an input an `ObjectHandler` pointing to different databases. In general, a `mambah` tool can be seen as a transformer function that takes as input a database to update its variables or to fill new columns. In this sense, taking as input MCParticles one can design a `mambah` tool to easily implement a filter function that selects `mcParticles` elements within the experimental geometrical acceptance, as the one mentioned above.

Pipelines

Together with `MambahAlgorithm`, the `mambah` framework provides also the `Pipeline` class, a special algorithm obtained running a sequence of algorithms that run one after the other. Since pipelines are algorithms themselves, pipelines of pipelines are admitted. To allow a dynamic and easy manipulation of the sequence of algorithms, the `Pipeline` object includes two lists:

- a `list_of_algorithms`, which can be accessed by the client application to the `Pipeline` member functions to configure the algorithms to run;
- a `_compiled_sequence`, which is a private sequence of algorithms that is built on-demand when the pipeline is executed.

Any modification of the `list_of_algorithms` triggers a reset of the `_compiled_sequence` that will be rebuilt at the next attempt to run the pipeline.

Furthermore, pipelines are useful to define conceptual blocks of algorithms that can be dispatched (for example committed to PyPI) and imported into a Python module as a single instance. This is possible configuring the pipelines in the form of YAML-database of variables as allowed by the `ConfigurablePipeline` class. The YAML data format provides a flexible and human-readable format for nested configuration files, as shown in Table 5.2. A similar data structure can be used effectively to export or load trained neural network models, allowing `mambah` to implement advanced algorithms in form of `Pipeline` objects.

```
Condition1:
  Condition2:
    Condition3:
      Feature1:  Value1
      Feature2:  Value2
      ...
    Condition4:
      Feature1:  Value3
      Feature2:  Value4
      SpecializedCondition4:
           Feature2:  SpecializedValue4

    Feature4:  Value5
```

Table 5.2: Schematic representation of YAML data format. The conditions (`Condition*`) collect set of configuration slots (`Feature*`) accessed by a search in cascade, where values closer to leaves overwrite the ones closer to the root.

5.3 The `mambah.sim` package

Data organization, database management and algorithm configuration make `mambah` the perfect starting point to build an efficient simulation framework named `mambah.sim`. The `mambah.sim` module provides a set of useful classes and tools to simulate the particle decays together with the detector responses.

The module is separated in various packages that focus on different aspects of the simulation. First of all, there is the *generation phase* that produces particles according to physics models for the collisions and for the decay. It is followed by the *reconstruction process* which is intended as the application of efficiency and resolution effects to the contents of Monte Carlo databases (`mcParticles` and `mcVertices`), thus producing reconstructed objects stored in the corresponding databases (`protoparts` and `vertices`). Lastly, there is the reconstruction of the *detector responses* that includes the simulation of variable specific to some detectors, such as Particle Identification variables.
In the following we will describe each of the previous packages, looking a little more closely to the LHCb case.

5.3.1 Generation Phase

The first step in any simulation system consist of generating particles according to some physical principle. Then, the particles produced are stored into a database of *true* variables, which has to be independent of any reconstruction sequence that may follow. As said above, in `mambah` the information of the generated particles is collected by two databases, `mcParticles` and `mcVertices`, which remain unchanged after the reconstruction process, but are modified by specific transformation (such as geometrical acceptance filter).

The generation phase is traditionally divided into two steps: the simulation of some process producing particles that includes heavy and resonant states, never directly de-

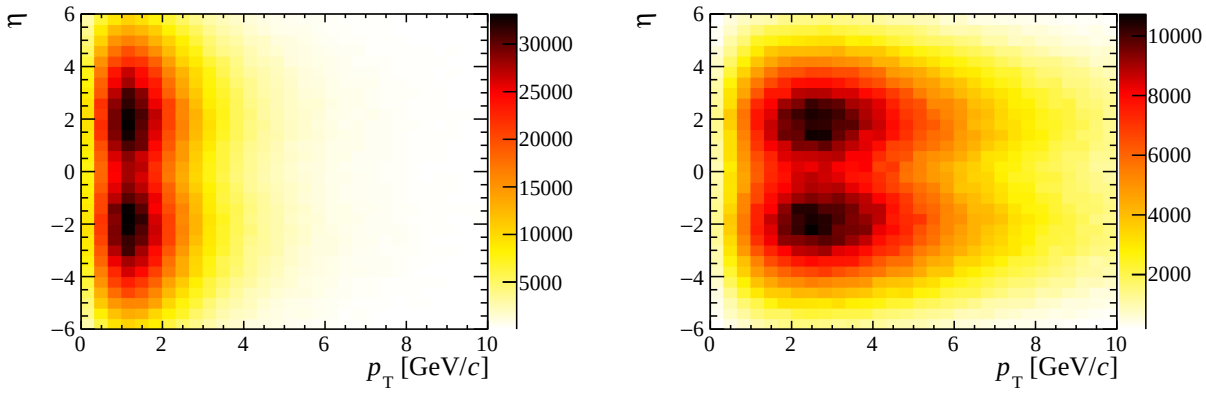


Figure 5.5: Distribution in pseudorapidity and transverse momentum of a generic charm (left) or beauty (right) hadron. Plots reproduced from Ref. [30].

tectable, followed by the simulation step in which the decay chain of these objects is reproduced. In `mambah.sim`, maybe improperly, the first step is named *Generator* and is delegated to the homonymous module `mambah.sim.generation.Generator`. The second step is named *MCDecay* (with `MCDecay` the related module) and `mambah.sim` implements two models for its description: one based on the `zfit/phasespace` package [71] and another one based on EvtGen [23].

Generator

The Generator provides a simple interface for the generation phase. The module should be implemented in derived classes reading the configuration from the constructor and implementing the member function. To date, the only physical model implemented is the so-called *particle-gun*, which consist of producing a single particle per event according to predefined kinematic distributions. In the current implementation, the `ParticleGun` generator is designed for heavy flavour hadrons (beauty and charm) whose production is theoretically predicted with the fixed-order next-to-leading-logarithm (FONLL) model[4] [72]. The theoretical prediction is represented as histograms of the predicted transverse momentum p_T and pseudorapidity η of a mixture of beauty and charmed hadrons. Then, all b- and c-hadrons are supposed to be produced according to these sample distributions. The parameterization strategy is the same used by RapidSim [30], and the corresponding kinematic distributions are depicted in Figure 5.5.

In the next future, the generation phase will be extended to allow generating events the Pythia program [21, 22] exploiting its interface with NumPy `numpythia`, and to import simulated events from any generator compatible with the `HepMC` format.

Decay Tools

Decay tools are used to described the decay of the generated particles. The base class `MCDecay` provides the common interface to register and configure the decay modes. Derived classes makes then available algorithms that interpret the registered modes to randomly choose a decay channel and to define the kinematics of the daughter particles.

[4] A full description of this topic is beyond the scope of this thesis.

Decay can be described through the `zfit/phasespace` [71], a Python implementation based on TensorFlow 2 depending only on packages that, up to now, can be installed through `pip`[5]. This represents the most portable solution, but lacks of the dynamic contribution to the description of the decay, leading to distributions of the kinematic variables of the daughter particles which are often unphysical. On the other hand, `zfit/phasespace` is lightning fast and provides native support for GPU computing.

`mambah.sim` provides a second decay tool to describe the decay chain of generated particles and is based on EvtGen [23], a celebrated package to simulate the physics of the heavy hadron decays. It has been binded to Python through the `evtgen-python` package using Cling, an interactive C++ interpreter for Root [73]. EvtGen itself depends on several external libraries (Pythia, Taula and Photos) and this complex dependency tree of Python, C and C++ code has not yet been translated into `pip` packages, and it is not clear this is feasible. Hence, in order to empower `mambah.sim` with the outstanding description of the decay mechanisms provided by EvtGen, the installation of this package is necessary. However, it should be pointed out that this Python interface problem makes hard exploiting opportunistic computing resources and public services such as Colaboratory, when using EvtGen.

5.3.2 Reconstruction Process

The particles produced at the generator-level are stored within the Monte Carlo databases together with all the kinematic information. Since the detectors cannot *see* directly heavy states or resonances, the reconstruction process will involve only long-lived particles. Hence, only the portion of `mcParticles` and `mcVertices` which contains these particle species are passed through the sequence of algorithms that describes the reconstruction process and that outputs the objects successively stored within `protoparts` and `vertices`. In addition, it should be noticed that even if the input objects of reconstruction algorithms are *true* particles, they are treated as *tracks* in order to simulate the detector responses. Tracks may be reconstructed or not, so we are interested to derive a boolean selection mask that describes this behaviour: it is called *efficiency corrections*. Simultaneously, the reconstructed tracks are affected by *resolution effects* which will have to be considered before storing the ProtoParticles.

Efficiency model

The efficiency corrections can be performed using a `mambah` tool that extracts the candidates of interest from `mcParticles` (or `mcVertices`) through the `ObjectHandler` methods (see Section 5.2.2). Considering `mambah.sim` for LHCb application, the boolean selection mask necessary to consider the efficiency contribution is parameterized through a neural network implemented in TensorFlow and previously trained. In this particular case, modelling the tracking efficiency as a function of the momentum components and the decay vertex coordinates, the neural network was trained as a simple classification problem over a labeled dataset obtained from Run 1 full simulation[6]. Importing the neural network in

[5] `pip` is a *package manager* for Python packages.

[6] The simulated samples considered are the same that allow to parameterize the efficiency corrections in Lamarr.

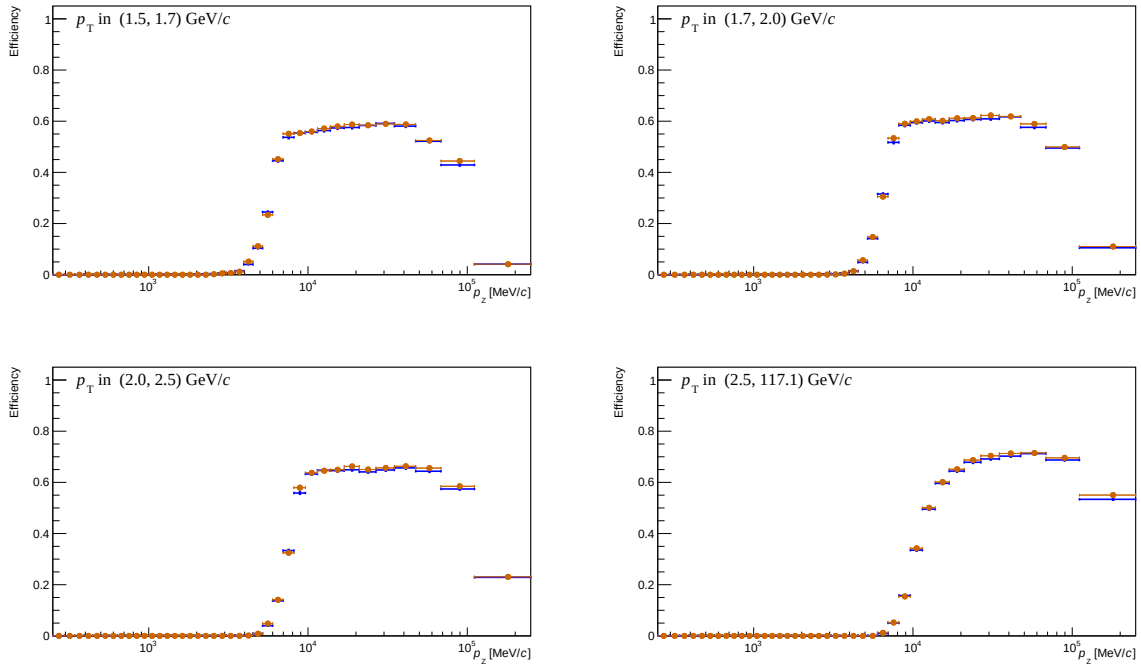


Figure 5.6: Comparison of the efficiency modeled with a deep neural network (represented with orange markers) superposed to the efficiency obtained from full simulation (represented with blue markers). The efficiency is defined here as the ratio between the number of fully reconstructed tracks (from the VELO to the calorimeter) to the number of charged particles generated. The agreement of the projection is exceptionally good.

YAML-format, `mambah` allows to rebuild the computational graph and to treat it as an algorithm that can be added in sequence to other algorithms. Hence, taking the necessary features from the Monte Carlo databases, `mambah.sim` is able to compute the boolean selection mask and to fill the reconstructed databases according to the results.

The efficiency is modeled as a function of the three components of the momentum of each track and the position of its origin vertex (or the point closest to the beam axis when the origin vertex is not accessible in the full simulation). A 5-layer deep neural network is trained on the fully simulated sample, minimizing a Bernoulli cross-entropy to evaluate the probability that a simulated particle is matched with a reconstructed particle. This approach to the efficiency parameterization has been recently discussed in Ref. [74].
Figure 5.6 presents the comparison between the efficiency modeled with the neural network and simulated with full simulation four bins of transverse momentum, as a function of the longitudinal momentum.

In Lamarr, to avoid evaluating an additional neural network, the same function is approximated by means of a multidimensional lookup-table.

Resolution Effects

The particle candidates that survive the efficiency selection are passed through another `mambah` tool which performs the smearing of the track momentum components. Building a parameterization for the smearing is simplified by the `mambah` high-level functions which

allow to access easily to the variables within a database. In this case, one can directly obtain the momentum components using the Functors `PX`, `PY` and `PZ`.

The resolution transformer of the `mambah.sim` module parameterizes the smearing function with a neural network built and trained with TensorFlow for LHCb application. Again, the neural network was trained over a dataset obtained from Run 1 full simulation, but this time the loss function was defined to compute a fit regression problem, in which the inputs were the true momentum components and the outputs were the smearing parameters. Retrieved the computational graph, `mambah.sim` allows to update the `protoparts` elements with the resulting variables.
To date, the other tracking variables related to resolution effects, such as the smearing of the decay vertex coordinates or the products of track fitting, are not yet implemented as computational graph. Instead, similarly to what done for Lamarr, they are parameterized extracting the distribution directly from data histograms.

5.3.3 Particle Identification

At the end of the reconstruction process, the `protoparts` and `vertices` databases are able to provide track kinematics parameters as *expected* from the tracking system. In addition, `mambah.sim` allows to parameterize the event multiplicity `nTracks` extracting a random number from the corresponding distribution for the decay $K_s^0 \to \pi^+\pi^-$, following what done in Lamarr. The kinematic parameters and the number of tracks per event are therefore accessible information to `mambah.sim` that can use them to evaluate the generative models introduced in Chapter 4. Exported as YAML-format objects, the various PID models can be effectively concatenated using the powerful methods available to the `mambah`'s `Pipeline` class. Such configuration allows that the output of a neural network can fed the input of another one, following a logic similar to the one shown in Figure 5.4.

To ensure a self-consistent system, `mambah.sim` has to provide the trained models with all the parameters belonging to the conditional space reported in Table 4.1. It is therefore necessary to build a further parameterization for the `isMuon` response. Latter can be described as a function of the momentum p, the pseudorapidity η and the detector occupancy `nTracks`, whose contribution can be effectively parameterized as a neural network. The structure of the computational graph is very similar to the one described for the tracking efficiency. However, this time, the output of the algorithm is directly injected to the input of another one.

Imagining to follow the stream schematically described in Table 4.1, the triplet $(p, \eta, \texttt{nTracks})$ can be used to derive directly the RICH detector responses and, adding the `isMuon` check, also to compute the MUON system responses. After having evaluated these two independent graphs, the outputs from both the systems feed the two remaining neural networks which, again, can be computed independently, allowing to produce huge samples efficiently. The `mambah.sim` framework is therefore able to provide track kinematics information and high-level PID variables, that together can be used by other analysis software (such as Bender) to convert a track into a particle candidate.

5.3.4 Conversion to the LHCb dataset format

The data structure at the base of `mambah` and its capacity to describe algorithms in terms of pipelines make `mambah.sim` a powerful simulation framework predisposed to exploit

hardware acceleration devices.

Sometime one can be interested in focusing on a particular step of the pipeline and to do this, it is useful to store the `EventStore` at a given step and reload it to test the behaviour of the last step that can then be pipelined to the main program. This kind of persistence rely on Python serialization through the `pickle` package and is pretty immediate. The main limitation of this approach is that the data can be hardly read from C++ program, as for example an official LHC application based on GAUDI. For this reason, the `mambah`'s `EventStore` class supports a SQLite3 serialization of the dataframes. SQLite3 is a pretty consolidated library with C/C++ APIs that can be used to read the SQLite database from any C/C++ application, including GAUDI-based applications. This data structure can be then used to build a GAUDI-based converter which allows to export the content of `mambah` databases in a format readable for the official LHC applications. Such an application, still private to the LHCb Collaboration, is named `Ficino`.

In the future, we expect `mambah` to make available several of the combination algorithms that are today only available as part of the LHCb software stack. This will allow an even faster validation of the new parameterizations, simply comparing the output of some `mambah` algorithm to the LHCb data, without relying on GAUDI and ROOT at any step of the simulation.

5.4 Framework Validation

To validate the simulation framework, a prototype of `mambah.sim` has been implemented and we have focused on the same decay channel selected for Lamarr: $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to pK^-\pi^+$. Also in this case, looking at decay modes different from the ones used to compute the parameterization represents a powerful validity test to verify the framework robustness. Then, after having evaluated entirely the pipeline from the generation phase to the output of the last neural network, the whole `EventStore` is exported as SQLite data format. The container of dataframes, converted by `Ficino` to be read by the GAUDI-based application, is finally transformed into `nTuples` by Bender.
The `nTuples` allows to access to the invariant mass of Λ_c^+ represented in Figure 5.11. A schematic representation of the sequence of steps followed in `mambah.sim`, in Lamarr, and during data taking of real physics events is presented in Figure 5.7. This representation clearly shows that the logical step described by Lamarr is split in two applications in the case of `mambah.sim`: the one devoted to compute the parameterizations is directly performed by `mambah.sim` itself, while the second one acts as simple adapter between intermediate data format (SQLite) and LHCb event model.

A comparison of the CPU-cost of the various step (with `mambah.sim` limited to *single-thread* operations) is presented in Figure 5.8, which shows that the time needed to simulate one event is less than the time requested to test the possible combinations of tracks, fit the secondary vertices and produce `nTuples` as output. This is a radical inversion on the CPU-cost hierarchy with respect to *full* simulation, where the CPU-cost is dominated by the first steps of the production to simulate the radiation-matter interactions.
The CPU cost of the *ultra-fast* simulation obtained with `mambah.sim` is dominated by the cost of the PID models, and in particular to the expensive pre-processing relying on a *quantile transformation*, and on EVTGEN, that is run on every single event before any acceptance or efficiency-related data reduction.

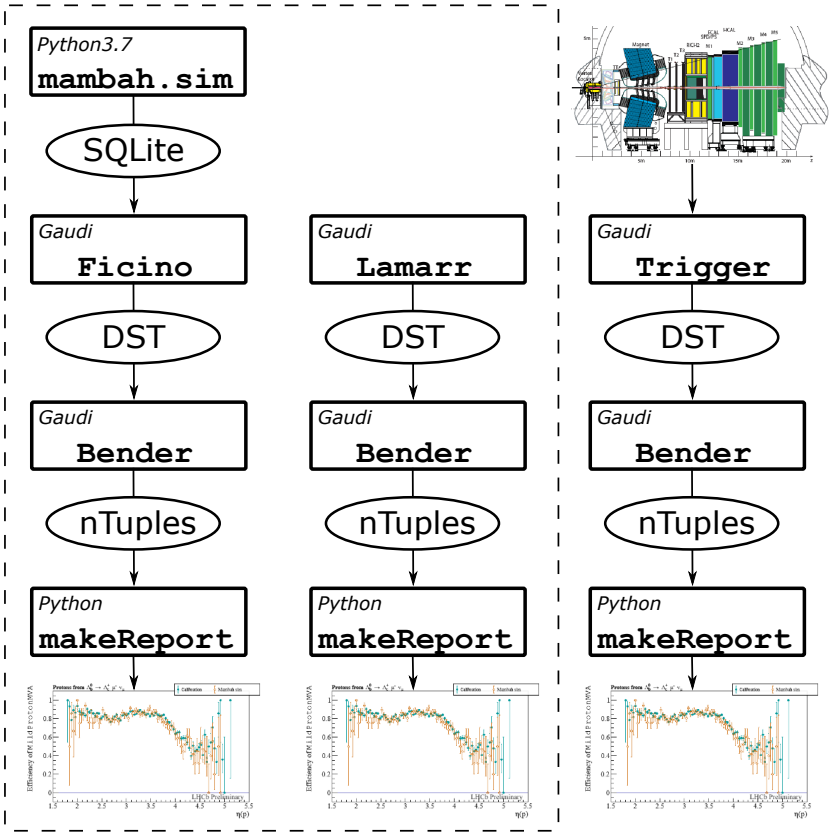


Figure 5.7: Schematic and simplified representation of the dataflows used to obtain the datasets compared in the validation. The left and middle flows represent simulation systems, the right one instead describes the data flow of real physics events.

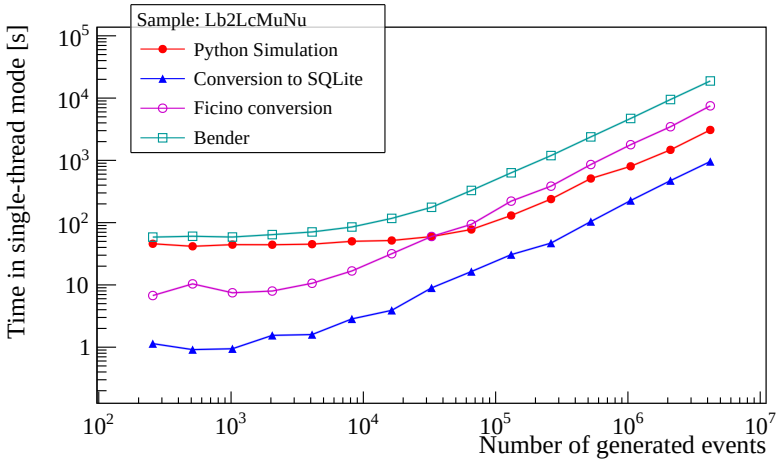


Figure 5.8: CPU time requested by each step of the data-flow when producing the sample of simulated Λ_b^0 decays described in text. The number of events on the x-axis is the number of *generated* events, before any acceptance or efficiency requirement.

In the following we consider several aspects of the simulation: the kinematics, the tracking and the particle identification, comparing the distributions or the efficiency produced by `mambah.sim` with the data collected through the calibration stream in 2016 [15] (with upward magnetic field).

5.4.1 Kinematics

As mentioned above, the kinematics of the Λ_b^0 daughters is defined by dynamic of its semileptonic decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}$, simulated in `mambah.sim` using the EvtGen implementation of the *BaryonPCR* model, named after the authors of Ref. [75]. On the other hand, the data sample to which the simulation is compared is obtained identifying $\Lambda_c^+ \mu^-$ pairs, with an invariant mass smaller than the Λ_b^0 mass and that come from the same vertex. Indeed, the chosen decay is a three-bodies process where the third particle, a neutrino $\bar{\nu}_\mu$, takes away some of the energy that cannot be exactly quantified. This results, within the data sample, in the impossibility to reject feed-down and partially reconstructed events with additional light particles produced in the Λ_b^0 decays, that are not reconstructed or simply ignored.

In Figure 5.9 we report the distribution of the momentum, the pseudorapidity and the transverse momentum of the proton produced in $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}$ decays followed by $\Lambda_c^+ \to p K^- \pi^+$. The agreement of the pseudorapidity distributions in data and simulation is good, while some mismatch at low transverse momentum is evident.
The disagreement could be due to a mis-modelling of the trigger selection efficiency or to a mis-modelling of the semileptonic form factors, but it reassuring that the disagreement is only slightly larger for samples simulated with `mambah.sim` than it is for the sample obtained with Lamarr.

5.4.2 Tracking

The tracking efficiency cannot be simply compared to real data for the obvious reason that the information of the non-reconstructed tracks is missing. Nonetheless, it is possible to compare the variables related to the quality of the track and to the uncertainty related to its parameters.

The criterion used in LHCb to determine whether a track is due to a random combination of hits in the tracker rather than to a real particle depositing energy in the detector, is defined by means of a neural network trained on simulated data, and its output is named *ghostProbability*. The parametrization of the *ghostProbability* in `mambah.sim` and Lamarr is based on its distribution in *full* simulation: as for the `ProbNN` variables, the neural network is not run, but its output is parametrized and predicted as a function of the kinematic variables of the input particles. The comparison between the distributions of *ghostProbability* in calibration data and in the *ultra-fast* simulation is presented in Figure 5.10.

The quality of the representation of the resolution of the momentum, as measured from the curvature of the reconstructed track in the magnetic field, can be assessed comparing the mass of the Λ_c^+ obtained combining the four-momenta of the three daughters: proton, kaon and pion. The comparison of the two histograms of the invariant mass as obtained in the calibration samples and in real data is presented in Figure 5.11. Note that the combinatorial background, not simulated, is only present in the calibration samples, but

the resolution function is *much better* reproduced than it is within the version of Lamarr discussed above.

Finally, we evaluate the quality of the uncertainty associated to the track parameters by comparing the compatibility of the track with the position of the primary vertex. Note that in the definition of this compatibility test, both the uncertainty on the position of the primary vertex and the covariance matrix of the track parameters are considered. The good agreement displayed in Figure 5.12 is therefore a first evidence of the good parameterization of these quantities. The covariance matrix of the track parameters, modeled on a statistical basis as a function of the *momentum only*, is however not sufficiently well reproduced in ultra-fast simulation to result in a good prediction of the Λ_c^+ vertex quality. The latter, measured as a reduced χ^2, is shown in Figure 5.13. The agreement between the two curves is much worse than it is for the detachment measure, which indicates that the parameterization of the covariance matrix, good on average, needs to reflect the dependence on the *slopes* of the tracks. A detailed study of the covariance parameterization will be an upcoming development of the *ultra-fast* simulation.

5.4.3 Particle Identification

The parameterization of the Particle Identification detectors, described with GANs as detailed in this thesis, has been validated comparing the distributions of the Combined Log-Likelihood of the proton hypothesis versus the pion and kaon hypotheses, as shown in Figure 5.14. The efficiency of selection criteria based on the `ProbNN` variables has also been

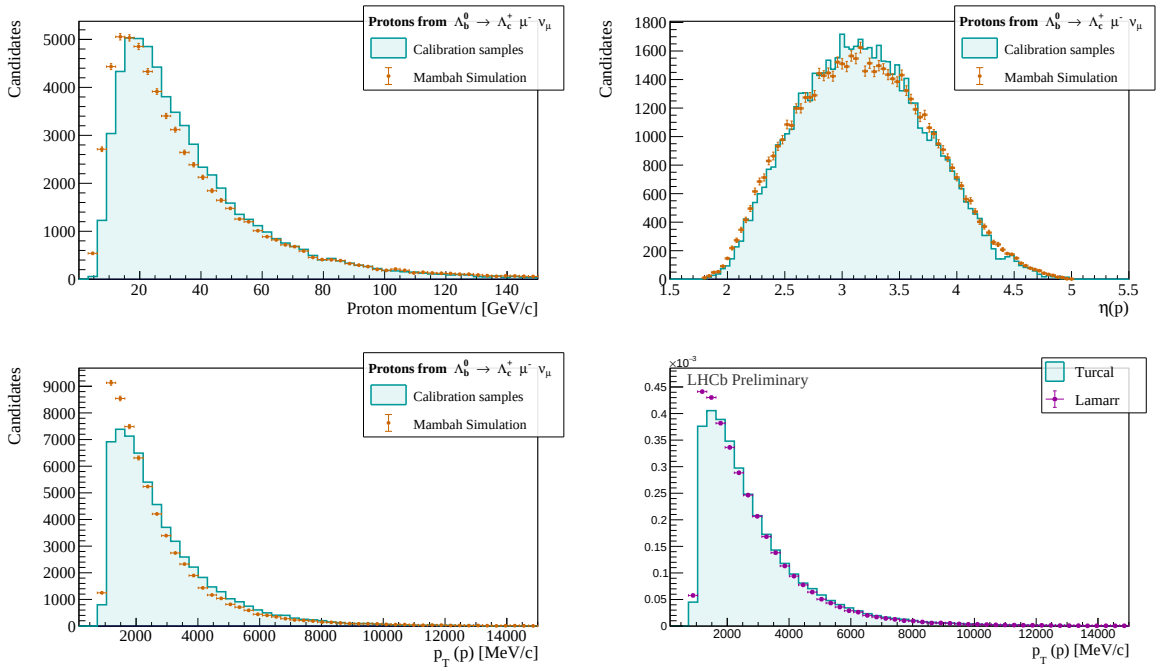


Figure 5.9: Kinematic variables of the proton produced in $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}$ decays followed by $\Lambda_c^+ \to pK^-\pi^+$. The momentum and the pseudorapidity distributions are shown on the first row, while the transverse momentum is reported in the second row, where the comparison between Lamarr and calibration data is also reported.

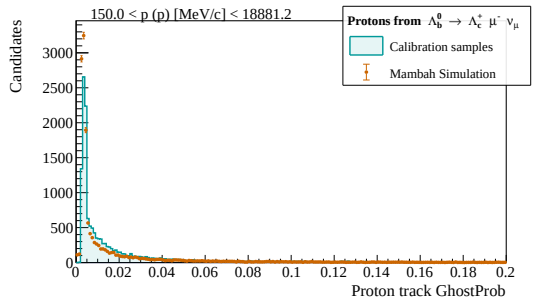


Figure 5.10: Ghost probability of the proton track as obtained from a sample of Λ_b^0 decays simulated with `mambah.sim` and from calibration data.

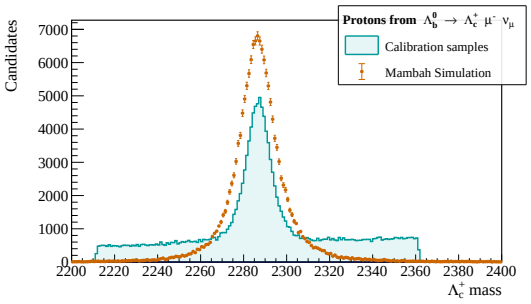


Figure 5.11: Invariant mass of the $\Lambda_c^+ \to pK^-\pi^+$ decay as obtained from ultra-fast simulation and with the calibration samples. The agreement of the resolution is astonishing, while the simulated sample does not include combinatorial background events.

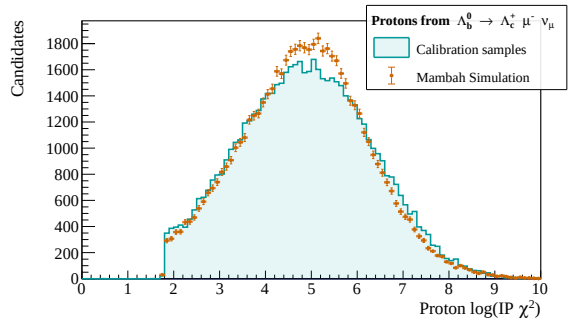


Figure 5.12: Comparison of the distribution of a measure of the compatibility of the proton track with the primary vertex in data and ultra-fast simulation. Protons were obtained from semileptonic Λ_b^0 decays.

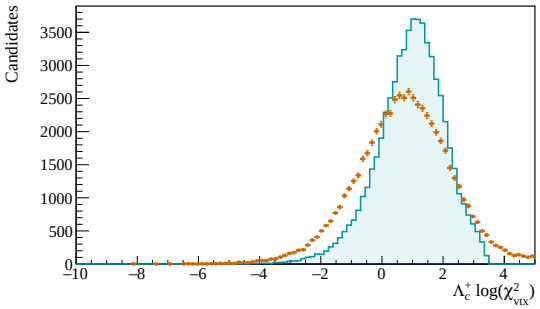


Figure 5.13: Vertex quality of the reconstructed Λ_c^+ particle in calibration data and in simulation. Protons were obtained from semileptonic Λ_b^0 decays.

compared in data and in the simulation samples, as reported in Figure 5.15: an acceptable level of agreement can be observed. It should be noticed once again that the training of the generator was performed on protons from Λ^0 decays, with a different kinematic coverage and a different production mechanism (resulting into a different average occupancy). The good agreement that we observe is therefore a *validation of the method* rather than a validation of the neural network, whose validation was discussed already in Section 4.

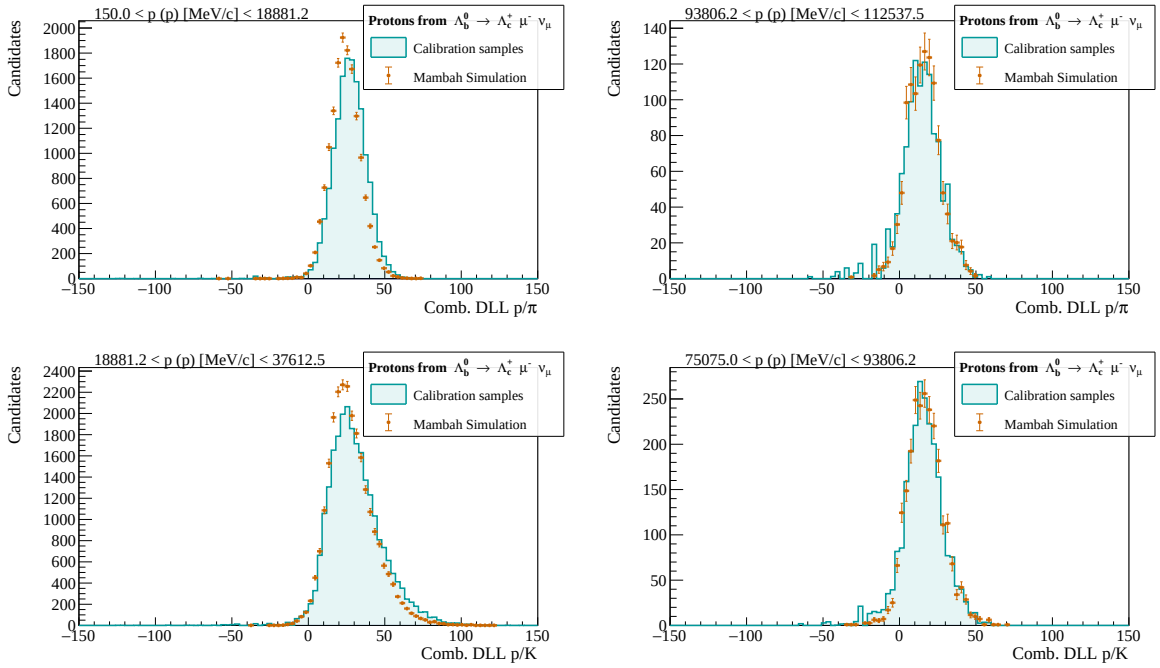


Figure 5.14: On top (bottom), the differential log-likelihood between the proton hypothesis and the pion (kaon) hypothesis, as simulated or measured for the proton candidates from semileptonic Λ_b^0 decays. Two well populated proton momentum bins were chosen randomly to highlight the good reproductions of the variation of the distribution at varying momentum.

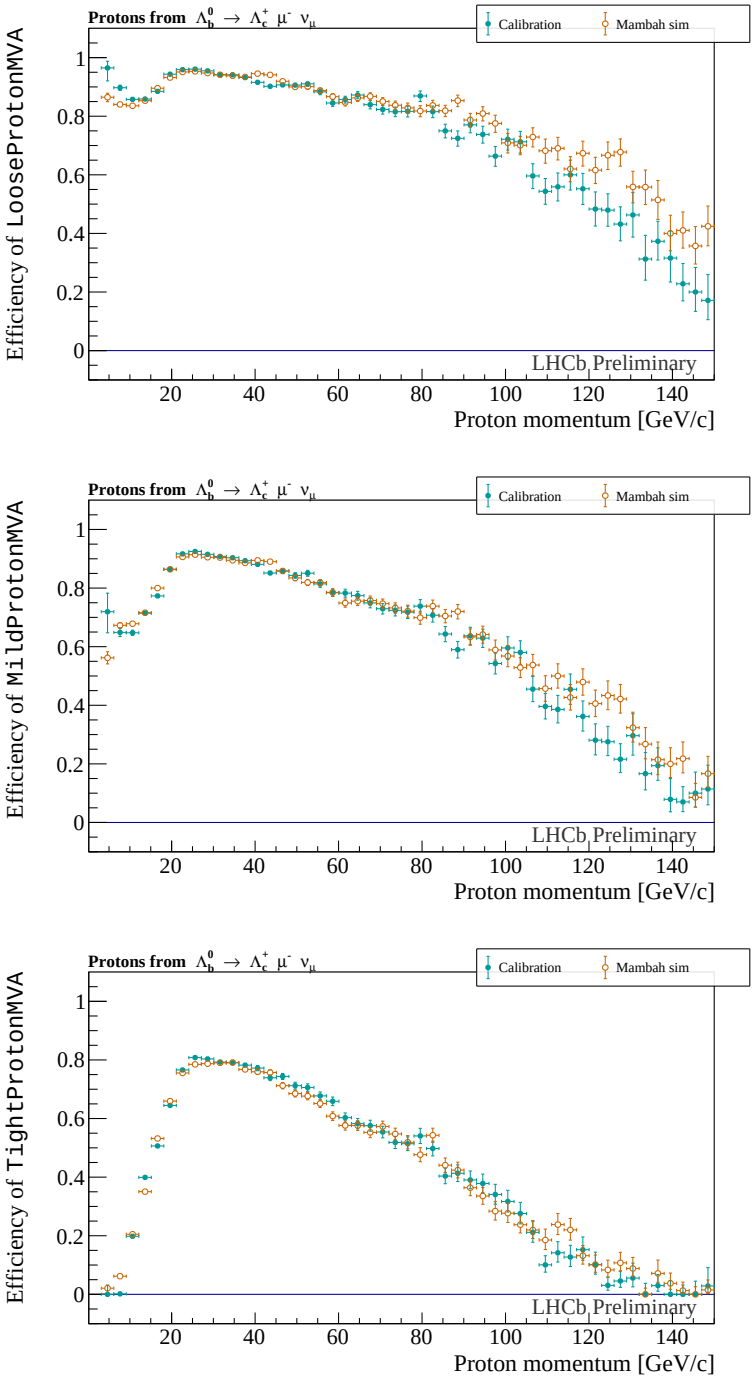


Figure 5.15: Performance plots of the `mambah.sim` Prototype reporting the efficiency of three different requirements on the output of a FNN trained to identify protons (namely `ProbNNp`) as evaluated on protons tagged through the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to pK^-\pi^+$. The efficiency is compared, in bins of the proton momentum, for a dataset selected without introducing bias on the particle identification of the proton (cyan shaded area), and an ultra-fast simulation sample where the PID variables are modeled through a Generative Adversarial Network trained using protons from $\Lambda^0 \to p\pi^-$ decays, only (orange markers).

Conclusion

Despite its clear success, the Standard Model is still far from being a complete theory. While there are hints of New Physics, none of them sharply points towards a specific extension of the theory. This results into the necessity of increasing the statistical power of the experiments, reaching unprecedented precision levels in order to transform those hints into evidences.

In order to achieve this ambitious goal, the LHCb experiment is and will be subject to upgrades to increase significantly the amount of data collected. Together with larger and larger samples, the upgraded version of the detector will see the progressive shift of the data processing workload towards the *online* resources. This implies a lower use of the computing resources by the offline reconstruction, in favor of the simulation production which becomes more and more dominant.
The availability of large data samples is not sufficient to reach the precision necessary for the physics analyses. Indeed, it is often crucial to be able to rely on simulated samples at least of the same size as the collected data. So far, the majority of simulated samples has been produced through the *full simulation* approach, which includes a complete simulation of the radiation-matter interactions within the detector. A similar strategy is not sustainable for the computing resources available, reason why *ultra-fast* simulation has been introduced.
Ultra-fast solutions avoid to simulate the interactions that occur within a detector, and provide instead parametric or non-parametric functions synthesizing the high-level responses of the detector. Among non-parametric solutions, methods based on Machine Learning algorithms and, in particular, based on Generative Adversarial Networks are being studied.

Generative Adversarial Networks (GAN) are a powerful class of generative models based on simultaneous training of two neural networks: a generative model that produces synthetic data given some noise source, and a discriminative model that distinguishes generator's outputs from true data. The training process between the two models can be seen as a competition, named *minimax game*, that achieves the optimal configuration when both discriminator and generator are not longer able to improve their performance exploiting the information of the counterpart: a similar condition is known as Nash equilibrium. Despite the basic idea is straightforward, obtaining good results or being able to converge in the training process needs some non-trivial expertise. To succeed in both of the challenges, it is necessary to modify the loss function in order to avoid the vanishing gradient problem. Loss functions with good properties are the Wasserstein and Cramér distances, with which it is possible to define the homonym GANs.

This thesis focused on the implementation of GAN models based on Cramér distance for the parameterization of the Particle Identification system of LHCb. The two neural

networks, generator and discriminator, were trained using the energy distance (multivariate generalization of Cramér distance) in order to reduce mode collapse phenomena and to ensure the existence of a non-zero gradient to drive the generator learning. To correctly map the latent space into the reference one through the transformation induced by the generator, the minimax game was reformulated in order to be able to produce probability distributions as a function of the track kinematics parameters (momentum and pseudorapidity) and of the detector occupancy (total number of tracks). The loss functions were therefore rewritten in a conditional form, and the two neural networks were modified to take into account variables from the conditional space.
During my thesis work I have proved the effectiveness of generative models trained directly over calibration samples in reproducing faithful distribution for the high-level variables of the Particle Identification system. In order to ensure that the generator synthesizes the target space, the training process was modified to statistically subtract the residual background of the calibration samples, applying weights as prescribed by the ${}_s\mathcal{P}lot$ technique.
Despite the loss function represents a distance measure between the generated distribution and the target one, its value cannot be used as an *unbiased* metric to assess the quality of the synthetic sample produced. To this end, I developed a scoring method based on the outputs of one of the most robust Machine Learning algorithm: a Boosted Decision Tree (BDT) classifier. At the end of any adversarial training, ten BDTs were trained over a mixed sample both of generated and reference elements in order to distinguish them. This third *independent* algorithm was therefore able to provide a score to assess the quality of generator outputs. Then, these scores were used to choose the best data preprocessing strategy, and to tune hyperparameters in order to obtain highly faithful distributions.

Adopting all these algorithms has allowed to reproduce distributions with various shapes, for different ranges of the kinematic parameters and of the detector occupancy, and to synthesize the distinct behaviour of four long-lived particles (muon, pion, proton and kaon) exploiting the corresponding information from the calibration samples. It was therefore possible to provide generative models to parameterize not only the responses of RICH detectors and MUON system, but also the distributions of combined Particle Identification variables. Each model was trained over a dataset obtained concatenating the reference and conditional spaces according to the specific subsystem described. A correct map of the latent space into a different reference space must take into account all the contributions to the final results of all effects out of the experimental control, leading to uncertainties (or fluctuations) on the target value. In this thesis I have shown that GAN model can effectively parametrize the high-level response of the Particle Identification system of LHCb including contributions from detectors which are not explicitly simulated.

The second important personal contribution of this thesis is related to the design and development of `mambah`, a *batch-grained* Python framework aimed to provide and manage user friendly data structure for High Energy Physics applications. The `mambah` package is built over an event manager (the `EventStore` class) implemented as a container of databases, and over well-defined tasks or algorithms (the `MambahAlgorithm` class) which allow to concatenate effectively different computational graphs and to include easily their respective dependencies. The inherent *batch-grained* structure of `mambah` makes it suitable to consume programs written with the most modern software solutions for parallel computing like TensorFlow or competitors, and to support efficiently the usage of hardware accelerators such as GPUs or FPGAs. Within the `mambah` project, I have partially

dealt with the implementation of database management functions, while my major contribution has concerned the integration of the PID models developed with the simulation framework based on `mambah` and named `mambah.sim`.

The `mambah.sim` module provides a set of classes and tools, developed within the powerful `mambah` framework, to simulate the particle decay together with the detector responses. The current state of the module allows to generate heavy hadrons only by the particle-gun approach, while their decays can be described according to the `zfit/phasespace`, an optimized package for phase space decays, or through EvtGen, a celebrated but slower package necessary to reproduce flavour physics contributions. Hadrons are propagated until obtaining long-lived particles, that are then kept within `mambah`'s databases. Its high-level access functions ensure to extract easily the information stored, that can be used as input of a well-defined pipeline to reproduce the detector response. This thesis, following the development of `mambah.sim`, was mainly focused on simulating the LHCb detector response, but it should be pointed out that the highly customizable features of `mambah` ensures to reproduce, in principle, any high energy physics experiment.
As said, the simulation framework describes the detector response with a chain of algorithms (the pipeline) that, taking the information from the databases, transform it or create new one in order to reproduce the high-level responses of the various subsystem of LHCb. To the Monte Carlo particle databases a filter transformation is applied to simulate the reconstruction efficiency of the tracker. The particles that survived the selection are subjected to a second transformation which degrades the kinematic information adding smearing effects and synthesizes the fitting track variables to measure the compatibility between reconstructed tracks and the corresponding primary vertex.
I have exploited the output datasets from these first two step to reproduce the likelihood functions computed by the RICH and MUON systems for the long-lived particles traversing them. This high-level response was obtained implementing the GAN models trained within the simulation framework, and feeding them with the kinematic parameters and the event multiplicity computed by the previous algorithms. Then, the likelihood functions resulting from the trained models fed a second layer of generative models in order to obtain also a parameterization for the combined Particle Identification variables used by LHCb, such as CombDLLs or ProbNNs. Technically, this implementation consists of a graph of computational graphs, where the single nodes are generators trained within a GAN system. A similar combination of algorithms is made trivial and effective thanks to the `mambah`'s pipelines, which can transparently offload them to hardware accelerators whenever they are available.

In order to validate this simulation framework, a large sample of Λ_b^0 was generated and decayed according to $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to pK^-\pi^+$. The decay mode selected represents a semileptonic process with a non-trivial dynamic, which needs to be describes through the EvtGen package. On the other hand, the decay channel includes all the particle species for which a PID parameterization exists, and it represents therefore the perfect laboratory to check the proper implementation of the developed models and to test the potential of the framework. Moreover, data samples containing these decays are collected by exclusive trigger lines with special care not to bias the detector response on implicit or explicit requirements over proton candidates identification. This allows also to test the simulation capabilities of `mambah.sim` with respect to the selection efficiency of protons acting on the PID variables.

The resulting databases, filled with kinematic parameters and PID information, were exported as SQLite data format and read by the LHCb applications thanks to a specific converter named `Ficino`. The database content was then transformed into `nTuples`, which allowed to access *particles* information after that LHCb applications had converted `mambah` tracks (or ProtoParticles) into particles, exploiting data provided. The distributions of the variables extracted from the `nTuples` were used to prove the validity of the framework described.

The preliminary studies have shown that `mambah.sim` is able to simulate particles with the proper kinematics. In particular, it was discussed the good consistency in reproducing momentum and pseudorapidity distributions for proton candidates. Furthermore, it was proved that the generative models developed are able to generalize effectively the behaviour of the studied particles, reproducing good-looking distributions for the PID variables of decay modes different from the training ones. In particular the agreement between the efficiencies of the Particle Identification criteria between data and simulation is an astonishing result that highlights the great potential of this approach to ultra-fast simulation and its implementation within the `mambah` framework.
There is still room for improvement, in particular we notice some evidence of disagreement in the low-p_T region which is relevant to a non-negligible amount of physics analyses within LHCb. Further developments will extend the impressive agreement achieved at large p_{T} to extend further the number of analyses that may profit from *ultra-fast* simulation, offering a viable alternative to *full* simulation for the upcoming and future upgrades of the LHCb experiment.

Acknowledgements

First of all I would like to express my deep gratitude to my master thesis advisor, Lucio Anderlini. His support has been fundamental not only to complete this thesis, but also to introduce me to the Machine Learning world and to drive my future career by valuable advice. I will always be in debt with him.
I would also like to thank Piergiulio Lenzi for his availability, Giacomo Graziani for the precious comments about the work, Artem Maevskiy for helping with background subtraction, Luca Clissa for various suggestions about Machine Learning algorithms and Denis Derkach for his hospitality which, unfortunately, I could not enjoy because of the COVID-19 emergency.
Finally, special thanks are given to *Istituto Nazionale di Fisica Nucleare* that allowed me to spend three months at CERN, improving my skills in Machine Learning and data analysis, and giving to me the opportunity to enhance my professional network within the LHCb Collaboration.

Bibliography

[1] Particle Data Group, M. Tanabashi *et al.*, *Review of Particle Physics*, Phys. Rev. D **98** (2018) 030001.

[2] LHCb Collaboration, R. Aaij *et al.*, *Observation of CP Violation in Charm Decays*, Phys. Rev. Lett. **122** (2019) 211803, arXiv:1903.08726.

[3] LHCb Collaboration, R. Aaij *et al.*, *Search for Lepton-Universality Violation in* $B^+ \to K^+\ell^+\ell^-$ *decays*, Phys. Rev. Lett. **122** (2019) 191801, arXiv:1903.09252.

[4] LHCb Collaboration, R. Aaij *et al.*, *Test of Lepton Universality with* $\Lambda_b^0 \to pK^-\ell^+\ell^-$ *Decays*, Journal of High Energy Physics **2020** (2020) 40, arXiv:1912.08139.

[5] A. Andreazza *et al.*, *What Next: White Paper of the INFN-CSN1*, Frascati Phys. Ser. **60** (2015) 1–302.

[6] LHCb Collaboration, J. Alves, A.Ãugusto *et al.*, *The LHCb Detector at the LHC*, JINST **3** (2008) S08005.

[7] R. Aaij *et al.*, *Selection and Processing of Calibration Samples to Measure the Particle Identification Performance of the LHCb Experiment in Run 2*, EPJ Tech. Instrum. **6** (2019) 1, arXiv:1803.00824.

[8] LHCb Collaboration, R. Aaij *et al.*, *LHCb Detector Performance*, Int. J. Mod. Phys. A **30** (2015) 1530022, arXiv:1412.6352.

[9] M. Adinolfi *et al.*, *Performance of the LHCb RICH Detector at the LHC*, Eur. Phys. J. C **73** (2013) 2431, arXiv:1211.6759.

[10] F. Archilli *et al.*, *Performance of the Muon Identification at LHCb*, JINST **8** (2013) P10020, arXiv:1306.0249.

[11] A. Hoecker *et al.*, *TMVA - Toolkit for Multivariate Data Analysis*, arXiv:physics/0703039.

[12] M. De Cian, S. Farry, P. Seyfert, and S. Stahl, *Fast Neural-Net Based Fake Track Rejection in the LHCb Reconstruction*, Tech. Rep. LHCb-PUB-2017-011. CERN-LHCb-PUB-2017-011, CERN, Geneva, 2017.

[13] M. Pivk and F. R. Le Diberder, *sPlot: A Statistical Tool to Unfold Data Distributions*, Nucl. Instrum. Meth. A **555** (2005) 356, arXiv:physics/0402083.

[14] R. Aaij *et al.*, *A Comprehensive Real-Time Analysis Model at the LHCb Experiment*, JINST **14** (2019) P04006, arXiv:1903.01360.

[15] L. Anderlini *et al.*, *Computing strategy for PID calibration samples for LHCb Run 2*, Tech. Rep. LHCb-PUB-2016-020. CERN-LHCb-PUB-2016-020, CERN, Geneva, 2016.

[16] LHCb Collaboration, *Computing Model of the Upgrade LHCb Experiment*, Tech. Rep. CERN-LHCC-2018-014. LHCB-TDR-018, CERN, Geneva, 2018.

[17] LHCb Collaboration, *LHCb Trigger and Online Upgrade Technical Design Report*, Tech. Rep. CERN-LHCC-2014-016. LHCB-TDR-016, CERN, 2014.

[18] LHCb Collaboration, *Upgrade Software and Computing*, Tech. Rep. CERN-LHCC-2018-007. LHCB-TDR-017, CERN, Geneva, 2018.

[19] LHCb Collaboration, I. Bediaga *et al.*, *Physics Case for an LHCb Upgrade II - Opportunities in Flavour Physics, and Beyond, in the HL-LHC Era*, arXiv:1808.08865.

[20] C. Bozzi, *LHCb Computing Resources: 2020 Requests and Preview of the Subsequent Years*, Tech. Rep. LHCb-PUB-2019-003. CERN-LHCb-PUB-2019-003, CERN, Geneva, 2019.

[21] T. Sjostrand, S. Mrenna, and P. Z. Skands, *PYTHIA 6.4 Physics and Manual*, JHEP **05** (2006) 026, arXiv:hep-ph/0603175.

[22] T. Sjostrand, S. Mrenna, and P. Z. Skands, *A Brief Introduction to PYTHIA 8.1*, Comput. Phys. Commun. **178** (2008) 852, arXiv:0710.3820.

[23] D. J. Lange, *The EvtGen Particle Decay Simulation Package*, Nucl. Instrum. Meth. A **462** (2001) 152.

[24] GEANT4 Collaboration, S. Agostinelli *et al.*, *GEANT4: A Simulation Toolkit*, Nucl. Instrum. Meth. A **506** (2003) 250.

[25] J. Allison *et al.*, *Geant4 Developments and Applications*, IEEE Trans. Nucl. Sci. **53** (2006) 270.

[26] J. Allison *et al.*, *Recent Developments in Geant4*, Nucl. Instrum. Meth. A **835** (2016) 186.

[27] D. Müller, M. Clemencic, G. Corti, and M. Gersabeck, *ReDecay: A Novel Approach to Speed Up the Simulation at LHCb*, Eur. Phys. J. C **78** (2018) 1009, arXiv:1810.10362.

[28] DELPHES 3 Collaboration, J. de Favereau *et al.*, *DELPHES 3: A Modular Framework for Fast Simulation of a Generic Collider Experiment*, JHEP **02** (2014) 057, arXiv:1307.6346.

[29] R. Brun and F. Rademakers, *ROOT: An Object Oriented Data Analysis Framework*, Nucl. Instrum. Meth. A **389** (1997) 81.

[30] G. A. Cowan, D. C. Craik, and M. D. Needham, *RapidSim: An Application for the Fast Simulation of Heavy-Quark Hadron Decays*, Comput. Phys. Commun. **214** (2017) 239, arXiv:1612.07489.

[31] B. Knuteson, *Systematic Analysis of High-Energy Collider Data*, Nucl. Instrum. Meth. A **534** (2004) 7, arXiv:hep-ex/0402029.

[32] G. Brooijmans *et al.*, *Les Houches 2015: Physics at TeV Colliders - New Physics Working Group Report*, arXiv:1605.02684.

[33] M. I. Jordan and T. M. Mitchell, *Machine Learning: Trends, Perspectives, and Prospects*, Science **349** (2015) 255.

[34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. http://www.deeplearningbook.org.

[35] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media, 2nd ed., 2019.

[36] A. Halevy, P. Norvig, and F. Pereira, *The Unreasonable Effectiveness of Data*, IEEE Intelligent Systems **24** (2009) 8.

[37] D. H. Wolpert, *The Lack of A Priori Distinctions Between Learning Algorithms*, Neural Computation **8** (1996) 1341.

[38] W. S. McCulloch and W. Pitts, *A Logical Calculus of the Ideas Immanent in Nervous Activity*, The Bulletin of Mathematical Biophysics **5** (1943) 115–133.

[39] F. Rosenblatt, *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*, Psychological Review **65** (1958) 386–408.

[40] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*, Nature **323** (1986) 533–536.

[42] M. Abadi *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. Software available from https://www.tensorflow.org/.

[43] I. J. Goodfellow *et al.*, *Generative Adversarial Networks*, arXiv:1406.2661.

[44] M. Mirza and S. Osindero, *Conditional Generative Adversarial Nets*, arXiv:1411.1784.

[45] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, *Unrolled Generative Adversarial Networks*, arXiv:1611.02163.

[46] M. Arjovsky and L. Bottou, *Towards Principled Methods for Training Generative Adversarial Networks*, arXiv:1701.04862.

[47] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein GAN*, arXiv:1701.07875.

[48] I. Gulrajani *et al.*, *Improved Training of Wasserstein GANs*, arXiv:1704.00028.

[49] M. G. Bellemare *et al.*, *The Cramer Distance as a Solution to Biased Wasserstein Gradients*, arXiv:1705.10743.

[50] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, arXiv:1312.6114.

[51] D. J. Rezende, S. Mohamed, and D. Wierstra, *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*, arXiv:1401.4082.

[52] L. de Oliveira, M. Paganini, and B. Nachman, *Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis*, Comput. Softw. Big Sci. **1** (2017) 4, arXiv:1701.05927.

[53] M. Paganini, L. de Oliveira, and B. Nachman, *CaloGAN : Simulating 3D High Energy Particle Showers in Multilayer Electromagnetic Calorimeters with Generative Adversarial Networks*, Phys. Rev. D **97** (2018) 014021, arXiv:1712.10321.

[54] S. Vallecorsa, F. Carminati, and G. Khattak, *3D Convolutional GAN for Fast Simulation*, EPJ Web Conf. **214** (2019) 02010.

[55] P. Musella and F. Pandolfi, *Fast and Accurate Simulation of Particle Detectors Using Generative Adversarial Networks*, Comput. Softw. Big Sci. **2** (2018) 8, arXiv:1805.00850.

[56] V. Chekalina *et al.*, *Generative Models for Fast Calorimeter Simulation: the LHCb Case*, EPJ Web Conf. **214** (2019) 02034, arXiv:1812.01319.

[57] D. Belayneh *et al.*, *Calorimetry with Deep Learning: Particle Simulation and Reconstruction for Collider Physics*, arXiv:1912.06794.

[58] B. Hashemi *et al.*, *LHC Analysis-Specific Datasets with Generative Adversarial Networks*, arXiv:1901.05282.

[59] LHCb Collaboration, A. Maevskiy *et al.*, *Fast Data-Driven Simulation of Cherenkov Detectors Using Generative Adversarial Networks*, arXiv:1905.11825.

[60] G. Sassoli, *Generative Adversarial Networks for the Ultra-Fast Simulation of the Muon Detector of the LHCb experiment at CERN*, Bachelor's thesis, Università degli Studi di Firenze, 2019.

[61] Y. Xie, *sFit: A Method for Background Subtraction in Maximum Likelihood Fit*, arXiv:0905.0724.

[62] C. Weisser and M. Williams, *Machine Learning and Multivariate Goodness of Fit*, arXiv:1612.07186.

[63] F. Pedregosa *et al.*, *Scikit-Learn: Machine Learning in Python*, Journal of Machine Learning Research **12** (2011) 2825.

[64] L. Breiman, *Arcing the Edge*, Tech. Rep. 486, Department of Statistics, Berkeley, 1997.

[65] F. Chollet *et al.*, *Keras*, `https://keras.io`, 2015.

[66] A. Paszke *et al.*, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, in *Advances in Neural Information Processing Systems 32* (H. Wallach *et al.*, eds.), pp. 8024–8035. Curran Associates, Inc., 2019.

[67] R. Aaij *et al.*, *Allen: A High Level Trigger on GPUs for LHCb*, Comput. Softw. Big Sci. **4** (2020) 7, `arXiv:1912.09161`.

[68] *CUDA Toolkit*, `https://docs.nvidia.com/cuda/`.

[69] Wes McKinney, *Data Structures for Statistical Computing in Python*, in *Proceedings of the 9th Python in Science Conference* (Stéfan van der Walt and Jarrod Millman, eds.), 56 – 61, 2010.

[70] E. Rodrigues, *The Scikit-HEP Project*, EPJ Web of Conferences **214** (2019) 06005.

[71] A. Navarro and J. Eschle, *phasespace: n-Body Phase Space Generation in Python*, Journal of Open Source Software **4** (2019) 1570.

[72] M. Cacciari, M. Greco, and P. Nason, *The p_T Spectrum in Heavy-Flavour Hadroproduction*, Journal of High Energy Physics **1998** (1998) 007.

[73] V. Vasilev, P. Canal, A. Naumann, and P. Russo, *Cling: The New Interactive Interpreter for ROOT 6*, J. Phys. Conf. Ser. **396** (2012) 052071.

[74] S. Cheong *et al.*, *Parametrizing the Detector Response with Neural Networks*, JINST **15** (2020) P01030, `arXiv:1910.03773`.

[75] M. Pervin, W. Roberts, and S. Capstick, *Semileptonic Decays of Heavy Λ Baryons in a Quark Model*, Phys. Rev. C **72** (2005) 035201.