UNIVERSITÀ
DEGLI STUDI
FIRENZE

PhD Program in Smart Computing
Dipartimento di Ingegneria dell'Informazione (DINFO)

The *flash-simulation* paradigm and its implementation based on Deep Generative Models for the LHCb experiment at CERN

Matteo Barbetti

Dissertation presented in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Smart Computing

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

The *flash-simulation* paradigm and its implementation based on Deep Generative Models for the LHCb experiment at CERN

Matteo Barbetti

Advisor:

Dr. Lucio Anderlini

Head of the PhD Program:

Prof. Stefano Berretti

Evaluation Committee:
Prof. Denis Derkach, *HSE University*
Prof. Michael Williams, *MIT*

XXXVI ciclo — May 2024

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH (CERN)

Friday 10th May, 2024

The *flash-simulation* paradigm and its implementation based on Deep Generative Models for the LHCb experiment at CERN

Author: Matteo Barbetti *Supervisor:* Lucio Anderlini

INFN and Università degli Studi di Firenze

Abstract. The LHCb experiment is dedicated to precision measurements of hadrons containing b and c quarks at the Large Hadron Collider (LHC) at CERN. During the first two Runs of the LHC, spanning from 2010 to 2018, the LHCb Collaboration invested more than 90% of the computing budget to simulate the detector response to the traversing particles produced in heavy hadron decays. Since 2022, the LHCb experiment has relied on a renewed detector and a novel data-acquisition strategy designed to acquire data at a rate enhanced by a factor of ten. Enabling an equivalent increase in simulation production is a major challenge, requiring a technological shift and diversifying the simulation strategies for specific purposes. Data processing and data analysis technologies have been evolving quickly during the last ten years. New industrial standards and huge communities behind open-source software projects arose, transforming the landscape of computer science and data processing. The fast development of *Machine Learning* and *Cloud* technologies provides modern solutions to address challenges well known to the High Energy Physics community, operating distributed data processing software on the nodes of the *Worldwide LHC Computing Grid* for the last three decades. In this Thesis, I present a study to adopt these new technologies to evolve the LHCb simulation software using machine learning models trained on multi-cloud resources to parameterize the detector response and the effects induced by the reconstruction algorithms. The resulting detector simulation approach is known as *flash-simulation* and represents the most challenging and radical option in the landscape of opportunities to accelerate the detector simulation. To encode in a machine learning model the intrinsic randomness of the quantum interactions occurring within the detector, the experimental uncertainties, and the effect of missing variables, parameterizations are designed as *Generative Models*, and in particular as *Generative Adversarial Networks*. The LAMARR project, arising as the official flash-simulation option of the LHCb experiment, enables connecting the trained models in long data-processing pipelines to simulate various effects in the detection and reconstruction procedure. Pipelines can be deployed in the LHCb Simulation software stack by relying on the same physics generators as the other simulation approaches and serializing the results with the format of the official reconstruction software. In this Thesis, I address the most compelling challenges in the design of a flash-simulation solution, including the identification of a strategy to train and validate reliable parameterizations, the definition and distribution of heavy hyperparameter optimization campaigns through opportunistic computing resources, the combination of multiple parameterizations in a data processing pipeline, and its deployment in the software stack of one of the major experiments at the LHC. Future work will extend the validation of flash-simulation techniques for additional heavy hadrons, decay modes, and data-taking conditions, paving the way to the widespread adoption of flash-simulations and contributing to a significant decrease in the average computational cost of detector simulation.

a Francesca,

porto sicuro nel
mio mare in tempesta

Preface

Machine Learning and Artificial Intelligence have been drastically reshaping the landscape of Computer Science for the last ten years. A stronger and stronger separation between the algorithm and its vectorized and optimized implementation was made possible by the wide adoption of Python with its numerical extensions (NumPy first, and then TensorFlow, PyTorch, and JAX), impressing a new thrust to the developments involving hardware accelerators. At the same time, the statistical modeling of physics quantities has become central in top-priority industrial applications, leading to an exponential expansion of the *Data Science* community and the emergence of hundreds of new tools and approaches to manage, process, and visualize complex data. The vast amount of data generated by the digital world we live in justified the introduction of computing infrastructures to enable scaling the computations on multiple sites, minimizing the effort for porting the applications from one center to another and identifying in the Web and the Web browsers the main entrance to the distributed, often virtual, computing resources. It is what we call today the Cloud.

Aware of this paradigm shift in Computer Science technologies, and proud of the long tradition in the development of computing infrastructure for *Big Science*, the National Institute for Nuclear Physics (INFN) has been investing great effort to renew its approach to data processing and computational science. The glorious developments of the *Worldwide LHC Computing Grid* (WLCG) connecting tens of national-scale data centers for the last three decades are being reviewed and simplified in light of the advancement of the data and Cloud industries, while Machine Learning is playing a first-citizen role in the development of the applications to process and analyze the experimental data.

This Ph.D. Thesis stems from a joint effort of the Department of Information Engineering of the University of Firenze and the National Institute for Nuclear Physics, with the partnership of the Universities of Pisa and Siena. The aim is to contribute to the ongoing technological shift in computing technologies, focusing in particular on the simulation of the LHCb experiment, one of the four large detectors surrounding the interaction points of the Large Hadron Collider (LHC), the world's largest and most powerful proton and heavy ion accelerator, built at CERN.

The LHCb experiment was designed to complement the physics program of the general-purpose detectors of the LHC, named ATLAS and CMS, with measurements of the particles produced at small angles with respect to the beam axis. In particular, hadrons

containing b and c quarks, known as heavy hadrons, are sufficiently light on the energy scale of the LHC to be produced at small angles and their study constitutes the highest priority in the LHCb physics program. During the first years of data-taking, the LHCb detector demonstrated great performance. Its pioneering data acquisition system, heavily relying on real-time software selections, was capable of unprecedented flexibility enabling impressive extensions to the originally planned physics program in several directions.

Despite the breadth of its physics program, the study of heavy hadrons remains central in the analysis activities in LHCb, and the vast majority of the computation resources pledged to the Collaboration are invested in simulating the decays of heavy hadrons and the subsequent interactions of the decay products with the material constituting the detector, the subsequent reconstruction procedures and the final selection strategy. In practice, thousands of data samples simulating the decay and the subsequent reconstruction of well-defined decay modes are produced every year. Some of them are used to perform feasibility studies for new data analyses, others investigate potential contributions of background in high-precision analyses, and others are requested to build a statistical model for the expected signature for some phenomena while searching the collected data for evidence.

Historically, the simulation of the LHCb detector was designed as a first, virtual prototype of the experiment. To perform feasibility studies on the most challenging physics analyses while defining the specifications for the radiation detectors and the data acquisition pipelines. That simulation, known as the *Detailed Simulation*, is based on first-principle models for radiation-matter interaction and is indeed capable of an impressive degree of accuracy in the determination of the physics performance of an apparatus even before construction. During the years following the start of the data acquisition, the Detailed Simulation has been patched, tuned, and refined multiple times, improving the accuracy of the physics models and the reliability of the detector description. Unfortunately, simulating each quantum interaction of hundreds of particles per event in a volume of several tens of cubic meters can be computationally expensive. Great effort has been invested by the High Energy Physics (HEP) community at large, and by the LHCb physicists in particular, to optimize the simulation and reusing, where possible, results from previous computations. Nevertheless, the detector simulation remains too expensive to be considered affordable for the planned upgrades of the HEP experiments, when the larger amount of collected data will require a larger amount of simulated events to be accurately interpreted.

In addition, physicists started to notice that a particle traversing a detector is associated with reconstructed features that depend almost exclusively on the kinematics of such particle and on the overall occupancy of the detector, rather than on the specific production mechanism. When performing the most precise data analyses of the data collected by the LHCb experiment, physicists started to ignore the simulated detector response to the traversing particles replacing them with the features of other reconstructed particles, produced otherwise but believed, from statistical reasoning, to be equivalent, in terms of detector response, to the simulated particle. The scientific community started then to imagine completely new simulation frameworks obtained by combining parameterizations and such resampling techniques, and capable of predicting the detector response to new decay modes from calibration data.

In 2019, with my Master Thesis titled *"Techniques for parametric simulation with deep neural networks and implementation for the LHCb experiment at CERN and its*

future upgrades", I proposed to use Generative Models as building blocks for complex parameterizations, possibly learned from calibration samples, and combined into pipelines where each model computes the output based on the quantities generated from the previous ones [1]. In that work, I discussed the challenges of integrating Machine Learning models in the LHCb simulation software and proposed an alternative and independent pure-Python simulation framework, employing in-memory columnar representation of batches of events and cross-table relations to remove dependencies on the LHCb software stack.

During the last three years, I had the opportunity to expand that pioneering work deepening my understanding of the Generative Models, employing distributed Cloud resources to improve their training, and deploying them in the official LHCb simulation software while preserving the ability to run the parameterization pipelines independently by adopting tabular, multi-language event model representation.

These contributions to the collective effort towards simulation speed up via Machine Learning technologies are discussed in depth in this Thesis. I introduce the scientific background and the LHCb experiment, focusing on its computing model and simulation software stack in Chapter 1. In Chapter 2, I discuss in more detail the technological shift in the computing landscape introduced by the widespread adoption of Machine Learning and Cloud technologies and present the development of HOPAAS a custom service to ease the distribution of hyperparameter optimization campaigns through multiple sites and different Cloud providers [2]. The technologies enabling Deep Learning and the state-of-the-art in terms of Generative Models are reviewed in Chapter 3. Chapter 4 describes in detail the models developed for the simulation of the LHCb experiments and their training [3–5]. Most of the models were obtained with the PIDGAN software package [6], a Python library exposing APIs to train multiple flavors of Generative Models, that I have developed as part of my Ph.D. program and made available as Free Open Source Software on GitHub[1]. Finally, in Chapter 5, I discuss the combination of the many parameterizations in a pipeline and discuss the results of the validation campaign of charged particles performed studying the semileptonic decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$. To integrate the parameterization in the LHCb software stack, while retaining the ability to run the same parameterization as an independent software package, I contributed to the development of `scikinC` [7], a transpiler for scikit-learn and Keras models trained in Python, enabling using them as shared objects from other applications.

During the three years of my Ph.D. I had the opportunity to discuss my results at several international conferences. In particular, I had two posters assigned at the 21$^{\text{st}}$ International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2022) titled *"Hyperparameter Optimization as a Service on INFN Cloud"*, and *"Lamarr: LHCb ultra-fast simulation based on machine learning models"*. At the 26$^{\text{th}}$ International Conference on Computing in High Energy and Nuclear Physics (CHEP 2023), organized in Norfolk (Virginia, USA), I was assigned the LHCb talk titled *"The LHCb Ultra-Fast Simulation Option, Lamarr"*. Besides, I was a lecturer for the third and fifth editions of the Hackathon of Machine Learning (Advanced Level), organized in Bari in November 2022, where I presented the talk *"Bayesian hyperparameter optimization"*; and in Pisa in November 2023 where I contributed to the lecture on *"Generative Models"*.

The remarkable results achieved in the development of this alternative approach to detector simulation were noticed by colleagues of the Pisa unit of the National Institute

[1] `https://github.com/mbarbetti/pidgan`

of Nuclear Physics who have started developing an analogous simulation for the CMS experiment, while the new *National Research Centre for High Performance Computing, Big Data and Quantum Computing* – ICSC, funded as part of the *Next Generation EU* program has identified the developments of detector simulations accelerated with Machine Learning techniques as one of its flagship activities.

The High Energy Physics community has a long and glorious tradition of software engineering and distributed computing which makes the emerged computing model well established and widely adopted. Rethinking the model and, in particular, the simulation, from its fundamental building blocks is a major challenge, and other years of development will be needed before a massive adoption of new simulation technologies will be able to significantly reduce the average computational cost of a simulated collision event. Among the options considered by the community, flash-simulation, obtained by combining *machine-learnt* parameterizations of the detector in data processing pipelines, is the most ambitious and radical. During my Ph.D., I had the opportunity to face and address some of the most fundamental challenges in the development of a full-fledged flash-simulation, from the construction of the parameterizations with Generative Models, and the representation of *particle-to-particle* correlation effects, to the efficient deployment of the pipelines in the production software stack of a major experiment. Future work will address the validation of the pipelines on various decay modes and for different data-taking conditions, heading the adoption of ultra-fast simulation for the challenging data analysis program of the current and upcoming Runs of the LHC experiments.

Contents

1 Scientific background and the LHCb experiment

The present knowledge of elementary particles and their interactions is collected within a successfully theory called Standard Model, which continue to predict the majority of the experimental results obtained to date. Nevertheless, the existence of several questions still unanswered, such as the dark matter, the neutrino mass, or the matter-antimatter asymmetry, brings out that the Standard Model is not a complete theory. In this scenario, the High Energy Physics experiments play a key role, providing the instruments for testing the validity of the theory and, eventually, pointing possible directions for its expansion. This chapter is dedicated to illustrate the scientific context in which my thesis work has been developed. In particular, Section 1.1 reports an overview of the Standard Model, briefly lingering on its theoretical aspects and describing some of the directions explored to find hints of New Physics. Such investigations often need for the study of extremely rare decays or very high energy process that require to rely on dedicated experimental apparatuses. Section 1.2 offers a detailed description of one of those apparatuses, the LHCb experiment which is designed for the study of Heavy Flavor Physics, looking for anomalous effects beyond the Standard Model.

1.1 The Standard Model and beyond: an overview

Since the 1930s, physicists witnessed a proliferation of particles thanks to the increasing number of studies and experiments investigating the fundamental structure of the 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 how they interact, called *Standard Model* and indicated as SM hereafter. The SM is a quantum field theory representing particles as spin-$\frac{1}{2}$ fields and their interactions as spin-1 fields. Gravity is the unique known interaction to not be included within the SM since it is still not clear how to describe it as a quantum field theory. On the other hand, the SM describes our best comprehension of all the other known interactions, namely the Electromagnetic interaction, associated to a neutral massless mediators named *photons* γ, the Weak interaction, associated to the charged massive mediators $W^{\pm}$ and the neutral massive mediator Z^0, and the Strong interaction, associated to a neutral massless mediator named *gluon* g. In the quantum field theory formalism, the Electromagnetic and Weak forces can be unified into the

Electroweak interaction, associated to the aforementioned two neutral (γ and Z^0) and charged ($W^{\pm}$) mediators. In 2012, the discovery of the Higgs boson [8, 9] by the ATLAS and CMS Collaborations at the LHC has confirmed the existence of a third kind of field, scalar (spin $s = 0$), defining the inertial properties (namely, the mass) of matter and interaction fields.

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 called *fermions*. The SM divides fermions into two families: *leptons* and *quarks*. Leptons are elementary particles which do not interact through the Strong force. The *electron* (e^-), essential constituent of the ordinary matter and responsible for all the properties exhibited by the chemical elements, is the lightest charged lepton. Two other charged leptons are know: the *muon* (μ^-) and the *tau* (τ^-). All three have the same unity charge, $-e$. While no electron decay is neither observed nor expected in the SM, so ensuring to the ordinary matter to be stable, muons and taus can decay to lighter leptons through the Weak interaction:

$$\mu^- \to e^- \bar{\nu}_e \nu_\mu \qquad \tau^- \to \mu^- \bar{\nu}_\mu \nu_\tau \qquad \tau^- \to e^- \bar{\nu}_e \nu_\tau$$

Such decays involve the emission of neutral leptons called *neutrinos* that, due to their charge, are only allowed to interact via the Weak force. The Weak decays conserves a quantum number, called *lepton number* that is shared by charged and neutral leptons. Hence, the three charged leptons correspond to as many neutrino states referred to as ν_e, ν_μ, and ν_τ.

Contrary to leptons, quarks are allowed to interact also via the Strong force. The *Quantum Chromo-Dynamics* (QCD) is the theory that describes such interactions by relying on a quantum number, called *color charge*, that quarks share with gluons. An important difference between Strong and Electromagnetic interactions is the self-interaction property of the mediator particles: possible for gluons, while impossible for photons. The self-interaction explains why the Strong bound-states, called *hadrons*, are so different

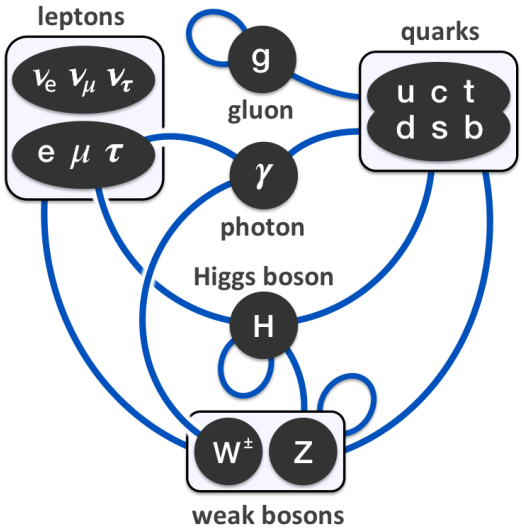


Figure 1.1: Summary of interactions between particles described by the Standard Model.

Family	*Flavor*			*Charge*	*Spin*
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	
Quarks	u ($\bar{u}$)	c ($\bar{c}$)	t ($\bar{t}$)	$+2/3$ ($-2/3$)	$1/2$
	d ($\bar{d}$)	s ($\bar{s}$)	b ($\bar{b}$)	$-1/3$ ($+1/3$)	

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

from the Electromagnetic bound-states, for example *atoms*, and why the constituents of the latter (electrons and nuclei) can be observed as free, while quarks can only be observed within hadrons. In the ordinary matter, the most common elementary particles are the *down* quark (d), having charge $-\frac{1}{3}e$, and the *up* quark (u), having charge $-\frac{1}{3}$, which are combined to form protons, with the bound-state uud, and neutrons, with the bound-state udd. The property of a quark of being up or down is named *hadronic flavor*. Six different flavors exist for quarks: *down* (d), *up* (u), *strange* (s), *charm* (c), *bottom* (b), and *top* (t). For the two latter flavors, the alternative names *beauty* and *truth* are often used. A schematic representation of elementary particles and how they interact is depicted in Figure 1.1.

For any elementary matter particle (i.e., leptons and quarks), a respective *antiparticle* exists. For example the positron (e^+) is the antiparticle of the electron (e^-), while an *anti-up* ($\bar{u}$) denotes the antiparticle of the up quark (u). The charge values of both leptons (quarks) and anti-leptons (anti-quarks) are reported in Table 1.1. Excluding the charge, (almost) all the properties of a particle are (almost) the same as those of the respective antiparticle. Hence, as quarks, also anti-quarks interact through the Strong force and, for example, it is possible to observe bound-states formed by a quark (q) and an anti-quark ($\bar{q}$), called *mesons* ($q\bar{q}$). As previously observed, the SM admits the existence of bound-states of three quarks, like protons (uud) and neutrons (ddu) that are generally called *baryons* (qqq). Following this formulation, also bound-states formed by three anti-quarks exist and are called *anti-baryons* ($q\bar{q}q$). It should be pointed out that, due to the number of flavors, the complexity of QCD, and the existence of bound state of particles and antiparticles, the variety of hadrons is huge, however providing a formal characterization of their properties is probably beyond the scope of this thesis and any interested reader is referred to the reviews of the Particle Data Group (PDG) [10].

While the quark flavor is conserved in the Strong interactions described by the QCD Lagrangian, decays of hadrons towards lighter states with different flavor are observed, and interpreted as Weak decays of quarks through the emission of a charged W boson. Notably, another important feature of the SM is that Weak and mass eigenstates[1] of quarks do not coincide. This results into the need of defining a mixing matrix to describe the Weak interaction[2] of the mass eigenstates: it is called *Cabibbo–Kobayashi–Maskawa matrix*, or

[1] In quantum physics, the *eigenstate*s correspond to states of the system where certain measurement will always yield the same value, i.e., the observables (like energy, position, or momentum) have definite, predictable values.

[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 is 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 phenomena in flavor-changing processes of the SM where the CP symmetry is violated[3]. 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 physics Beyond the Standard Model (BSM), also known as 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[4] and hadron colliders (see next Section). 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, $\alpha + \beta + \gamma = (179^{+7}_{-6})^\circ$, is consistent with the SM expectation [10].

Both *beauty* and *charmed* mesons ($q\bar{q}$ states made of b- or c-quark coupled with a lighter one) offer decay channels to study CP-violation. In 2019, the LHCb Collaboration at CERN has observed, for the first time, the matter-antimatter asymmetry in the D^0 meson [11], confirming the extremely small contribution of $D^0 - \bar{D}^0$ mixing expected by the SM. More recently, the LHCb Collaboration has measured the time-dependent CP-violation in the decays of B^0 and $\bar{B}^0$ mesons [12] offering the most precise single measurement of the CKM angle β to date, overcoming in precision the current world average [10].

Heavy flavor physics allows indirect search for NP also by 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 species should be produced

[3]The fundamental discrete symmetries are C, P and T. C is the *charge conjugation* symmetry and relates two states where the second represents the antiparticle of the first. P is the *parity symmetry* and relates two states changing the sign of the three space coordinates $\mathbf{x} \to -\mathbf{x}$. Lastly, T is the *time reversal symmetry* and relates two states that differ for the sign of the temporal coordinate. While the CPT symmetry is conserved in all the physical phenomena, CP and T symmetries can be violated. The study of processes where CP is violated is crucial to understand the abundance of matter over antimatter observed in the present-day Universe.

[4]B-factories are particle collider experiments designed to produce and detect a large number of B mesons so that their properties and behavior can be measured with small statistical uncertainty. Read more on `https://en.wikipedia.org/wiki/B-factory`.

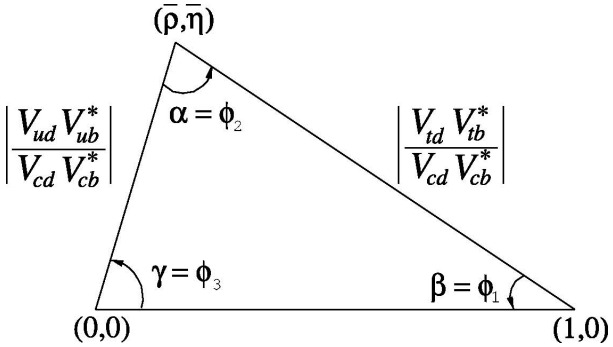


Figure 1.2: Sketch of the unitarity triangle. Figure reproduced from Ref. [10].

with the same probability in particle decays, once mass differences are taken into account. To 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 *Flavor-Changing Neutral Currents* (FCNC), for example, are allowed in the SM 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 [13–16] or, in the case of violation, to highlight effects beyond the SM.

Despite its clear success, the SM is still far from being a complete theory. While there are signs of SM, none of them sharply points to a specific extension of the theory, nor tells us by which kind of experiments NP will be discovered [17]. 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 High Energy Physics (HEP) experiments, like the Large Hadron Collider discussed in the next Section, play a key role, facing the technological challenges to move forward in understanding the Universe and its laws.

1.2 The LHCb experiment at CERN

Approved by the CERN Council in 1994 and fired up for the first time in 2008, the *Large Hadron Collider* (LHC) [18] is the accelerator currently operating at CERN and the largest ever built. It is located at the French-Swiss border and is hosted in the same tunnel previously used by the Large Electron-Positron collider (LEP) operating from 1989 to 2000. The tunnel, with a 27 km long circumference and an average 100 m depth underground, contains two beam pipes for as many particle beams accelerated in opposite directions. Differently from LEP that delivered electron-positron collisions, protons or ions (notably the lead ones) circulate into LHC at higher energy. In addition, contrary to proton-antiproton machines (such as Tevatron operating at Fermilab until 2011) where antiprotons are difficult to produce, accumulate and uniformly squeeze in direction and energy, the employment of protons in both beams ensures a higher accelerator capability to deliver collisions. Such a property is described by the *instantaneous luminosity* $\mathcal{L}$, defined as

$$\mathcal{L} = \frac{1}{\sigma(\sqrt{s})} \frac{dN}{dt} \tag{1.3}$$

where dN/dt and $\sigma(\sqrt{s})$ are the rate and the cross-section[5] of a given process, respectively, at the $\sqrt{s}$ energy scale. Whenever a collider relies on two highly-relativistic beams as at LHC, the luminosity definition can be reduced to [19, 20]:

$$\mathcal{L} \simeq n_b \, \nu_{rev} \, \frac{N_1 N_2}{4\pi \sigma_x \sigma_y} \tag{1.4}$$

being n_b the number of circulating bunches of particles, ν_{rev} the revolution frequency, $N_{1,2}$ the number of particles per bunch, and $\sigma_{x,y}$ the effective beam transversal dimensions. Because of the variation of $N_{1,2}$ due to beam-beam collisions which eject particles from the beams, and because of the increase in $\sigma_{x,y}$ due to beam warming, $\mathcal{L}$ decreases during the acquisition period.

The design pp target instantaneous luminosity at LHC was 10^{34} cm$^{-2}s^{-1}$, but in 2017 a value twice this limit was reached. Since the total quantity of collected data is typically indicated with the integrated luminosity $\int \mathcal{L}\, dt$, disposing of a high instantaneous luminosity is a key requirement to study rare phenomena like FCNC processes or to search for physics BSM. A second crucial component is to collide particles with high center-of-mass energy $\sqrt{s}$ to have access to decays, namely increasing their cross-section, that involve particles with high mass, such as the Higgs boson. During the first data taking period (2009-2012), the so-called Run 1, LHC operated with a collision energy of 7 TeV and then 8 TeV. With the start of the Run 2 (2015-2018), the collision energy reached 13 TeV, 1 TeV below the LHC design energy. In April 2022, LHC has resumed the data taking with the start of Run 3 where is operating with a new maximum collision energy of 13.6 TeV, to be increased to the design one at a later stage.

The LHC is served by other smaller and less powerful particle accelerators which gradually accelerate protons up to 450 GeV before transferring them to the LHC storage rings, where they are further accelerated for about 20 minutes before reaching the operational energy. In Figure 1.3, a schematic representation of the accelerators complex of CERN is shown. Once reached the LHC ring, protons or lead ions circulating in opposite beams are made to collide at four crossing points, where the main experiments are located:

- The CMS [21] and ATLAS [22] detectors are general purpose experiments, being conceived to the study of the Higgs boson, to the analysis of the top quark and to search for physics BSM. Since no claims of how NP will be exhibited is known, the geometry of the two detectors covers the largest possible fraction of the solid angle and is equipped with a versatile detector structure. The main differences between the two experiments are the magnetic field that is employed to reconstruct the particle trajectories, which is purely solenoidal for CMS and also toroidal for ATLAS, and the installed detector technologies;

- The ALICE [23] experiment is dedicated to the study of the matter under extreme temperature and pressure conditions, where *Quark-Gluon plasma* QGP with quarks free from the confinement in hadrons is formed. The ALICE detector is optimized to study heavy-ion collisions, where the QGP formation is expected;

[5] Given a particle approaching another particle, the *cross-section* for this process is the probability that the two particles interact with each other.

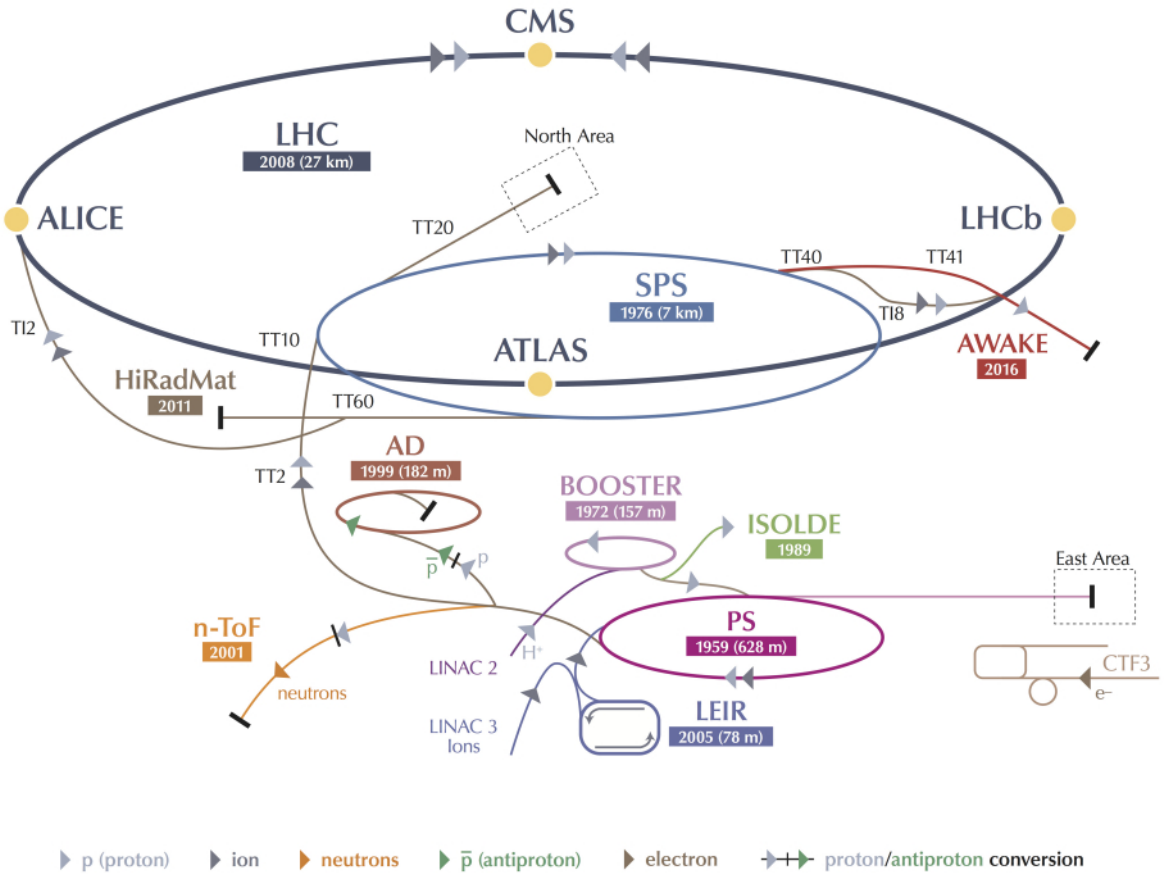


Figure 1.3: Scheme of the CERN accelerator complex with the largest experiments indicated. Reproduced from `https://home.cern/resources/faqs/facts-and-figures-about-lhc`.

- The LHCb [24] detector, detailed in the rest of this chapter, was originally designed for high-precision measurements of the b- and c-quark decays, while can now be considered a general purpose experiment covering the forward region.

LHCb is the LHC experiment specialized in studies of b-physics. The experiment has a wide physics program covering many important aspect of Heavy Flavor, Electroweak, and QCD physics. The large production cross-section of $b\bar{b}$ pairs at the energy reachable in high energy proton collisions ($\sigma_{b\bar{b}} \simeq 500$ µm) makes the LHC the most copious source of B mesons in the world. To exploit this large number of b-hadrons, the LHCb experiment has been developed as a *single-arm detector*, in contrast with the other three large LHC detectors (ATLAS, CMS, and ALICE) which are called 4π detectors since they cover a solid angle of nearly 4π srad. The instrumented region covered by the LHCb detector is $\theta \in [10, 300\,(250)]$ mrad in the horizontal (vertical) plane with respect to the beam axis, corresponding to the pseudorapidity[6] values $2 < \eta < 5$. Such a choice is motivated by Figure 1.4, illustrating the production cross-section of $b\bar{b}$ pairs as a function of the emission angles of the two quarks with respect to the beam axis. According to this distribution, the geometrical acceptance of LHCb for $b\bar{b}$ pairs is about 24%, and is highlighted through

[6]The *pseudorapidity* η is defined as $\eta = -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]$, where θ is the angle between the particle momentum and the beam axis.

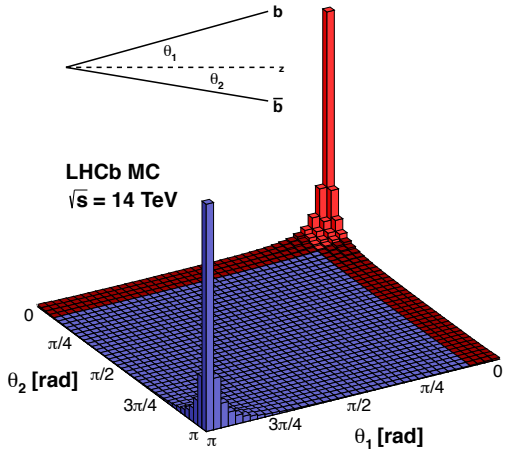


Figure 1.4: Polar angular distribution with respect to the beam axis for the $b\bar{b}$ pairs production cross-section at a $\sqrt{s} = 14$ TeV energy, according to the PYTHIA8 generator [25]. Red squares only consider the $b\bar{b}$ pairs in the LHCb detector acceptance. Figure reproduced from Ref. [26].

red squares in Figure 1.4. The key features of the LHCb detector include:

- Excellent vertex and proper time resolution.
- Precise particle identification, especially for kaon-pion separation.
- Precise invariant mass reconstruction. This feature is required to efficiently reject background due to random combinations of tracks (combinatorial background) and implies a good momentum resolution.
- Versatile trigger scheme. High efficiency is required in both leptonic and hadronic B decay channels, in order to collect high statistics samples and study the variety of decay modes with small branching ratios.

A scheme of the LHCb spectrometer layout, with the main detectors indicated, is shown in Figure 1.5. Conceptually, these can be grouped into two subsystems:

1. **Tracking system.** System composes of a vertex locator and a set of tracking stations upstream and downstream a large dipole magnet. The Tracking system allows to measures the charged particle momenta, identify the collision vertices and reconstruct the particle decays.
2. **Particle identification (PID) system.** System composes of two Ring Imaging Cherenkov (RICH) detectors, a calorimeter system, and five muon stations. The PID system combines the momentum information with the velocity or the energy to separate the different charged particle species and identify some neutral particles, such as neutral pions or photons.

In the following Sections, a brief description of both the systems is depicted together with some figures of merit for their performance, while full information can be found in the indicated references.

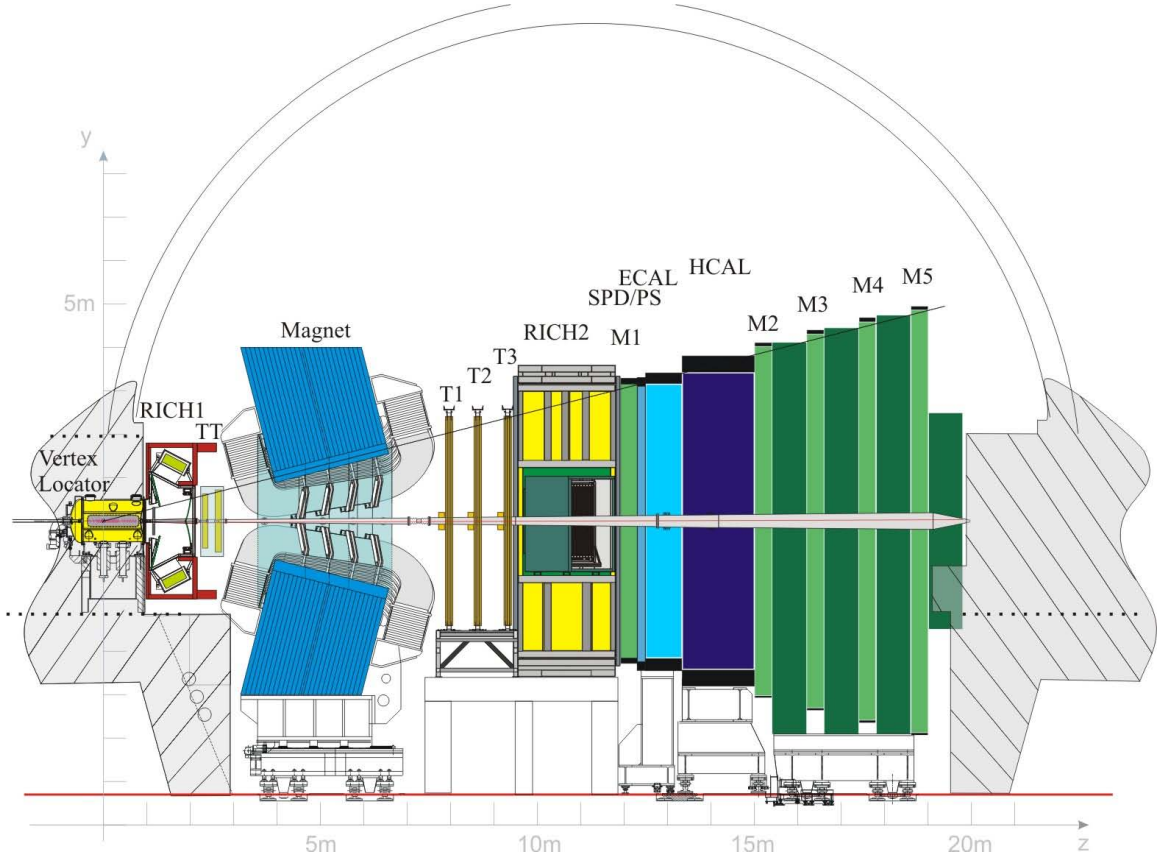


Figure 1.5: A schematic representation of the LHCb detector in the non-bending vertical plane. The definition of non-bending is referred to the magnetic field, which bends charged particles trajectories in a plane orthogonal to the represented one. The origin of the Cartesian reference system is centered on the beam-beam interaction point. The z axis coincides with the beam axis and is directed towards downstream detectors, the y axis is vertical and defined to be parallel to the weight-force direction pointing upward. The x axis is horizontal and form right-hand reference system with the axes defined above. Figure reproduced from Ref. [24].

1.2.1 Tracking system

Aiming to achieve a precise determination of the decaying particles mass, to measure the particle momenta, and to reconstruct the positions of the pp collisions and of the decay vertices, the trajectories of charged particles are made bent by employing a magnetic field. At the LHCb detector, a magnetic field with a 4 T $\cdot$ m bending power is generated by a dipole magnet [27], which allows to reconstruct the particle trajectories by combining the information from:

- the *Vertex Locator* (VELO) detector [28, 29], a set of silicon tracking stations surrounding the pp interaction region;
- the *Trigger Tracker* (TT) [30], tracking stations providing some trajectory reference coordinates upstream the magnet;
- the *Inner* [31] and *Outer Trackers* [32, 33] (T1, T2 and T3), a set of three tracking stations installed downstream the magnet.

Particles that are reconstructed with the information from the whole LHCb Tracking system have a resolution on the momentum ranging from 0.5 to 1% for $5 < p < 200$ GeV/c [34].

The VELO detector

The *Vertex Locator* (VELO) is the first detector encountered by the particles produced in the LHC collisions. It is devoted to the early reconstruction of track segments, and to measure the position of primary and secondary vertices. One of the main goals of the VELO is an excellent resolution in the separation between the two vertices, since a well-defined and displaced decay vertex is a typical signature for the production of a b- or c-hadrons. As schematically shown Figure 1.6 (top), the VELO consists of 23 silicon tracking stations arranged transversally with respect to the beam axis z, covering a region of about one meter. Each station consists of two movable modules, a feature required considering that the minimum sensor distance from the beam is 8.2 mm during data taking, and lower than the radial aperture of the LHC beam during the injection or technical development periods. Thus, a mechanical system is used to retract the two module halves by 3 cm and protect them from the radiation, as illustrated in Figure 1.6 (bottom left). When the beam safety conditions are restored, a closing procedure brings back the VELO stations to their nominal position.

The key requirements for the VELO detector performance are:

- a larger angular coverage than the nominal LHCb acceptance, which is achieved by the chosen sensor positions;
- an efficient reconstruction of straight-line tracks, being the magnetic field effect negligible in the VELO z region and a high rejection of fake tracks;
- an excellent resolution in the separation of the primary and secondary vertices.

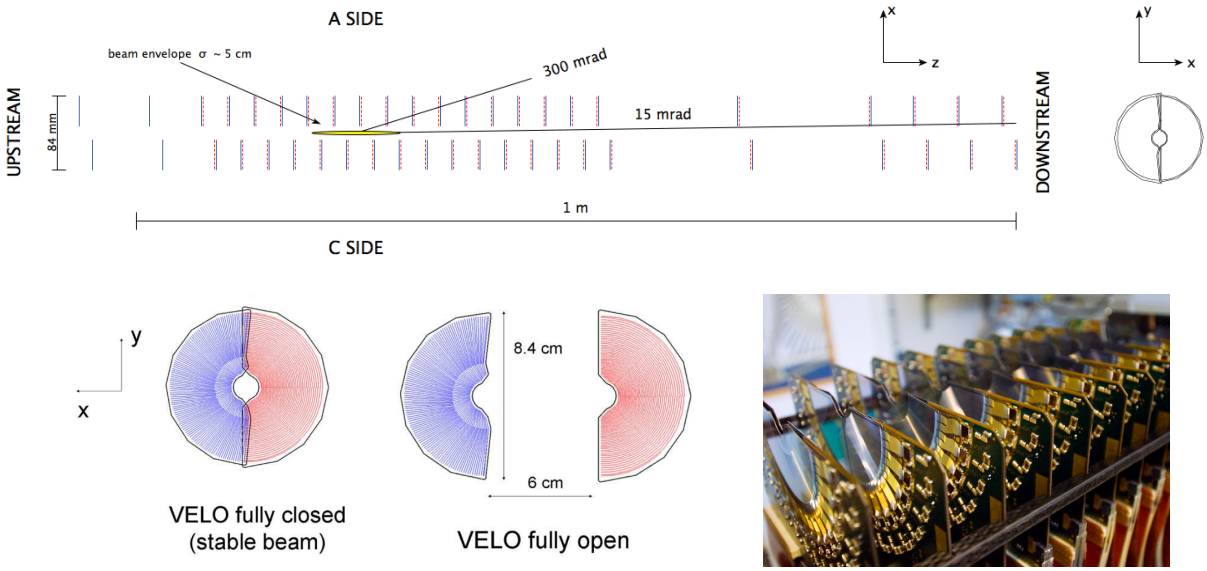


Figure 1.6: The top figure shows the positions of the VELO stations along the beam axis z. The bottom left figures compare the fully closed and open positions of the VELO, while on the right a picture of one half of the detector is reported. Figures reproduced from Ref. [29].

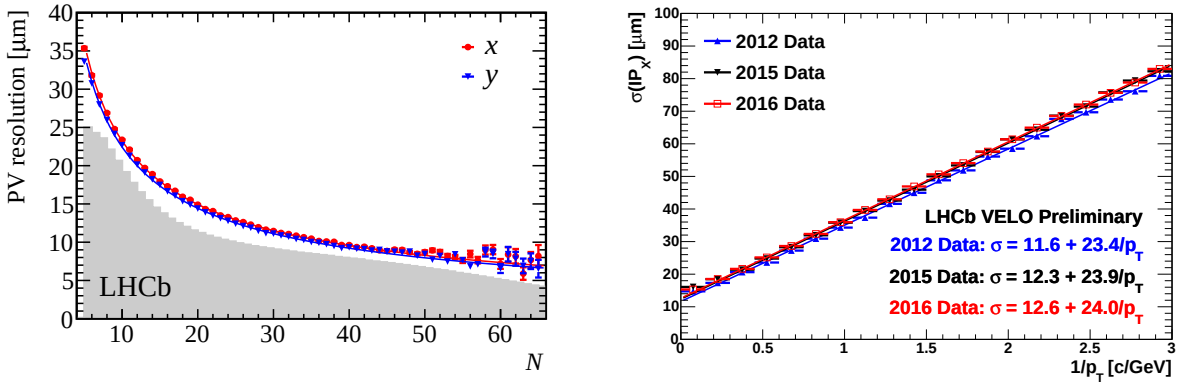


Figure 1.7: Figures of merit for the performance of the VELO detector. The left plot presents the primary vertex resolution as a function of the number of particles originating in the vertex; the right one the impact parameter resolution measured on 2012 (in blue), 2015 (in black) and 2016 (in red) data. Figures from Refs. [34] and [35], respectively.

Figure 1.7 reports two figures of merit illustrating the VELO performance, whose detailed discussion can be found in Ref. [29]. The left plot shows the primary vertex resolution as a function of the number of particles associated to the vertex for 2012 data reconstructed with only one vertex. The right plot reports the resolution on the x coordinate of the *impact parameter*, namely the distance between the linearly extrapolated track and the reconstructed vertex position. This is affected by the multiple scattering and steeply increase with the inverse of the particle transverse momentum. Thanks to the geometry of the VELO and its proximity to the beam, LHCb exhibits the best resolution of the impact parameter coordinates among all the LHC experiments, ranging from 10 to 80 μm for transverse momenta in $0 < 1/p_{\mathrm{T}} < 3$ $(\,\mathrm{GeV}/c)^{-1}$.

The dipole magnet and the tracking stations

The LHCb magnet, schematically represented in Figure 1.8 (left), has a bending power of about $4\ \mathrm{T}\cdot\mathrm{m}$ and is generated by two coils that reach a maximum peak intensity of 1.1 T directed as the y axis. The current in the magnet, and hence the magnetic field polarity, is periodically inverted. Indeed, the LHCb detector is designed to be symmetric with respect to the x coordinate, an essential characteristic to mitigate the charge-dependent detection asymmetries that could affect the measurements of CP-violation phenomena. The magnetic field profile evolution with z is shown in Figure 1.8 (right) compared to the positions of the tracking detectors.

The tracking stations located upstream the magnet are called *Trigger Tracker* (TT) or *Tracker Turicensis*. These are located immediately upstream the dipole magnet to set a reference for charged particle trajectories before the deflection. This improves the precision of the measurement of particle momenta, reduces the fraction of fake tracks (i.e., tracks resulting from a random combination of hits), and allows the reconstruction of long-living particles that decay outside the VELO, such as Λ or K^0_{S} [36]. The TT detector consists of two stations distant ~ 30 cm with two layers each. All layers are equipped with 512 silicon strips and cover the full LHCb acceptance. The first and last layer measure the x track coordinate, the other two are inclined by $\pm 5°$, forming two axes called u and v. By combining the information from all the sensors, ambiguities in the

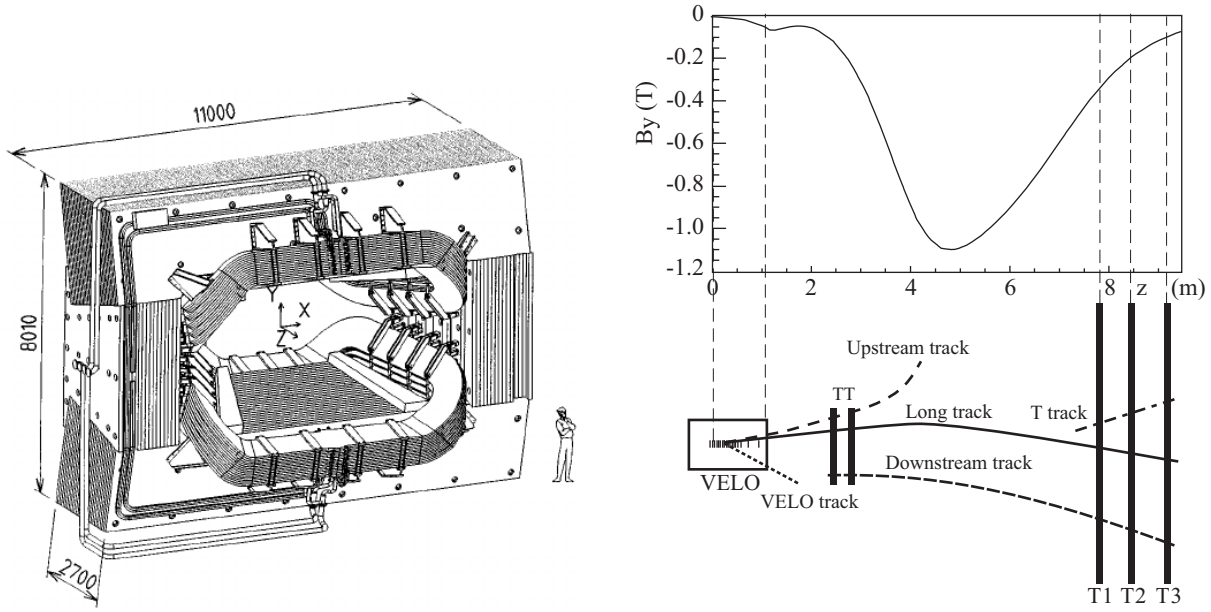


Figure 1.8: Schematic representation of the LHCb magnet (left) and evolution of the magnetic field intensity as a function of the z coordinate (right). Figures reproduced from Refs. [24] and [34], respectively.

track reconstruction are removed.

Downstream the magnet, three other tracking stations (T1, T2, and T3) each equipped with four detection planes reconstruct the particle trajectory after its bending in the magnetic field. Each T station employs two different tracking technique, depending on the closeness to the beam pipe. In the inner region, with a harsher environment due to the larger detector occupancy, the T stations are equipped with the *Inner Tracker* (IT) [31] which relies on silicon detector with high radiation resistance, granularity, and spatial resolution. The IT stations only cover a region of $\sim 2\%$ of the LHCb acceptance (5×6 m^2), however they measure about 20% of the particle flux, due to the low-angle peak in particle distributions. At the radial values, where the detector occupancy decreases to 10%, the IT stations are replaced by the *Outer Tracker* (OT) [32, 33] ones, which employ straw tubes, gas detectors developed to minimize the material budget before the calorimeters. The achieved spatial hit resolution are 50 μm and 200 μm for the IT and OT detectors, respectively.

Once a particle has traversed the whole Tracking system, the corresponding track is reconstructed by combining the information coming from all the detectors in two stages [37]. Firstly, pattern recognition tasks are executed to bundle together the hits (i.e., energy deposits) left by the particles in the detectors. Two strategies, called *forward tracking* and *track matching*, are followed. In the former, a starting track seed is reconstructed in the modules of the VELO detector. Since the magnetic field intensity is negligible at the VELO z coordinates (see Figure 1.8), a straight-line model is assumed to determine the track positions and slopes. Then, the reconstructed track segment is extrapolated to the TT first modules and corresponding hits are looked for. This rough momentum estimation is used to predict the position in the T stations by assuming that the magnetic field effect produce a kink in the particle trajectory (see Figure 4.5). In the latter strategy, track seeds are reconstructed in the VELO detector and in the T stations at the same

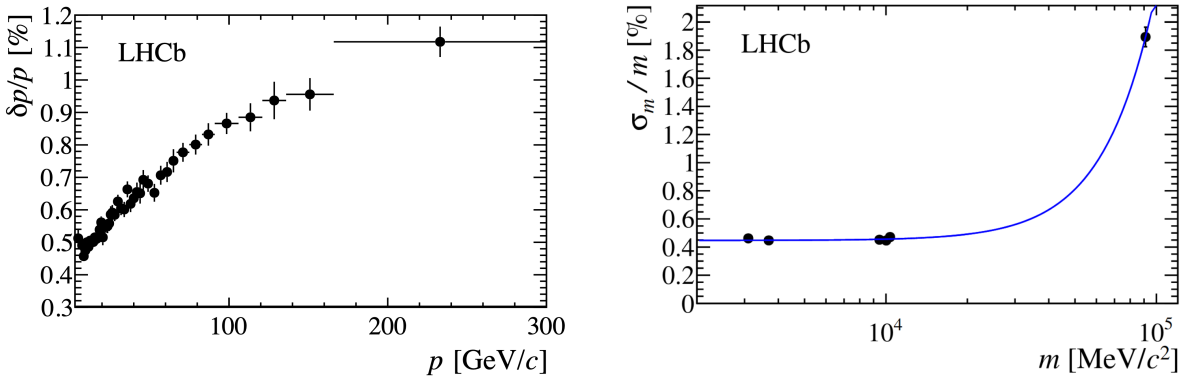


Figure 1.9: LHCb relative resolution on the momentum (left) and on the mass for di-muon resonances (right). Figures reproduced chefrom Ref. [34].

time, and are extrapolated forward and backward to the magnet plane. A compatibility criterion to match the two track segments is then set based on their distance in x and y at the magnet bending plane. Secondly, the bundled hits are fitted to determine the track parameters, including the particle momenta. A Kalman filter model [38] is employed in order to account for the multiple scattering effects. The resulting χ^2/ndf of the track fit is used to evaluate the track reconstruction quality. In addition, a dedicated neural network classifier is used to measure the probability that a track results from the random combination of hits coming from different particles [39]. The output of this classifier, called `ghostProb`, is typically used within selection algorithms.

The relative momentum resolution as a function of the momentum is depicted in Figure 1.9 (left). Particles that are reconstructed with the information from the whole Tracking system have a resolution on the momentum ranging from $\sim 0.5\%$ to $\sim 1\%$ for $5 < p < 200$ GeV/c [34]. For tracks originating in the same vertex, the invariant mass is calculated as:

$$Mc^2 = \sqrt{\left(\sum_i E_i\right)^2 - \left(\sum_i p_i c\right)^2} \tag{1.5}$$

indexing i the particles and indicating E_i and p_i their energy and momentum, respectively. The resolution achieved by LHCb on the determination of this quantity is presented in Figure 1.9 (right) for di-muon resonances, and ranges from $\sim 0.5\%$ to $\sim 2\%$ for $3 < M < 100$ GeV/c^2 [34].

1.2.2 Particle Identification system

With the available measurement of a charged particle momentum, its mass can be derived once its velocity or energy is known. The discrimination of all the particle species in the momentum range of interest ($\sim$ 2-150 GeV) demands for a redundancy of experimental strategies. Detectors devoted to Particle Identification (PID) at the LHCb experiment are:

- the Ring Imaging Cherenkov (RICH) system [40], exploiting the Cherenkov effect to discriminate among charged particles, notably protons, pions and kaons;

- the Calorimeter system [41], where the showers of particles produced by charged or neutral particles interacting with the detector material are absorbed to measure the incoming particle energy. The system is composed of an Electromagnetic calorimeter (ECAL), optimised for the identification of neutral pions, photons, electrons and positrons, and a Hadronic calorimeter (HCAL). To give a fast measurement of the detector occupancy employed for the hardware level of the trigger and to induce the start of the shower before the calorimeters, two scintillator planes called Scintillating Pad Detector (SPD) and PreShower (PS) are installed upstream ECAL.

- Muons are capable to escape HCAL and are tracked by five other dedicated stations [42] downstream the calorimeters. Matches between extrapolated tracks and energy deposits in these detectors provide thus the identification for muons.

The RICH system

A charged particle traversing a dielectric medium of refractive index n with a velocity $\beta > 1/n$ emits Cherenkov photons with a characteristic angle θ_c depending on its mass m and momentum p as [43]:

$$cos\theta_c = \frac{1}{n\beta} = \frac{1}{n}\sqrt{1 + \left(\frac{mc^2}{pc}\right)^2} \tag{1.6}$$

In order to measure θ_c, allowing charged particles to be identified when combined with the momentum information from the tracking system, two RICH detectors located upstream and downstream the magnet and schematised in Figure 1.10 are used. The former, that covers the full LHCb acceptance, is optimised for particles with $p < 60\,\mathrm{GeV}/c$, which could exit the detector acceptance because of the magnetic deflection. The latter efficiently identifies particles with $p \in [15, 110]\,\mathrm{GeV}/c$ in the acceptance region $[15, 120]$ mrad. The photons emitted by the charged particles in the RICH radiators are conveyed with a system of spherical and planar mirrors outside the LHCb acceptance and, independently on their emission point, form a ring on a lattice of Hybrid photodetectors (HPDss). Depending on the momentum and assuming a mass hypothesis, a test ring for each particle species is compared against the positions of the activated photodetectors. A RICH likelihood function is evaluated and, when combined with the information from the calorimeter and muon systems, is associated to tracks to define a particle identification classifier to be used in physics analyses. Usually, this is expressed as a differential log-likelihood (DLL) between two h_1 and h_2 particle hypotheses:

$$\mathrm{DLL}_{h_1,h_2} = \log\left(\frac{h_1 \text{ likelihood}}{h_2 \text{ likelihood}}\right) \tag{1.7}$$

The theoretical evolution of the Cherenkov angle for all particle species as a function of the momentum and for all the radiators employed in the RICH system is presented in left Figure 1.11. The right figure presents the θ_c angle reconstruction performance in the RICH2 detector for tracks in 2010 and 2011 data reconstructed as isolated, i.e. whose associated Cherenkov rings do not overlap with any other one. Despite the resolution effects, an excellent proton-kaon separation up to $p \simeq 50\,\mathrm{GeV}/c$ is obtained, while the RICH2 refractive index limits the active identification of kaons and protons to $p \leq 9.3\,\mathrm{GeV}/c$

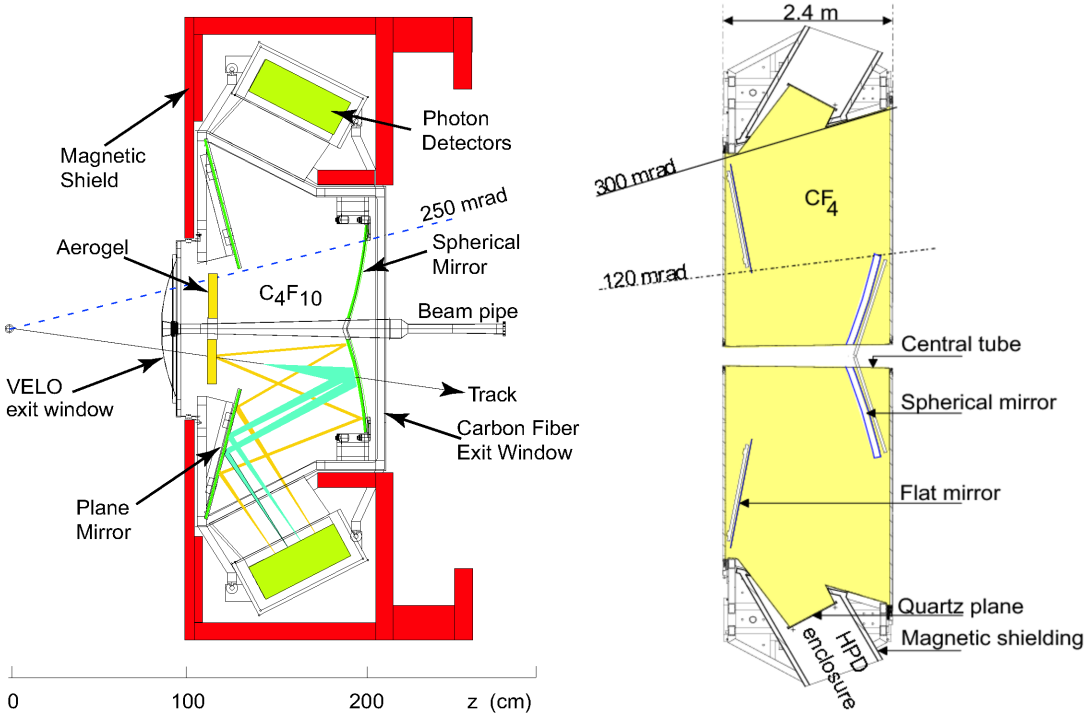


Figure 1.10: Scheme of the RICH1 (left) and RICH2 (right) detectors. Figures from Ref. [24].

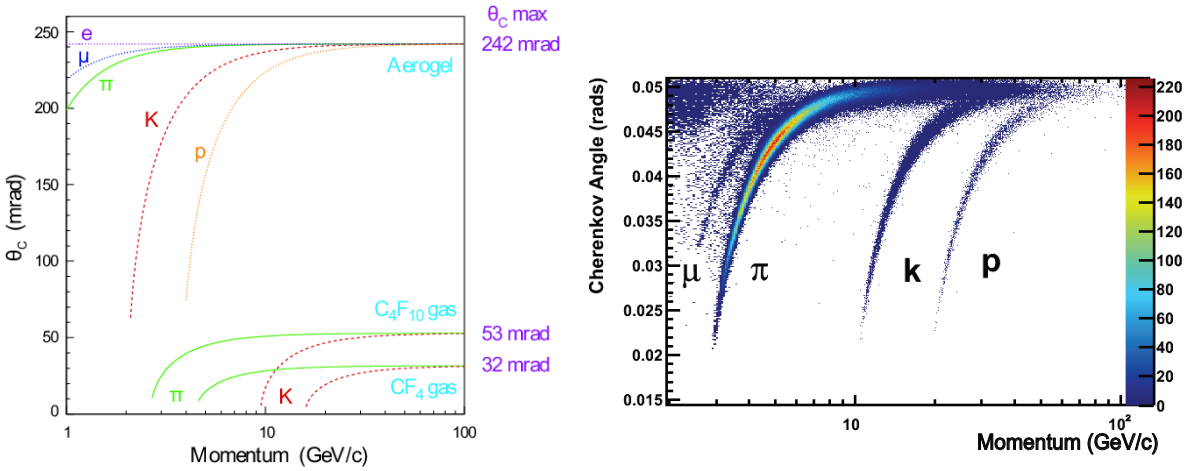


Figure 1.11: Theoretical evolution of the Cherenkov angle θ_c with the momentum for all particle species in the radiators employed by the LHCb RICH detectors (left) and its measurement in the RICH2 detector for 2010 and 2011 data only considering reconstructed tracks producing isolated Cherenkow rings. Figures from Refs. [24] and [44], respectively.

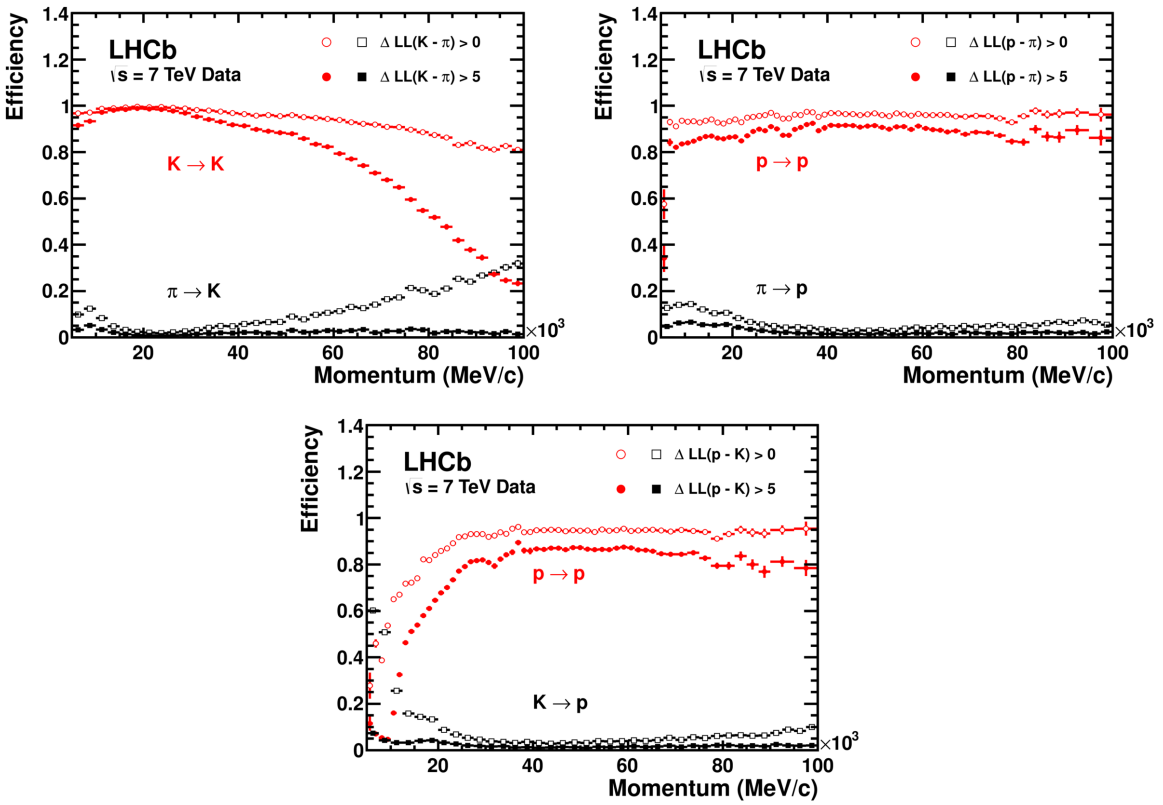


Figure 1.12: PID efficiency and misidentification rates for the discrimination of kaons from pions (top left), protons from pions (top right) and proton from kaons (bottom) as a function of the momentum. For all plots, the efficiency and misidentification rates are compared considering two different applied thresholds on the respective DLL_{h_1,h_2}. Figures from Ref. [44].

and $p \leq 17.7\,\mathrm{GeV}/c$, respectively. The main figures of merit parameterizing the RICH performance are:

- the PID efficiency, defined as the fraction of particles correctly identified;
- the misidentification rate, i.e. the number of particles assigned to the wrong mass hypothesis;

Examples for both are presented in Figure 1.12 [44] with two applied thresholds on the relevant DLL_{h_1,h_2} variable. Combining the information from the two RICH detectors, protons are distinguished from kaons and pions in the full momentum range, while a steep decrease in the kaon PID efficiency is found. Integrating over the full momentum range, the kaon PID efficiency with the looser threshold is measured as $\sim 95\%$ with a pion misidentification rate of $\sim 10\%$, as $\sim 85\%$ and $\sim 3\%$ with the tighter one [44], respectively.

The Calorimeter system

Within the calorimetric system, particles are identified through the characteristics of the showers they produce when interacting with the detector material. It is composed

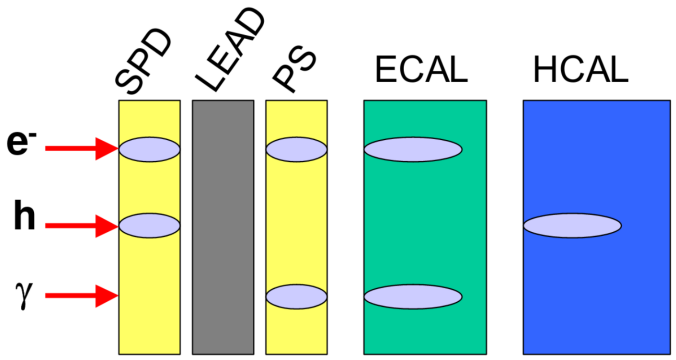


Figure 1.13: Scheme for the shower development in the LHCb calorimetric system for different particle hypotheses. Figure from Ref. [45].

of two scintillator planes, the SPD and the PS separated by a lead converter, and two calorimeters, ECAL and HCAL. The system is intended to:

- distinguish among the hadron, $e^{\pm}$, π^0, γ hypotheses relying on the different induced showers, as schematically represented in Figure 1.13.;
- measure the particles transverse energy, used in the hardware data trigger;
- efficiently reconstruct π^0 and γ particles.

The PS and SPD detectors [45] are two scintillating planes. The former is used to distinguish photons from charged-particles-induced showers and facilitates the match with the information from the tracking detector; the latter discriminates electromagnetic and hadronic showers.

The ECAL detector is a sampling calorimeter of *shashlik* type. To measure the longitudinal transverse shower development and to optimize the separation between neighbour showers, each calorimeter layer is segmented in the transverse direction. The cell dimensions increase with the distance from the beam because of the lower occupancy in the detector and are 4.04×4.04 cm^2, 6.06×6.06 cm^2 and 12.12×12.12 cm^2 in the inner, middle and outer region, respectively. The achieved resolution on the energy measurement is [46]

$$\frac{\sigma_E}{E} = \frac{(9.6 \pm 1.4)\%}{\sqrt{E\,[\mathrm{GeV}]}} \oplus (3.7 \pm 0.1)\% \oplus \frac{(395 \pm 30)\ \mathrm{MeV}}{E\,[\mathrm{GeV}]} \tag{1.8}$$

indicating $\oplus$ the sum in quadrature $a \oplus b = \sqrt{a^2 + b^2}$.

Also HCAL is a sampling calorimeter but, differently than ECAL, employs alternating iron and scintillating tiles oriented in parallel to the beam axis to improve the angular resolution. A transverse segmentation is also available, with two cell sizes of 13.13×13.13 cm^2 and 26.26×26.26 cm^2 in the inner and outer regions, respectively. The total HCAL thickness is not enough for the full confinement of the induced hadronic shower, and the consequent achieved energy resolution is:

$$\frac{\sigma_E}{E} = \frac{69\%}{\sqrt{E\,[\mathrm{GeV}]}} \oplus 9\%. \tag{1.9}$$

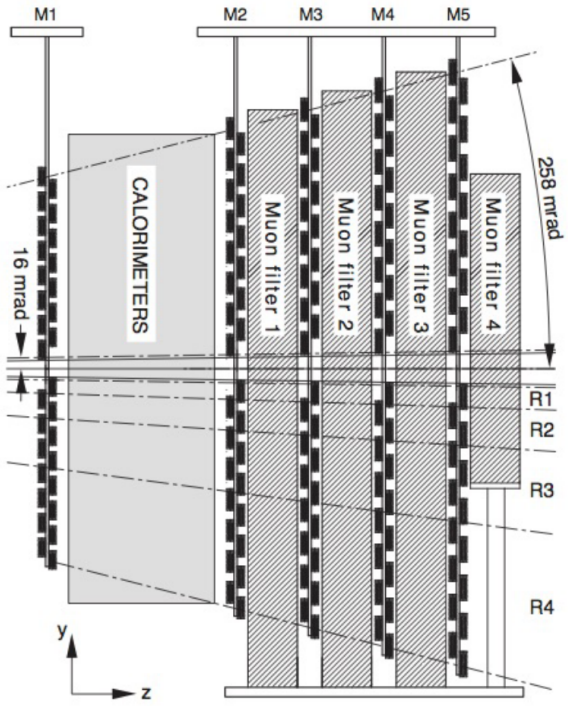


Figure 1.14: Scheme for the muon chambers location in the LHCb detector. Figure from Ref. [24].

The MUON system

The LHCb detector is completed by five stations for the muon identification and reconstruction [42], named from M1 to M5. The first one is located upstream the calorimeters and equipped with Electron Multiplier (GEM) detectors in the central region and with Multi-Wire Proportional Chambers (MWPC) elsewhere. It is used to improve the early muon transverse momentum p_T measurement, achieving $\sim 20\%$ resolution with the only muon stations, that is used in the hardware trigger. The other stations are installed downstream HCAL, equipped with MWPC detectors, and 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. The angular acceptance of the chambers is [20,306] mrad ([16,258] mrad) in the horizontal (vertical) plane, while the minimum momentum required to muons to cross all the chambers is $p = 6$ GeV/c.

The computation of the MUON likelihoods follows from performing a loose binary selection of muon candidates, called `isMuon` and implemented via FPGA. 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 [47].

For those particles that have passed the `isMuon` criterion, we compute D^2 defined as the average squared distance significance of the hits in the muon chambers with respect to the linear extrapolation of the tracks from the tracking system. The average distance

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.

significance is defined as follows:

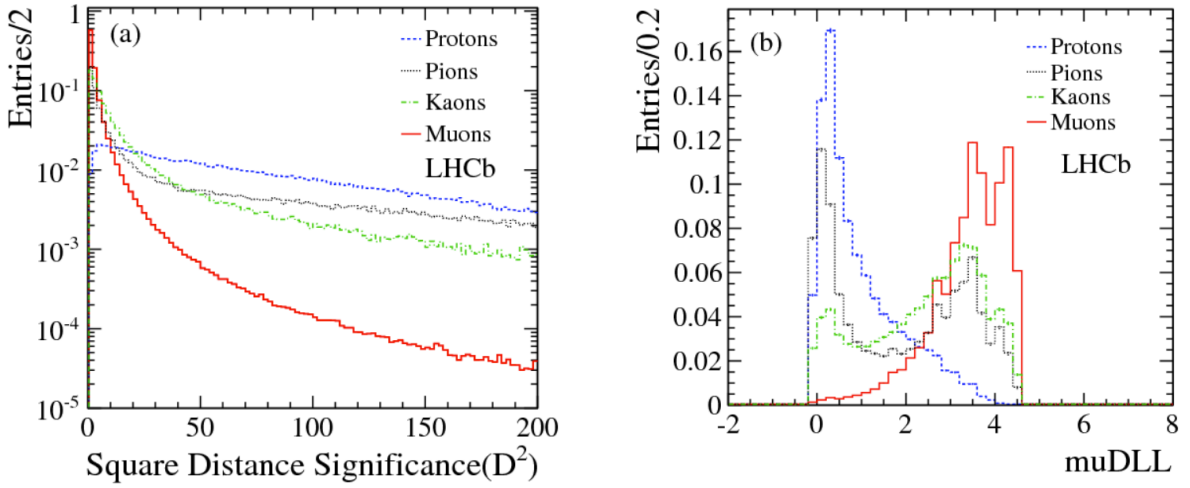
$$D^2 = \frac{1}{N} \sum_{i=\mathrm{M2...M5}} \left[\left(\frac{x^{(i)}_{\mathrm{closest}} - x^{(i)}_{\mathrm{track}}}{\mathrm{pad}^{(i)}_x} \right)^2 + \left(\frac{y^{(i)}_{\mathrm{closest}} - y^{(i)}_{\mathrm{track}}}{\mathrm{pad}^{(i)}_y} \right)^2 \right] \tag{1.10}$$

where the index i runs over the the total number of stations (denoted by N) containing hits within the FOI, $(x, y)_{\mathrm{closest}}$ are the coordinates of the closest hit to the extrapolated track point $(x, y)_{\mathrm{track}}$ 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 1.15a, 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 1.15a. Instead, the likelihood for the non-muon hypothesis `MuonBgLL` 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 `MuonBgLL` is used as discriminating variable and is called muDLL. Its distribution, for different particle hypotheses, is reported in Figure 1.15b. The resulting muon identification efficiency and hadron misidentification probabilities depend on the momentum. With the combined likelihood approach, average values ∼93% with a hadron misidentification rate below 0.6% are achieved [47].

Global particle identification

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 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 $\Delta\mathrm{LL}_{comb}(X - \pi)$, where X represents either the electron, muon, kaon, or proton mass hypothesis [34].

The second approach relies on machine learning algorithms to define a multivariate 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 ANNPID, was implemented as *feed-forward neural network* (FNN) with a single hidden



(a) (b)

Figure 1.15: 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. Figure reproduced from Ref. [47].

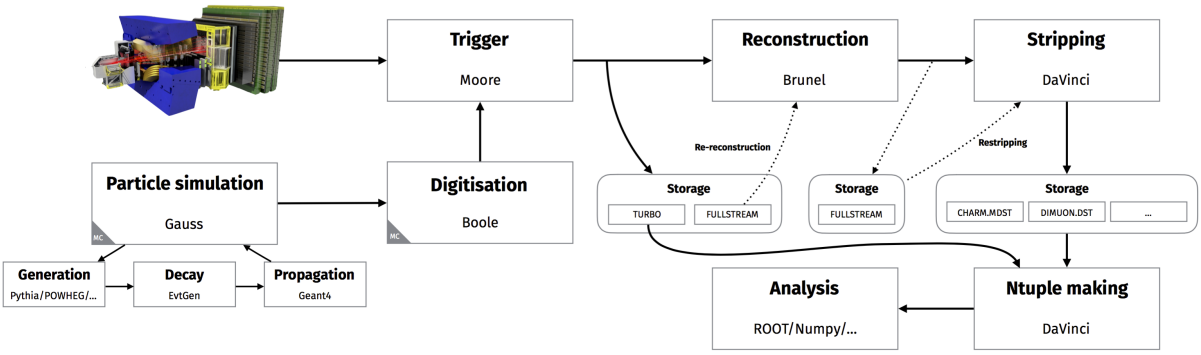


Figure 1.16: Scheme for the LHCb data processing sequence. Figure from Ref. [54].

layer activated through a sigmoid function, by relying on the TMVA toolkit [48]. Trained to perform particle identification, it is also called `ProbNN` and combined the information provided the variables listed in Table 1.3.

1.2.3 Data trigger and processing

Raw detector signals need to be processed [50–52] to reconstruct high-level objects like tracks, vertices and particles and to select events with interesting physics signatures. A graphical summary of the LHCb data processing sequence running within the GAUDI framework [53], with the names of the software applications dedicated to each step, is presented in Figure 1.16.

Early detector information is exploited online to discard the events that do not contain signatures for interesting physics signal. The trigger, as illustrated in Figure 1.17, is composed of three stages for both LHC Run 1 and Run 2. The first stage, called Level 0 (L0), is implemented on dedicated custom FPGA cards and runs synchronously

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 [39]
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. [49].

with the LHC bunch-crossing rate with a fixed 4 μs latency. Based on the information from the Calorimeter and MUON systems, the L0 trigger reduces the data from 1 Tb/s originating from the 30-million per-second visible collisions to $\sim$ 1 MHz. For the events accepted by at least one of the L0 trigger selections, the information from all detectors is merged by dedicated acquisition boards and transferred to the software trigger. As presented in Figure 1.17, in both LHC Run 1 and Run 2, this comprises two stages and is controlled by the Moore application with the output data rate for the two periods indicated. In the former, HLT1, a partial event reconstruction is performed, including the finding of tracks and the muon PID algorithm. Based on these objects, inclusive trigger lines, i.e. selection algorithms dedicated to generic one or two-body signatures rather than a specific physics channels, are defined. As an example, to distinguish b- and c- from light-quark hadrons, which is crucial in the core physics program of the LHCb experiment, the main signatures are the displacement of the decay vertex and the large transverse momentum of the decay products. The HLT2 trigger stage completes the event

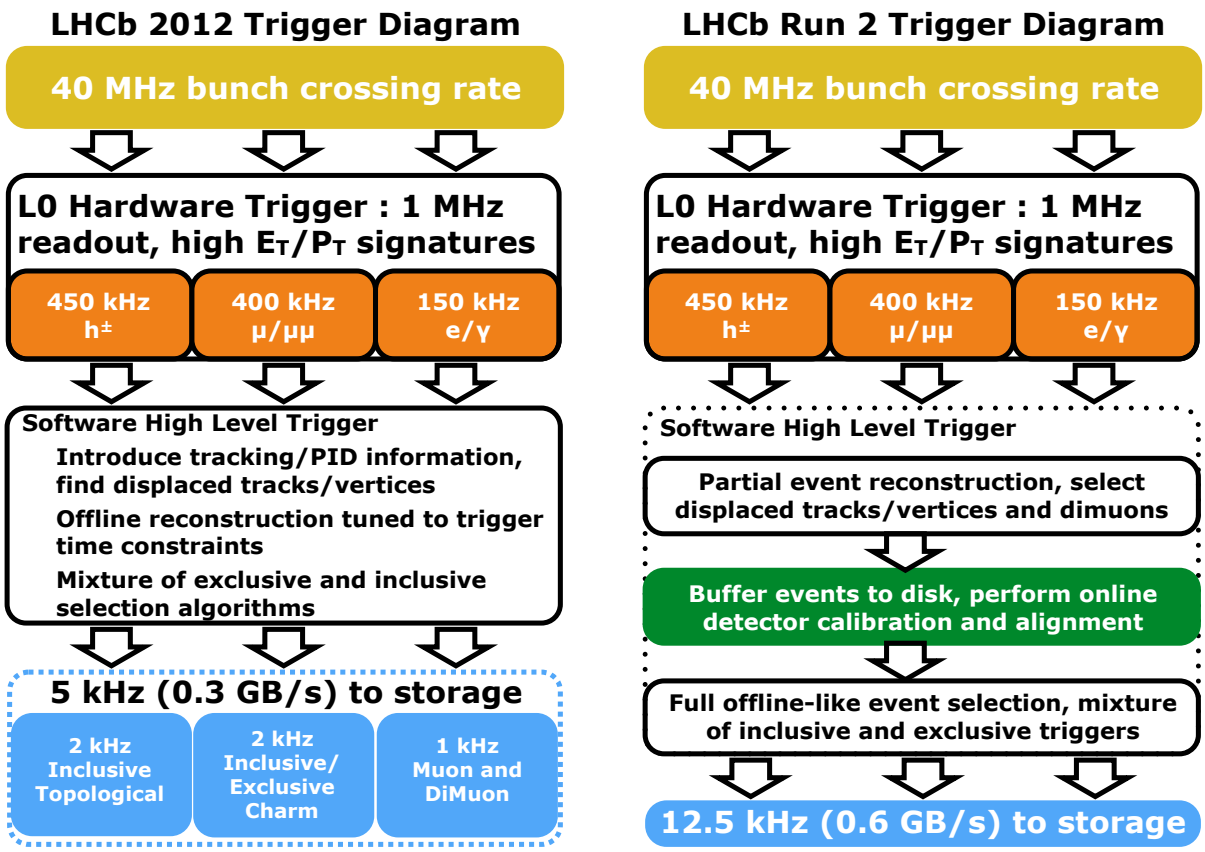


Figure 1.17: Schemes for the LHCb data trigger software during the LHC Run 1 (left) and Run 2 (right) periods of data-taking. Figure from Ref. [55].

reconstruction with a more accurate fit of the reconstructed tracks and the decoding and processing of the Calorimeter and RICH systems. With the complete description of the reconstructed events, both inclusive and exclusive trigger lines are acquired.

Raw data of the events that are not discarded by the trigger are distributed through the Worldwide LHC Computing Grid (WLCG) [56], and reconstructed with an application called BRUNEL making use of the calibration and alignment constants that are progressively updated as a result of the continuous improvement in the understanding of the detector. The produced output is called `Full` stream and contains all the reconstructed objects describing the collision. This could be in principle used for physics analyses but, considering the needs of the whole LHCb Collaboration, is prohibitive in terms of computing resources. Instead, a further data reduction step, called *stripping*, is typically applied. This corresponds to a set of algorithms implemented offline to only select and save on disk the data that are relevant for physics analyses. The output of the *stripping* is finally available to analysts and is processed with the DAVINCI application to produce an output data format that can be easily processed and analyzed. With the start of the Run 2, a more efficient paradigm for data-taking only reconstructing signal candidates that would be selected by the stripping, called `Turbo` [57], was conceived and developed, providing offline-quality reconstructed objects in real time.

1.3 Detailed and fast simulations

1.3.1 The simulation software: Gauss, Gaussino, and Boole

To obtain physics information from the acquired data disentangling detector-induced effects from genuine physics phenomena is of primary importance. This is one of the main tasks for which a highly reliable simulation is crucial for a High Energy Physics (HEP) experiment.

Historically, computer-based simulations of the HEP experiments started to be developed well in advanced with respect to the construction of the detectors. The main objective, at the time, was to develop a synthetic, virtual version of the detector to define the specifications on the detector construction and on the data acquisition pipeline to ensure the physics goal of the experiment.

To serve to this purpose, the simulation must be designed from first-principle models of the collision between accelerated protons, of the subsequent hadronization, of the decays of the unstable and quasi-stable particles, and finally of the interaction of the radiation with matter. Simulation frameworks were developed by the LHC collaborations to implement these steps for the particular geometry and technological choices of the respective communities. At the same time, the physics models describing the hadronic collisions and the radiation-matter interaction are common to the four experiments, and are relevant to an even wider community spanning from researchers studying cosmic rays to nuclear medicine applications. Hence, the physics models are grouped in common software packages on which the Simulation frameworks of the four experiments rely.

Some of the most important dependencies in the category of physics models are:

- *Pythia8* [25], describing the collision of the protons accelerated by the LHC and the subsequent hadronization of quarks and gluons in baryons and mesons;
- *EvtGen* [58], defining the decay models for heavy hadron decays also implementing data-driven models capable of reproducing accurately complex decay patterns, in particular for semileptonic decays, not fully reproduced by purely theoretical models;
- *Geant4* [59, 60], implementing the models describing radiation-matter interactions enabling to predict the ionization energy deposited by traversing particles in the detector material.

On top of these libraries, the experiments develop their own simulation framework. For example, within the LHCb Collaboration, these packages are glued together by the Gauss [61] application which is consistently integrated within the LHCb data flow, as depicted in Figure 1.16.

The Gauss application is organized to combine in a single application four different aspects related to the simulation:

- the *Generation*, that relies on Physics Generators to predict the outcome of a proton-proton or ion-ion collision;
- the *Geometry*, that enables describing the different geometrical configuration of the material constituting the detector and providing a desired or undesired target to the crossing particles;

- the *Simulation*, that is specific of the representation of the interaction of radiation with matter;
- the *Persistency*, that enables to store the data with a serialization protocol common to the other LHCb applications and easy to read and process in the subsequent steps of the data processing.

In recent year, several efforts started to share between experiments also the implementation of these frameworks, decoupling the implementation of experiment-specific geometry from an experiment-independent detector description language, and the serialization format from the Event Model representation used internally by the framework. The most popular among these experiment-agnostic frameworks is GAUSSINO [62] that, started as a refactoring of the GAUSS project, is now an independent project serving multiple experimental communities [63] and becoming an external dependency for GAUSS [64].

Independently on whether relying on GAUSSINO or not, the GAUSS application produces generated data using the same serialization format as the other LHCb applications, but its output is not ready to the reconstruction software. Indeed, after the computation of the energy deposits performed by GAUSS, an additional layer of simulation is needed to implement the electronic response and parameterizing the electronic noise on the various detector channels. This last step in the simulation data flow is named *Digitization* and is demanded to a different application, named BOOLE.

Once digitized, the simulated data provides identical format to the raw data obtained by a run of the LHCb experiment so that the same reconstruction software can be used to process the two kind of datasets. In addition to the digitized information, the simulated datasets include what in jargon is named the *Monte Carlo truth*, representing the generator-level information before any effect related to detection and digitization is applied. The Monte Carlo truth is propagated through all the subsequent steps of the data analysis to retain the ability of reconstructing the generated events, greatly improving the ability of the analyst to identify effects in the reconstructed quantities induced by the detector or by the reconstruction algorithms.

1.3.2 Sustainability of the detailed simulation

The LHCb Simulation framework, designed before the construction and providing impressively accurate prediction of the detector response event before the first collisions, has been object of an intense work of optimization to further improve the reliability of the simulated samples used to interpret the acquired data. In addition, the maintainer of the libraries implementing the physics models and the theory community at large, have translated the physics results obtained by the LHC experiments into improvements to the effective models used in the generators and in the GEANT4, providing a further, extremely precious contribution to the quality and reliability of collision simulation.

Unfortunately, the simulation based on PYTHIA8 and GEANT4 is extremely expensive in terms of computing resources. The simulation of the high-energy proton collisions requires to reproduce the thousands of intermediate particles resulting from the interactions and that give origin to large graphs, where nodes represent elementary interactions and edges represent the intermediate particles. In addition, these generators implement the physics models with Monte Carlo techniques, by drawing random numbers from complex multi-variable joint probability distributions, often defined by the squared modulus of

a complex wave-function. Conditioning those distributions is of prohibitive complexity, so that, to generate a particles through a specific physics process, it is customary to just rerun the generator multiple times until the physics process, by chance, is generated and continue the simulation for that specific event. While simple and robust in terms of induced biases, this approach is not particularly efficient from a computing point of view, making the generation of specific physics processes relatively slow.

Nonetheless, little effort is being spent by the community on alternative techniques to condition the generators because the computing resources needed by even the most ineffective generator is almost negligible in front of the computational cost of the radiation-matter interaction. When high energy particles interact with matter they have a large chance of transferring a large amount of energy to an atomic electron. If the energy is sufficient, the electron may be expelled by the atomic orbitals (ionization) and becoming an ionizing radiation itself. This cascade results in an exponential growth of the ionizing particles that a simulation framework must track to compute the energy of the original particle that gets finally deposited in a given region of the detector. Similar exponentially growing particle shower are caused by hadronic interactions of the high energy particles with the atomic nuclei. In this case, the physics models to describe the interaction are even more complex and the computation of *hadronic showers* may be as computational expensive as that of the aforementioned *electromagnetic showers*. In the LHCb experiment, also the RICH detectors require large computational power to be simulated, as each charged particle traversing them may emit Cherenkov photons, but also deposit ionization energy in the Cherenkov radiator, generating high-energy secondary electrons that will start emitting Cherenkov light as well. Tracking each photon through the complicated geometry of curved mirror projecting the Cherenkov radiation on the light sensors of the RICH system is responsible for a relevant fraction of the time spent to simulate collision events.

After the successes of the first two Run of the LHC experiment, notably achieving the discovery of the Higgs boson in 2012 [8, 9], the LHC is undergoing a major upgrade to increase the rate of proton-proton collisions by a factor ten with respect to the design value of the LHC. The upgraded accelerator will be name *High Luminosity Large Hadron Collider* (HL-LHC). As a consequence, the LHC collaborations will have the ability of accessing rarer decays and perform precision measurements on a larger number of physics phenomena that have been just discovered using the already-collected data. To perform those new analysis and to enter the precision regime for such new decay modes, a larger amount of simulated samples will be required. Statistical models were built by the collaborations trying to observe correlation patterns between the integrated luminosity of data samples collected in a given period and the number of simulated samples requested by the data analysis experts to perform the analysis. The statistical model was then used to build forecast of the simulation requests expected for the experiments at the HL-LHC. The resulting predictions are reported in Figure 1.18.

The unit for CPU request used in the Figure and, in general, by the HEP community is the HEP-SPEC 2006, or HS06, which corresponds to a standard benchmark of CPU performance measured on a set of HEP-specific applications in 2006. By definition, the HS06 measures an amount of work and is therefore independent of the CPU architecture provided by different sites of the WLCG. As a consequence, the exponential decrease in the cost of computing is reflected in a decrease of the cost of each HS06. To convert the HS06 in a quantity easier to interpret for the reader, I report that the National Institute for Nuclear Physics, estimated a cost of 10 €per operated HS06 in 2024. The cost includes

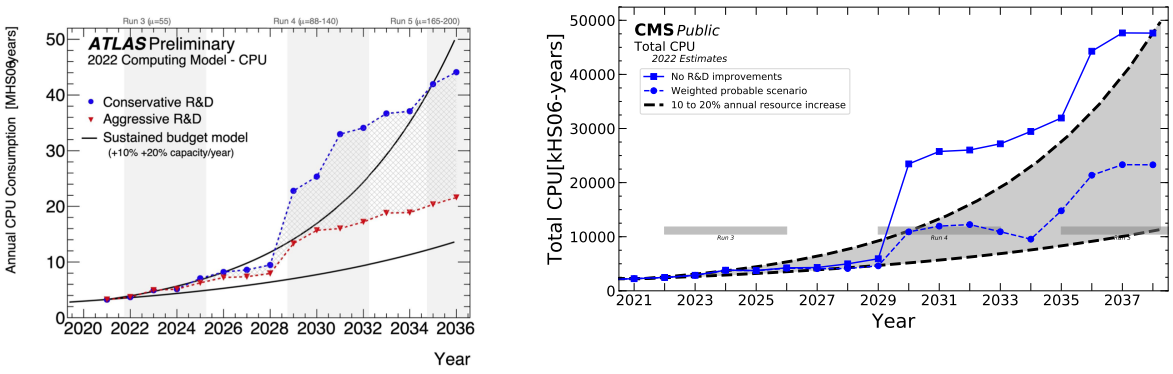


Figure 1.18: Predicted requests of computing power from the two major experiments at the LHC for the upcoming years. The predictions for ATLAS and CMS are reported on the left and on the right, respectively. The sudden increase in the requests during the years 2027–2030 is aligned with the planned start of the High-Luminosity LHC. The CPU requests are dominated by Simulation. The Figures were reproduced from Ref. [65], and Ref. [66], respectively.

the acquisition of the CPU, of the servers, the maintenance, and personnel costs. Similar estimates apply for other countries. Applying the conversion factor to the vertical axes of Figure 1.18, we conclude that the cost of CPU resources for the next Run of the LHC is of the order of several million euros. The budget considered as sustainable by the government agencies funding the LHC program are marked as black lines for an optimistic and a less-optimistic scenario, highlighting that even in the most richest future, the cost of CPU processing, dominated by the simulation, must decrease by at least a factor two to be affordable.

The situation for the LHCb requests is being updated very often at the time of writing because the LHCb Collaboration anticipated a major upgrade of the detector to LHC Run 3. Because of an incident with primary vacuum of the vertex locator happened in 2023, the data taking has been postponed to 2024 and new data are therefore expected to start flowing soon. In any case, a similar sudden growth to that observed in the ATLAS and CMS plots is expected on an earlier schedule for LHCb, though it is not clear whether it will happen in 2024 or in 2025, or in the most pessimistic scenario if it will overlap with the restart of the LHC in 2029.

1.3.3 A bird's eye view on the fast simulation options

Multiple efforts are ongoing within the HEP community aiming at a reduction in the cost of the detector simulation. To distinguish the various options to reduce the cost of the simulation from the simulation approach discussed so far, we will refer to the latter as ***Detailed Simulation***. In the past the wording *Full Simulation* was preferred and can still be found in some references, but they indicate the exact same approach.

While details may introduce very important practical differences, all techniques to reduce the cost of the simulation can be ascribed to two main categories:

- Methods replacing part of the GEANT4 computations with results obtained otherwise, either reusing previously simulated events, or parameterizing the output expected by

GEANT4. These methods aim at producing a simulation output identical to those obtained with the Detailed Simulation, so that they do not perturbate the subsequent steps in the data flow, and all the analysis-level variables defined for the Detailed Simulation will be made available by the reconstruction algorithms obtained with these methods. However, depending on the method, the probability distributions of the reconstructed analysis-level features may be drastically different from those of the Detailed Simulation. For example, belongs to this category the trivial method of switching off the interactions in part of the detector. The analysis-level quantities will be computed for the missing detectors, but they will correspond to empty events. Methods in this category are generically labeled as ***Fast Simulations***.

- Methods replacing the whole simulation step, and sometimes the subsequent digitization and reconstruction steps with parameterizations. Historically these methods played a major role in leading order studies to determine the geometry of completely new detectors, or even to quickly rule out experimental ideas as technically impossible before any further investment. Traditionally, parameterizations were provided by analytical formulae or simple histograms while today they are being replaced more and more by non-parametric techniques such as neural networks or boosted-decision trees. Since calling *Parametric Simulation* a simulation based on non-parametric models sounds clumsy or worse, multiple branding attempts were made calling methods in this category *Rapid* or *Ultra-Fast Simulations*. In this Thesis I will adopt the wording ***Flash Simulations*** introduced for the first time by Italian members of the CMS Collaboration [67].

The different methods belonging to the same or to the two categories are not necessarily competing with each other. As mentioned in the introduction, today the samples obtained by means of the Detailed Simulation are used for a wide pletora of different tasks, spanning from the tuning of the reconstruction algorithms to feasibility studies before starting a new challenging data analysis. The ambition of identifying a fast or flash simulation option adequate for all the needs is utopian, while providing a palette of simulation options the analyst can choose from might lead to an effective reduction of computing costs. For example, an analyst may request a small amount of events from Detailed Simulation to validate the selection efficiency and the projections of some physics quantities, and much larger amount of events from Flash Simulation to train an efficient multivariate classifier to reject backgrounds without fears for over-training while trying to let the classifier to exploit non-trivial correlations.

Other potential applications for Flash Simulation include the construction of statistical models for the resolution of reconstructed quantities for decay modes as a function of the kinematics of the particles involved in the reconstruction, or the bulk simulation of a large cocktail of different decays modes to check for potential peaking contributions in a search.

In the next Sections, I will review some of the most important or promising development to cope with the projected limitations to the computing budget, more emphasis will be given to the options explored or adopted within the LHCb Collaboration.

1.3.4 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-modeling of the detector response, with underestimated resolution effects and overestimated reconstruction efficiencies [68].

ReDecay approach mitigates these problems simulating only the signal process and re-using 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 [68]. 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 [68].

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[7] (for additional detail refer to Ref. [68]).

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 [69].

1.3.5 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. 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 [70].

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 [70].

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[8], 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 [70].

[7] 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.

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

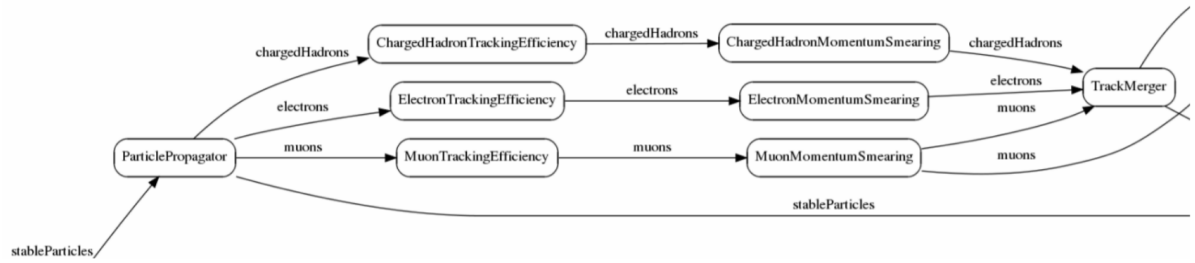


Figure 1.19: Scheme of the DELPHES data flow through the tracking system modules.

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 [70].

Software implementation

The DELPHES software is a modular framework written in C++ and based on the ROOT analysis framework [71]. 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 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 1.3.5 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 [70].

1.3.6 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: modeling 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 [72].

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. [72]).
RapidSim utilises the Root software package [71] 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[9] 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 [72].

[9]The simple design allows users to extend RapidSim to include alternative detector geometries.

2 Technologies for fast and scalable computing

The landscape of modern computing has been largely shaped by Moore's Law, a principle positing that the number of transistors on a microchip doubles approximately every two years, leading to an exponential increase in computing power. However, as we venture deeper into the 21st century, we are increasingly confronted with the physical limitations that prevent further miniaturization of transistors due to intrinsic and unavoidable issues, such as heat dissipation and quantum effects. As a result, the relentless pace of Moore's Law is being challenged, prompting a paradigm shift in computing architectures and algorithms. This chapter aims to explore these challenges and potential solutions in the context of modern computing (Section 2.1), exploring in particular parallel (Section 2.2) and Cloud computing (Section 2.3). Particular emphasis will be given to the problem of training complex Machine Learning models (Sections 2.4 and 2.5) in a distributed, multi-cloud environment (Section 2.6).

2.1 Overview on modern computing

Experimental High Energy Physics (HEP) has a long and glorious tradition of shaping the landscape of computing resources [73–75]. Indeed, as soon as the detector became readable with electronic devices, they quickly became digital data, stored and processed with computing technologies. The technological evolution of computing during the last three decades, however, has changed significantly the analysis techniques and the practices for storing, handling, and processing data. Sometimes, walking through the theorists' floor of the Physics Department, one may hear discussions on re-analyses of the whole dataset collected by the Large Electron Positron (LEP) experiments, in operation at CERN between 1989 and 2000 and famous for collecting the world's most precise and abundant datasets on the vector bosons $W^{\pm}$ and Z^0. The whole dataset of LEP, motivating at the time the construction of a world-leading computing infrastructure, can today be stored on a pen drive and processed on a common laptop.

Indeed, the computing power made available to researchers has increased exponentially during the last decades, roughly following Moore's law predicting that the computing power per chip doubles approximately every two years. In recent years, however, such exponential growth has been challenged by the physical size of the silicon lattice, with lithographic technologies capable of engraving transistors of few nanometers, and with

cooling challenges due to the increasing amount of power dissipated by the devices per unit of surface [76].

These challenges require huge scientific and technological efforts to be tackled and, to mitigate the slowdown in the exponential growth of available computing power, engineers have started exploring alternatives, notably *parallelizing* the computing workloads on multiple chips. While guaranteeing a stable exponential decrease in the cost per floating-point operation (FLOP), to profit from this additional computing power, the software development techniques and technologies have become more and more complicated and motivated the introduction of new programming languages and paradigms.

Nonetheless, if a computing workload is designed to run in parallel on multiple devices, it may become easy to scale the number of concurring processes on a variable number of processors, as long as the computing infrastructure has been properly designed. This reasoning, in the context of Experimental HEP, gave origin to the *Worldwide LHC Computing Grid* (WLCG) [56], a project aiming at a global collaboration of around 170 computing centers in more than 40 countries to share computing resources and provide the computing power to process the hundreds of petabytes of data produced every year of operations from the Large Hadron Collider (LHC) experiments [77]. In more recent years, with the increasing amount of available digital data relevant to commercial applications, an alternative approach towards scalability, named *Cloud*, was introduced and connected to the concept of web service. If the Grid is conceived to provide scalability to *High-Throughput Computing* (HTC) applications accessing and processing a large amount of static data, the Cloud provides scalability to the access of services, such as websites or web applications, to the public. The strategic importance of Cloud infrastructures for companies and the competition among big players[1] resulted in a flourishing of future-proof standards with commercial support and documentation, and attracted large communities of users developing and supporting open-source alternatives to the best commercial products.

A notable example of such a transition is provided by the *containerization* of applications. While possible since the early eighties, the practice of encapsulating applications together with all the dependencies except the kernel became common with the introduction of Docker [78] in 2013 and the Open Container Initiative (OCI) standard in 2015 [79]. Containers have impressed a new pace to the development of applications, providing a tool for versioning software stacks, and automated testing on multiple platforms and environments. With time, it became common to design complex software stacks composed of multiple containerized applications orchestrated by some higher-level tool, such as `docker-compose`[2] or *Kubernetes*[3]. Applications designed with such techniques are often easier to scale with respect to monolithic applications since the provided orchestrators are designed to easily increase the computing power for some sets of containers, resulting in a wiser usage of compute and network resources. Today, containers are ubiquitous in any Cloud environment and represent a fundamental building block for the vast majority of the web applications accessible through the Internet.

Such an evolution in distributed computing has been receiving attention from the scientific communities, including those involved in Experimental HEP, and today the Grid and Cloud approaches co-exist and often overlap in most of the computing centers

[1]Among the major Cloud service providers there are Amazon with AWS, Microsoft with Azure, and Google with GCP.

[2]`https://docs.docker.com/compose`

[3]`https://kubernetes.io`

supporting scientific data processing.

Also in terms of hardware, the great demand for computing power from the commercial consumers motivated a great effort to optimize the processors. The instruction sets of CPUs have been greatly enhanced to include operations between vectors and to ease vector-matrix products with processor-optimized libraries, such as Intel oneMKL[4] and oneAPI[5]. The increased power consumption associated with the introduction of these technologies opened the gate to competitors focusing on low-energy processors, such as ARM, which are found to be perfectly suited for running the simulation and reconstruction code of HEP experiments [80], despite the different instruction set they are based on. At the same time, the exponential growth in the entertainment and video game industries, motivated a fast development of *Graphics Processing Units* (GPUs) to elaborate images, represented as large matrices with strong spatial correlation, on a separate, dedicated hardware accelerator. The same operations used in GPUs to display fancy images on a screen during a game session can be used to decode an image to extract digital information with unprecedented speed and can be tweaked to process acoustic waves and recognize spoken words or even to study correlations between the words occurring in a sentence. More recently, commercial computing providers started to serve algorithms on lower-consumption devices explicitly designed for the purpose and synthesized on *Field Programmable Gate Arrays* (FPGAs), providing yet another flavor to hardware acceleration.

Unfortunately, the evolution of storage technologies is not sufficiently fast to keep the pace of computing in terms of cost reduction. Despite the effort of using magnetic tape as much as possible, the most important technology for managing large amounts of data relies on magnetic disks, as in the last decades. Figure 2.1 shows a comparison of the evolution of the cost per FLOP and terabyte of data, highlighting a faster reduction of the former. In practical terms, such a difference results in a shift in the computing models of the various HEP experiments, moving more and more complex processing closer to the detector to discard immediately the largest fraction of uninteresting data, making better usage of the expensive storage resources. Such a shift in the computing paradigm sets unprecedented challenges for scientific Collaborations requiring precise planning of the data analysis steps well ahead of data taking. In such a scenario, a fast, reliable, and tunable simulation, obtained from mixed Cloud and Grid resources is crucial to the success of future HEP experiments.

2.2 Parallel computing

As mentioned above, *parallel computing* has become the solution to increase the computing power provisioned to applications beyond the physical constraints on frequency and power dissipation by dividing the task into multiple procedures that can, or must, be run at the same time. Within this rather wide definition, the concept of parallel computing can be implemented in many different ways and on several different scales. Since, in most cases, multiple parallelization techniques can be adopted in the same application, it is

[4]Intel oneMKL is a library of optimized math routines for science, engineering, and financial applications. Read more on `https://en.wikipedia.org/wiki/Math_Kernel_Library`.

[5]oneAPI is an open standard, adopted by Intel, for providing a unified interface to be used across different computing accelerator architectures, including GPUs, AI accelerators and FPGAs. Read more on `https://en.wikipedia.org/wiki/OneAPI_(compute_acceleration)`.

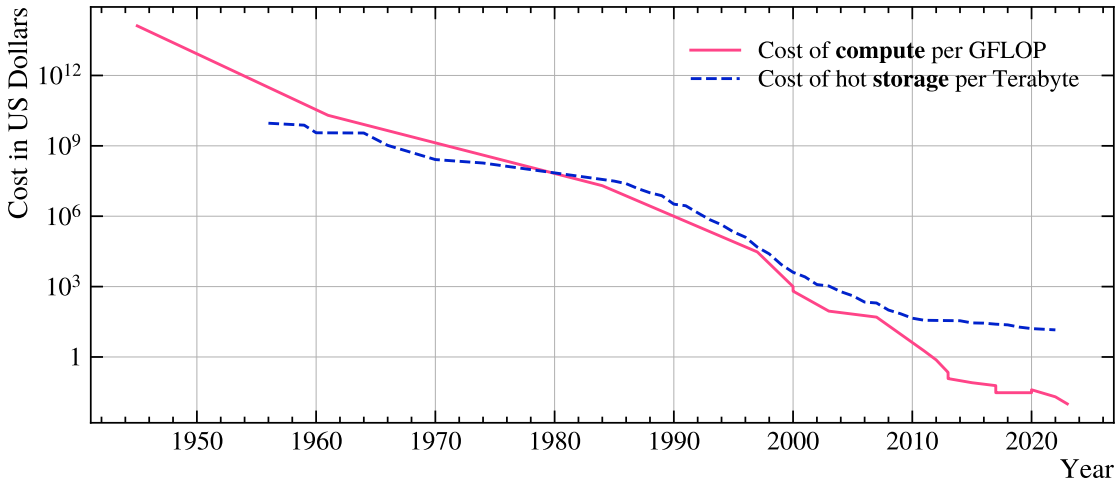


Figure 2.1: Historical average cost for storing (blue line) and processing (red line) data. The information is taken from `https://hblok.net/blog/storage` and `https://aiimpacts.org/wikipedia-history-of-gflops-costs`, respectively.

common to depict the parallelization options as a multi-dimensional space, where each axis represents a given direction in the parallelization and the hyper-volume corresponding to a given configuration can be read as the maximal, theoretical speed-up achievable by the application. For example, if an application can run on two computers in the same room, but also on two distinct data centers, then the hyper-volume of the configuration, 2×2, corresponds to a maximal speed-up of factor 4 with respect to the serial application. Complications on data management, non-parallelizable portions of the application, and requirements of synchronization between the processors may significantly reduce the theoretical speed-up. Depending on the application and the surrounding infrastructure, some configurations may be less expensive than others while providing the same effective speed-up.

Referring to the formulation of the parallel paradigm as a multi-dimensional space, the configuration dimensions may vary depending on the perspective of the author, but they can usually be ascribed to the following seven cases:

1. **Multi-site processing.** Conceptually very simple, it relies on the share of workloads on machines belonging to *different computing centers*, but it comes with challenging requirements in terms of data management. Notably, input data must be available on the site where the application that accesses it is running, or easily accessible through the network. The WLCG infrastructure responds to this need with a design aiming to maximize the locality of data access, or, in other words, that tries to execute predetermined portions of the applications in those data centers holding copies of the input data. The portions of the applications running on different data centers are usually considered as totally asynchronous and are not coordinated. Indeed, disposing of an orchestration layer connecting applications running on different sites would be extremely expensive in terms of maintenance and would open to important security threats. Recently, multi-cloud solutions relying on VPNs to ensure a secure connection between multi-site applications are being explored [81], but they have not yet been widely adopted for scientific workloads.

2. **Multi-machine (or cluster) processing.** Data management and synchronization become easier if the machines running concurrent portions of the application are physically close to each other and can rely on a private, fast, and local network, by forming a *cluster*. The data center of the WLCG usually relies on batch systems such as HTCondor [82] and Slurm [83] to manage queues of independent jobs, ensuring a fair share between the tenants and effective balancing among the computing nodes. If stronger coordination of the application segments is needed, libraries such as OpenMP[6] are often used to ease the development of applications crossing the borders of a single machine while dealing with the complexity of synchronizing and sharing the data through the processors.

3. **Multi-processing.** From an application development perspective, everything becomes much easier if the processors running different segments of the application reside on the *same machine*. The synchronization between tasks does not need to rely on the network and processors can communicate through signals and even share some read-only chunk of the memory. When applications are designed to process data, multi-processing is usually the easiest solution to achieve parallelization. Libraries such as Snakemake [84] enable building complex graphs of dependencies of different portions of the application and scheduling them in parallel on the available processors, using the local storage resources to pass intermediate results through the workflow. Most languages, including C, C++, and Python, support multi-processing in the standard libraries, making it easy for the application to exploit this dimension of parallel computing even without relying on third-party tools. The most important limitation of the multi-processing approach is that they cannot share chunks of memory that are not made immutable. Applications requiring a large amount of almost-immutable memory (such as the geometrical description of a HEP experiment), need to replicate that amount of memory for each process, often saturating or exceeding the resources available to the node.

4. **Multi-threading.** To overcome the limitations of multi-processing in terms of capabilities to share memory, processors started to support multi-threading and symmetric access to the memory and other resources from multiple cores. On Intel processors, multi-threading is also used to increase the utilization of a single core by simulating two *virtual (logical) cores* for each physical CPU. This technique is called *Hyper-threading* (HTT) and may enable a performance enhancement of up to 30% with respect to the configuration with a single thread per core [85]. To simulate the two virtual cores, HTT relies on the long registers available to modern CPUs, which can be split in two and used to process two independent code segments. HTT provides users with a simple way to profit from long registers and, when the application is designed to take advantage of the instruction sets operating on large registers (i.e. the *superscalar operations* discussed below), they can usually achieve even more significant performance boost. Figure 2.2 reports an example of multi-threading for the GEANT4 [59, 60] application, representing the major consumer of CPU resources for Experimental HEP communities. While extremely beneficial in terms of memory consumption, coding multi-threading applications usually represents

[6]OpenMP is an API that supports multi-platform shared-memory multi-processing programming in C, C++, and Fortran. Read more on `https://en.wikipedia.org/wiki/OpenMP`.

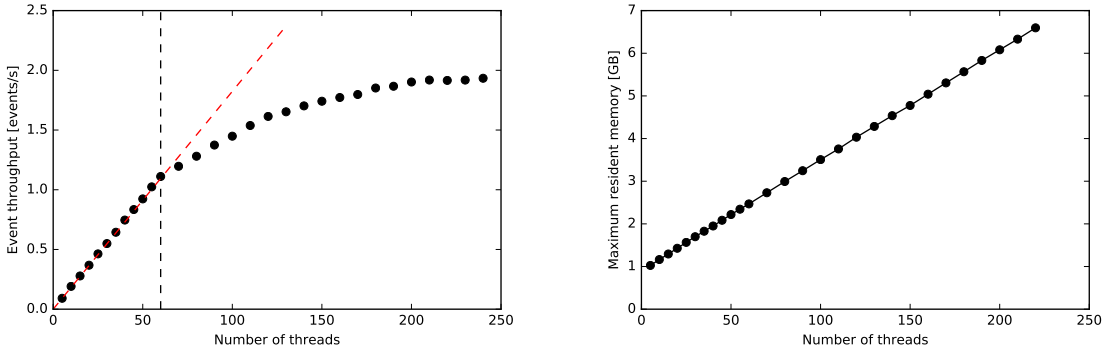


Figure 2.2: On the left, throughput of the GEANT4 application as a function of the number of threads on an Intel Xeon Phi processor with 64 physical cores. Using a number of threads up to twice the number of physical cores (*Hyper-threading*) enables a significant speed up. Beyond 128 threads, computing operations are scheduled during I/O-bound operations, and a further speed-up is enabled. On the right, the memory allocated by the multi-threaded application is shown as a function of the number of threads. The vast majority of the memory is shared between the threads, enabling 200 concurrent executions with less than 10 gigabytes of allocated memory. For comparison, the same concurrency implemented with multi-processing would require more than 200 gigabytes of memory. Figures reproduced from Ref. [91].

a major challenge with respect to multi-processing because concurrent access to the memory may bring the application in an unexpected, non-reproducible state. Nonetheless multi-threading has wide support in most languages and standard libraries exist for example for C++11 and Python. New languages designed to ease multi-threading programming, such as Rust [86], are increasing in popularity. In scientific applications [87–90], the preferred way of benefiting from multi-threading is relying on frameworks that enforce strict read-only constraints on vulnerable memory chunks or providing self-contained multi-thread functions to perform generic operations, such as those implementing tensor algebra or image processing steps. Multi-threading is a challenging but extremely powerful tool to make applications parallel when application segments are strictly coupled. In terms of performance, the major limitation of multi-threading is the latency introduced by *context switching*. Indeed, when switching from one thread to another, the operating system must suspend a sequential program, store the status of the processing (the context) somewhere in the memory, retrieve the context of the new thread, and resume the new process. This may introduce a latency of a few milliseconds making multi-threading unsuitable for parallelizing a large number of very fast code segments.

5. **Asynchronous execution.** While, strictly speaking, not belonging to the category of parallel computing, *asynchronous programming* enables concurrency by executing multiple loosely coupled code segments in the same thread, overcoming the cost of spawning new threads and switching from one thread to another when passing from one code segment to another. Asynchronous applications are common in scenarios that are subject to the latency introduced by accessing external resources (such as magnetic disks or remote services through the network) while managing several hundreds of concurrent operations. Asynchronous programming has its foundation

in the *callback principle*, in which the code segment processing data accessible with some latency is scheduled at the earliest convenience after the data becomes available. A common example of such an application is provided by web servers that may have to handle several hundreds of concurrent requests from the Internet while accessing storage to respond to each request. In such a scenario, a serial application would serve one request at a time, in a sequence, and would access the storage resource independently for each request, resulting in a CPU idling most of the time while waiting for data fetched from the storage. Multi-threading would enable to distribute on multiple threads the code to serve the requests, by effectively using the time a CPU would otherwise idle to submit a request to the storage service. However, for an increasing number of requests, the number of threads may become unsustainable for the operating system and, the time wasted switching from one context to another just to check whether the data has been fetched from the storage may become significant. Asynchronous programming comes to the rescue by providing concurrency in a single thread.

6. **Single-Instruction-Multiple-Data (SIMD) paradigm and vectorization.** Modern processors have large registers with up to 512 bits, that, relying on the appropriate instruction set, can be used to apply the same operation to multiple data in a single calculation. For example, a 512-bit register can hold 16 single-precision floating point values. This means that summing up two floating point numbers, or 16 pairs of floating point numbers has the same computational cost: one clock cycle. To profit from these so-called *superscalar* operations, however, the data must be represented in the computer memory as contiguous blocks of uniform data type, in a way that it is fast for the processor to pick from the memory the 16 values to fill the register in a single read operation. Indeed, the latency introduced by composing the register with floating points scattered in independent locations is so that, as fast as the RAM is, it would completely vanish the benefit of summing them up in a single clock cycle. In such a scenario HTT is usually more profitable. The speed-up granted by *vectorization* is important and often exploits resources that would not be possible to use as efficiently with techniques like Hyper-threading, but designing applications to represent data in memory in contiguous blocks of homogeneous type is sometimes a major challenge. In practice, the community involved in *High-Performance Computing* (HPC) uses to map algorithms to tensor operations, leaving to the libraries and underlying runtime the task of optimizing tensor algebra [92,93]. For communities with a long tradition in scientific computing, however, it is often difficult to restate algorithms designed to operate on graphs, as customary in Object-Oriented Programming (OOP), into tensor algebra, making vectorization one of the most challenging optimization axes to explore [94].

7. **Hardware acceleration.** A further option to parallelize a workload is to rely on co-processors to perform parts of the computation with specialized hardware. GPUs and FPGAs are the most adopted co-processors and provide parallelism via vectorization and/or multi-threading. Using co-processors requires dedicated libraries and, in some cases, specialized programming languages. In general, hardware acceleration is too broad as a task to be covered here, but *heterogeneous computing* is becoming central in most of the HPC scenarios, and co-processors are being used also for some HTC applications as, notably, the trigger for the LHCb experiment [95].

To summarize, an application may ideally be parallelized by running on *multiple sites*, accessing in each site the data locally available as instructed by the WLCG, on *multiple machines* in each site, coordinated by batch systems such as HTCondor or Slurm, in *multiple processes* on the same machine, and with *multiple threads* spawned by each processor to optimize the ratio between available memory and computing power. The application may have been designed with an in-memory data representation made of contiguous blocks of homogeneous data types to enable *vectorization* and, on some nodes, it may offload to *hardware accelerators*, such as GPUs, some portion of the algorithm to achieve further speed-up.

Developing software capable of benefiting from all of the dimensions of the configuration space is more an art than a science. Adopting and relying on standard solutions and infrastructures is often a shortcut to improve performance, but it may require remapping the application into algorithms and data structures completely different from those that may appear as natural while describing the task to a human being.

Cloud computing and Machine Learning represent two modern, commercial-level, often off-the-shelf solutions to deal with different aspects of parallel computing. Cloud computing has been designed to transparently scale on multiple processors and machines, possibly on different sites, while machine learning libraries are designed to represent computations as tensor operations which are implemented targeting multi-threading, vectorization, and hardware acceleration. In the following sections, I will focus on Cloud computing and Machine Learning, in the context of scientific computing.

2.3 Cloud computing for scientific data processing

Definition (*Wikipedia*, 2024)**.** Cloud computing is the on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user. Cloud computing relies on sharing of resources to achieve coherence and typically *pay-as-you-go* model.

To achieve the goal of managing users' resources while providing the necessary customization layers to profit from the resources, and to grant rapid elasticity to follow the needs of customers, Cloud providers need to pool the resources with a multi-tenant architecture in which resources can be quickly reallocated from one tenancy to another according to policies (and prices) agreed a priori.

Cloud infrastructures rely on *virtualization* and *containerization* technologies to achieve elastic scalability and implement efficient metrics to measure the consumption of resources to bill customers with the exact amount of resources requested. While in the scientific community, billing and accountability are treated differently, the approach established by commercial Cloud providers is very interesting since it enables the treatment of burst-type requests of resources which are becoming more and more common for quasi-interactive data analysis.

Since 2019, the Italian National Institute for Nuclear Physics (INFN) has been provisioning computing resources with the Cloud model, under the name of *INFN Cloud*[7]. While the WLCG remains the de-facto standard for the production model of the LHC experiments, INFN Cloud aims to provide computing resources to an increasing number

[7] `https://www.cloud.infn.it`

of smaller experiments, whose computing models rely on the flexibility and scalability offered by the Cloud infrastructure.

2.3.1 Services

To provision managed resources to the users, while enabling customization, Cloud infrastructures are typically organized as a stack of layers of increasing complexity in terms of management and operational burden. In the following, the different levels (layers) of service in use at INFN Cloud are described. Terms are imported from commercial Cloud solutions, even if the exact meaning may differ depending on the context.

The deepest level of management exposed via web interfaces is the virtualization layer, which in INFN is based on OpenStack[8], and is generically referred to as ***Infrastructure as a Service*** (IaaS). Users accessing the IaaS are *Cloud resource administrators* and are enabled to instantiate, administer, or delete the virtual machines within the tenancies they manage.

Since multiple sites that provision resources through the Cloud model may rely on different infrastructure solutions, a further layer of abstraction, common to all the sites has been introduced and named ***Platform as a Service*** (PaaS). The PaaS orchestrates the requests of resources addressing them towards the underlying infrastructures, dealing with the differences in their interfaces and implementations. The PaaS also introduces automation tools to configure the virtual machines according to predefined models (templates) using tools such as *Ansible*[9] and *TOSCA*[10] to describe the application to deploy. Also the users managing resources through the PaaS are Cloud resource administrators, but they are only allowed to manage their own resources rather than a whole tenancy.

Deployed either with the IaaS or the PaaS, the virtual machines are intended to host services that can be accessed by a broader community than the single administrator. These applications are commonly referred to as ***Software as a Service*** (SaaS). For analysis workloads, the most typical example of SaaS is *JupyterHub*[11]. Users accessing the SaaS are not Cloud administrators and have access only to a limited amount of resources in a containerized environment, which should limit the threats to the overall infrastructure with respect to accessing the underlying virtual machines as administrators.

A special kind of SaaS is Kubernetes, sometimes referred explicitly to as *Kubernetes as a Service* (KaaS). Kubernetes is an orchestrator of containers that may handle multi-tenancy. It can be used to spawn custom containers upon user request providing a last layer of customization known as *Container as a Service* (CaaS).

2.3.2 Data management

To achieve transparent and elastic scalability, a key ingredient of Cloud computing solutions is effective data management. In general, Cloud providers tend to favor centralized storage solutions with location-independent access to the data rather than developing and maintaining tools enabling the locality of data, as customary in the context of the WLCG. Applications requiring data locality implement it via an additional layer, based

[8] `https://www.openstack.org/software`
[9] `https://www.ansible.com/overview/how-ansible-works`
[10] `https://www.oasis-open.org/committees/tosca/faq.php`
[11] `https://jupyterhub.readthedocs.io`

on technologies such as *Spark*[12] and *Hadoop*[13], designed to map computations on the nodes with the fastest access to the data. Despite important pioneering works to adopt Spark for analysis tasks in HEP [96], its strong optimization for tabular, non-nested data is hindering its adoption.

To discuss Cloud storage solutions, it is useful to introduce the following, potentially overlapping, macro-categories:

- **Data warehouse.** A data warehouse is a storage solution designed to report transactional operations to ease reporting and analyzing the data. In general, a minimal data warehouse is a *relational database*. Data warehouses may combine multiple databases and data sources, albeit maintaining the tabular and relational structure of the data. In the context of scientific data processing, databases and data warehouses are used to store the experimental conditions of the data taking. In some cases, databases may hold the metadata of the datasets (stored with other technologies) to ease indexing and retrieval.

- **Data lake.** Data lake solutions, nevertheless, are much more flexible in terms of data structure, and, in general, they may work even with binary data without any data structure. The minimal unit of a data lake is a generic *storage system*, and multiple storage systems can be combined in a data lake. In scientific data processing, the data lake approach is used to store scientific data in data centers all over the world and make them accessible through networks with protocols such as XRootD[14], WebDAV[15], and more recently Amazon S3[16].

- **Distributed file system.** The data sources of a data lake may be structured as file systems. File systems have a hierarchical tree-structured organization that groups files semantically similar or correlated in a *path*. Usually, file systems enable modifying the files, which might be mapped into non-contiguous chunks of the underlying storage system. File systems distributed on multiple nodes might be complicated to implement if concurrent editing of the files is needed, which requires adopting some precautions to prevent or handle race conditions. File systems distributed on multiple sites are rare, as the network latency between geographically distant locations usually hinders performance in the procedure dealing with concurrency. A notable exception is the *CERN Virtual Machine File System* (`cvmfs`) [97] which distributes the software environments used by the LHC experiments to process data through the network of sites member of the WLCG. To limit issues related to concurrency, `cvmfs` only distributes a sealed portion of the file system accessible all over the world in read-only mode.

- **Object storage.** An increasingly important alternative to file systems is provided by object storage. Objects are chunks of data stored in a *flat* address space. Each object has a unique identifier to which metadata can be associated with other services (such as a data ware house). Demanding the complexity of handling metadata to

[12] `https://spark.apache.org`
[13] `https://hadoop.apache.org`
[14] `https://xrootd.slac.stanford.edu/index.html`
[15] `http://www.webdav.org`
[16] `https://aws.amazon.com/s3`

other software components, object storage is much more scalable than file systems. Nonetheless, objects are immutable. They can be created, read, and deleted, but not modified, simplifying concurrent access and mitigating latency-related problems. Distributed file systems and object storage solutions have different use cases and coexist in most Cloud solutions, possibly organized and accessed through a *common data lake* infrastructure.

2.4 Machine Learning

Machine Learning (ML) is the branch of Science dedicated to the development and study of statistical algorithms capable of extracting *knowledge* from data and generalizing the *learned patterns* to unseen data. The aim is to perform tasks without explicit instructions, but only relying on data analyzed with techniques developed in the context of Statistics, Data Mining, and Artificial Intelligence (AI). A more precise and 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.

The evolution of Machine Learning over recent decades has undergone a rapid expansion characterized by a collection of surprising results, both in terms of variability of the tasks tackled and in terms of the performance achieved in each task. Nowadays, Machine Learning represents a pivotal technology with widespread use cases that go beyond specialized scientific research and leave room also to commercial applications. The first commercial success achieved by ML-based algorithms dates back to the 1990s when obtaining favorable results was often attributed more to artistry than technological prowess. The point break was marked by realizing that the skills required to make those algorithms work reduced as the data available for training increased. Thus, from one side, the growing digitization of society and the corresponding amount of data recorded that ushered in the era of *Big Data* has played a key role in the recent progress in Machine Learning. On the other hand, the voracious demand for computing power has encouraged the development and optimization of more and more fast CPUs and laid the foundation for the advent of general-purpose GPUs [98]. Hence, if until the 1990s the progress in the field of Machine Learning was stalled due to the limited computing resources available, within a couple of decades the conditions have completely changed, up to the present days where ML-based models exhibit performance in the *Natural Language Processing* (NLP) [99–101] or *Image Generation* [102, 103] problems unimaginable until a few years ago.

Back to a more mathematical formulation, the majority of the Machine Learning algorithms focus on function approximation problems, for which task T is embodied in the parametric function f_θ. The experience E may consist of a sample of known input-output pairs (x, y) from which extracts $f : x \to y$. In this context[17], the learning problem corresponds to improving the performance P defined by the *loss function* (or *cost function*) $L(y, f(x))$. Hence, training the model f_θ means finding the values for θ that optimize the performance metric:

$$\theta^* = \arg\min_{\theta \in \Theta} \mathbb{E}_{(x,y)\sim p} \left[\mathcal{L}(y, f_\theta(x)) \right] \tag{2.1}$$

[17]The scenario described above goes by the name of *supervised learning* because of the presence of outcome variables that drives the learning process.

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

In general, the ultimate objective of an ML-based algorithm is to make predictions, namely succeeding in generalizing the outcomes of data never seen before only by relying on the instances *explored* during the training procedure. However, it should be pointed out that exhibiting good performance P (i.e., minimizing as much as possible the loss function $\mathcal{L}$) is strictly connected to the quality of the training sample available, which should be representative of the tackled task and with sufficient statistics for learning the correct patterns to generalize to new instances. Thus, to monitor the generalization capabilities of the ML-based model, it is customary to retain a portion of the dataset from the training set and use it for validating the algorithm. This sample, never used during the training phase, is typically called *test sample*.

2.4.1 Classes of learning problems

Finding the solution to a learning problem depends both on the training set and the specific task investigated. Many different classes of Machine Learning problems exist and it may be useful to classify them by using the following criteria:

- **Supervised/unsupervised learning.** In *supervised learning* problems the training data consists of input-output pairs and the algorithm aims to build a parameterization of the relation that links the input instances to the desired solutions (output), referred to as *labels*. Each entry of the input sample is represented by a set of *features* that are used by the model to predict a qualitative outcome (for *classification tasks*), or a quantitative output (for *regression tasks*). Contrary to supervised learning, in *unsupervised learning* problems, algorithms learn patterns exclusively from unlabeled data. Nonetheless, also in this case, the model can be trained either to perform a classification task or a regression task. When the training data consists of a combination of unlabeled instances and labeled ones, we talk about *semi-supervised learning* (or *weak supervised learning*) problems. With the proliferation of Large Language Models (LLMs) we have witnessed in recent years [99–101, 104–106], this learning paradigm is becoming more and more popular. Notably, it is typically employed to train models by combining a small amount of human-labeled data (more expensive and processed only using the supervised learning paradigm) with a large amount of unlabeled data (cheaper and processed only relying on unsupervised learning techniques). Lastly, *reinforcement learning* defines a totally different paradigm where an intelligent *agent* that operates in a dynamic environment learns the best strategy to pursue the assigned task based on a reward-penalty scheme.

- **Batch and online learning.** In *batch learning* problems, the model is not designed to learn incrementally from data, but instead, it relies on the overall instances available. From a computational perspective, the cost of this kind of strategies is generally high, a characteristic that forces us to pursue it offline: this is why it is also referred to as *offline learning*. A complementary strategy is *online learning*, where instead the model is implemented and trained to improve by adding new 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 aims to build a model that is used to generalize new instances, then it represents a *model-based learning* problem.

As one may imagine, these criteria are not exclusive and can be combined according to the chosen strategy and the faced task.

Issues on learning process

From the definition of the learning strategies discussed above, it follows that the performance achieved by the trained model on a specific task depends on both the quality of the training dataset and the algorithm chosen to drive the training itself.

Referring to an instance-based learning system, it is self-evident the key role played by the data sample. Nevertheless, also for the model-based learning problem, the training set assembly is crucial. Indeed, the presence of noisy instances or irrelevant features can prevent from accomplishing the task, while relying on non-representative training data prevents the generalization at all. It is therefore evident the importance of the *data preprocessing* step, where data may be manipulated, filtered, and/or augmented before being analyzed.

Also choosing the model demanded to extract the patterns within data is crucial. Indeed, the latter can prove to be either too complex or too simple for the available training set. When the first case occurs, we are typically dealing with a poorly populated training sample and a model so descriptive that it learns even the noisiness of the data seen during the training. The result is a model that does not generalize well, a phenomenon called *overfitting*. A way out of this drawback is *regularization*[18], a technique to make the model simpler constraining its degrees of freedom. The complementary phenomenon is the *underfitting*, that occurs when the model is too simple to learn the underlying structure of the data. In such cases, the solution is to modify the adopted model to increase its degrees of freedom or to reduce the action of the regularization strategies eventually applied.

2.4.2 Choosing the best algorithm

Solving a learning problem corresponds to finding the solution set of an optimization problem like the one defined in Eq. (2.1). In other words, the learning process aims to find the best-suited set of parameters defining the model that we have chosen to parameterize the patterns underlying the data sample. By design, the only information accessible to the model is provided by the training sample, thus the optimization problem is performed on such a set of data. Nonetheless, the ultimate objective of the learning problem is to build a model that succeeds in the generalization task. It is therefore necessary to define a metric to monitor the actual performance of the trained model. To this end, the data sample available is split into (at least) two sub-sets: the *training set* and the *test set*. The error made on new cases is called *generalization error*, and we have access to an estimate of this error by evaluating the trained model on the test set. It is customary to define a

[18]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).

third independent sub-set, called *validation set*, and aim to perform further studies on the quality of the model output.

The time to reflect a bit on how to define a model has come. In general, we can describe it as a *simplified* version of the observations. The simplification is intended to remove any noisy contributions to the features that are unlikely to be generalized to new instances. Doing so, it is customary 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 [107], 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. This is called the *No Free Lunch* (NFL) theorem, and it ensures that a priori better-working models simply do not exist. This statement implies that the only way to find the best-performing model for a given task is to evaluate all the possible alternatives. Such studies are generally collected within the *hyperparameter optimization* problem that is further described in Section 2.6.

2.5 Deep Learning

Deep Learning (DL) is a branch of Machine Learning that achieves power, scalability, and flexibility by representing the world as a nested hierarchy of *concepts* [98]. Starting from basic concepts, typically expressed by the feature space representing the input instances, Deep Learning allows to compute data representations more and more abstractly by relying on the previous, less abstract computations. Studying models that involve a structured hierarchy of the learned concepts, this special class of Machine Learning algorithms is typically referred to as "deep".

Differently from what one may guess, Deep Learning is not a new technology, but its early studies and applications date back to the 1940s. This common mistake is due to the fact that it was relatively unpopular for several decades because of the incapability to demonstrate its validity unless disposing of large data samples and an amount of computing power inconceivable for that period. Nowadays, the availability of huge datasets for almost every field of application and the technological progress achieved by modern computation have encouraged a rapid evolution of Deep Learning algorithms that have witnessed an actual rebranding operation, until to become the de-facto standard for the majority of Machine Learning applications.

Since its inception, researchers have seen in Deep Learning a reliable approach to *Artificial Intelligence*. It is therefore not surprising that the earliest learning algorithms developed in this context were inspired by biological learning systems, like the one exhibited by the human brain (Figure 2.3). As proof of this, in 1943 the neurophysiologist Warren McCulloch and the mathematician Walter Pitts published a landmark paper [108], in which they proposed a simplified computational model inspired by animal neurons. This represented the first *Artificial Neural Network* (ANN) architecture and, since then, many other architectures have been invented [109–115].

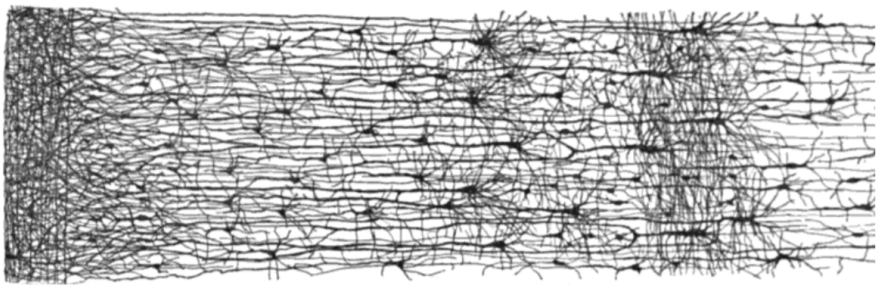


Figure 2.3: Drawing of a multiple layers in a biological neural network (human cortex). Figure reproduced from `https://en.wikipedia.org/wiki/Cerebral_cortex`.

2.5.1 Perceptron and multilayer perceptron

The *perceptron* [116] is the simplest ANN architecture, where input-output values are numbers, unlike the binary elements of the McCulloch-Pitts proposal. The perceptron algorithm relies on a slightly different artificial neuron called *threshold logic unit* (TLU) and is schematically represented in Figure 2.4a. The TLU computes a weighted sum of its inputs:

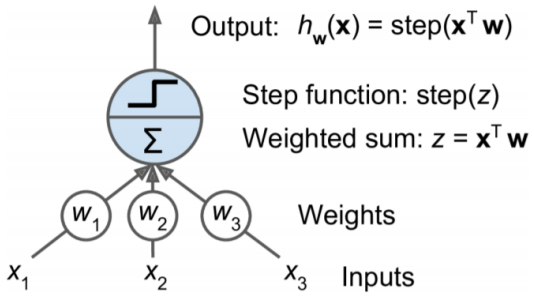
$$z = w_1\, x_1 + w_2\, x_2 + \cdots + w_n\, x_n = \mathbf{x}^T\, \mathbf{w} \tag{2.2}$$

where the inputs $\{x_i\}_{i=1,\ldots,n}$ denote the features used to build a predictive model. Then, a *step function* is applied to that weighted, obtaining as result

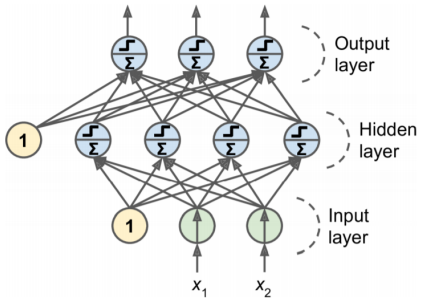
$$y = h_{\mathrm{w}}(\mathbf{x}) = \mathrm{step}(z) \tag{2.3}$$

The step function is employed to emulate the behavior of a biological neuron, that fires new electrical impulses when it receives a sufficient amount of chemical signals. The most common step functions used for perceptrons are the Heaviside step function and the sign function.

A perceptron is composed of a single layer of TLUs, where each TLU is connected to all the inputs. When all the neurons in a layer are connected to every neuron in the



(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 2.4: Schematic representation of Perceptron and Multilayer Perceptron. Figures reproduced from Ref. [117].

previous one, the layer is called a *fully connected layer* (or *dense layer*). The perceptron inputs are sent to special neurons called *input neurons*, which output whatever input they have received. Combining all the input neurons forms the *input layer*. It is customary to add to this layer an extra bias feature ($x_0 = 1$): the latter is represented by a special type of neuron, called *bias neuron*, which outputs 1 all the time. Hence, training a TLU corresponds to finding the best-suited set of weights $\mathbf{w} = (w_0, w_1, \ldots, w_n)$ that, used in combination with the input data according to (2.2) and (2.3), allows the optimization of the loss function $\mathcal{L}$.

The perceptron algorithm can be successfully used for reproducing the output of simple linear binary classifications, such as the AND/OR operators, but it fails in solving some trivial problems, like the XOR operator. To overcome these limitations, we can stack multiple perceptrons together obtaining a new ANN architecture called *multilayer perceptron* (MLP), also known as *feed-forward neural network* (FNN).

A FNN-based model typically aims to approximate some target function f^*. In a regression problem, for example, we assume that the input $\mathbf{x}$ is mapped to the output $\mathbf{y}$ via the relation $\mathbf{y} = f^*(\mathbf{x})$. Thus, an FNN can be employed to define a parametric mapping $\hat{\mathbf{y}} = f_\theta(\mathbf{x}; \boldsymbol{\theta})$, where the parameters $\boldsymbol{\theta}$ are the solution of the optimization problem defined in (2.1). Since the information flows from the input $\mathbf{x}$, through the intermediate computations used to define f_θ, and finally to the output layer $\hat{\mathbf{y}}$, such models are called "feed-forward". Notably, FNNs are typically represented by composing together many different intermediate functions: for instance, $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$ describes a map three-layer deep. The internal layers of a FNN, namely the ones different from input/output layers, are called *hidden layers*. A schematic representation of a FNN model with a single hidden layer is depicted in Figure 2.4b.

Activation functions

Ensuring that FNNs behave as *universal approximators* [118], hence potentially parameterizing any function and solving any task, requires interposing non-linear components between each linear combination of features that results from the various FNN layers. In the perceptron example depicted in Figure 2.4a, such a transformation is encoded in the function h that allows, with a simple step function, the addition of a non-linear component to the linear combination of the input features $\mathbf{x}^T \mathbf{w}$. Increasing the number of neurons and layers, and hence of the non-linear transformations, the networks acquire more and more descriptive capabilities, becoming perfect candidates to approximate the target function f by finding the best set of parameters θ that solve the optimization/learning problem. Either used within the hidden layers or as output of the network, these non-linear transformations are called *activation functions*. A non-exhaustive list of the more commonly used activation functions is provided in Table 2.1.

2.5.2 Training Deep Models

As mentioned above, training a neural network (NN) corresponds to finding the set of parameters $\boldsymbol{\theta}$ that minimizes the loss function $\mathcal{L}$ measuring the performance of the model on tackled task. Considering a regression problem, for example, we aim to solve an optimization problem like the one reported in (2.1) and to define an approximator $f(\mathbf{x}; \boldsymbol{\theta})$ capable of reliable predictions.

Name	*Function,* $f_\theta(z)$	*Range*
Identity	z	$(-\infty, \infty)$
Binary step	$\begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$	$\{0, 1\}$
Logistic or sigmoid	$\dfrac{1}{1 - e^{-z}}$	$(0, 1)$
Hyperbolic tangent (tanh)	$\dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$(-1, 1)$
Rectified linear unit (ReLU)	$\begin{cases} 0 & z \leq 0 \\ z & z > 0 \end{cases}$	$[0, \infty)$
Leaky rectified linear unit (Leaky ReLU)	$\begin{cases} \alpha\, z & z \leq 0 \\ z & z > 0 \end{cases}$ with α fixed	$(-\infty, \infty)$
Parametric rectified linear unit (PReLU)	$\begin{cases} \alpha\, z & z \leq 0 \\ z & z > 0 \end{cases}$ with α learnable	$(-\infty, \infty)$
Softmax	$\dfrac{e^{z_i}}{\sum_{j=1}^{J} e^{z_j}}$ with i = 1, ..., J	$(0, 1)$
Maxout	$\max_i z_i$ with i = 1, ..., J	$(-\infty, \infty)$

Table 2.1: Non-exhaustive list of the activation functions generally used for training neural networks, both for regression and classification tasks.

Let's now try to derive the best set of parameters by relying on this equation. It should be pointed out that, in general, we don't know the joint probability p for $(\mathbf{x}, \mathbf{y})$, and then we are unable to compute directly the expected value $\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p}\left[\mathcal{L}(\mathbf{y}, f(\mathbf{x}; \boldsymbol{\theta}))\right]$. The only way out is to rely on the information provided by the training set $\mathcal{T}$ to obtain an estimator for the average value of the loss function:

$$\hat{\boldsymbol{\theta}}^* = \arg\min_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{m} L(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta})) = \arg\min_{\boldsymbol{\theta}\in\Theta} J(\boldsymbol{\theta}) \tag{2.4}$$

where Θ is the parameter space, $\mathbf{x}_i$ is the i-th input[19] of $\mathcal{T}$, $\mathbf{y}_i$ is the i-th 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 mentioned so far.

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* (GD), are therefore preferred as baseline solutions for training neural networks. Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The algorithm consists of updating the parameters according to the local gradient of $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta\, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.5}$$

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

where η weights the gradient contributions to the parameters update. It represents a crucial training hyperparameter and is called *learning rate*. To be more precise, step (2.5) is the parameters update of the *Batch Gradient Descent* algorithm, where the gradient is computed over all the instances available. It is a powerful tool in optimum search, but it suffers from slowness for large training sets. An extreme solution is *Stochastic Gradient Descent* (SGD), whose updates rely only on a single computation of the gradient on an instance randomly chosen:

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \, \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta}) \tag{2.6}$$

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

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \sum_{i=1}^{\ell} \mathcal{L}(\mathbf{y}_i, f(\mathbf{x}_i; \boldsymbol{\theta})) \tag{2.7}$$

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

Despite the use of mini-batch-based GD or SGD optimizers, training a very large deep neural network may prove to be awfully slow. A huge speed boost can come from the deployment of faster optimizers, such as *Adam* [119]. Adam, reported in Algorithm 2, combines the ideas of two popular optimization algorithms: *momentum optimization* and *RMSProp* [120]. The first algorithm introduces the *momentum vector* $\boldsymbol{m}$, used to speed up the learning process along directions already found in previous steps of the iterative process (line 6). The second algorithm, instead, keeps track of squared gradients from most recent iterations into the vector $\boldsymbol{s}$ (line 7), so that the latters can be used to drop the learning rate faster along steep dimensions than for dimensions with gentler slopes. The parameters update step is reported on line 10.

Algorithm 2 Adam [119] 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 $\boldsymbol{\theta}_0$
1: $\boldsymbol{m}_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
2: $\boldsymbol{s}_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
3: $t \leftarrow 0$ (Initialize timestep)
4: **while** $\boldsymbol{\theta}_t$ not converged **do**
5: $\quad t \leftarrow t + 1$
6: $\quad \boldsymbol{m}_t \leftarrow \beta_1 \, \boldsymbol{m}_{t-1} - (1 - \beta_1) \, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})$
7: $\quad \boldsymbol{s}_t \leftarrow \beta_2 \, \boldsymbol{s}_{t-1} + (1 - \beta_2) \, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1}) \odot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_{t-1})$
8: $\quad \hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m} \,/\, (1 - \beta_1^t)$
9: $\quad \hat{\boldsymbol{s}}_t \leftarrow \boldsymbol{s} \,/\, (1 - \beta_2^t)$
10: $\quad \boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} + \eta \, \hat{\boldsymbol{m}}_t \oslash \sqrt{\hat{\boldsymbol{s}}_t + \varepsilon}$
11: **end while**
12: **return** $\boldsymbol{\theta}_t$ (Resulting parameters)

Back-propagation algorithm

At this point is evident that training neural networks requires computing gradients, often of complicated functions. Thus, 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 [121] plays a key role in this sense, allowing the information from the cost $J(\boldsymbol{\theta})$ to flow backward through the network to compute the gradient.

Describing neural networks in terms of *computational graphs*, the back-propagation algorithm simply consists of performing a Jacobian-gradient product for each operation in the graph, according to the *chain rule* of calculus. Suppose that $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$, $g : \mathbb{R}^n \to \mathbb{R}^m$, and $f : \mathbb{R}^m \to \mathbb{R}$. If $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i} \tag{2.8}$$

In vector notation, this may be equivalently written as

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z \tag{2.9}$$

where $(\partial \mathbf{y}/\partial \mathbf{x})$ is the $m \times n$ Jacobian matrix of g. Hence, using equation (2.8), the algorithm can measure the contribution to $J(\boldsymbol{\theta})$ of each connection from the previous below until it reaches the input layer.

2.6 The hyperparameter optimization problem

As discussed above, recent years have witnessed Machine Learning becoming an incredibly valuable tool in practically every field of application, from scientific research to industry. Increasingly complex models achieve surprising results in a wide range of applications, such as language modeling [100], image generation [102], and medical diagnosis [122]. Generally speaking, most of the ML techniques rely on the optimization of an *objective function* with respect to some internal parameters, describing the performance of the algorithm. Usually, when the optimum of the objective function is a minimum, the name of loss (cost) function is adopted. As briefly discussed in Section 2.5.2, the fastest iterative optimization techniques rely on (Stochastic) Gradient Descent techniques [123], such as Adam [119] or RMSprop [120]. Unfortunately, for a wide class of optimization problems, the gradient of the loss function with respect to the model parameter is extremely expensive to compute or cannot be defined at all. For example, optimization problems involving noisy loss functions in contexts where analytical derivatives cannot be computed, cannot rely on gradient-descent techniques, requiring the adoption of slower, often heuristic, methods. A widely adopted option is to define a *surrogate model* describing the variations of the loss function across the parameter space together with its uncertainty, driving the optimization algorithm to explore those regions where improvements were not statistically excluded from previous evaluations. Such techniques are referred to as *Bayesian optimization* (BO) methods [124, 125] and, although representing a field of study too broad to be covered here, have been an active area of research in ML in the last decade, especially from a technological perspective aiming to integrate BO-techniques with the modern computing ecosystems [2, 126–130].

Tuning the performance of ML models may benefit from the optimization of the *hyperparameters*, defined as all those parameters that are not learned during the model training procedure, but rather encode some arbitrariness in the architecture of the model itself or in the procedure to train it [124]. In practice, hyperparameter optimization (HPO) studies require training the model multiple times to explore the hyperparameter space, searching for the best-performing model according to the NFL statement (as discussed in Section 2.4.2). Since training ML models is computationally expensive, HPO campaigns should focus as much as possible on those regions of the hyperparameter space where the model performs better to reduce the time needed for finding the best configuration. Nevertheless, the loss is often a noisy function of the hyperparameters as multiple training procedures may result in different performances because of the intrinsic randomness of the stochastic gradient-descent techniques.

Since Bayesian techniques do not rely on gradient computation, the exploration of the hyperparameter space can be performed in parallel by relying on many independent trainings, or *trials*, that can be potentially offloaded to different, distributed computing resources. In general, accessing more resources enables the exploration of larger hyperparameter spaces, possibly resulting in better models. Disposing of opportunistic access to computing resources may provide valuable contributions to HPO campaigns, but coordinating studies on resources from different providers and different computing paradigms challenges the adoption of existing HPO services [126–130]. Hence, a part of my Ph.D. has been dedicated to the conception and development of a Cloud-based solution designed to coordinate HPO studies across multiple machines submitting simple queries to a central, managed service. The solution, called Hopaas [2] is described in detail in Section 2.6.1, while an early Hopaas application relying on HPC resources and aimed at the LHCb *flash-simulations* is discussed in Section 2.6.2.

2.6.1 Hyperparameter optimization as a service

Hopaas, which stands for *Hyperparameter Optimization as a Service*), implements a set of REST APIs to orchestrate HPO studies across multiple computing instances. Computing nodes from multiple HPC centers can concur *dynamically* to the same optimization study, requesting to the Hopaas server a set of hyperparameters to test, and then sending back the outcome of the training procedure. Several trials of one or more studies can be tracked and monitored through the web interface provided by the Hopaas service. A reference implementation, with a server instance[20] deployed on INFN Cloud resources and a simple client package [131] wrapping the REST APIs to Python functions is discussed in the following.

Hopaas API specification

We refer to a *trial* as a single training attempt with a specific set of hyperparameters to test. A *study* represents an optimization session and includes a collection of trials. In practice, a study is unambiguously defined by the set of hyperparameters to optimize, the range of values where searching the optimum, and the modality in which this search is carried out (e.g., grid search, Bayesian methods [124], or evolutionary algorithms [132]).

[20]Visit `https://hopaas.cloud.infn.it` for additional details.

API	Description	HTTP method	Request path
`version`	Provides the version of the HOPAAS backend.	GET	`/api/version`
`ask`	Creates a new trial, contributing to a new/existing study. The POST body request should include the set of settings to refer unambiguously to a study. The API response contains the hyperparameters to test.	POST	`/api/ask/token`
`tell`	Provides the final score of a trial to the backend optimizer chosen for the study.	POST	`/api/tell/token`
`should_prune`	Provides an intermediate score to the backend optimizer. If the study includes a *pruner* strategy, the API response is a boolean value saying whether or not to continue the current trial.	POST	`/api/should_prune/token`

Table 2.2: Minimal description of the REST APIs provided by the HOPAAS service.

The core activity of the HOPAAS service is to manage distributed optimization studies by providing sets of hyperparameters to requesting computing nodes, the so-called HOPAAS clients. The creation, intermediate updates, and finalization of a trial is controlled from the client-side by using a set of REST APIs. Such APIs, named `ask`, `tell`, and `should_prune`, implement these actions upon POST HTTP requests with user authentication based on an *API token* in the request path. A minimal description of the HOPAAS REST APIs is depicted in Table 2.2 and further detailed in the rest of this section.

A computing node ready to test a set of hyperparameters, whether it comes from on-premises, Cloud, or HPC resources, will simply need a network connection with the HOPAAS server to take part to an optimization campaign. In particular, it will query the HOPAAS server via the `ask` API, including in the request body all the information needed to define a study unambiguously. The HOPAAS server will define a new trial, possibly assigning it to an existing study, or creating a new one. Once created the trial, the HOPAAS server provides it with a unique identifier that is included in the HTTP response together with the set of hyperparameters to be evaluated for the study.

Usually, the evaluation of a set of hyperparameters consists of training a model defined by those hyperparameters aiming at the resulting value of the objective function. The evaluated performance metric may correspond to the loss function computed during the training procedure but, in general, it can be any numerical score obtained processing a given set of hyperparameters. Once the evaluation is completed, the computing node will

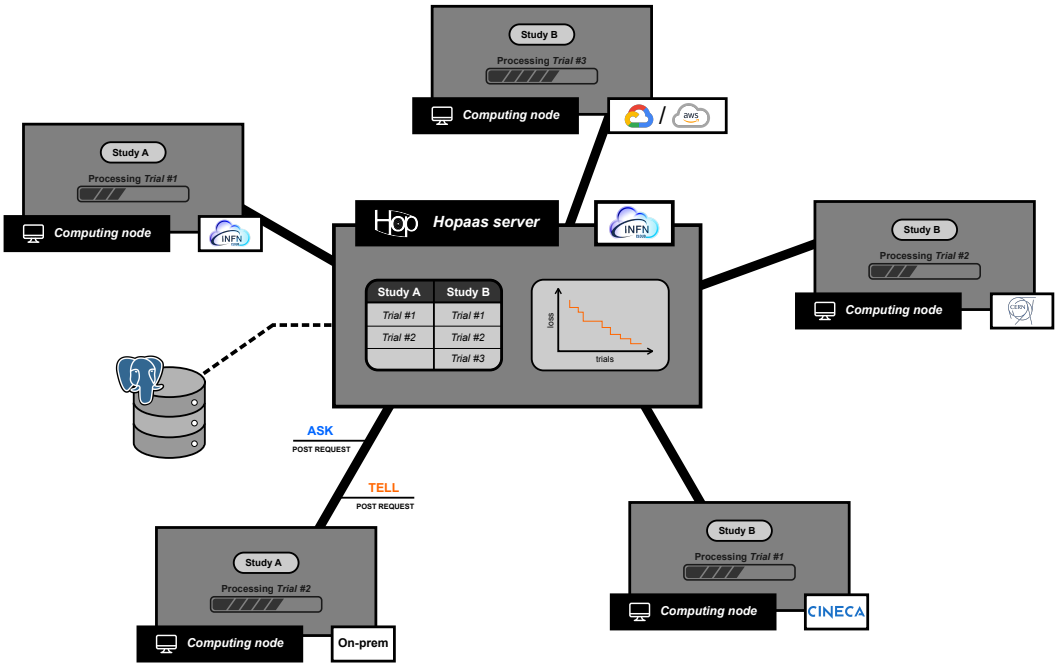


Figure 2.5: A HOPAAS server orchestrating multiple studies across multiple sites.

finalize the trial using the `tell` API, whose body will include the unique identifier of the trial and the final evaluation of the objective function.

The HOPAAS server may serve multiple `ask` requests from different sources, assigning them to one or different studies, while updating the surrogate model each time a new evaluation is made available by querying the `tell` API. A schematic representation of the orchestration of studies in multiple sites is reported in Figure 2.5.

Depending on the specific ML algorithm, intermediate evaluations of the objective function can be accessed during the training procedure and used to abort non-promising trials (*pruning*) without wasting computing power to take the training procedure to an end. Optionally, the computing node may update the HOPAAS server with intermediate evaluations of the objective function by querying the `should_prune` API for monitoring and pruning purposes. The body of a `should_prune` request will contain the unique identifier of the trial, the intermediate value of the loss function, and an integer number encoding the progress of the training procedure, the so-called *step*. The HTTP response will indicate whether the study should be early terminated, or it is sufficiently likely to result in an improvement over the previous tests.

A reference Python frontend was developed aiming at a facilitated access to the HOPAAS service from Python applications [131]. While Python is a primary choice for many scientific applications, it should be noticed that the client simply wraps the REST APIs into classes and functions, as the HOPAAS protocol is designed to be language-agnostic, relying on widely adopted web communication standards. In addition, the HOPAAS client is also framework-agnostic since the evaluation of the objective function for a given set of hyperparameters can be implemented with any framework and environment.

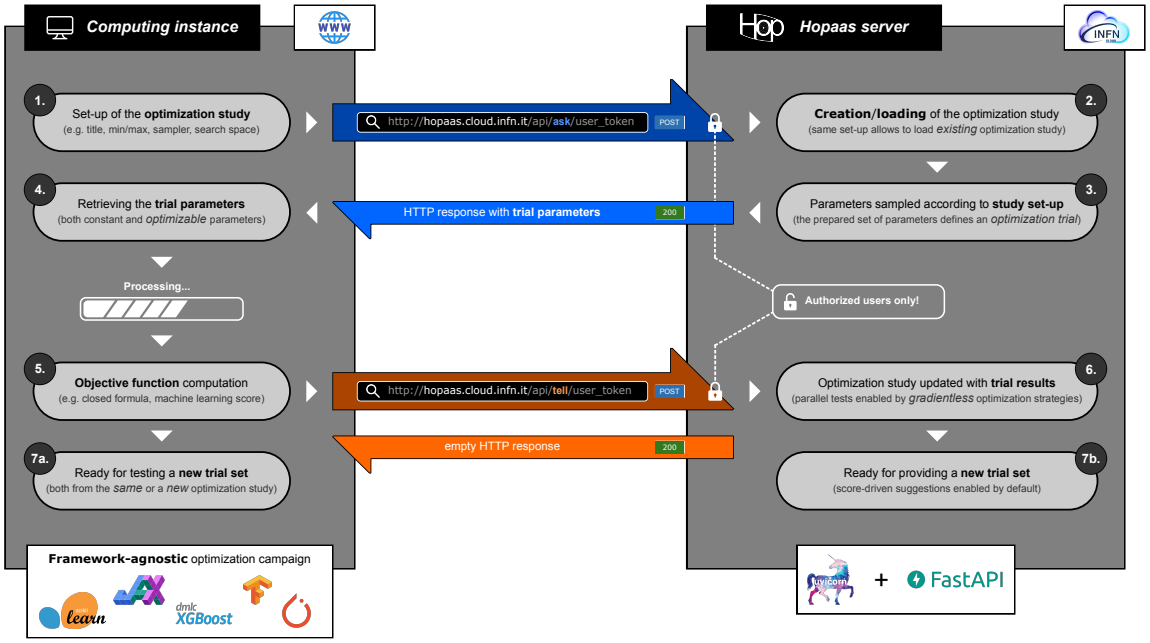


Figure 2.6: Workflow of an optimization study with a client-server approach based on REST APIs.

Implementation

The reference implementation for the HOPAAS service running on INFN Cloud relies on containerized applications orchestrated with `docker-compose`. The web server implementing the REST APIs is a scalable set of Uvicorn[21] instances running an application based on the FastAPI framework[22]. The BO algorithms are provided by integrating the backend with OPTUNA[23], while future extensions to additional frameworks are planned. The access to the Uvicorn instances from the Internet is mediated by an NGINX[24] reverse proxy accessed via the encrypted HTTPS protocol. A PostgreSQL[25] instance is part of the `docker-compose` configuration to provide shared persistency to the multiple instances of the web application backend. The workflow of the interaction between the HOPAAS server and computing nodes is depicted in Figure 2.6.

The same HOPAAS server is designed to serve web-based user access. A web application, developed in HTML, CSS, and JavaScript, is shipped to the client browser as defined by a set of web-specific APIs in Uvicorn. The web pages of the frontend provide dynamic visualizations by fetching data from specialized APIs at regular intervals. Plots showing the evolution of the loss reported by different studies and trials are obtained with the CHARTIST library[26].

The user authentication and authorization procedure of the web application is managed to rely on *access tokens* as defined by the OAuth2 standard, using the INFN GitLab

[21] `https://www.uvicorn.org`
[22] `https://fastapi.tiangolo.com`
[23] `https://optuna.org`
[24] `https://www.nginx.com/solutions`
[25] `https://www.postgresql.org/about`
[26] `https://gionkunz.github.io/chartist-js`

instance as an identity provider. Support for INDIGO IAM[27] is also planned for the future [133]. Once authenticated, users can generate multiple API tokens through the web application. Each API token has a validity period defined at generation and can be revoked at any time. Tokens with shorter validity are more appropriate for usage in public or untrusted contexts.

2.6.2 Hyperparameter optimization on HPC centers

The reference implementation of the HOPAAS service presented in the previous Section has found, in the LHCb efforts to provide the experiment with reliable flash-simulated samples, its early adopters. As deeply discussed in Chapter 1, the detailed simulation of HEP detectors has a high CPU cost, often committing the largest fraction of pledge resources available to the LHC experiments. This has resulted in a joint effort within the HEP community to produce alternative and faster simulation solutions to meet the increasing request for simulated samples of the various physics working groups for their analyses. Two major strategies are emerging and aim to reduce the computational cost for simulations by providing (parametric) shortcuts that point either upstream of the reconstruction step, referred to as *fast-simulations*, or downstream of the reconstruction step, then offering the fastest option for simulation and called *flash-simulations*. LAMARR [3–5], which will be described in Chapters 4 and 5, is the flash-simulation option for LHCb and allows to reproduce analysis-level quantities by using advanced ML algorithms. In particular, most of the parameterizations rely on *Generative Adversarial Networks* (GAN) [134], a powerful algorithm that allows to learn the probability distributions of data by the competition, namely an *adversarial training*, of two neural networks. By design, the performance exhibited by GAN-based models is intimately connected to the combination of hyperparameters chosen for the training and may require intensive optimization campaigns to parameterize accurately the target distributions.

During my Ph.D., I had the opportunity to access several GPU-accelerated instances, provided by on-premises, Cloud, and HPC resources. The pilot project for providing the CERN experiments with a Cloud broker solution [135] gave me access to GPU-equipped instances from AWS and GCP and raised the issues of how resources coming from different providers could concur to the same optimization study. This motivated the design of a proof-of-concept [136] aimed to thin as much as possible the requirements in terms of both dependencies and security to combine different computing nodes regardless of their provenance (e.g., on-premises, Cloud, or HPC resources). The solution then evolved in HOPAAS, relies on a centralized service that coordinates the optimization studies via HTTP requests. This avoids any connection among nodes and only adds a network requirement: the access to the Internet.

Several optimization studies have been orchestrated by the HOPAAS service using *diverse* computing instances, from scientific providers (like INFN Cloud and the CINECA HPC systems) and commercial Cloud providers (like AWS and GCP). Most of the resources have been provided by the CINECA supercomputer MARCONI100 (technical specifications in Table 2.3) which has allowed to perform the dozens of optimization studies with hundreds of trials on each study required to find the best-suited set of hyperparameters to build reliable parameterizations for the LAMARR framework. HOPAAS has played a

[27] `https://indigo-iam.github.io`

	Marconi100	*Leonardo*
Nodes	980	3456
Processors	IBM Power9 AC922, 2.60 GHz	Intel Xeon 8358, 2.60 GHz
Cores	32 cores/node	32 cores/node
Accelerators	4 × NVIDIA V100 GPU/node	4 × NVIDIA A200 GPU/node
RAM	256 GB/node	512 GB/node
Node performance	∼ 32 TFLOPS	∼ 77 TFLOPS

Table 2.3: Technical specifications of the CINECA HPC systems.

key role in this respect, succeeding in coordinating more than twenty concurrent instances coming from on-premises, INFN Cloud, and Marconi100, and allowing to perform HPO campaigns for more than $\mathcal{O}(10^4)$ GPU hours. Notably, using Marconi100 has required a custom network configuration [137] to enable communications between *working nodes* and the Hopaas server, complying with the network policies typically in use at HPC centers. Marconi100 was dismissed in June 2023 to leave the place to its successor, Leonardo (technical specifications in Table 2.3), one of the most powerful supercomputing systems worldwide, capable of executing more than 250 PFLOPS. At the time of writing, Hopaas has not been tested on Leonardo nodes yet due to stricter network policies that prevent working nodes (the only ones equipped with GPUs) from having access to the Internet at all. Developments are undergone to relax the network constraints aiming to integrate Leonardo within the distributed LHC computing infrastructure. Further developments of Hopaas are planned to exploit the large computing power offered by Leonardo and to rely on Cloud solutions for defining automatic pipelines performing optimization and validation studies for the Lamarr parameterizations whenever new data are available[28].

Most of the concepts briefly reported will have space to be expanded and clarified in the next Chapters, while have been introduced here to provide the discussion on Hopaas with the context that has motivated its conception and development.

[28]The solution briefly discussed aims to define pipeline, based on the MLOps paradigm, to deploy and maintain Machine Learning models in production reliably and efficiently.

3 Deep generative models

Year after year, architecture after architecture, model after model, we are witnessing a continue and disruptive evolution of the performance achieved by Deep Generative Models that, nowadays, populate the majority of the Artificial Intelligence domains, from image generation to natural language processing. Most of the enhancements results from an active scientific community that, in addition to design novel architectures, deeply investigates the limitations and drawbacks exhibited by the previous models, developing ever-changing strategies to overcome them. The High Energy Physics experiments are watching carefully the latest achievements aiming at the application of tuned versions of the most promising algorithms for Fast Detector Simulation. This Chapter is devoted to Deep Generative Models, from a general introduction depicted in Section 3.1 to a precise theoretical description aimed to characterize either Generative Adversarial Networks (Section 3.2) or Normalizing Flows (Section 3.3).

3.1 Introduction

Deep Generative Modeling (DGM) is a quite active field in Machine Learning, where neural networks are trained to approximate high-dimensional, non-trivial probability distributions by relying on large data samples. Originally focused on Computer Vision applications, nowadays DGM problems populate various Artificial Intelligence (AI) domains, studying models for the generation of images, text, audio, and even more complex data structures, like sequences (e.g., videos) or graphs (e.g., proteins). Such kind of techniques has become more and more popular also in High Energy Physics, where are mostly investigated to reproduce the response of the detectors to traversing particles. The common objective of all these use-cases is to learn an unknown or intractable probability distribution from a *finite* sample of instances distributed accordingly to the target distribution. It represents a quite general problem that has been deeply studied by Statistics for decades, but that remains computationally challenging to solve, especially in high dimensional space [138].

In general, the objective of a Generative Modeling problem is to learn, from a finite set of data, a representation for an intractable probability distribution p_{x} defined over $\mathbb{R}^n$, where n is typically large, and p_{x} is a non-trivial distribution. If in statistical inference problems we aim to find an expression for the target probability distribution, the ultimate

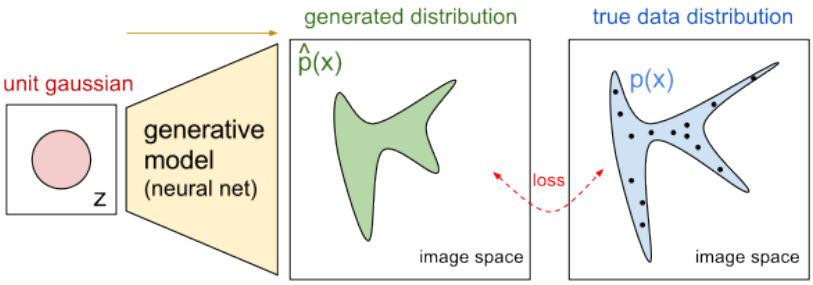


Figure 3.1: Schematic representation of a generic Generative Modeling problem. The target distribution $p(\mathbf{x})$ is approximated with the distribution $\hat{p}(\mathbf{x})$ induced by the *generator* (or *generative model*) when fed by elements sampled from a Gaussian distribution. Figure reproduced from `https://openai.com/research/generative-models`.

goal of Generative Modeling problems is to obtain a *generator*

$$g : \mathbb{R}^{\ell} \to \mathbb{R}^{n} \tag{3.1}$$

namely a function g able to map elements from a tractable distribution p_{z} defined over $\mathbb{R}^{\ell}$ to points in $\mathbb{R}^{n}$ that resemble the given data. In other words, we assume that for each points $\mathbf{x} \sim p_{\mathrm{x}}$ there is at least one element $\mathbf{z} \sim p_{\mathrm{z}}$, such that $g(\mathbf{z}) \approx \mathbf{x}$. Since the element $\mathbf{z}$ that results in a given $\mathbf{x}$ is generally unknown, it is customary to refer to it as the latent variable and call $\mathcal{Z}$ the *latent space*. Without any loss of generality, p_{z} is usually assumed to be a multivariate Gaussian in $\mathbb{R}^{\ell}$. Hence, disposing of the map function g allows us to generate new instances $\mathbf{x}$ distributed according to p_{x} by simply applying g to an element $\mathbf{z}$ drawn from $p_{\mathrm{z}} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. A schematic representation of this process is depicted in Figure 3.1.

Deriving g from first principle is infeasible for most of the datasets of interest. For example, it may be challenging to model the process that transform an element sampled from a multivariate Gaussian into the 2-D image representing the energy deposited within a calorimeter by an electromagnetic shower. Therefore, in recent years, it has become common relying on *universal approximators*, like neural networks, to parameterize the generator function. Notably, two major techniques are in use to drive the neural network training so that it approximate a generator map. In the first case, the training is driven by measuring the similarity between the generated sample and the reference one. In the second case, instead, the training is driven by "inverting" the generator map. Despite the differences, both the strategies come with their own pros and cons. Quantifying the similarity of two probability distributions from samples become more and more complex as the dimensions increase. On the other hand, inverting the generator is a non-trivial operation in most the cases, especially when it is modeled with a neural network that presents nonlinear components by design [138].

The Deep Generative Models most commonly used follows:

1. **Generative Adversarial Networks.** Also called GAN, this model relies on the competition between two neural networks, called *discriminator* and *generator*. The discriminator is trained to perform a classification task, while the generator to perform a simulation task. Further details discussed in Section 3.2.

2. **Variational Autoencoders.** Also called VAE, this model relies on two neural networks implementing an *encoder-decoder* system. The encoder define a map from the reference space to a low-dimensional latent space. The decoder tries to *invert* the encoder action, defining a map from the latent space to the synthetic space. The training is driven minimizing the differences between the input and output spaces, and regularizing the latent space. Further details in Ref. [139].

3. **Normalizing Flows.** By relying on a sequence of invertible and differentiable transformations, this model allows to explicitly learn the data distribution $p(\mathbf{x})$. Further details in Section 3.3.

4. **Diffusion Models.** Inspired by the non-equilibrium thermodynamics, this model works by destroying training data through the successive injection of Gaussian noise, and then learning to recover the data by reversing this noising process (i.e., *denoising* process). After the training, we can use the Diffusion Model to generate data by simply passing randomly sampled noise through the learned denoising process. Further details in Ref. [140].

As schematically represented in Figure 3.2, all four models disposes of a latent space $\mathcal{Z}$ that, used by the different-implemented generator maps, allows to produce a *synthetic space* aiming at making it as similar as possible with the reference space.

The rest of this Section is dedicated to provide further details for two of the four Deep Generative Models presented above. In particular, Section 3.2 offers a complete review on the training issues that typically affect GAN-based systems, and how to mitigate them using regularization strategies. A brief introduction on Normalizing Flows and how to implement them relying on autoregressive models are discussed in Section 3.3.

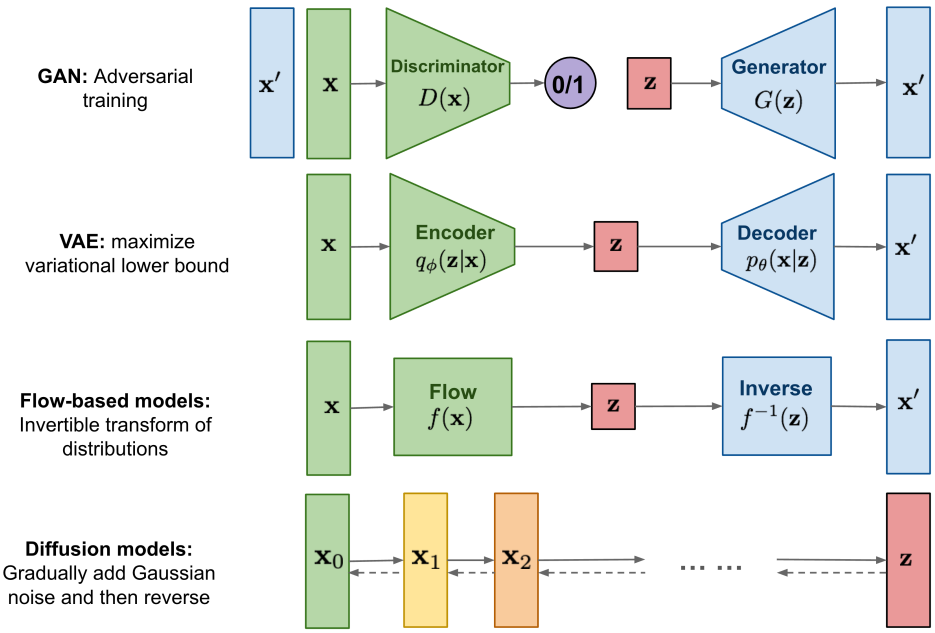


Figure 3.2: Schematic representation of the learning strategies pursued by the various Deep Generative Models presented. Figure reproduced from Ref. [140].

3.2 Demystifying Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a powerful class of generative models that rely on the simultaneous training of two neural networks [134]:

- the generator G is trained to perform a simulation task, aiming at reproducing as accurately as possible the reference data sample, starting from some noise source;
- the discriminator D is trained to perform a classification task, aiming at distinguishing reference data from the output of the generator.

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 $\mathbf{z}$ are sampled according to known distribution p_z. Then, the generative model $G(\mathbf{z}; \boldsymbol{\theta}_g)$ maps the latent space to the data space, inducing a distribution p_g to generator outputs. The discriminative model $D(\mathbf{x}; \boldsymbol{\theta}_d)$ outputs a single scalar, readable as the probability that $\mathbf{x}$ comes 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 $\log(1 - D(G(\mathbf{z})))$. Therefore, defining the $V(D, G)$ function as follows

$$V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_r}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z}[\log(1 - D(G(\mathbf{z})))] \tag{3.2}$$

the minimax game can be written in this form:

$$\min_G \max_D V(D, G) \tag{3.3}$$

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 3.3.

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 (3.3) with respect to D, in fact, one can find the *optimal* discriminator D^* for fixed generator G:

$$D^*_G(\mathbf{x}) = \max_D V(D, G) = \frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})} \tag{3.4}$$

If we substitute expression (3.4) in the loss function (3.2), it follows that

$$V(D^*, G) = -\log(4) + 2 \cdot JS\left(p_r \| p_g\right) \tag{3.5}$$

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, $\{\mathbf{z}_i\}_{i=1,\dots,\ell}$
    Sample ℓ elements from the data space, $\{\mathbf{x}_i\}_{i=1,\dots,\ell}$
    $g_d^{(s-1)} \leftarrow \nabla_{\boldsymbol{\theta}_d} \frac{1}{\ell} \sum_{i=1}^{\ell} \left[\log D\left(\mathbf{x}_i; \tilde{\boldsymbol{\theta}}_d^{(s-1)}\right) + \log\left(1 - D\left(G\left(\mathbf{z}_i; \boldsymbol{\theta}_g^{(t)}\right); \tilde{\boldsymbol{\theta}}_d^{(s-1)}\right)\right)\right]$
    $\tilde{\boldsymbol{\theta}}_d^{(s)} \leftarrow \tilde{\boldsymbol{\theta}}_d^{(s-1)} + \eta\, g_d^{(s-1)}$
  end for
  $t \leftarrow t + 1$
  Sample mini-batch of ℓ elements from the latent space, $\{\mathbf{z}_i\}_{i=1,\dots,\ell}$
  $g_g^{(t-1)} \leftarrow \nabla_{\boldsymbol{\theta}_g} \frac{1}{\ell} \sum_{i=1}^{\ell} \log\left(1 - D\left(G\left(\mathbf{z}_i; \boldsymbol{\theta}_g^{(t-1)}\right); \tilde{\boldsymbol{\theta}}_d^{(s)}\right)\right)$
  $\boldsymbol{\theta}_g^{(t)} \leftarrow \boldsymbol{\theta}_g^{(t-1)} - \eta\, g_g^{(t-1)}$
  $\boldsymbol{\theta}_d^{(t)} \leftarrow \tilde{\boldsymbol{\theta}}_d^{(s)}$
end while
return $\boldsymbol{\theta}_g^{(t)}$ (Resulting generator parameters)

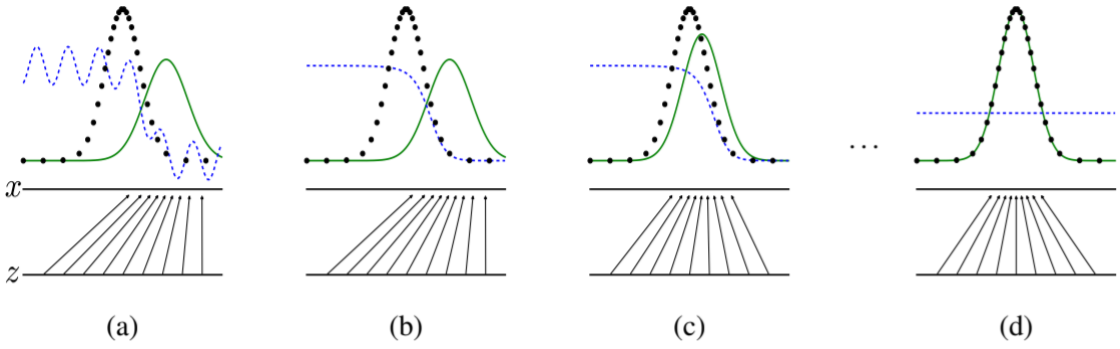


Figure 3.3: GANs are trained by simultaneously updating both of the discriminator D (dashed line in blue) and of the generator G. The goal of D is to distinguish between the data sample (dotted line in black) from the generator one (solid line in green). The lower horizontal lines and the upward arrows represent the latent space $\mathcal{Z}$ and the generator map $G : \mathbf{z} \to \mathbf{x}$. 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 $D(\mathbf{x}) = 1/2$. Pedagogical example reproduced from Ref. [134].

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 (3.3) corresponds to minimize the Jensen-Shannon divergence (JSD):

$$\min_G \max_D V(D,G) = \min_G V(D^*,G) = \min_G JS\left(p_r \| p_g\right) \tag{3.6}$$

Since the JSD 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 [134].

GANs can be extended to a conditional model [141] if both the generator G and discriminator D are conditioned on some extra information $\mathbf{y}$, where latter can be any kind of auxiliary information. One can perform the conditioning by adding new nodes containing $\mathbf{y}$ to the input layer. In doing so, the loss function (3.2) can be rewritten as follows:

$$V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_r}[\log D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z}|\mathbf{y})|\mathbf{y}))] \tag{3.7}$$

3.2.1 Causes and solutions for training instability

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 JSD between true data distribution and the generated one [142].

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 [142], but despite this the set of hyperparameters for which latter converges is generally very small.

Least Squares GANs

The typical solutions developed in the literature for overcoming the training instabilities rely on changing the loss function that drives the training procedure, aiming at defining a more robust competition system. Among the proposals, the Least Squares Generative Adversarial Networks (LSGAN) [143] are remarkable. The core idea is to extend the labeling range available to the discriminator, and disentangle it from the one of the generator during the evaluation of the loss. Notably, suppose we use the a-b score scheme for the discriminator, where a and b are the labels for fake and real data, respectively. Then, we can reformulate the minimax (actually, a "minimini") game, as follows:

$$\begin{cases} \min_D V_{\text{LSGAN}}(D,G) = \frac{1}{2}\mathbb{E}_{x\sim p_r}[(D(\mathbf{x}) - b)^2] + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_z}[(D(G(\mathbf{z})) - a)^2] \\ \min_G C_{\text{LSGAN}}(D,G) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_r}[(D(\mathbf{x}) - c)^2] + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_z}[(D(G(\mathbf{z})) - c)^2] \end{cases} \tag{3.8}$$

where c denotes the value that G want D to believe for fake data.

Also for LSGAN we can find the optimal discriminator D^* for a fixed generator G by minimizing $V_{\text{LSGAN}}(D,G)$ defined in (3.8):

$$D^*(\mathbf{x}) = \min_D V_{\text{LSGAN}}(D,G) = \frac{b\,p_r(\mathbf{x}) + a\,p_g(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})} \tag{3.9}$$

which, substituted in the generator loss in (3.8), results in

$$\begin{cases} b - c = 1 \\ b - a = 2 \\ 2\,C(D,G) = 2\,C(D^*,G) = \chi^2_{\text{Pearson}}(p_r + p_g \| 2\,p_g) \end{cases} \tag{3.10}$$

where χ^2_{Pearson} denotes the f-divergence between $p_r + p_g$ and $2\,p_g$ if a, b, and c satisfy the conditions of $b - c = 1$ and $b - a = 2$.

In the practice, two configuration for the parameters a, b, and c are the most employed when one relies on LSGAN. The first one aims to satisfy the condition that ensure to minimize the f-divergence once G and D are trained with the loss functions in (3.8). In particular, by setting $a = -1$, $b = 1$, and $c = 0$, we get the following loss functions:

$$\begin{cases} \min_D V_{\text{LSGAN}}(D,G) = \frac{1}{2}\mathbb{E}_{x\sim p_r}[(D(\mathbf{x}) - 1)^2] + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_z}[(D(G(\mathbf{z})) + 1)^2] \\ \min_G C_{\text{LSGAN}}(D,G) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_r}[(D(\mathbf{x}))^2] \end{cases} \tag{3.11}$$

Another method is to make G generate samples as real as possible by setting $c = b$. For example, by using the 0-1 score scheme, we obtain the following loss functions:

$$\begin{cases} \min_D V_{\text{LSGAN}}(D,G) = \frac{1}{2}\mathbb{E}_{x\sim p_r}[(D(\mathbf{x}) - 1)^2] + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_z}[(D(G(\mathbf{z})))^2] \\ \min_G C_{\text{LSGAN}}(D,G) = \frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_r}[(D(\mathbf{x}) - 1)^2] \end{cases} \tag{3.12}$$

The authors of Ref. [143] declared to having observed similar performance with both the loss functions defined above.

Unfolding the training instability problem

A complete analysis of the reasons behind the massive instability of GANs training is reported in a paper by Martin Arjovsky and Léon Bottou [144]. Here, they notice that generator updates get consistently worse as the discriminator gets better, although the optimization problem (3.6) follows from achieving the optimal discriminator D^*.

As mentioned above, with an *optimal* discriminator trained with (3.3), 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 JSD between p_r and p_g is maxed out[1]. The only way this may occur is if the supports of distributions are disjoint or lie in low dimensional manifolds. Actually, these conditions are not so rare. Considering the generator map $G : \mathcal{Z} \to \mathcal{X}$, for instance, if the dimensionality of $\mathcal{Z}$ is less than the dimension of $\mathcal{X}$ (as it is in most of the case), then the support of p_g can only lie in a manifold with measure 0 for $\mathcal{X}$.

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

Under this assumption, it can be proven that 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

$$\lim_{\|D-D^\star\|\to 0} \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{z}\sim p_z} \left[\log\left(1 - D\left(G_{\boldsymbol{\theta}}(\mathbf{z})\right)\right)\right] = 0 \tag{3.13}$$

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 (3.13) points out that using the loss function (3.2) 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, and it corresponds to add continuous noise to the inputs both of discriminator and generator (for more clarification, see Ref. [144]). 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* JSD:

$$\mathbb{E}_{\mathbf{z}\sim p_z,\varepsilon'} \left[\nabla_{\boldsymbol{\theta}} \log\left(1 - D^*_\varepsilon\left(G_{\boldsymbol{\theta}}(\mathbf{z}) + \varepsilon'\right)\right)\right] = 2 \cdot \nabla_{\boldsymbol{\theta}} JS\left(p_{r+\varepsilon} \| p_{g+\varepsilon'}\right) \tag{3.14}$$

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

3.2.2 Wasserstein distance as stability solution

Before defining a 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 crucial property of these distances we are interested in is their impact on the *convergence* 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 [145].

Therefore, in optimizing $\boldsymbol{\theta}$ we would like to define a model inducing a 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:

$$W(p_r, p_g) = \inf_{\gamma\in\pi(p_r,p_g)} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\gamma}\left[\,\|\mathbf{x}-\mathbf{y}\|\,\right] \tag{3.15}$$

where $\pi(p_r, p_g)$ denotes the set of all joint distributions $\gamma(\mathbf{x}, \mathbf{y})$ whose marginals are respectively p_r and p_g. Even if the Wasserstein distance is much weaker than the JSD (for more clarification, see Ref. [145]), $W(p_r, p_g)$ 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 JSD 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 (3.15) is computationally intractable, and we need a different formulation. Luckily, the Kantorovich-Rubinstein duality comes to our rescue:

$$W(p_r, p_g) = \sup_{\| f \|_L \leq 1} \left(\mathbb{E}_{\mathbf{x} \sim p_r} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim p_g} [f(\mathbf{x})] \right) \tag{3.16}$$

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:

$$\min_G \max_{C \in \mathcal{C}} W(C, G) = \min_G \max_{C \in \mathcal{C}} \left(\mathbb{E}_{\mathbf{x} \sim p_r} [C_{\boldsymbol{\omega}}(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \sim p_z} [C_{\boldsymbol{\omega}}(G_{\boldsymbol{\theta}}(\mathbf{z}))] \right) \tag{3.17}$$

where C (the *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 $\boldsymbol{\omega}$ and $\boldsymbol{\theta}$. Training WGANs can still be described by Algorithm 3 adopting the appropriate changes on the loss function. Even if we have found a new formulation for the loss function that prevents, by design, any vanishing gradient problem, it strictly depends on out ability to make the critic a Lipschitz function ($C \in \mathcal{C}$). Several techniques are available in the literature, but here we will focus on two among the most used.

Implicit Lipschitz constraint

Historically, the way to constrain the critic network to induce a 1-Lipschitz function follows a reduction of the expressivity of the FNN by clipping the weights $\boldsymbol{\omega}$ within a predefined range. However, such a technique was soon overcome since source of optimization difficulties due to a violation of the *universal approximation theorem* for the critic network. The solution, proposed in Ref. [146], relies on an *implicit* property of the optimal solution f^* for the optimization problem defined in (3.16). As long as f^* is a 1-Lipschitz, it has gradient norm 1 almost everywhere under p_r and p_g. This property provides a metric-informed constraint for the critic network that can then regularized to resemble a 1-Lipschitz function without any loss of expressivity, since the regularization term, is naturally satisfied whenever the critic network approximate the optimal solution f^*. In practice, to enforce the critic to reach the condition just described, we can add a regularization term to the loss function $W(C, G)$:

$$W(C, G) = \underbrace{\mathbb{E}_{\mathbf{x}_r \sim p_r}[C(\mathbf{x})] - \mathbb{E}_{\mathbf{x}_g \sim p_g}[C(\mathbf{x}_g)]}_{\text{Original critic loss}} + \underbrace{\lambda \cdot \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}} [(\|\nabla_{\hat{\mathbf{x}}} C(\hat{\mathbf{x}})\|_2 - 1)^2]}_{\text{Gradient penalty}} \tag{3.18}$$

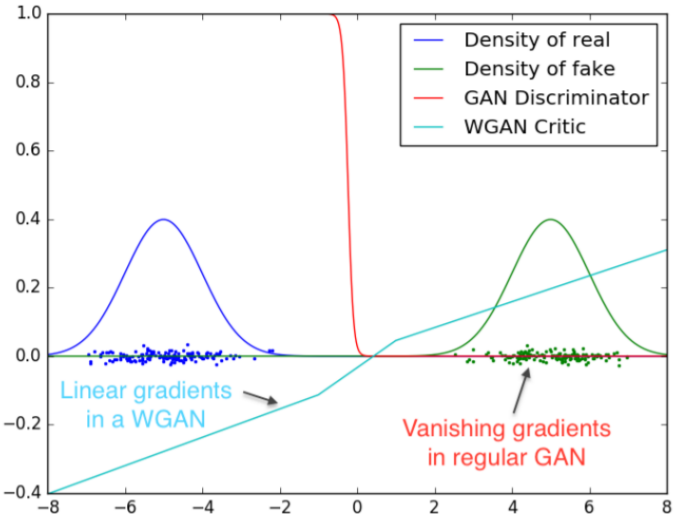


Figure 3.4: Optimal discriminator and critic when learning to distinguish two Gaussians. Here it's clearly shown that the discriminator of original GAN saturates and results in vanishing gradients. On the other hand, WGAN critic is able to provide very clean gradients on all parts of the space. Reproduced from Ref. [145].

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

Once ensured the Lipschitzianity of the critic, the loss function resembles the Wasserstein distance, whose properties (statement 2 in particular) ensures that $W(C,G)$ is continuous everywhere and differentiable almost everywhere. This means that we can always train the critic function till optimality without any problems since it, by design, cannot saturate. In Figure 3.4 we represent this behaviour comparing it with the one of the original discriminator that, as expected, provides no reliable gradient information.

Explicit Lipschitz constraint

Contrary to the GP-like regularizations that aim to induce the Lipschitzianity of the critic by requiring that some implicit property is satisfied, the *explicit methods* allow to define regularization terms that explicitly penalize any violation of the Lipschitz constant K:

$$d_{\mathcal{Y}}(f(\mathbf{x}), f(\mathbf{y})) \leq K \cdot d_{\mathcal{X}}(\mathbf{x}, \mathbf{y}) \qquad \text{for any } \mathbf{x}, \mathbf{y} \in \mathcal{X} \tag{3.19}$$

where the metric spaces $(\mathcal{X}, d_{\mathcal{X}})$ and $(\mathcal{Y}, d_{\mathcal{Y}})$ are the domain and the codomain of the function f, respectively. However, relying on this explicit formulation within the training is unfeasible in most of the case, since exploring the whole parameter space defined by a neural network is computationally too expensive. On the other hand, one could propose to approximate K by relying on pairs $(\mathbf{x}, \mathbf{y})$ randomly sampled from p_r and p_g but it may ends up with an approximation far from being accurate. A viable solution is to change the method for sampling the pairs, aiming at statistically exploring such directions where the Lipschitz constant K is massively violated. This is the idea behind the *Adversarial*

Lipschitz Regularization method [147] that allows to rewrite the critic loss function as follows:

$$W(C,G) = \underbrace{\mathbb{E}_{\mathbf{x}_r \sim p_r}[C(\mathbf{x})] - \mathbb{E}_{\mathbf{x}_g \sim p_g}[C(\mathbf{x}_g)]}_{\text{Original critic loss}} + \underbrace{\lambda \cdot \mathbb{E}_{\mathbf{x} \sim p_{r,g}} \left(\frac{|C(\mathbf{x}) - C(\mathbf{x} + \mathbf{r}_{adv})|}{\|\mathbf{r}_{adv}\|_2} - 1 \right)_+^2}_{\text{Adversarial Lipschitz penalty}} \tag{3.20}$$

where $p_{r,g}$ is a combination of the real and generated distributions (meaning that a sample $\mathbf{x}$ can come from both), λ is the regularization hyperparameter, and the adversarial perturbation is defined as

$$\mathbf{r}_{adv} = \underset{\mathbf{r};\, \|\mathbf{r}\|_2 > 0}{\arg\max} \frac{|C(\mathbf{x}) - C(\mathbf{x} + \mathbf{r})|}{\|\mathbf{r}\|_2} \tag{3.21}$$

The introduction of this term is called *Adversarial Lipschitz penalty* and defines WGAN-ALP systems [147].

3.2.3 Cramér distance as biased gradients solution

The use of Wasserstein distance with the regularization terms reported in (3.18) and (3.20) improves significantly the training procedure, 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, and in part to the fact that the Wasserstein distance is an *ideal divergence*[2].

Another important requirement for distance measure used in Machine Learning algorithms is to satisfy the *unbiased sample gradients* condition. This property ensure that[3], 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

$$\mathbb{E}_{x \sim q}[\nabla_\theta \rho(\hat{q}_\ell, q_\theta)] = \nabla_\theta \rho(q, q_\theta) \tag{3.22}$$

where ρ is a general distance, and q_θ denotes the usual parameterized distribution. Better said, having unbiased sample gradients allows to rewrite relation (2.1) as (2.4) saying to solve the same optimization problem. Unfortunately, Wasserstein distance does not dispose of unbiased sample gradients.

The most dangerous consequence of not satisfying condition (3.22) is that the minimum of the expected sample loss $\hat{\theta}^*$ is in general *different* from the minimum of the real Wasserstein loss θ^*:

$$\hat{\theta}^* = \underset{\theta \in \Theta}{\arg\min}\, \mathbb{E}[W(\hat{q}_\ell, q_\theta)] \neq \underset{\theta \in \Theta}{\arg\min}\, \mathbb{E}[W(q, q_\theta)] = \theta^* \tag{3.23}$$

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 3.5). Once again, the solution lies in changing the loss function.

[2] 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. [148]).

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

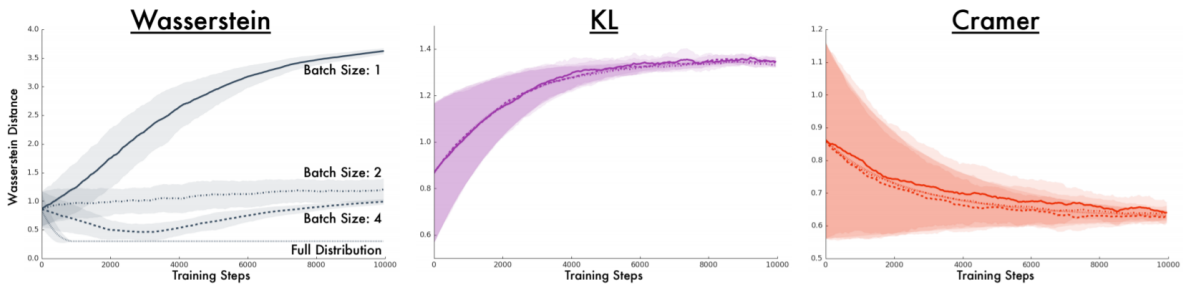


Figure 3.5: Learning curves of three different adversarial models: using Wasserstein distance, Kullback–Leibler divergence and Cramér distance respectively. For the last two, having unbiased sample gradients, the batch size does not prevent from reaching the minimum. Instead, the convergence of Wasserstein distance is heavily conditioned by hyperparameter setting. Plots reported from Ref. [148].

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

$$l_n(p,q) = \sup_{f \in \mathcal{F}_m} | \, \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)] \, | \tag{3.24}$$

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 $n = 2$. By choosing the definition (3.24) 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 $\mathbf{x}, \mathbf{x}'$ and $\mathbf{y}, \mathbf{y}'$ be independent random variables distributed according to p and q respectively. For

$$f^*(\boldsymbol{a}) = \mathbb{E}_{\mathbf{y}'\sim q}\left[\, \|\boldsymbol{a} - \mathbf{y}'\|_2 \, \right] - \mathbb{E}_{\mathbf{x}'\sim p}\left[\, \|\boldsymbol{a} - \mathbf{x}'\|_2 \, \right] \tag{3.25}$$

the energy distance can be written as

$$\mathcal{E}(p,q) = \mathbb{E}_{\mathbf{x}\sim p}[f^*(\mathbf{x})] - \mathbb{E}_{\mathbf{y}\sim q}[f^*(\mathbf{y})] \tag{3.26}$$

Starting from expression (3.26), 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 (3.25) as follows

$$f_h^*(\boldsymbol{a}) = \mathbb{E}_{\mathbf{x}'_g\sim p_g}\left[\, \|h(\boldsymbol{a}) - h(\mathbf{x}'_g)\|_2 \, \right] - \mathbb{E}_{\mathbf{x}'_r\sim p_r}\left[\, \|h(\boldsymbol{a}) - h(\mathbf{x}'_r)\|_2 \, \right] \tag{3.27}$$

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

$$\min_G \max_h \mathcal{L}_g = \min_G \max_h \left(\mathbb{E}_{\mathbf{x}_r\sim p_r}[f_h^*(\mathbf{x}_r)] - \mathbb{E}_{\mathbf{x}_g\sim p_g}[f_h^*(\mathbf{x}_g)] \right) \tag{3.28}$$

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 $\mathcal{L}_g$:

$$\mathcal{L}_{\text{critic}} = -\mathcal{L}_g + \lambda \cdot \mathbb{E}_{\hat{\mathbf{x}} \sim p_{\hat{\mathbf{x}}}} [\, (\|\nabla_{\hat{\mathbf{x}}} f_h^*(\hat{\mathbf{x}})\|_2 - 1)^2 \,] \tag{3.29}$$

where $\hat{\mathbf{x}} = \varepsilon \mathbf{x}_r + (1 - \varepsilon)\mathbf{x}_g$ 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 (3.28) is based on the loss maximization with respect to the map h, and not with respect to the critic itself[4] (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 [148].

3.3 Brief introduction to Normalizing Flows

Relying on a series of invertible and differentiable transformations (namely *bijective* transformations), Normalizing Flows offer a general mechanism to define expressive probability distributions, starting from a (simple) base distribution. According to George Papamakarios and others [149], Normalizing Flows operate by pushing a simple density through a series of transformations to produce a richer, potentially more multi-modal distribution, like a fluid flowing through a set of tubes.

Let $\mathbf{x}$ be a d-dimensional real vector, and suppose we would like to define a joint distribution over $\mathbf{x}$. The main idea behind Flow-based models it to represent $\mathbf{x}$ as a transformation T of a real vector $\mathbf{z}$ sampled from $p_{\text{z}}(\mathbf{z})$:

$$\mathbf{x} = T(\mathbf{z}) \qquad \text{where} \qquad \mathbf{z} \sim p_{\text{z}}(\mathbf{z}) \tag{3.30}$$

where $p_{\text{z}}(\mathbf{z})$ is typically referred to as *base distribution* of the Flow-based model. Both the transformation T and the base distribution $p_{\text{z}}(\mathbf{z})$ can be described by a set of parameters (denote them as θ and φ, respectively), hence inducing a family of distributions over $\mathbf{x}$ parameterized by $\{theta, \varphi\}$.

By requiring that T is *invertible*, both T and T^{-1} are *differentiable*, and that also $\mathbf{z}$ is a d-dimensional vector, under these conditions the probability density function of $\mathbf{x}$ is well-defined and can be obtained by a (simple) change of variables:

$$p_{\text{x}}(\mathbf{x}) = p_{\text{z}}(\mathbf{z}) |\det J_T(\mathbf{z})|^{-1} \qquad \text{where} \qquad \mathbf{z} = T^{-1}(\mathbf{x}) \tag{3.31}$$

Equivalently, we can also write $p_{\text{x}}(\mathbf{x})$ in terms of the Jacobian of the T^{-1} transformation:

$$p_{\text{x}}(\mathbf{x}) = p_{\text{z}}\left(T^{-1}(\mathbf{x})\right) |\det J_{T^{-1}}(\mathbf{x})| \tag{3.32}$$

[4]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 $h(\mathbf{x}_r) \approx 0$ for $\mathbf{x}_r \sim p_r$.

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 $\mathbf{x}_r$ from the data space
      Sample $\mathbf{z}, \mathbf{z}'$ from the latent space
      Sample a random number ε from $\mathcal{U}(0, 1)$
      $\mathbf{x}_g \leftarrow G\left(\mathbf{z}; \boldsymbol{\theta}^{(t)}\right)$
      $\mathbf{x}'_g \leftarrow G\left(\mathbf{z}'; \boldsymbol{\theta}^{(t)}\right)$
      $\hat{\mathbf{x}} \leftarrow \varepsilon \mathbf{x}_r + (1 - \varepsilon)\mathbf{x}_g$
      $f_r^* \leftarrow \left\| h\left(\mathbf{x}_r; \tilde{\boldsymbol{w}}^{(s-1)}\right) - h\left(\mathbf{x}'_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2 - \left\| h\left(\mathbf{x}_r; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2$
      $f_g^* \leftarrow \left\| h\left(\mathbf{x}_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) - h\left(\mathbf{x}'_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2 - \left\| h\left(\mathbf{x}_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2$
      $\mathcal{L}_g \leftarrow f_r^* - f_g^*$
      $f_i^* \leftarrow \left\| h\left(\hat{\mathbf{x}}; \tilde{\boldsymbol{w}}^{(s-1)}\right) - h\left(\mathbf{x}'_g; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2 - \left\| h\left(\hat{\mathbf{x}}; \tilde{\boldsymbol{w}}^{(s-1)}\right) \right\|_2$
      $\mathcal{L}^{(i)}_{\text{critic}} \leftarrow -\mathcal{L}_g + \lambda \cdot (\|\nabla_{\hat{\mathbf{x}}} f_i^*\|_2 - 1)^2$
    end for
    $\tilde{\boldsymbol{w}}^{(s)} \leftarrow \tilde{\boldsymbol{w}}^{(s-1)} + \eta \cdot \nabla_{\boldsymbol{w}} \left(\frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}^{(i)}_{\text{critic}} \right)$
  end for
  $t \leftarrow t + 1$
  for $i = 1, \ldots, \ell$ **do**
    Sample $\mathbf{x}_r$ from the data space
    Sample $\mathbf{z}, \mathbf{z}'$ from the latent space
    Sample a random number ε from $\mathcal{U}(0, 1)$
    $\mathbf{x}_g \leftarrow G\left(\mathbf{z}; \boldsymbol{\theta}^{(t-1)}\right)$
    $\mathbf{x}'_g \leftarrow G\left(\mathbf{z}'; \boldsymbol{\theta}^{(t-1)}\right)$
    $\hat{\mathbf{x}} \leftarrow \varepsilon \mathbf{x}_r + (1 - \varepsilon)\mathbf{x}_g$
    $f_r^* \leftarrow \left\| h\left(\mathbf{x}_r; \tilde{\boldsymbol{w}}^{(s)}\right) - h\left(\mathbf{x}'_g; \tilde{\boldsymbol{w}}^{(s)}\right) \right\|_2 - \left\| h\left(\mathbf{x}_r; \tilde{\boldsymbol{w}}^{(s)}\right) \right\|_2$
    $f_g^* \leftarrow \left\| h\left(\mathbf{x}_g; \tilde{\boldsymbol{w}}^{(s)}\right) - h\left(\mathbf{x}'_g; \tilde{\boldsymbol{w}}^{(s)}\right) \right\|_2 - \left\| h\left(\mathbf{x}_g; \tilde{\boldsymbol{w}}^{(s)}\right) \right\|_2$
    $\mathcal{L}^{(i)}_g \leftarrow f_r^* - f_g^*$
  end for
  $\boldsymbol{w}^{(t)} \leftarrow \tilde{\boldsymbol{w}}^{(s)}$
  $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta \cdot \nabla_{\boldsymbol{\theta}} \left(\frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}^{(i)}_g \right)$
end while
return $\boldsymbol{\theta}^{(t)}$ (Resulting generator parameters)

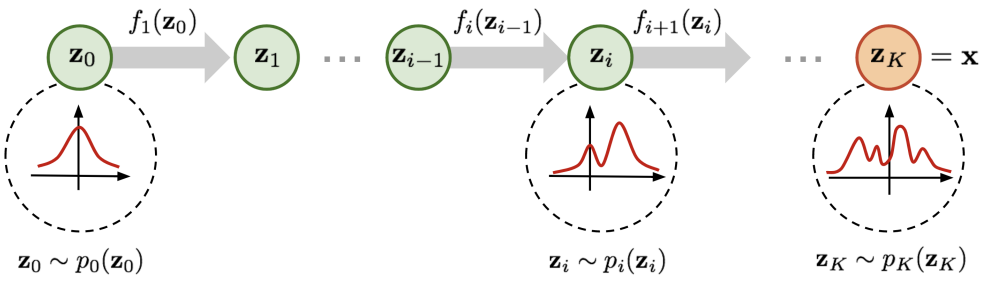


Figure 3.6: Pedagogical representation of a Flow-based model transforming the base distribution $p_{\mathrm{z}}(\mathbf{z})$ into the target one $p_{\mathrm{x}}(\mathbf{x})$ step by step. Figure reproduced from Ref. [150].

The Jacobian $J_T(\mathbf{z})$ is the $d \times d$ matrix of all the partial derivatives of T given by:

$$\begin{bmatrix} \frac{\partial T_1}{\partial z_1} & \cdots & \frac{\partial T_1}{\partial z_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial T_d}{\partial z_1} & \cdots & \frac{\partial T_d}{\partial z_d} \end{bmatrix} \tag{3.33}$$

Hence, we can define a by Flow-based model by implementing T (or T^{-1}) with a neural network and assuming, without any loss of generality, $p_{\mathrm{z}}(\mathbf{z})$ to be a multivariate Gaussian.

An important property of invertible and differentiable transformations is that they are *composable*. Given two such transformations T_1 and T_2, their composition $T_1 \circ T_2$ is also invertible and differentiable. Consequently, we can build complex transformations by composing multiple instances of simpler transformations, and be sure to still satisfy the requirements for calculating the density $p_{\mathrm{x}}(\mathbf{x})$. In practice, it is common to chain together multiple transformations to obtain $T = T_K \circ \cdots \circ T_1$, where each T_k transforms $\mathbf{z}_{k-1}$ into $\mathbf{z}_k$, assuming $\mathbf{z}_0 = \mathbf{z}$ and $\mathbf{z}_K = \mathbf{x}$. A schematic representation of the "flow" followed by the base distribution $p_{\mathrm{z}}(\mathbf{z})$ through the transformation $T_1, \ldots, T_K$ is depicted in Figure 3.6.

Interestingly, a Flow-based model enables two different operations: *sampling* from the model via Eq. (3.30), and *evaluating* the model's density via Eq. (3.32). The two operations have different computational requirements, since the first needs to sample from $p_{\mathrm{z}}(\mathbf{z})$ and compute the forward transformation T, while the second relies on the computation of both the inverse transformation T^{-1} and its Jacobian determinant [149].

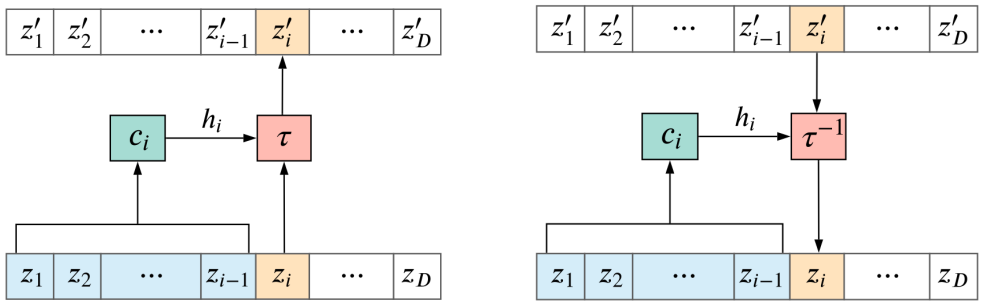
Since with Eq. (3.32) Flow-based models expose an explicit expression for the density $p_{\mathrm{x}}(\mathbf{x})$, the training criterion of such algorithms is simply the negative log-likelihood over the training data, namely

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \log p_{\mathrm{x}}(\mathbf{x}) \tag{3.34}$$

3.3.1 Autoregressive Flows

For constructing the Flows, we rely on a chain of invertible and differentiable transformations $T = T_K \circ \cdots \circ T_1$, where each transformation represents a (single) building block, that must have a tractable inverse and Jacobian determinant. In practice, either T_k or T_k^{-1} can be implemented using a neural network f_{θ_k} for which follows

$$f_{\theta_k} \to T_k: \quad \mathbf{z}_k = f_{\theta_k}(\mathbf{z}_{k-1}) \qquad\qquad f_{\theta_k} \to T_k^{-1}: \quad \mathbf{z}_{k-1} = f_{\theta_k}(\mathbf{z}_k) \tag{3.35}$$



(a) *Forward* (b) *Inverse*

Figure 3.7: Illustration of the i-th step of an Autoregressive Flow. Figure reproduced from Ref. [149].

How to use neural networks to define forward/inverse transformations that ensure an efficient computation of both the inverse and the Jacobian determinant without compromising the expressive power of the model is one of the major challenges in designing Flow-based models. A possible solution is to rely on *Autoregressive Flows* [151, 152], namely Flow-based models based on the following forward transformation:

$$z'_i = \tau(z_i; \boldsymbol{h}_i) \qquad \text{where} \qquad \boldsymbol{h}_i = c_i(\mathbf{z}_{<i}) \tag{3.36}$$

where τ is called the *transformer* and c_i the i-th *conditioner*. The forward transformation is illustrated in Figure 3.7a. By definition, the transformer is a strictly monotonic function of z_i (hence invertible) and is parameterized by $\boldsymbol{h}_i$. Conceptually, the transformer describes the action of the flow on z_i to result into z'_i. On the other hand, the conditioner controls the parameter $\boldsymbol{h}_i$ of the transformer, and can then modifies its behavior. No constraints are applied to the conditioner except for the list of variables that can take as input, hence can be parameterized with a neural network.

As long as the transformer is invertible, the above construction is invertible for any choice of τ and c_i. Given $\mathbf{z}'$, we can compute $\mathbf{z}$ iteratively as follows:

$$z_i = \tau^{-1}(z'_i; \boldsymbol{h}_i) \qquad \text{where} \qquad \boldsymbol{h}_i = c_i(\mathbf{z}_{<i}) \tag{3.37}$$

The inverse transformation is illustrated in Figure 3.7b. It should be noticed that if in the forward computation each $\boldsymbol{h}_i$, and therefore z'_i, can be computed in parallel, in the inverse computation all $\mathbf{z}_{<i}$ need to be computed before z_i, so that $\mathbf{z}_{<i}$ is available to the conditioner for computing $\boldsymbol{h}_i$.

Constructing an Autoregressive Flow thus requires to take a choice on how to implement either the transformer and the conditioner functions. Among the various species available in literature, in the following two types of transformers and a single conditioner are briefly discussed. For further details refer to Ref. [149].

Affine transformer

One of the simplest possible choices for the transformer is the class of *affine functions*:

$$\tau(z_i; \boldsymbol{h}_i) = \alpha_i \, z_i + \beta_i \qquad \text{where} \qquad \boldsymbol{h}_i = \{\alpha_i, \beta_i\} \tag{3.38}$$

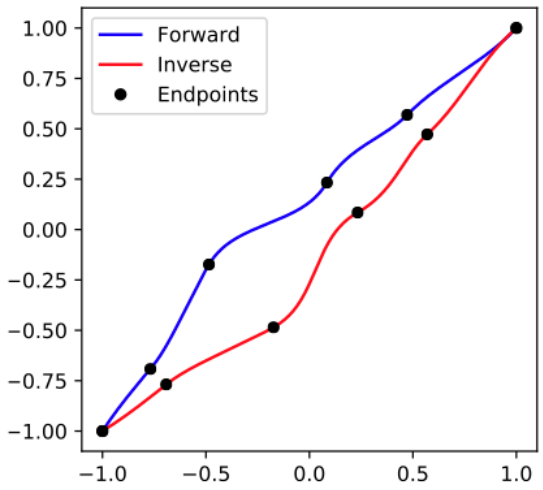


Figure 3.8: Example of a spline-base transformer with 5 segments. Each segment is a monotonic rational-quadratic function, which can be easily inverted [153]. The locations of the endpoints (black dots) parameterize the spline. Figure reproduced from Ref. [149].

Invertibility is guaranteed if $\alpha_i \neq 0$. To this end we can reformulate the problem as

$$\tau(z_i; \boldsymbol{h}_i) = \exp(\tilde{\alpha}_i)\, z_i + \beta_i \qquad \text{where} \qquad \boldsymbol{h}_i = \{\tilde{\alpha}_i, \beta_i\} \tag{3.39}$$

where $\tilde{\alpha}_i$ is now an unconstrained parameter. Autoregressive Flows with affine transformers are attractive because of their simplicity and analytical tractability, but suffer from a limited expressivity.

Spline-based transformer

To overcome the limits of the affine functions, we can implement the transformer as a monotonic *spline* [153], namely a piecewise function consisting of K segments, where each segment is represented by a function easily invertible (such as a polynomial).

Masked conditioner

This approach uses a single, typically feed-forward neural network that takes as input $\mathbf{z}$ and outputs the entire sequence $(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_d)$ in one pass. Despite efficient to evaluate as, given $\mathbf{z}$, all the parameters $\boldsymbol{h}$ can be obtained in one neural network pass, it requires that the FNN obeys to the autoregressive nature of the conditioner, namely that an output $\boldsymbol{h}_i$ cannot depend on inputs $\mathbf{z}_{>i}$. A simple way to remove invalid connections among the FNN neurons is by multiplying each weight matrix elementwise with a binary matrix, called *mask*, describing the autoregressive structure of the network. A general procedure for constructing masks for multilayer perceptron is described in Ref. [154].

4 Lamarr: the LHCb flash-simulation option

The detailed simulation of the physics processes occurring within the LHCb detector is tremendously expensive in terms of CPU hours, enough to take down the majority of the resources available to the experiment. Reducing the computational cost for simulation production is an unavoidable requirement to fulfill the upcoming and future demands for simulations expected from the physics working groups. In this thesis work, we propose Lamarr, the flash-simulation option for LHCb, designed to provide the experiment with the fastest solution for simulation production. This chapter is devoted to present and detail the Lamarr project. Section 4.1 provides a comprehensive introduction Lamarr, highlighting development motivations and design details. Section 4.2 describes the parameterizations adopted to reproduce the high-level response of the Tracking system, while Section 4.3 is devoted to detail the machine-learning-powered solutions chosen for the PID system models. Lastly, Section 4.4 describes the strategies currently under investigation to properly parameterize the electromagnetic calorimeter, that includes to face the particle-to-particle correlation problem. The integration of the Lamarr project within the LHCb Simulation Software is demanded to the next chapter.

4.1 Introduction

Disposing of statistically significant simulated samples is an unavoidable requirement for the analysis of most of the data collected by the LHCb experiment. The simulation of the high-energy collisions, of the decay processes, of the radiation-matter interactions occurring within the detector, of the electronics, and finally of the data acquisition pipeline is crucial to interpret signal, reject background contributions and perform efficiency studies. In order to build a library of interesting decay processes that may be relevant to be analyzed for the wide physics program of the LHCb Collaboration, analysts request thousands of simulated samples corresponding to different signals, while sharing the same detector description and acquisition conditions. Under these assumptions, it is not surprising that LHCb has spent more than 90% of the total CPU budget for simulation production during LHC Run 2 [69].

The upgraded version of the experiment is designed to collect data at a higher luminosity and with a software trigger capable to select interesting decay modes based on a real-time

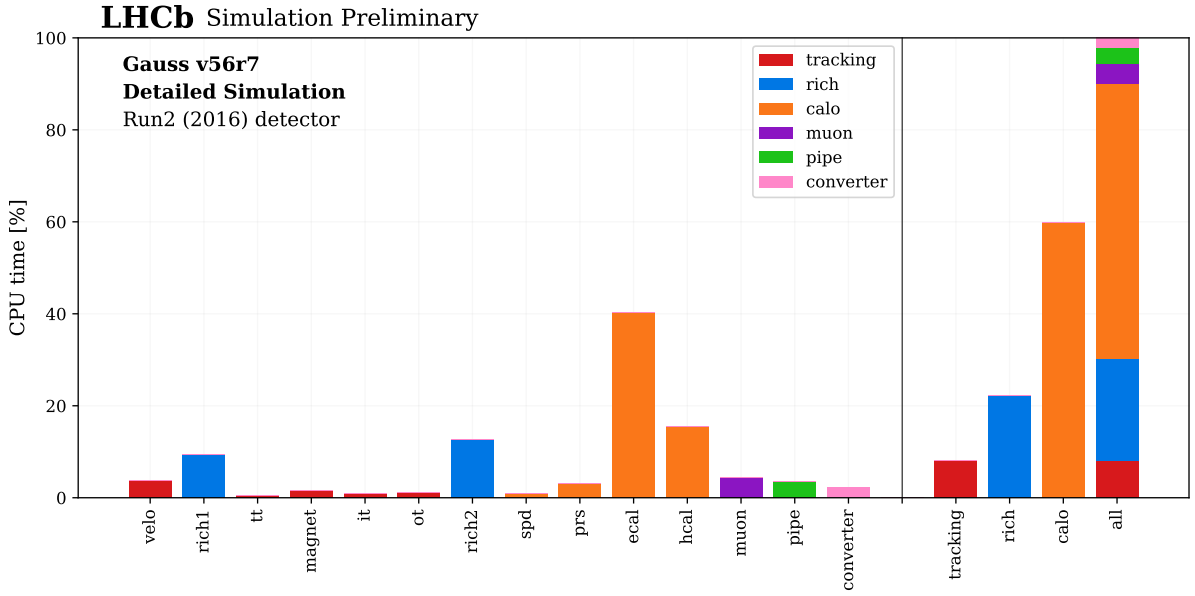


Figure 4.1: Percentage of CPU time spent by each LHCb sub-detector for the detailed simulation of 100 minimum bias events in the Run 2 configuration of the experiment. The timing data reported in this bar chart has been obtained using GAUSS [61] `sim10` (`v56r7`) and has been provided by the LHCb Performance and Regression (LHCbPR) tests system [156].

event reconstruction [155]. As a consequence, the upgraded LHCb detector is able to collect larger and pure datasets during LHC Run 3, enabling the study of rarer decay processes and particles. Meeting the upcoming and future requests for larger simulated samples, due to the wish of retaining constant the ratio between statistical uncertainties on the collected and simulated data, will set significant pressure on the pledged CPU resources.

Within the simulation pipeline, the major CPU consumer is GEANT4 [59, 60] that is responsible to reproduce the radiation-matter interactions between all the stable and quasi-stable particles (e.g., electrons, photons, muons, pions, kaons or protons) and the detector materials. The vast majority of the computing power is devoted to calculate the energy deposited in each active volume of the detector: for instance, in a typical minimum bias event, about 60% of the resources are dedicated to the Calorimeter system, while another 22% to the RICH detectors as depicted in Figure 4.1. This quasi-monopoly of CPU time is due to the cascade of secondary particles that characterizes both the detectors when traversed by long-lived particles and that require to be properly reproduced in the detailed simulation of the LHCb detector response.

Despite the amount of CPU dedicated to simulate raw data, most of the analyses only require very high-level quantities that result from the application of dedicated reconstruction algorithms: for example, a set of aligned hits on the tracking detectors can be combined into a *track* candidate. Spending the majority of a precious resource such as CPU to simulate quantities that will be used only indirectly and then discarded sounds rather sub-optimal, especially for all of those analyses that require large simulated samples to study the adopted selection strategy or to model the statistical distribution of various background contributions relying only on analysis-level variables. On the contrary, providing directly the high-level response of the LHCb detector offers a viable solution to

Figure 4.2: Logo of the LAMARR project.

reduce the pressure on the computing budget without any renounce for the analysts as long as the chosen parameterizations are able to correctly model the errors introduced in the detection and reconstruction steps. Similar parameterizations follow the so-called *flash-simulation* paradigm and can be effectively implemented relying on modern machine learning techniques.

The LHCb Collaboration is spending great efforts in reducing the computational cost for simulation production, in particular developing a simulation framework able to reproduce directly the high-level response of its spectrometer. This novel framework is named LAMARR[1] [3–5] and is the official flash-simulation option for the LHCb experiment. Figure 4.2 shows the logo of the LAMARR project.

My Ph.D. research activity has been mainly dedicated to the development of the LAMARR framework. In particular, large efforts have been spent to the design, training and validation of its underlying parameterizations. The following Sections are devoted to a in-depth description of the LAMARR design, highlighting its modular pipelines and detailing its atomic components, namely the parameterizations. The next chapter instead is dedicated to the LAMARR framework itself, describing how the latter is integrated with the LHCb Simulation Software and concluding with some outlook on its future development

4.1.1 Priorities for flash-simulated samples

Flash-simulation strategies are not designed to entirely replace the detailed simulations. As an example, regardless of the quality one could achieve with a (non-)parametric simulation, detector studies will necessarily remain a use-case for Detailed Simulation, since they require a *deterministic* relation between the physics processes and the sign left by long-lived particles when traverse the detector.

On the other hand, whenever one needs simulations to test if the adopted selection strategy rejects the background contributions sufficiently well or to design the statistical model describing the latter backgrounds before performing the final fit of an analysis, disposing of a flash-simulation option allows to satisfy the analysts request for simulated samples with a lower computational cost. During LHC Run 2, the analysts were used to adopt RAPIDSIM [72] for some of these applications. Operating as a *particle-gun* simulator generating heavy flavour particles with kinematics sampled from input p_T spectra [157], RAPIDSIM offers a rough description of the LHCb geometrical acceptance and a simplified parameterization of the tracking resolution function. The PID information was usually added by using the *resampling* or *transformation* techniques described in

[1]The framework name comes from Hedy Lamarr (1914–2000), who was an Austrian-born American film actress and inventor who pioneered the technology that would one day form the basis for today's communication systems. Read more on `https://en.wikipedia.org/wiki/Hedy_Lamarr`.

Section 4 of Ref. [49]. While relatively coarse with respect to the detailed simulation approach, RapidSim has been a precious tool for LHCb providing the analysts with abundant and cheap samples that would otherwise require thousands of CPU hours to be produced.

RapidSim could be significantly improved by adopting modern machine learning techniques to redesign its parameterizations, like the geometrical acceptance, the tracking resolution function, or the high-level response of the PID system. Integrating such novel simulation framework within the LHCb software stack also enables using the exact same MC physics generators as the official LHCb simulation, including Pythia8 [25] with LHCb-specific tuning [158] and EvtGen [58], in addition to the particle-gun approach. Moreover, providing the same reconstructed physics objects, such as tracks and photons, as the detailed simulation, enables injecting the simulated dataset in the same analysis pipelines. These are the core ideas motivating the Lamarr project.

The development of the Lamarr parameterizations has been driven by the opportunity of potentially replacing the largest fraction of fully-simulated events with synthetic samples obtained via flash-simulations. For example, exploring the simulated datasets in the LHCb bookkeeping engine for 2016, one finds 16.5 billion simulated events, split into 1.8 million files and corresponding to about 1.5 PB of data. In about 76% of the simulated events, the generated signal processes do not include any photons or electrons but only hadrons and muons. Hence, equipping Lamarr with parameterizations that describe the response of the Tracking and PID systems when traversed by charged hadrons (e.g., pions, kaons and protons) and muons, one virtually earns an alternative way to produce the vast majority of the simulated events. Adding the necessary parameterizations to simulate photons would enable generating another 15% of events, while to simulate the remaining 9% one should also model electrons.

Electrons are rather unique objects, challenging the tracking reconstruction algorithms since they are more prone to *multiple scattering*[2] effects while traversing the tracking stations [159]. In addition, electrons may emit *bremsstrahlung radiation*[3] while interacting with the tracking stations upstream the magnet, which makes analysts to search, among the energy clusters in the ECAL detector, candidates for emitted photons and using them to correct a posteriori the measurement of the momentum of particles believed to be electrons. Implementing a flash-simulation for the physics objects involved in such a process is a non-trivial particle-to-particle correlation problem that requires, in the first instance, having a correct model for the ECAL when traversed by photons and then to build a whole LHCb-specific parameterization for the electrons. This is mandatory for enabling Lamarr to virtually offer an alternative solution to simulate the remaining 9% of events involving electrons.

Figure 4.3 shows the pie charts of the simulated data split into the categories described above, summing up either the number of events (left) or the data size (right). From the comparison of the two plots, it is evident that simulations including photons and electrons require larger storage per event with respect to simulations not involving ECAL. The

[2]A charged particle traversing a medium is deflected by many small-angle scatters due to the Coulomb interaction with nuclei. The elastic scattering of charged particles by Coulomb interactions is called *Rutherford scattering*. Read more on `https://en.wikipedia.org/wiki/Rutherford_scattering`

[3]In particle physics, we call *bremsstrahlung* the electromagnetic radiation produced by the deceleration of a charged particle when deflected by another charged particle, typically an electron by an atomic nucleus. Read more on `https://en.wikipedia.org/wiki/Bremsstrahlung`.

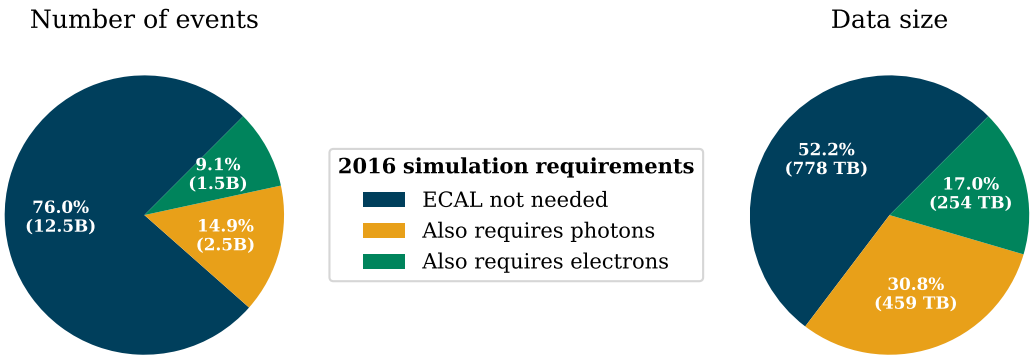


Figure 4.3: Pie charts of the number of events (left) and the corresponding data size (right) as obtained from the LHCb bookkeeping engine for 2016 data taking. The simulations that do not involve the ECAL detector are reported in blue; the ones that requires neutrals are highlighted in yellow, while including also the electrons we obtain the green portions.

reason is that, on average, raw information is more often needed to perform analyses involving the calorimeter and therefore preserved on disk. As a result, we expect that the use of flash-simulations in analyses involving photons or electrons would be less profitable, even if their modeling remains needed to ensure that Lamarr provides a full description of the LHCb detector.

In summary, simulating electrons would require a dedicated Tracking parameterization and a specialized effort in building the response of the PID system, including the ECAL model needed by analysts to reconstruct the energy of an electron track candidate interested by bremsstrahlung radiation. All that would be necessary on top of a parameterization for the Tracking and PID systems when traversed by hadrons and muons, which provides a viable solution to reduce the pressure on CPU offering the fastest option to simulate the majority of the events requested at LHCb for its physics program.

4.1.2 The Lamarr project design

The first attempt of providing LHCb with a flash-simulation option has originated from the desire for a customized version of the Delphes framework [160]. In order to profit as much as possible from the use of parametric strategies to accelerate the computation of high-level quantities, Delphes is designed as a pipeline of modules describing a generic multipurpose detector [70]. Despite the wish of integrating the LHCb software stack with Delphes, the Event Model of the two frameworks is so different that the effort to achieve a perfectly functional integration would supersede the benefit from reusing existing parameterizations which are, in any case, mainly designed for concentric detectors. Hence, at the end of May 2019, the LHCb Simulation Group renounced to rely on Delphes for flash-simulations, preferring to develop an independent project natively integrated with the simulation software stack and powered by machine learning LHCb-tuned models, then we named Lamarr [3–5].

Taking inspiration from the modular layout of Delphes, the Lamarr framework consists of multiple pipelines of parameterizations that follow one another to reproduce the high-level response of the LHCb sub-detectors when traversed by quasi-stable particles

as provided by the MC physics generators. In particular, the LAMARR pipeline can be logically split in two separated chains according to the charge of the generated particles.

A branch treating charged particles will handle the *Tracking* and the *Particle Identification* parameterizations. The parameterization of the Tracking, replacing the simulation and reconstruction of the mark left by charged particles in the tracking stations, is divided in four subsequent blocks describing:

- the geometrical acceptance (see 4.2.2);
- the reconstruction efficiency for different *categories* of tracks (see 4.2.3);
- the resolution effect, effectively smearing the reconstructed quantities with respect to the properties of generated particles (see 4.2.4);
- the uncertainties, obtained by the reconstruction algorithm from the minimization of the χ^2 of the trajectory of a particle through the marks left in the detector, and parameterized with generative models in LAMARR (see 4.2.5).

Charged particles for which a track is successfully reconstructed are associated to a *Particle Identification* (PID) interpretation based on the response of the RICH detectors, of the MUON system and of the calorimeters. The parameterization of such interpretation is symmetrically defined as follows:

- the simulation of the combined response of the two RICH detectors (see 4.3.2);
- the simulation of the response of the MUON system expressed as two likelihood values for the *muon* and *non-muon* hypothesis (see 4.3.3);
- the simulation of a loose, binary flag, named `isMuon`, implemented on programmable electronics in the real experiment, and parameterized with a neural network in LAMARR; (see 4.3.4)
- the simulation of reconstruction procedure combining the RICH and MUON responses into experiment-level classifiers, either as combined likelihoods or using multivariate classifiers. Since additional inputs (notably including the calorimeter response) are used in the real version of the reconstruction with marginal, but non-negligible effect, generative models are used to model the global response, effectively encoding the missing inputs as part of the latent space. Also, since the distributions of the global PID features are drastically different depending on the `isMuon` criterion, two different generative models are trained and they are evaluated selectively based on the predicted response of the `isMuon` pipeline (see 4.3.5).

We expect that charged particles leave a mark in the Tracking system that LAMARR characterizes in terms of geometrical acceptance, efficiency and resolution effects as further described in Section 4.2. The reconstructed tracking variables are then used to compute the response of the PID system for a set of traversing charged particles (i.e., muons, pions, kaons, and protons) as detailed in Section 4.3. In case of neutral particles (e.g., photons), the calorimeters play a key role and, since multiple photons can concur to the energy of a single calorimetric cluster, parameterizing particle-to-particle correlation effects is of major relevance. The solutions investigated so far are reported in Section 4.4. The LAMARR pipelines described above are schematically represented in Figure 4.4 using logical blocks.

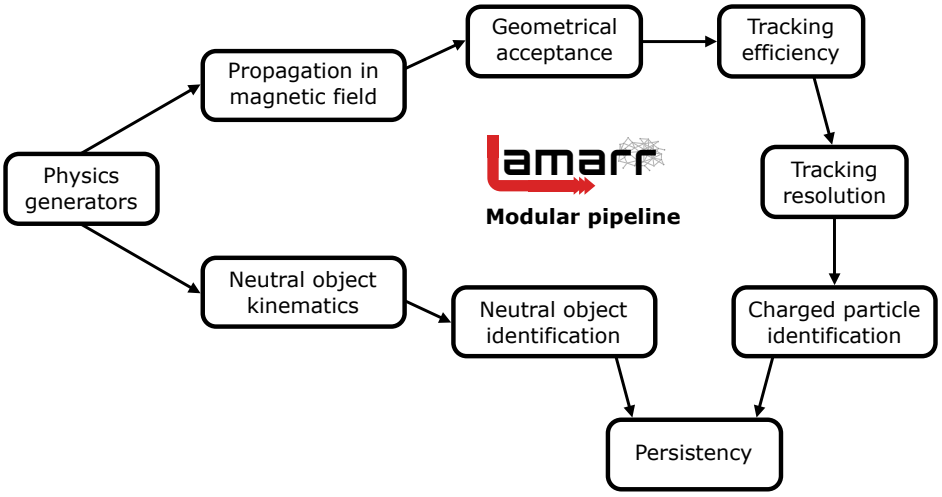


Figure 4.4: Scheme of the LAMARR modular pipeline. According to the charge of the particles provided by the physics generator, two sets of parameterizations are defined: the charged particles are passed through the Tracking and PID models, while the neutral ones follow a different path where the calorimeter modeling plays a key role.

In addition, disposing of a modular layout for flash-simulations gives the opportunity to design also hybrid solutions able to accelerate the production of detailed simulated samples. For instance, fast- or flash-simulation strategies can be adopted on top of a partial detailed simulation to reproduce the response of one or more computationally expensive detectors, like the RICH and the Calorimeter systems, reducing the CPU time needed for simulation production. The LHCb Collaboration is currently investigating several techniques that follow such philosophy using parametric functions [161] or relying on deep generative models [162] aimed at reducing the computational cost for the calorimeter simulation. One of the goal of the LAMARR project is also that of *expanding* these techniques, offering flash-simulation parameterizations for the whole spectrometer.

4.2 Charged particles pipeline: the tracking system

When detected and properly reconstructed, a charged particle traversing the LHCb spectrometer gives origin to a *track*. A track represents the trajectory of a charged particle through the detector, bent by the effect of the magnetic field and described by a collection of hits that fires in correspondence of the active volumes of the tracking detectors. To select a coherent set of hits forming a track, each hit is described in terms of *states*. To define a state, we need to specify the position of the hit along the track, the corresponding track momentum and the covariance matrix between the estimators of these parameters, track-level features extracted from a combination of hits using a Kalman-based[4] procedure [163]. By convention, only few states are defined for each track

[4]The *Kalman filter* is an algorithm that uses a series of measurement collected over time and the corresponding uncertainties to estimate unknown states of the investigated system relying on a parameterization of the joint probability distribution over the states for each time-frame. Read more on `https://en.wikipedia.org/wiki/Kalman_filter`.

at specific z positions that give them meaningful names, like `EndVelo` for states at the end of the VELO, `EndT` for states at the end of the T stations, or `ClosestToBeam` for referring to states close to the LHC proton beam. A specific track state is defined through the following state vector:

$$\vec{x} = \begin{pmatrix} x \\ y \\ t_x \\ t_y \\ q/p \end{pmatrix} \quad \text{where} \quad \begin{cases} t_x = \partial x/\partial z \\ t_y = \partial y/\partial z \\ q = \pm 1 \end{cases} \tag{4.1}$$

The corresponding uncertainties on $\vec{x}$ are given by a 5×5 state covariance matrix, C. The state vector and its covariance matrix are commonly referred to as the *track state*.

LAMARR does not have any notion of hit and does not rely on any explicit geometrical description of the detector, aiming to a fully parametric modeling of the detection and reconstruction procedure [160]. Hence, to completely define a reconstructed track, LAMARR needs to provide the following parameterizations:

- **Propagation.** Model for the trajectory of a charged particle through the dipole magnetic field;
- **Geometrical acceptance.** Model for the probability of a particle to lay within a sensitive area of the detector considering the effects of bending and multiple scattering;
- **Tracking efficiency.** Model for the probability of a track to be successfully reconstructed, declaring also the tracking stations involved to reconstruct the track;
- **Tracking resolution.** Model for the errors introduced to the estimates of the track parameters $\vec{x}$ by the reconstruction algorithms due to, for example, multiple scattering phenomena or imperfections in the detector alignment;
- **Track covariance matrix.** Model for the uncertainty assessed by the Kalman filter and encoded in the track covariance matrix C.

To the purpose of most of the analyses at LHCb, only the `ClosestToBeam` state is relevant, since it is used to evaluate the consistency of a track with a vertex defining quantities such as the *impact parameter*[5] (IP) or to fit decay vertices measuring quantities such as the lifetime of a particle.

The rest of this Section is devoted to discuss how to parameterize the high-level response of the LHCb Tracking system when traversed by charged particles. The parametric function describing the *propagation* of particles in the magnetic field is discussed in Section 4.2.1. How to use neural networks to parameterize both the *geometrical acceptance* and the *tracking efficiency* as a classification problem is detailed respectively in Sections 4.2.2 and 4.2.3. Lastly, Sections 4.2.4 and 4.2.5 show how the *tracking resolution* and the *covariance matrix* of the `ClosestToBeam` track state can be successfully parameterized using deep generative models.

[5]In the LHCb jargon, we call *impact parameter* the minimum distance of a track to a primary vertex.

4.2.1 Propagation through the magnetic field

Predicting the position at which each charged particle crosses the tracking stations and the three-momentum of the corresponding track states does not require to know exactly the trajectory of the traversing particle. Indeed, the tracking stations are positioned in a region of the spectrometer with negligible magnetic field, resulting into straight track segments. In addition, the difference between the momentum along the bending axis (x-axis) as measured upstream and downstream of the magnetic field can be considered constant if one neglects dissipative processes, such as radiation-matter interactions.

Combining these two considerations together, the effect of the magnetic field on the path of a particle can be modeled by a single change of direction of the momentum vector in the xz-plane, in correspondence of a point referred to as z_{kick}. Such parameterization is called *single-kick dipole approximation* and is schematically reported in Figure 4.5.
Using such an approximation, parameterizing the trajectory of charged particles throughout the spectrometer only requires the definition of two parameters:

- Δp_x – constant variation of the momentum p along the bending axis;
- z_{kick} – coordinate of the point where the effect of the magnet is condensed.

The combination of the magnet bending power with the structure of the Tracking system ensures that charged particles experience a variation on the momentum direction Δp_x of about 1.23 GeV/c [165]. Figure 4.6 shows that this constant value describes effectively the dynamics of particles traversing the LHCb magnetic field in a wide momentum range. The two peaks exhibited in the Figure result from Detailed Simulation and reports the momentum variation as measured for particles positively and negatively charged ($q = \pm 1$). In particular, by considering the magnet polarity, it results that

$$\Delta p_x \equiv p'_x - p_x \simeq \begin{cases} -q \cdot 1.23 \text{ GeV}/c & \text{in MagUp conditions} \\ +q \cdot 1.23 \text{ GeV}/c & \text{in MagDown conditions} \end{cases} \tag{4.2}$$

where p_x and p'_x represent the projections along the bending axis of the momentum measured upstream and downstream of the magnetic field, respectively.

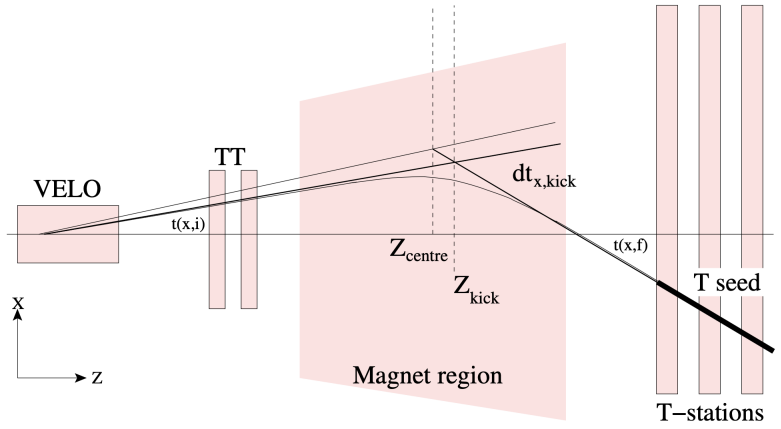


Figure 4.5: Graphical representation of the *single-kick approximation*. The main component of the magnetic field is perpendicular to the drawn plain. Figure reproduced from Ref. [164].

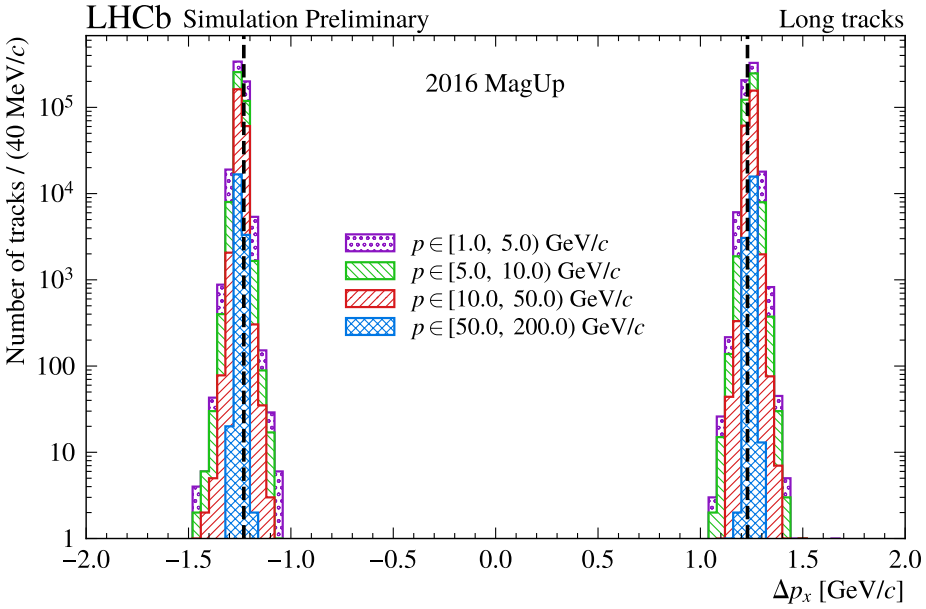


Figure 4.6: Distribution of the *momentum variation* along the bending axis (x-axis) between the track state at the end of the T stations (`EndT`) and the one upstream of the magnetic field (`ClosestToBeam`). Two vertical dashed lines highlight the adopted parameterization that fix the momentum variation: $\Delta p_x = 1.23$ GeV/c. Stacked histograms with different colors and filling strategies show the Δp_x distributions in four p-bins as obtained from Detailed Simulation.

The z_{kick} coordinate depends on the transverse momentum p_T of the particle, reason why the single-bending-point parameterization is also called *p_T-kick approximation*. By considering negligible the variation of the y-projection of the momentum p_y, the conservation law stands that p_z changes with a value given by $p'_z \simeq \sqrt{p_x^2 + p_z^2 - {p'_x}^2}$. Hence, by using a set of trivial trigonometric formulas and by observing the kinematic constraints of the physical system considered, the z_{kick} coordinate can be expressed as follows:

$$z_{\text{kick}} = \frac{x' - x + z \cdot t_x - z' \cdot t'_x}{t_x - t'_x} \qquad \text{where} \qquad \begin{cases} t_x = p_x / p_z \\ t'_x = p'_x / p'_z \end{cases} \tag{4.3}$$

where (x, z, t_x) denotes the coordinates and slope of a track state upstream of the magnetic field, while (x', z', t'_x) reports the same set of features for a downstream track state.

To parameterize the z_{kick} coordinate as a function of the momentum p, LAMARR relies on a parabolic model whose parameters were obtained by studying the propagation of quasi-stable particles within the LHCb Tracking system. Figure 4.7 shows the distribution of z_{kick} as obtained from Eq. (4.3) by using the reconstructed `ClosestToBeam` and `EndT` states provided by the detailed simulation of a sample of b-hadron decays. The kick z-coordinate is reported versus q/p with the momentum values taken from the physics generators. The result of the fit procedure is depicted in Figure 4.7 through a red solid line, together with the corresponding parameters.

4.2.2 Geometrical acceptance

The geometrical acceptance model is defined as the probability that a charged particle lays within the fiducial volume of the detector where reconstructing the track is, at

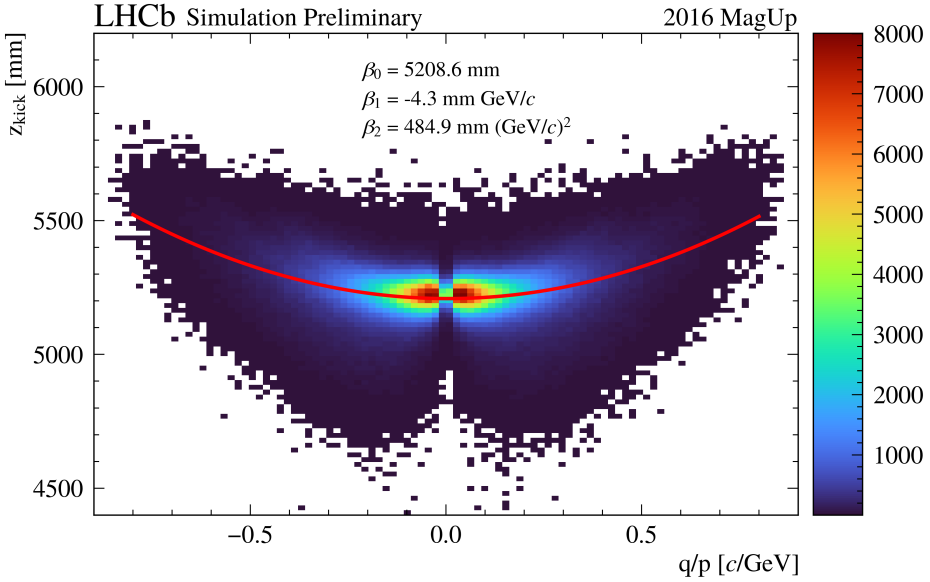


Figure 4.7: LAMARR parameterization of the *propagation* through the LHCb magnetic field of charged particles reconstructed as Long tracks. The x-axis reports the charge of the track over its momentum q/p, whereas the y-axis shows the single-kick approximation point z_{kick}. The color code represents the number of particles in simulation, while the red curve indicates the result of a second-order polynomial fit used to parameterize z_{kick} as a function of q/p. The coefficients β_i of the parabolic model employed in LAMARR are reported on the plot.

least a priori, possible. Most of the particles are either in acceptance or not, so such a probability is either 0 or 1. However, at the boundaries of the fiducial volume, multiple scattering phenomena may occur and, since they are not included in the propagation parameterization, disposing of intermediate probability values becomes relevant.

The current version of LAMARR relies on a *neural network* (NN) for modeling the geometrical acceptance of the LHCb spectrometer. Such *non-parametric approximator* was trained to predict the fraction of particles that lay in acceptance based on the following set of generator-level information:

- the origin vertex position (x, y, z) of the generated particle;
- the logarithm of the momentum $\log_{10}(p)$ of the generated particle;
- the slopes t_x and t_y of the generated particle;
- the pseudorapidity η of the generated particle;
- the azimuthal angle φ of the generated particle;
- the *specie* (electron, muon, or hadron) of the generated particle;
- the charge q of the generated particle.

Despite the large correlation between the slopes (t_x, t_y) and the doublet (η, φ), it was verified that explicitly including both of them results in better-performing models.

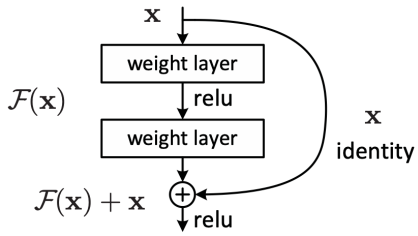


Figure 4.8: Schematic representation of a *residual block* composed by two weight layers that are linked to the previous ones via a *skip connection*. Figure reproduced from Ref. [111].

A training dataset was produced by simulating a cocktail of b-hadron decays with an official configuration of GAUSS [61] `sim10` (`v56r2`), involving PYTHIA8 [25], EVTGEN [58], and GEANT4 [59,60]. Following the steps of the official Data Analysis framework of LHCb, we defined an `inAcceptance` flag for each quasi-stable generated particle and stored it in a tabular dataset. A neural network equipped with *skip connections* [111] was then trained to predict the fraction of particles for which `inAcceptance` is verified, performing a *binary classification task*. Different particle species are prone to multiple scattering effects in different ways, showing specific behavior at the boundaries of the fiducial volume. Hence, to parameterize the "geometrical efficiency" of the LHCb detector a specie-sensitive neural network model was trained using the kinematic information of the MC truth.

Model design and training

The neural network currently adopted for the geometrical acceptance is constructed with a set of 10 fully connected layers of 128 neurons with ReLU activation functions. A final dense layer with a single neuron and a sigmoid activation function follows. The classification task was performed by minimizing the *binary cross-entropy* (BCE) using Adam [119] as optimizer. To prevent the vanishing gradient problem, the hidden layers are implemented as 1-D *residual blocks* [111] where each layer is directly linked to the previous one with what is called a skip connection, schematically represented in Figure 4.8. Using skip connections ensures having non-zero gradients during the *back-propagation* even if the learning gradient of an inner residual block is zero. This allows to scale NN-based models that can be safely trained without any adoptions of customary layer-based regularization strategies [166].

The neural network was implemented and trained using Keras [167] with TensorFlow [168] as back-end. The training procedure is split in two phases, a pre-training with very high learning rate and label smoothing to protect against numerical instabilities, followed by a fine-tuning phase to restore the statistics-motivated loss function, while drastically reducing the learning rate to preserve the stability of the optimization procedure. In practice, at the beginning of the training procedure, the BCE was used with label smoothing set to 0.05, and Adam initialized with a learning rate of 0.01. During the following 200 epochs, such learning rate was exponentially decreased up to 10^{-4}. Then, fixed the learning rate to the last scheduled value and restored to zero the label smoothing, the model weights were fine-tuned for 100 more epochs. The complete list of hyperparameters used to define and train the model for the geometrical acceptance is reported in Table 4.1

To train this 10-layer neural network, a dataset of $\mathcal{O}(10^8)$ detailed simulated particles was prepared. The training procedure was performed on a fraction of 50% of the dataset. An independent 10% portion of the sample was used, during the training, to ensure that no effect of *overtraining* was present. Lastly, the remaining 40% of the dataset was

	training	*fine-tuning*
skip connections [111]	✔	✔
input shape	(None, 12)	(None, 12)
input preprocessing	✔	✔
n hidden layers	10	10
n hidden neurons	128	128
hidden activation functions	ReLU	ReLU
hidden kernel regularizer	L2	L2
L2 regularization factor [169]	5×10^{-4}	5×10^{-4}
output activation function	sigmoid	sigmoid
output shape	(None, 1)	(None, 1)
optimizer	Adam	Adam
learning rate	0.01	1×10^{-4}
loss function	BCE	BCE
BCE label smoothing	0.05	✖
learning rate scheduling	ExpDecay	✖
scheduling decay rate	0.1	-
scheduling decay steps	20000	-
batch-size	25000	25000
batches per epoch	~ 200	~ 200
n epochs	200	100

Table 4.1: Hyperparameters of the NN-based model for the acceptance.

preserved for validation and performance measurement. The left plot of Figure 4.9 reports the learning curves of the trained model: comparing the BCE values obtained on the training and test samples, we do not observe evidence of overtraining. The right plot of Figure 4.9 shows the evolution of the *Area Under the Curve* (AUC) for the *Receiver Operating Characteristic* (ROC) curve, that illustrates the performance achieved by the neural network in accomplishing the binary classification task. The change in the definition of the loss function between the two phases of the training motivates the dip immediately after the beginning of the second phase. Then the fine-tuning procedure quickly adapts the weights to minimize the BCE without label smoothing. The fact that the model exhibits better performance on the test set rather than on the training one is probably due to an L2 weights regularizer [169] applied to the neural network.

Validation studies

The geometrical acceptance model has been validated by applying the parameterization to a dataset never used during the training and that counts about 3 million simulated particles. The neural network was trained to predict the fraction of generated particles laying within the LHCb Tracking system. Such fraction is parameterized as the probability that the generator-level kinematics and the specie of the input particles verify the `inAcceptance` flag. Hence, to assess the performance of the trained model, we compare the kinematic distributions of the particles *in acceptance* with the ones obtained by weighting the

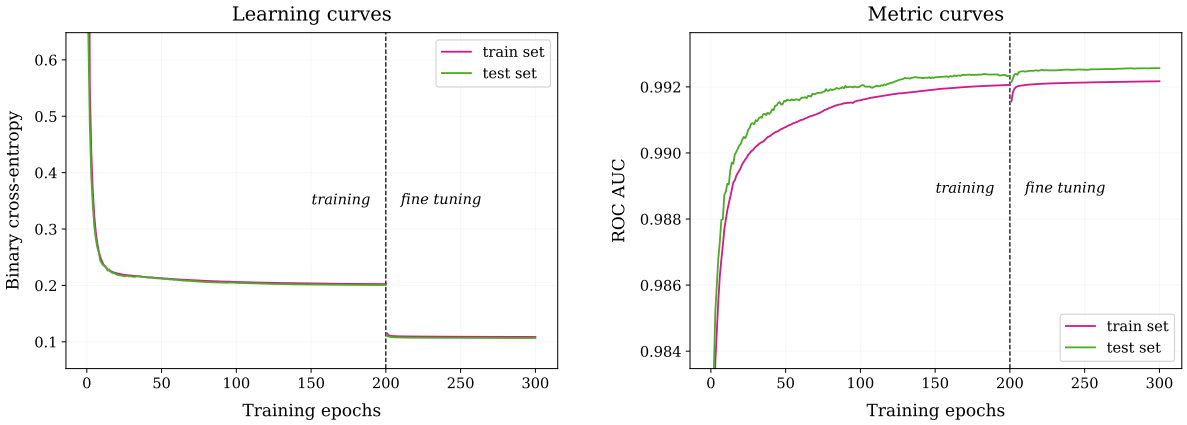


Figure 4.9: Learning and metric curves of a L2-regularized deep neural network [169] trained to parameterize the LHCb geometrical acceptance. The evolution of the *BCE loss function* is reported on the left. The right plot shows how the *ROC AUC score* improved during the training and fine-tuning steps. The training procedure consisted of 200 epochs where label smoothing and learning rate scheduling are used. A fine-tuning phase followed counting 100 epochs with label smoothing and learning rate scheduling strategies disabled.

kinematic distributions of the *generated* particles with the predicted probability.

The validation plots of the geometrical acceptance model for the electrons as a function of the pseudorapidity η in four longitudinal momentum p_z bins is depicted in Figure 4.10. The pseudorapidity distributions of the generated electrons are shown in grey, while what lays in acceptance is highlighted using blue-hatched histograms. The results of the trained specie-sensitive neural network is superimposed with a red solid-line. Figures 4.11 and 4.12 show the same comparisons for muons and hadrons (i.e., pions, kaons, and protons), respectively.

As highlighted by the reported histograms, the NN-based model succeeds in parameterizing the LHCb geometrical acceptance in a wide range of the kinematic space of the generated particles, taking into account correctly the specie of such particles.

4.2.3 Tracking efficiency

When a charged particle has kinematics such that lays in the LHCb detector acceptance, we expect that it deposits some energy in at least a portion of the Tracking system. The reconstruction algorithms can identify track segments involving only the VELO, the Trigger Tracker (TT), or the tracking stations installed downstream of the magnet (T1, T2, and T3), and then combine them into several track classes, depending on the set of detectors in which matching hits are found [163, 170]. The track classes of major relevance for analysis purposes are schematically shown in Figure 4.13 and further described in the following:

- **Long tracks.** Class of tracks that traverse the full Tracking system. They have hits in both the VELO and the T stations, an optionally in TT. Traversing the full magnetic field, they are characterized by the most precise momentum estimate, and therefore are the most important class of tracks for physics analyses.

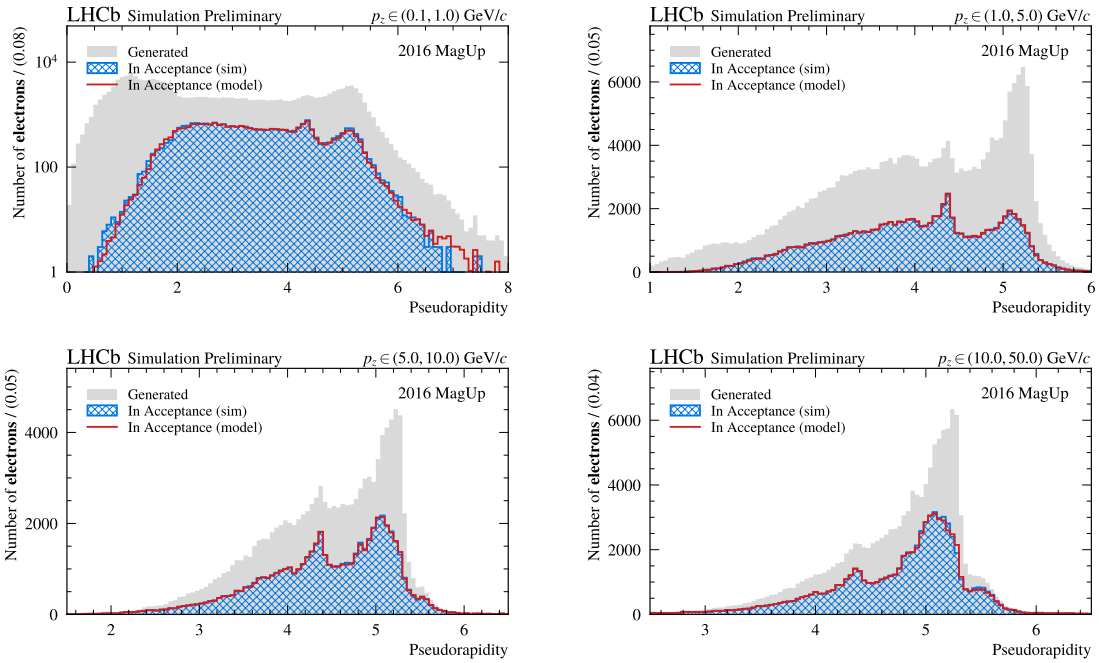


Figure 4.10: Validation plots of the acceptance model for *electrons* as a function of the pseudorapidity η in four longitudinal momentum p_z bins. The kinematics distributions of the generated electrons are represented as light grey shaded histograms. The distributions of electrons in acceptance are shown through blue-hatched histogram, while their parameterization as modeled by a deep neural network is superimposed using red solid-line histograms.

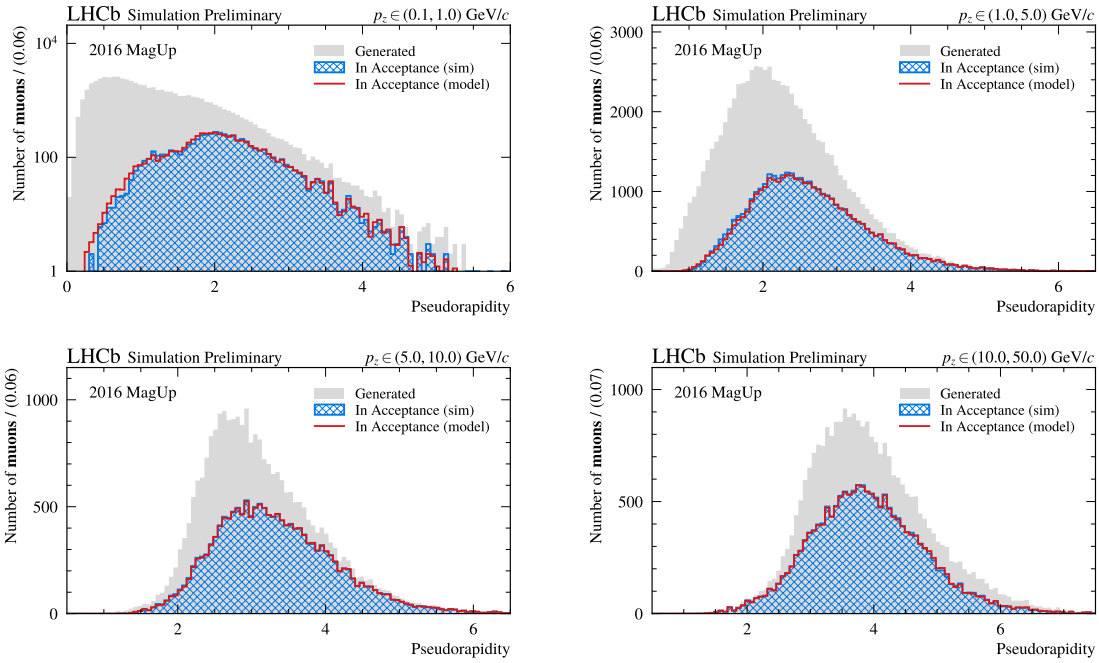


Figure 4.11: Validation plots of the acceptance model for *muons* as a function of the pseudorapidity η in four longitudinal momentum p_z bins. The kinematics distributions of the generated muons are represented as light grey shaded histograms. The distributions of muons in acceptance are shown through blue-hatched histogram, while their parameterization as modeled by a deep neural network is superimposed using red solid-line histograms.

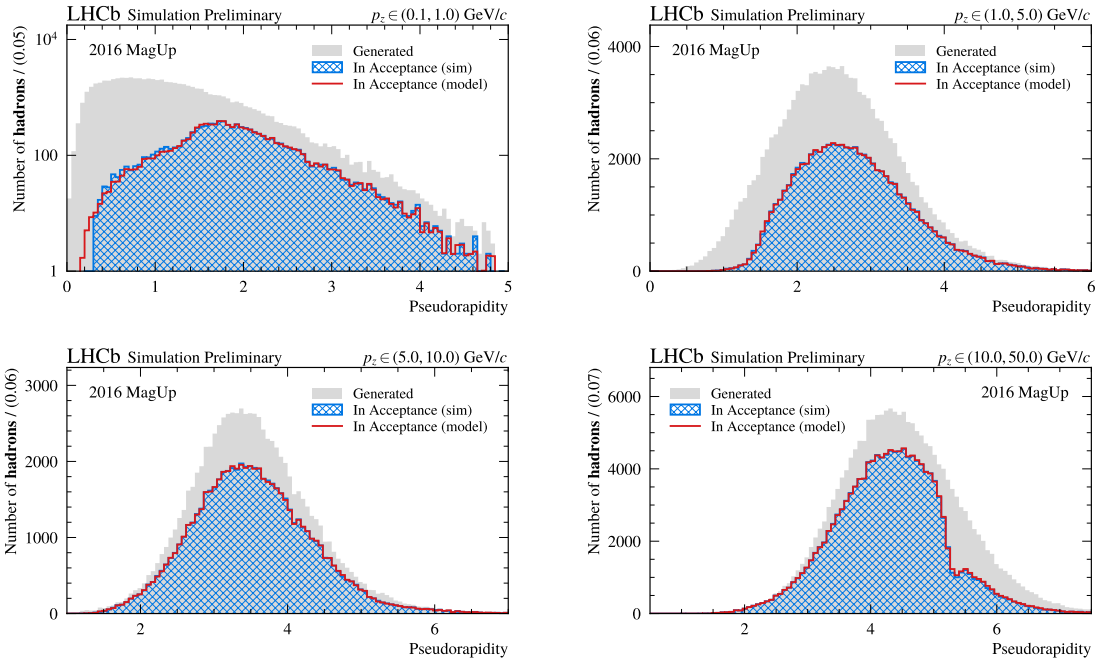


Figure 4.12: Validation plots of the acceptance model for *hadrons* as a function of the pseudorapidity η in four longitudinal momentum p_z bins. The kinematics distributions of the generated hadrons are represented as light grey shaded histograms. The distributions of hadrons in acceptance are shown through blue-hatched histogram, while their parameterization as modeled by a deep neural network is superimposed using red solid-line histograms.

- **Upstream tracks.** Class of tracks that only pass through the VELO and TT stations. In general, their momentum is too low to traverse the magnet and reach the T stations. However, they pass through the RICH1 detector and may generate Cherenkov photons if they have $p > 1$ GeV/c. They are therefore also used to understand backgrounds in the particle identification algorithm of the RICH.

- **Downstream tracks.** Class of tracks that only pass through the TT and T stations. They are important for the reconstruction of long lived particles, such as K^0_S and Λ, that decay outside of the VELO acceptance.

- **VELO tracks.** Class of tracks that only involve the VELO. They are typically large-angle or backward tracks, which are useful for primary vertex reconstruction.

- **T tracks.** Class of tracks that only involve the T stations. They are typically produced in secondary interactions, but are still useful during the treatment of RICH2 data for particle identification.

Given their definition, distinguishing VELO and T tracks from non-reconstructed particles is probably outside of the scope of flash-simulations. Actually, taking a particle in acceptance with hits in the VELO, the fact that no match is found in the TT detector, thus originating a VELO track, is due to second-order effects not directly modeled by LAMARR. In the same way, T tracks are typically produced by secondary interactions, whose parameterization is, as of today, beyond the scope of LAMARR. Hence, both VELO and T tracks are considered as not reconstructed in the current version of the tracking

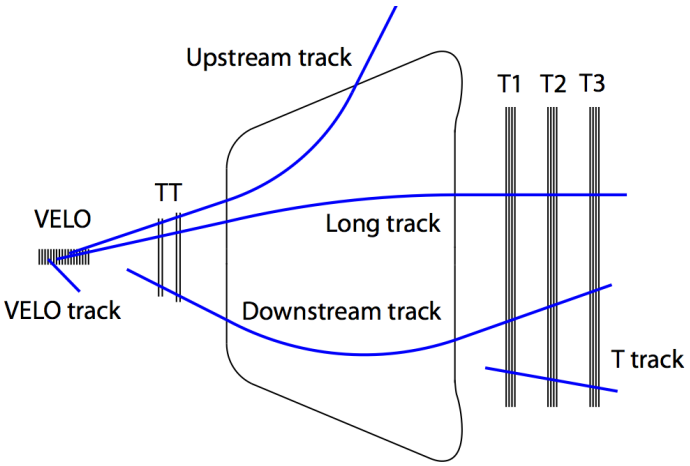


Figure 4.13: Schematic representation of the LHCb Tracking system that consists of a vertex locator (VELO), a large-area tracker located upstream of the magnet (TT), and three tracking stations placed downstream of the magnet (T1, T2, and T3). Track segments reconstructed involving one or more of these detectors can be combined into different track classes: namely *Long*, *Upstream*, *Downstream*, *VELO*, and *T* tracks.

efficiency model, whose aim is then to predict the probability that a charged particle is reconstructed as a Long, Upstream, or Downstream track.

For parameterizing the tracking efficiency, LAMARR relies on a neural network equipped with skip connections [111] that takes as input the same set of generator-level variables adopted by the model for acceptance described in Section 4.2.2. The training and validation of this *non-parametric approximator* can count on $\mathcal{O}(3 \times 10^7)$ particles resulting from Detailed Simulation and reconstructed by using the BRUNEL and DAVINCI applications [171]. Since we are interested only in the particles for which reconstructing the tracks is possible, such a dataset only contains candidates that verify the `inAcceptance` flag. Generalizing what has been done for the geometrical acceptance, this novel specie-sensitive neural network was trained to perform a *multi-class classification task*. The aim is to predict whether a generated particle in acceptance will be reconstructed based on its specie and kinematics. In case of a positive answer, the parameterization should also indicate which tracking detectors are involved in the reconstruction procedure, forecasting the probability that such particle will be reconstructed as a Long, Upstream, or Downstream track [5].

Model design and training

The current version of the tracking efficiency model relies on a 10-layer neural network with 128 neurons in each hidden layer and ReLU activation functions. The network output is moderated by a final dense layer with 4 neurons (one per track class, including the non-reconstructed one) with a softmax as an activation function. The training procedure was driven by the minimization of the *categorical cross-entropy* (CCE) by using Adam as an optimizer.

Most of the training strategies are the same adopted by the geometrical acceptance model. As described in Section 4.2.2, the use of skip connections is crucial to avoid the gradient vanishing problem [111]. The training procedure was split into two phases also

	training	*fine-tuning*
skip connections [111]	✔	✔
input shape	(None, 12)	(None, 12)
input preprocessing	✔	✔
n hidden layers	10	10
n hidden neurons	128	128
hidden activation functions	ReLU	ReLU
hidden kernel regularizer	L2	L2
L2 regularization factor [169]	5×10^{-4}	5×10^{-4}
output activation function	softmax	softmax
output shape	(None, 4)	(None, 4)
optimizer	Adam	Adam
learning rate	0.01	$\sim 1.6 \times 10^{-4}$
loss function	CCE	CCE
CCE label smoothing	0.05	✖
learning rate scheduling	ExpDecay	ExpDecay
scheduling decay rate	0.1	0.1
scheduling decay steps	35000	50000
batch-size	25000	25000
batches per epoch	~ 200	~ 200
n epochs	300	200

Table 4.2: Hyperparameters of the NN-based model for the efficiency.

in this case. During the first phase, the CCE was used with label smoothing set to 0.05, and Adam initialized with a learning rate of 0.01. During the following 300 epochs, such learning rate was exponentially decreased up to 10^{-4}. During the second phase, the decay steps of the scheduling were increased to slow the learning rate scheduling. Then, restored to zero the label smoothing, the model weights were fine-tuned for 200 more epochs. The complete list of hyperparameters adopted to define and train the model for the tracking efficiency is reported in Table 4.2.

The training dataset was arranged by selecting a fraction of 50% of the $\mathcal{O}(3 \times 10^7)$ particles generated via detailed simulations. Another 10% portion of the sample was used to monitor any signs of overtraining, while the remaining 40% was preserved for validation studies. The learning curves representing the evolution of the CCE during the training procedure are quite similar to the one in Figure 4.9 and not so descriptive.

While not really motivated from a theoretical perspective, since the task is really to assign a probability of belonging to a class rather than classifying occurrences, we found it helpful to show the performance of the neural network showing the history of the F_1 *score* for the four classes under investigation is preferred (Figure 4.14). The F_1 score is the harmonic mean of the precision and recall:

$$F_1 = 2\,\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2\,\text{TP}}{2\,\text{TP} + \text{FP} + \text{FN}} \tag{4.4}$$

where TP, FP, and FN represent the number of *true positive*, *false positive*, and *false*

negative instances resulting from the trained model, respectively.

Using the F_1 score for a multi-class classification requires projecting the task into a binary classification problem following a one-vs-all philosophy: the TP, FP, and FN are computed considering true the instances belonging to the class under study and false all the others. Following this strategy, the evolution of the F_1 scores for not reconstructed, Long, Upstream, and Downstream tracks is reported in Figure 4.14. We do not observe significant evidence of overtraining, actually, the presence of an L2 weights regularizer [169] ensures that the model shows better performance in recognizing Long and Upstream tracks on the test set rather than on the training one. The different F_1 scores achieved by the four classes reflect their statistical population within the training dataset. The model clearly succeeds in identifying not reconstructed and Long tracks, since they represent 68% and 23% of the generated particles, respectively. Upstream and Downstream tracks achieve worse performance, as expected by poorly represented classes: the remaining 4% and 5% portions of the training sample.

Validation studies

The ability of the trained neural network to parameterize the tracking efficiency is assessed on a sample of about 3 million particles never seen during the training procedure. The model aims to predict the fraction of particles in acceptance that are properly reconstructed by the LHCb detector based on the generator-level kinematics and their species. The neural network was also trained to identify the set of sub-detectors involved in the reconstruction

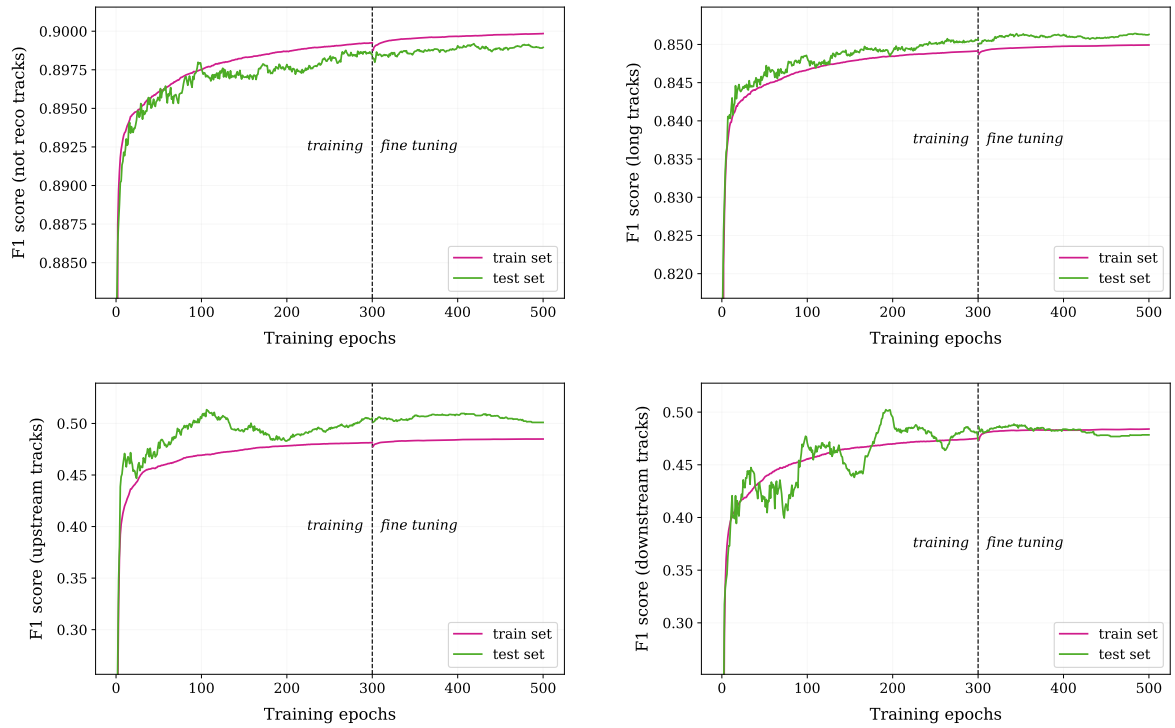


Figure 4.14: Metric curves reporting the F_1 *score* of the deep neural network trained to model the LHCb tracking efficiency for the four track classes provided by LAMARR. The training procedure consisted of 300 epochs where label smoothing and learning rate scheduling are used. A fine-tuning phase followed counting 150 epochs with label smoothing disabled.

process, to forecast the probability that a particle, once reconstructed, is associated with a *Long*, *Upstream*, or *Downstream* track. Similarly to what was done for the acceptance parameterization, we evaluate the performance of the trained model by comparing the kinematic distributions of the *reconstructed* particles with the ones obtained by weighting the kinematic distributions of the particles *in acceptance* with the probability predicted for the various track classes.

The validation plots of the tracking efficiency model for electrons as a function of the pseudorapidity η and of the z-coordinate of the origin vertex are depicted in Figure 4.15 in four bins of momentum p. The kinematic distributions of the electrons in acceptance are reported in grey, while stacked histograms (in hatched fill) are used to highlight if a particle is reconstructed as a Long (in green), Upstream (in red), or Downstream (in blue) track. The output of a neural network trained to perform this multi-class classification is superimposed with solid-line histograms. Similar comparisons are repeated for muons and hadrons (i.e., pions, kaons, and protons), as shown in Figures 4.16 and 4.17, respectively.

The agreement between the kinematic distributions resulting from detailed simulations and the ones induced by the NN-based model is pretty good, even on poorly populated tails. It is noteworthy that the neural network is able to reproduce the distribution of the pseudorapidity for high-energy hadrons that, when reconstructed as Long tracks, is characterized by a dip at $\eta \approx 4.4$. A corresponding enhancement in the probability of reconstructing these particles as VELO tracks suggests that multiple scattering phenomena affect the propagation of the tracks, deflecting them outside of the TT and preventing the reconstruction as Long tracks. Figure 4.18 shows this effect using a detailed simulated sample.

The major improvement fulfilled with respect to the tracking efficiency models reported in Ref. [3, 4, 172] is the use of a neural network sensitive to the specie of the generated particle, and able to parameterize successfully Upstream and Downstream tracks in addition to the Long ones [5]. This achievement is highlighted by the distributions of the origin vertex z-coordinate. Since Downstream tracks do not involve the vertex detector, they dominate the contribution of reconstructed tracks moving away from the VELO position, located at $z = 0$ mm, as is clearly shown in Figures 4.15 and 4.17. When we have to deal with muons, other effects that interest the z-region away from the origin position may happen. In particular, if we consider muons produced in the decay of a a quasi-stable hadron, namely $K^-(\pi^-) \to \mu^-\nu_\mu$ (and the charge conjugated process), because of the tiny mass of the neutrino, we expect that the muon trajectory is almost a perfect prosecution of the hadron one. As a result, we have a muon with an origin vertex detached to VELO, but that is still reconstructed as a Long track due to the path of the hadron misidentified as a muon. Hence, Long tracks represent the dominant contributions of the z-region away from the VELO position, as depicted in Figure 4.16.

4.2.4 Tracking resolution

Multiple scattering phenomena, imperfections in the LHCb detector alignment, and the finite granularity of the Tracking system may degrade the quality of track parameters $\vec{x}$, measured through the reconstruction algorithms and defined in (4.1). The trajectory of a charged particle traversing the spectrometer is reconstructed by using the energy deposited along the path that is described as a sequence of hits in the tracking stations. Since the sensors have a finite dimension, the hit position is known with errors, that we

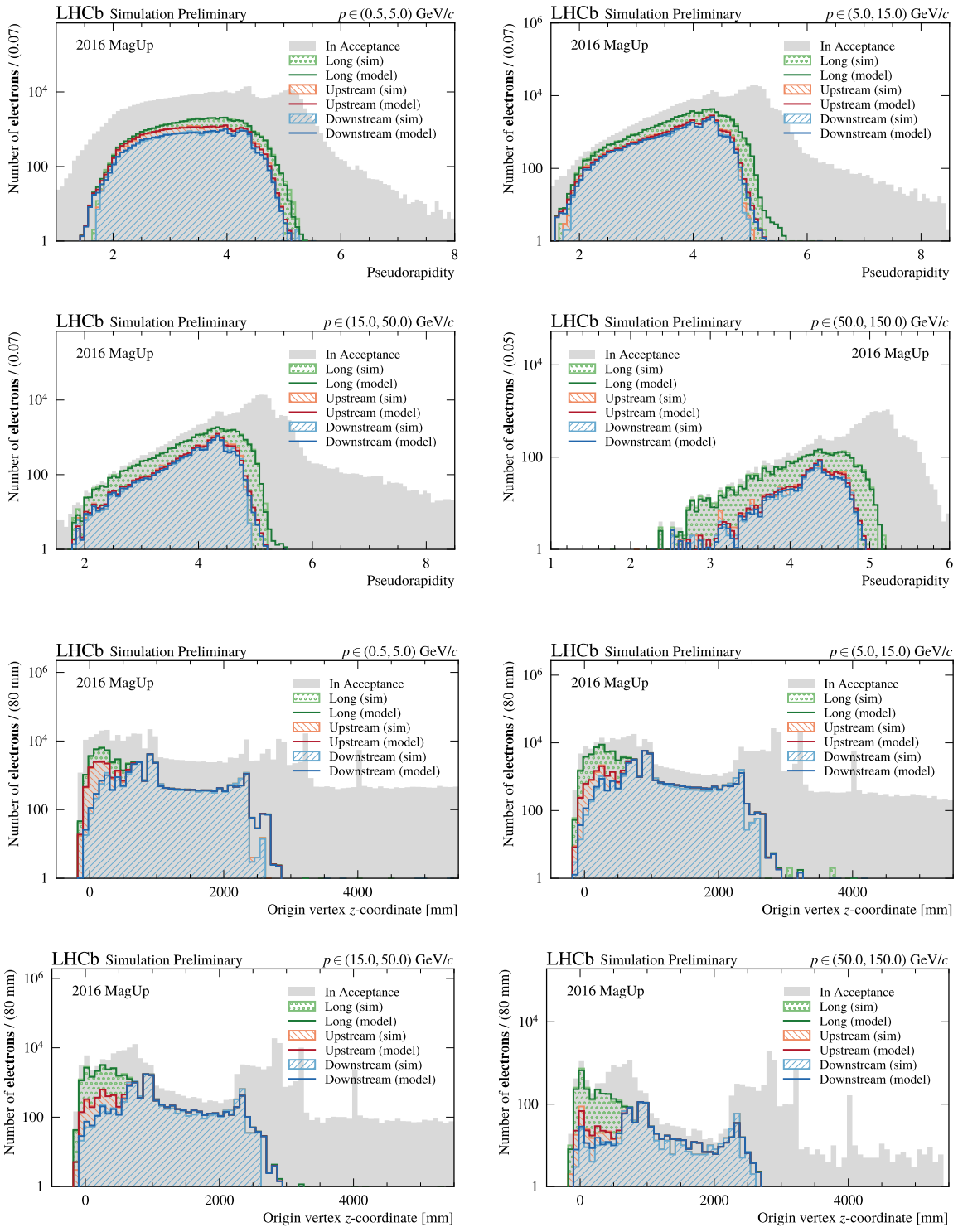


Figure 4.15: Validation plots of the tracking efficiency model for *electrons* as a function of the pseudorapidity η and of the origin vertex z-coordinate in four momentum p bins. The kinematic distributions of the electrons in acceptance are represented as light grey shaded histograms. The distributions of electrons reconstructed as Long (in green), Upstream (in red), and Downstream (in blue) tracks, are shown through stacked histograms (in hatched fill). The parameterization of the detectors involved for tracks reconstruction as modeled by a deep neural network is superimposed using solid-line stacked histograms.

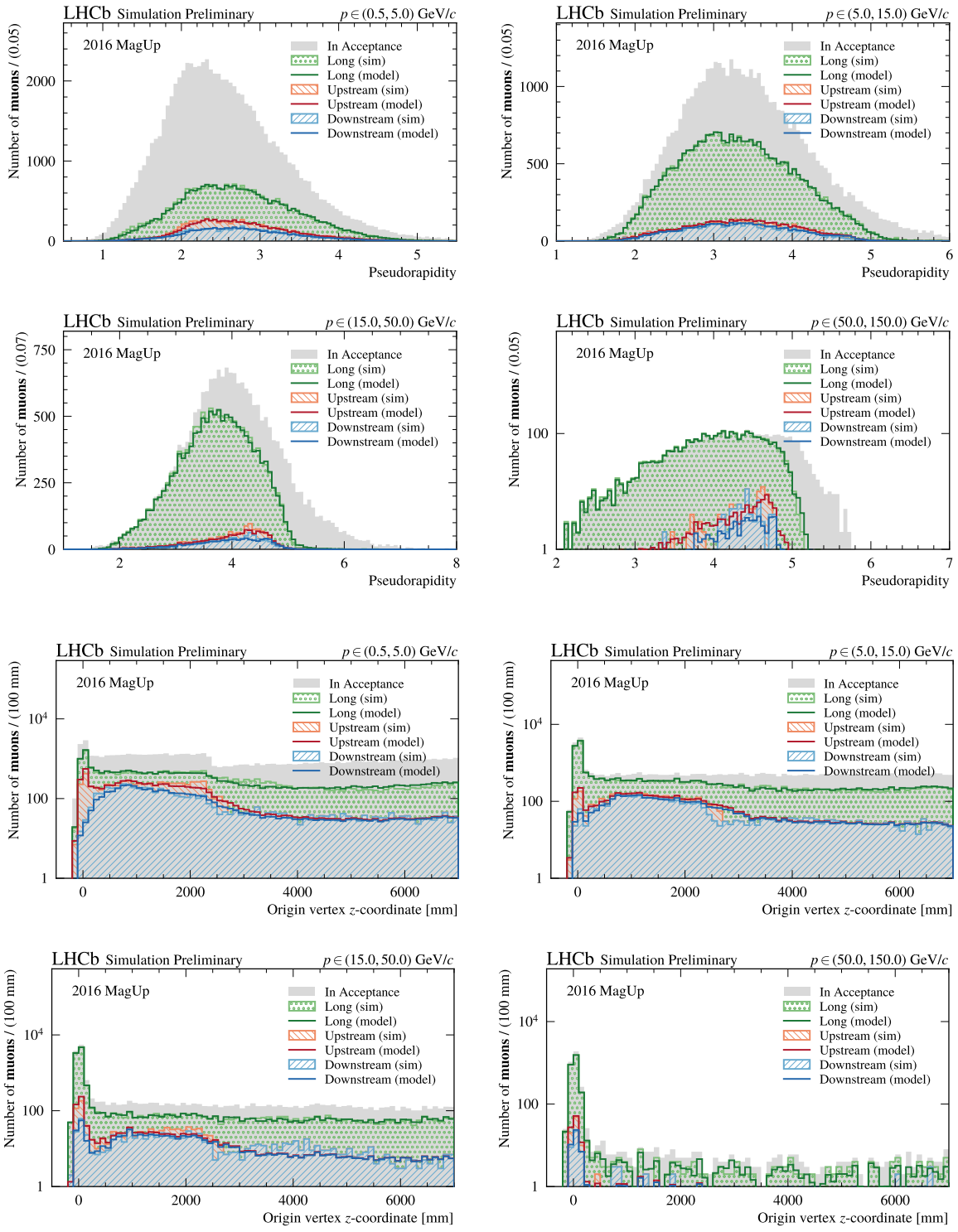


Figure 4.16: Validation plots of the tracking efficiency model for *muons* as a function of the pseudorapidity η and of the origin vertex z-coordinate in four momentum p bins. The kinematic distributions of the muons in acceptance are represented as light grey shaded histograms. The distributions of muons reconstructed as Long (in green), Upstream (in red), and Downstream (in blue) tracks, are shown through stacked histograms (in hatched fill). The parameterization of the detectors involved for tracks reconstruction as modeled by a deep neural network is superimposed using solid-line stacked histograms.

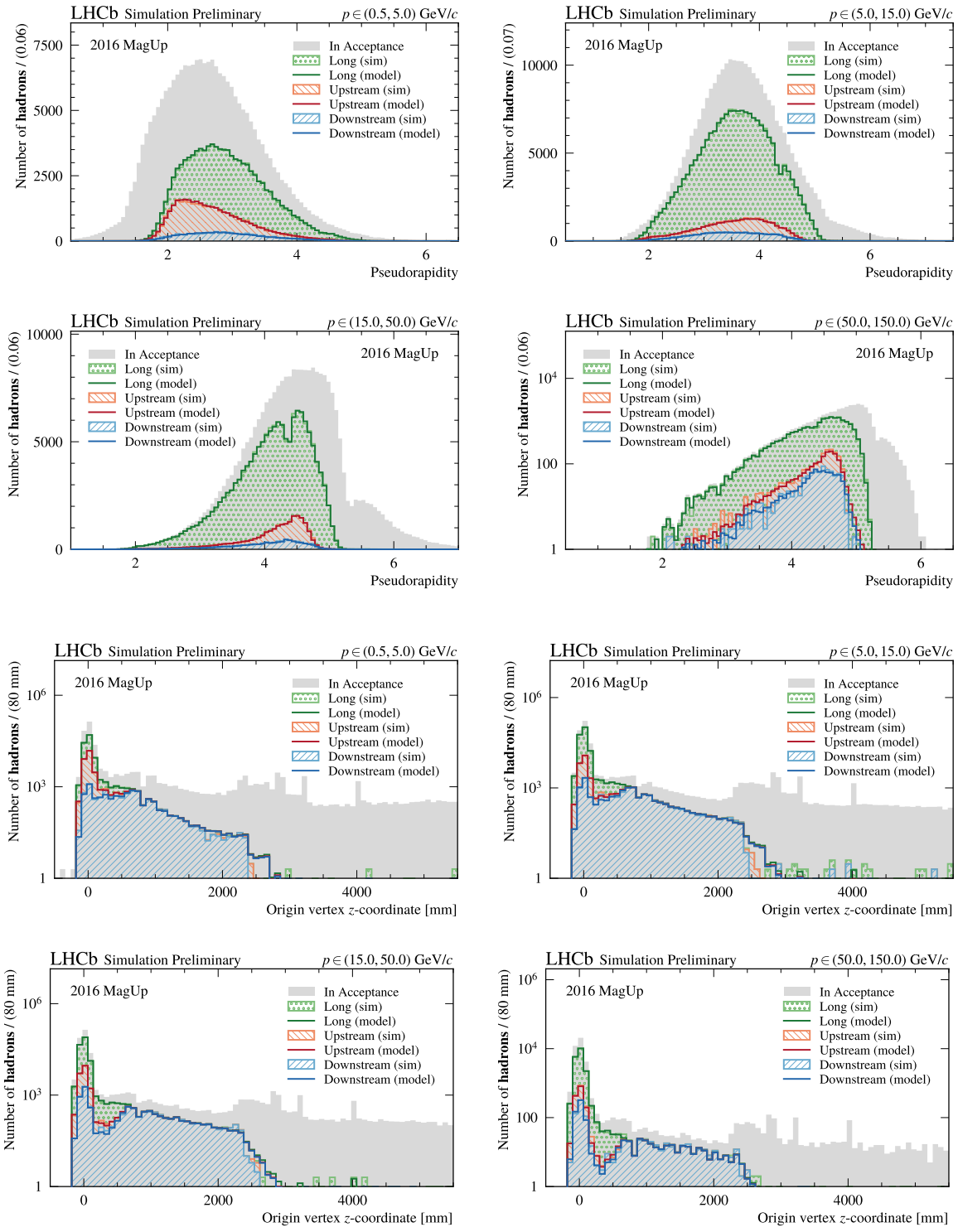


Figure 4.17: Validation plots of the tracking efficiency model for *hadrons* as a function of the pseudorapidity η and of the origin vertex z-coordinate in four momentum p bins. The kinematic distributions of the hadrons in acceptance are represented as light grey shaded histograms. The distributions of hadrons reconstructed as Long (in green), Upstream (in red), and Downstream (in blue) tracks, are shown through stacked histograms (in hatched fill). The parameterization of the detectors involved for tracks reconstruction as modeled by a deep neural network is superimposed using solid-line stacked histograms.

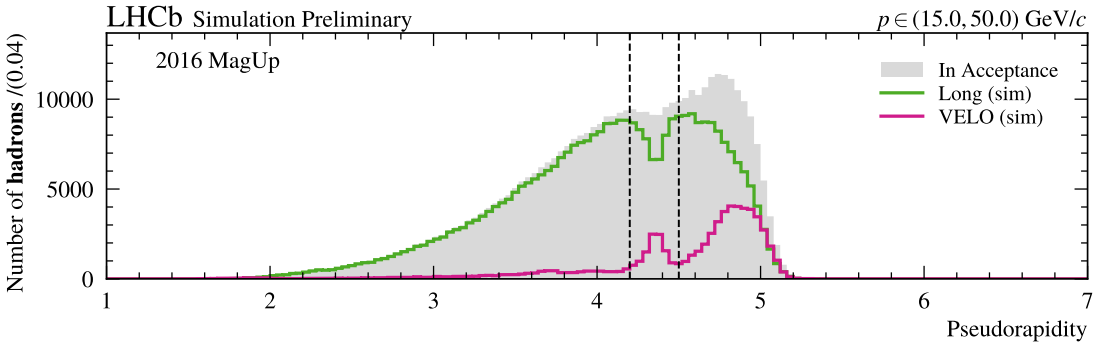


Figure 4.18: The distribution of the pseudorapidity of high-momentum *hadrons* in acceptance is represented as a light grey shaded histogram. The corresponding distribution for hadrons reconstructed as either Long or VELO tracks is shown through solid green and magenta lines, respectively. At $\eta \approx 4.4$ hadrons are more likely to be reconstructed as VELO tracks because of enhanced multiple scattering effects somewhere within the vertex detector.

expect to be of the order of the sensor dimensions. The resolution on the position of the hits combined with multiple scattering phenomena may lead to sequences of non-perfectly aligned hits, that degrade the performance of the reconstruction algorithms on measuring the kinematics of the traversing particles. Hence, the derivation of the track parameters cannot be disentangled from the resolution effects that end up affecting the position of the primary vertex (PV), the measurement of the impact parameter (IP), and the charged particle momentum p.

In LAMARR, we do not have any notion of hits or multiple scattering, then, to reproduce the tracking resolution, we need a model able to parameterize the errors introduced during the detection and reconstruction steps based on some set of generator-level information. The statistical nature of the underlying phenomena makes this *regression task* non-trivial since two charged particles generated with the same kinematics can be reconstructed as tracks with different kinematic properties. Hence, we are not simply interested in modeling the errors associated with a reconstructed particle, but rather in parameterizing the error distributions to properly account for the resolution effects. Deep generative models, like *Generative Adversarial Networks* (GAN), can be successfully used to extract probability distributions from the training dataset and produce new synthetic data according to the learned distributions [134].

LAMARR relies on a GAN-based model to parameterize the LHCb tracking resolution. As for the case of the acceptance and efficiency models, we expect that also for parameterizing the resolution the kinematics and the specie of the particles play a key role. In addition, since the combination of the tracking detectors involved in the reconstruction procedure strongly affects the ability to derive the track parameters, we cannot properly parameterize the resolution without knowing if particles are reconstructed as Long, Upstream, or Downstream tracks. Hence, to build a model for the resolution able to take into account all the aforementioned information learning *conditioned probability distributions*, such GAN system is implemented by following the design described in Ref. [141]. The complete list of input conditions follows:

- the true position (x, y, z) of the `ClosestToBeam` reconstructed track state;

- the logarithm of true momentum $\log_{10}(p)$ of the reconstructed particle;
- the true slopes t_x and t_y of the `ClosestToBeam` reconstructed track state;
- the specie (electron, muon, or hadron) of the reconstructed particle;
- the track class (Long, Upstream, or Downstream) of the reconstructed particle;
- the charge q of the reconstructed particle.

It is worth noticing that the kinematic information and the particle specie (and charge) are available from the MC physics generators, while the track class can be inferred for reconstructed particles by the NN-based efficiency model. The pipeline philosophy of LAMARR starts to take shape: we aim to provide analysis-level tracking variables, like the reconstructed momentum p_{reco}, chaining together the parameterizations for acceptance, efficiency, and resolution. The first two models allow the selection of the subset of generated particles reconstructed by the LHCb detector, while the ultimate goal of the resolution parameterization is to degrade the generator-level kinematic information according to what is expected from the Tracking system. In addition, since the reconstruction algorithms also provides a set of track quality variables resulting from the iterative Kalman procedure adopted to fit combination of hits into tracks, the GAN-based model was also designed to reproduce such analysis-level quantities, like the χ^2, the number of degrees of freedom of the track fit, and the `ghostProb` score [39]. The latter is a high-level classifier introduced in the LHCb reconstruction software to discriminate, a posteriori, between genuine tracks and random associations of aligned hits erroneously promoted to track by the reconstruction algorithm. These spurious tracks, not associated to a charged particle, are known as *ghost tracks* and this high-level classifier is known as *ghost probability*, or `ghostProb` in short. Technically, the `ghostProb` classifier is implemented as a FNN taking as input several low-level quantities produced as secondary results by the hit-clusterization and track-fitting algorithms, not available in a flash-simulation. Hence, the `ghostProb` is parameterized directly using a generative model. The correlations with the other quantities related to the track reconstruction is ensured by including the `ghostProb` model in the same generator parameterizing the track χ^2 and resolution effects. The complete list of features parameterized by the resolution model is reported in the following:

- the reconstruction errors on the position $(\delta x, \delta y, \delta z)$ of the `ClosestToBeam` state;
- the relative reconstruction error on the momentum $\delta p/p$;
- the reconstruction errors on the slopes δt_x and δt_y of the `ClosestToBeam` state;
- the track fit χ^2 per degree of freedom;
- the number of degrees of freedom of the track fit;
- the probability of reconstructing as a track a random combination of hits not associated with any charged particle (`ghostProb`) [39].

With the reconstruction error of a variable v, we refer to the difference between its reconstructed value v_{reco} and the one resulting from physics generators v_{true}, namely

$$\delta v \equiv v_{\text{reco}} - v_{\text{true}} \tag{4.5}$$

Model design and training

A GAN model, as deeply described in Section 3.2, consists of two neural networks, called *generator* G and *discriminator* D, which are trained simultaneously through a competition game. The discriminator is trained to perform a classification task, distinguishing elements of the reference dataset from synthetic data produced by the generator. At the same time, the generator training is driven by a simulation task, whose aim is to reproduce as well as possible the reference dataset, trying to fake the discriminator. The result is what the authors of Ref. [134] call a *minimax two-player game*.

For parameterizing the tracking resolution, the two players are implemented via 10-layer neural networks with 128 neurons in each hidden layer and Leaky ReLU activation functions. Both the neural networks are equipped with skip connections to limit the vanishing gradient problem [111]. To allow the generator to learn probability distributions conditioned by the above-listed features, such conditions x are used as input features together with the elements z of the *latent space*, represented as a 128-dimensional normal distribution $\mathcal{N}_{128}(\mathbf{0}, \mathbf{1})$. The last layer of the generator counts as many neurons as it is the number of the output features y, and it has a simple linear activation function. To accomplish the classification task, also the discriminator takes the conditions x as input, coupled with the corresponding resolution parameters, whether they are part of the reference dataset y or reproduced by the generator $G(z|x)$. The last layer of the discriminator consists of a single neuron with a sigmoid activation function, describing the probability that the input belongs to the reference sample.

The minimax game was implemented using BCE-based loss functions. In particular, the discriminator was trained to correctly classify stacks of resolution parameters and conditions in input, minimizing the following loss function:

$$\mathcal{L}_D = \frac{1}{2}\left[\mathrm{BCE}\left(\mathbf{1}, D(y|x)\right) + \mathrm{BCE}\left(\mathbf{0}, D(G(z|x))|x\right)\right] \tag{4.6}$$

On the other hand, the generator training was driven by faking the discriminator outcome. During the training, the generator improved its ability to mimic the reference dataset, minimizing the loss that follows:

$$\mathcal{L}_G = \mathrm{BCE}\left(\mathbf{1}, D(G(z|x))|x\right) \tag{4.7}$$

The minimization of the discriminator and generator losses was performed using two RMSprop optimizers [120] initialized with learning rates of 2×10^{-4} and 10^{-4}, respectively. During a training procedure of 500 epochs, both the learning rates were decreased by about two orders of magnitude via an exponential decay scheduling.

Despite Ref. [134] demonstrates that the solution of the minimax two-player game corresponds to minimizing the *Jensen-Shannon divergence*[6] (JSD) between the reference probability distributions and the ones induced by the generator, let a GAN systems converge is a non-trivial task. As discussed in Section 3.2.1, the instability of GAN training results from the ability of the discriminator to perfectly separate true instances from generated data, so that no feedback remains to the generator for improvement. To mitigate this vanishing gradient problem, the training of the discriminator was hindered by setting a label smoothing of 0.05 to the BCE functions and by adding Gaussian noise

[6]In statistics, the *Jensen-Shannon divergence* is a method of measuring the similarity between two probability distributions. Read more on `https://en.wikipedia.org/wiki/Jensen-Shannon_divergence`.

	Generator	*Discriminator*
skip connections [111]	✔	✔
latent space dim [134]	128	-
input shape	(None, 141)	(None, 22)
input preprocessing	✔	✔
n hidden layers	10	10
n hidden neurons	128	128
hidden activation functions	Leaky ReLU	Leaky ReLU
output activation function	linear	sigmoid
output shape	(None, 9)	(None, 1)
output preprocessing	✔	✖
optimizer	RMSprop	RMSprop
learning rate	1×10^{-4}	2×10^{-4}
loss function	BCE-based loss (4.7)	BCE-based loss (4.6)
BCE label smoothing	0.05	0.05
Gaussian noise stddev [144]	0.02	0.02
learning rate scheduling	ExpDecay	ExpDecay
scheduling decay rate	0.1	0.1
scheduling decay steps	290000	190000
batch-size	3000	3000
batches per epoch	~ 1000	~ 1000
n epochs	500	500

Table 4.3: Hyperparameters of the GAN-based model for the resolution.

with zero mean and a standard deviation of 0.02 to the discriminator input. This strategy prevents the discriminator from being a perfect classifier, ensuring the generator learns continuously during the training [144]. The complete list of hyperparameters used to build and train the model for the tracking resolution is reported in Table 4.3.

For training and validating the GAN-based resolution model, a dataset of $\mathcal{O}(2 \times 10^7)$ reconstructed particles was produced through Detailed Simulation. Similarly to what was done for the acceptance and efficiency parameterizations, a fraction of 50% of the dataset was used for training, while another 10% was retained to monitor any signs of overtraining. Finally, the remaining 40% of the dataset was preserved to assess the quality of the trained model on an independent never-seen data sample. The left plot of Figure 4.19 shows the traditional learning curves expected from GAN training. The aforementioned minimax game minimizes the generator loss, while at the same time, the discriminator loss is maximized. This competition continues up to reach the *Nash equilibrium*, which corresponds to the incapability of the discriminator to distinguish between reference and generated data, namely $\mathcal{L}_{D/G} = -\ln(0.5) \simeq 0.69$. The right plot of Figure 4.19 reports the evolution during the training of the JSD score computed between the univariate probability distribution induced by the discriminator taking as input true instances and synthetic data. The proof of the generator improvement during the training is demonstrated by the reported decrease in the JSD score. Lastly, from both the plots we can conclude that there are not any signs of overtraining.

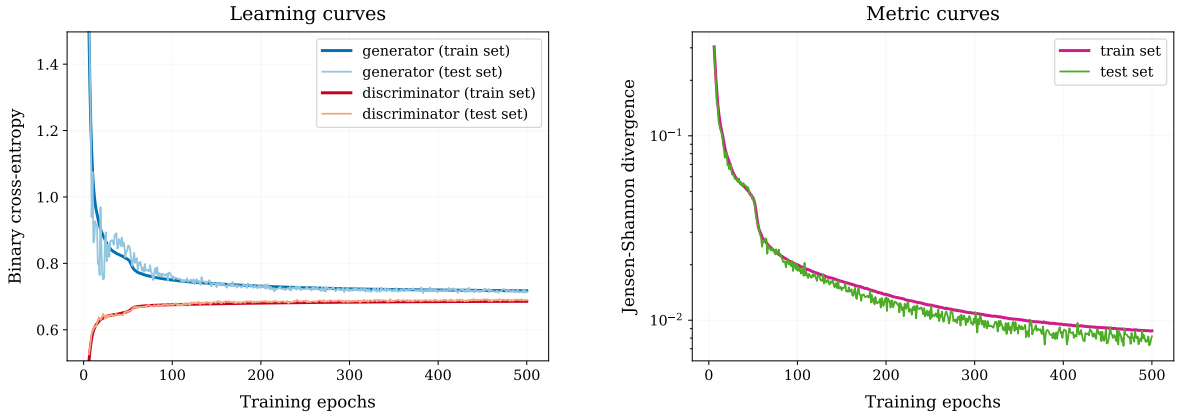


Figure 4.19: Learning and metric curves of a GAN model trained to parameterize the resolution effects of the LHCb Tracking system. The left plot reports the competition of the *generator* and *discriminator* networks, whose training is driven by the minimax two-player game. The evolution of the JSD score is depicted in the right plot to show the improvement of the similarity between the reference and generated samples during the training.

Validation studies

A generator properly trained via a GAN system can be used to reproduce the probability distributions underlying the reference dataset explored during the training procedure. To assess the performance of such a generator in parameterizing the tracking resolution, an independent sample of more than 2 million *reconstructed* particles was retained. The output of the trained generator can be split into two main categories: from one side, we have the errors on the track position and momentum introduced by the detection and reconstruction steps, while, on the other hand, we have the set of track quality variables resulting from the reconstruction algorithms. For the tracking errors, we are mostly interested in modeling the width of the distributions, namely the resolution, and correctly parameterizing how it changes as a function of the particle kinematics and the track class. Figures 4.20 and 4.21 investigate these dependencies in kinematic bins for the spatial and momentum resolutions, respectively. Assessing the performance of the trained generator also requires validating its ability to reproduce the distributions of the track quality variables, whose 1-D histograms are reported in Figures 4.22, 4.23, and 4.24.

The study of CP violation and rare decays in the heavy flavour sector, the core business of the LHCb physics program, requires accurately measuring decay vertices, impact parameters, and particle momenta, to highlight signal contributions, rejecting any background sources. It is therefore not surprising that the performance of the LHCb tracking detectors is often described in terms of PV, IP, and momentum resolution [29, 34, 173, 174]. Hence, for validating the GAN-based resolution model, we need the generator to succeed in reproducing the same metric curves typically used to describe the tracking performance.

Since the major contribution to the PV is the track covariance matrix (discussed in Section 4.2.5), the corresponding metric curves are not taken into account in these validation studies. On the contrary, the IP resolution is strongly related to the resolution of the `ClosestToBeam` track state. The upper triplet of plots in Figure 4.20 shows the resolution of the `ClosestToBeam` x-coordinate as a function of $1/p_T$ for tracks reconstructed

as Long, Upstream, and Downstream, reproducing what was reported in the LHCb performance paper of Ref. [34] for the IP_x resolution. The linear dependence on $1/p_T$ is a consequence of multiple scattering and the geometry of the vertex detector [29], and indeed, it is not present for Downstream tracks, whose reconstruction does not involve the VELO. The agreement between the resolution exhibited by Detailed Simulation and the one resulting from the GAN training is excellent, even if cases of slightly under-estimated resolution occur for Downstream tracks. Similar performances are achieved for the y- and z-coordinates of the `ClosestToBeam` track state as a function of $1/p_T$. Lastly, it should be pointed out that neither p_T or $1/p_T$ are part of the input conditions: the generator is then able to derive these dependencies from the true momentum p and slopes (t_x, t_y).

The lower triplet of plots in Figure 4.20 examines the resolution of the `ClosestToBeam` x-coordinate as a function of the azimuthal angle φ, reproducing what was also discussed in Ref. [29] for the IP_x resolution. The two peaks exhibited by Long and Upstream tracks in correspondence of $\varphi \approx \pm\pi/2$ result from regions of the VELO detector where the material density greatly increases due to the presence of multiple sensors overlapping. The GAN-based model struggles to properly reproduce this effect, aided by the fact that it corresponds to a poorly represented training sample. On the contrary, the generator succeeds in parameterizing the resolution of the x-coordinate as a function of φ for Downstream tracks. Similar performances are achieved also for the resolution of y- and z-coordinates. As for the case of p_T studies, neither the azimuthal angle φ is part of the input conditions, demonstrating the ability of GANs to learn non-trivial dependencies.

The Long tracks, involving the whole Tracking system in the reconstruction process, offer the highest-precision measurement of the momentum p of charged particles. The momentum resolution lays between 0.5% and 1.5% for particles below 20 GeV/c, stabilizing at about 0.5% for particles up to 80 GeV/c. In the same kinematic range, the precision offered by Downstream tracks is worse even if still decent. Low energy particles are reconstructed with a momentum resolution ranging from 0.5% to 10%, while increasing the energy the resolution improves oscillating around 5%. Upstream tracks have the worst performance, exhibiting momentum resolution from 10% to 100%. The upper triplet of plots in Figure 4.21 reports the resolution of the relative momentum $\delta p/p$ as a function of p for tracks reconstructed as Long, Upstream, and Downstream, reproducing the same metric curve provided in the LHCb performance paper of Ref. [34]. The correct parameterization of the momentum resolution is a non-trivial task since it is strongly correlated to the input kinematic conditions, besides exhibiting behaviors different based on the reconstructed track class. The GAN-based model succeeds in associating the correct momentum resolution ranges to the corresponding track class, while the presence of some mismatches with what is expected from Detailed Simulation suggests that there is still room for improvement.

The resolution of the relative momentum $\delta p/p$ is also investigated as a function of the track-steepness, namely $\sqrt{t_x^2 + t_y^2}$, as depicted in the lower triplet of plots in Figure 4.21. In this case, the generator exhibits excellent performance in reproducing the expected behavior either for Long or Upstream tracks. To fix the mismodeling on the Downstream tracks instead, some further hyperparameter optimization should be necessary.

The track quality variables provided by the trained generator are crucial to building a parameterization for the track covariance matrix that properly takes into account the correlations between how sequences of hits are combined into tracks and the corresponding uncertainties. The capability of the GAN-based model to reproduce the distribution of

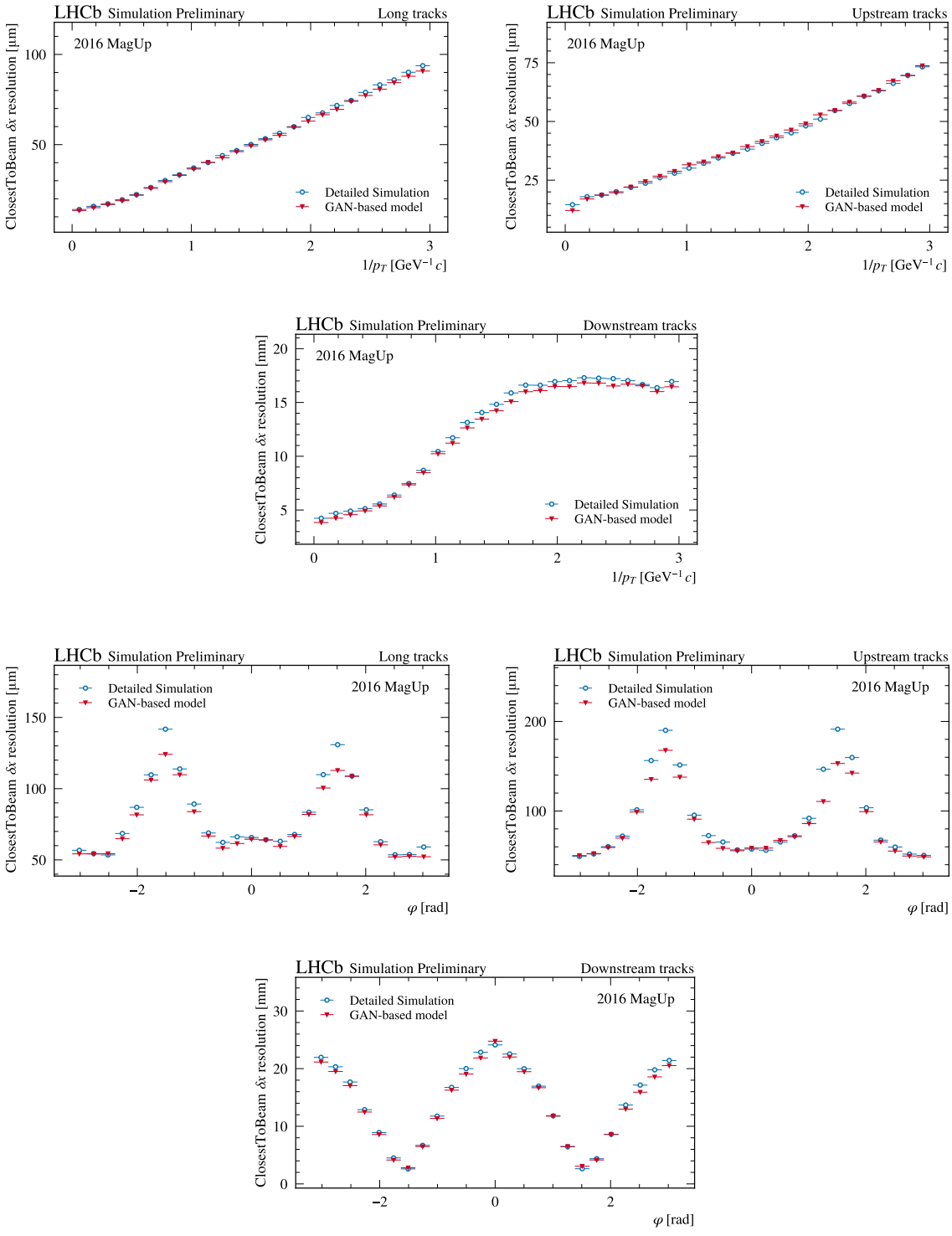


Figure 4.20: Resolution of the `ClosestToBeam` x-coordinate as a function of the reciprocal of the transverse momentum p_T (upper triplet of plots) and of the azimuthal angle φ (lower triplet of plots). The resolution resulting from detailed simulated *Long*, *Upstream*, and *Downstream* tracks is reported using blue circle-markers. The output of a GAN-based model trained to parameterize the tracking resolution is depicted through red triangular-markers. It is worth noticing that both p_T and φ are not part of the input variables to the GAN system, which succeeds in inferring them from MC momentum and slopes.

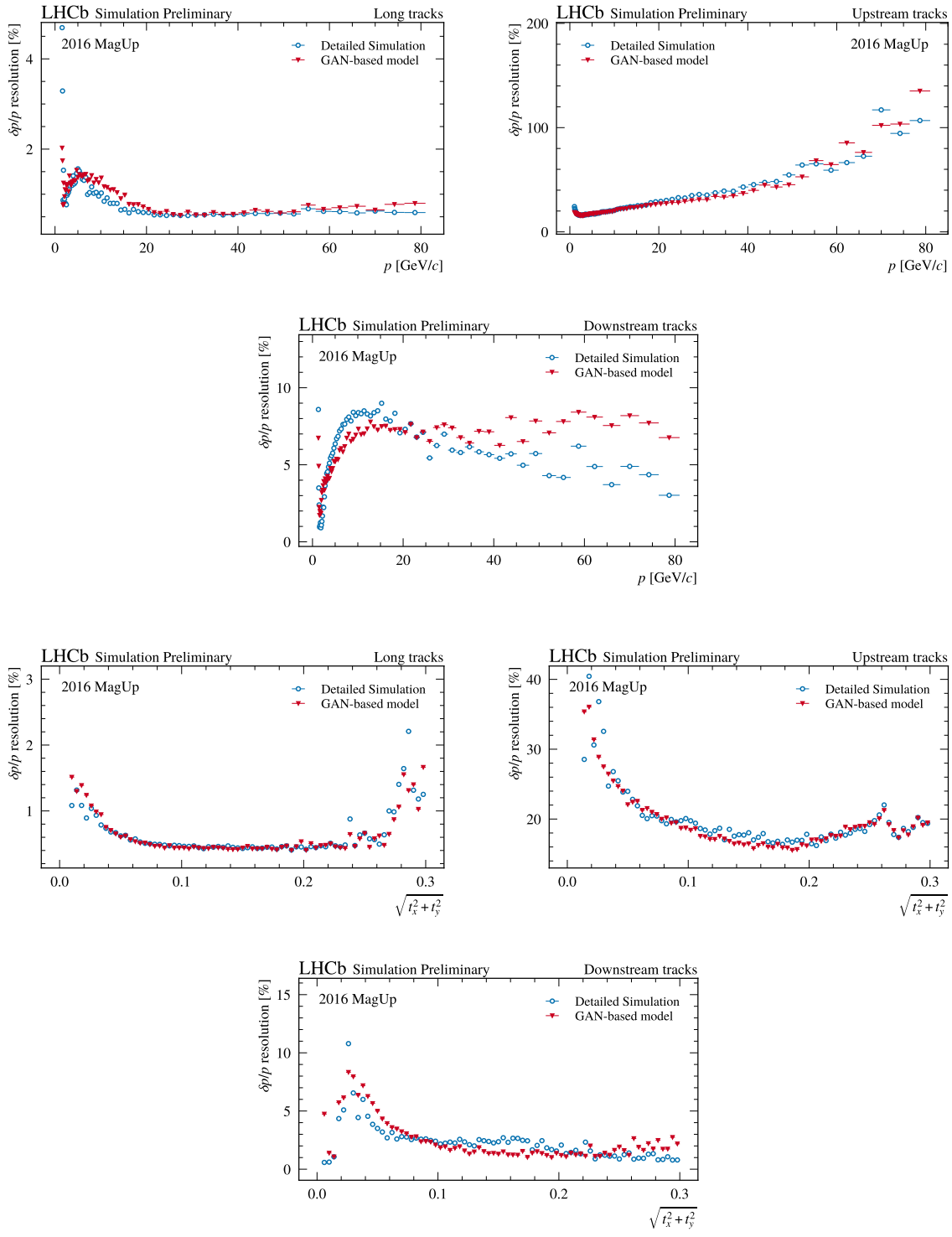


Figure 4.21: Relative momentum resolution as a function of the momentum p (upper triplet of plots) and of the track-steepness $\sqrt{t_x^2 + t_y^2}$ (lower triplet of plots). The relative resolution resulting from detailed simulated *Long*, *Upstream*, and *Downstream* tracks is reported using blue circle-markers. The output of a GAN-based model trained to parameterize the errors introduced in the LHCb detection and reconstruction phases are depicted through red triangular-markers.

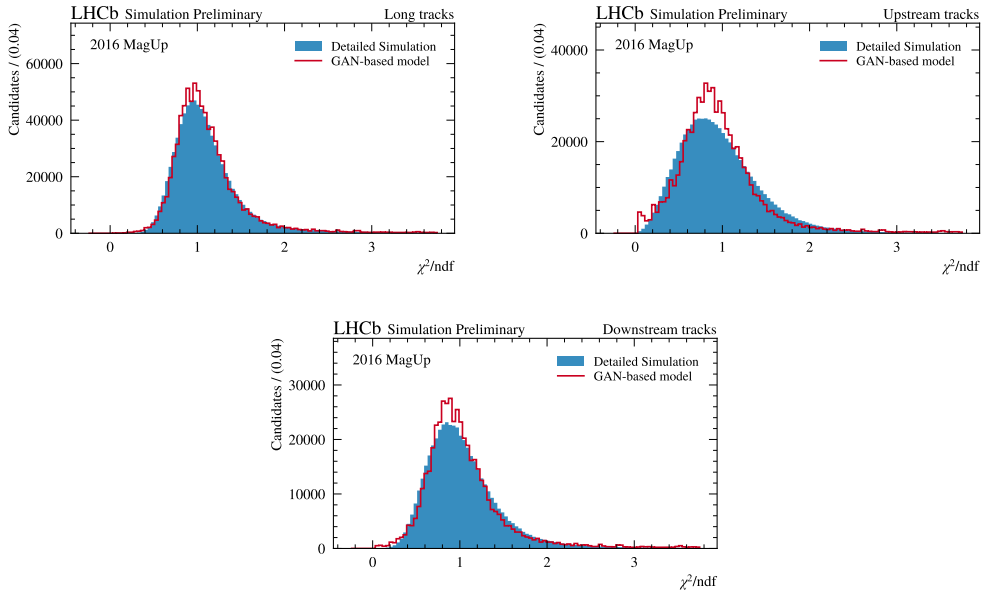


Figure 4.22: Validation plots of the track fit χ^2 per degrees of freedom (ndf) for tracks reconstructed as *Long* (upper left), *Upstream* (upper right), or *Downstream* (lower center). The distributions obtained from Detailed Simulation are represented as blue shaded histograms. The results of a GAN model trained to parameterize the high-level response of the LHCb Tracking system, including the track fit output, are shown using red solid-line histograms.

the track fit χ^2 per degree of freedom for Long, Upstream, and Downstream tracks is depicted in Figure 4.22. Its performance is decent even if the tendency to overestimate the population at the peak should be further investigated. Figure 4.23 reports the distribution of the number of degrees of freedom. As expected Long tracks are characterized, on average, by a greater number of degrees of freedom than the other track classes since typically involve larger sequences of hits. As highlighted in Figure 4.23, the generator takes correctly into account the condition of the track class, even if its performances are severely challenged by the discrete nature of the parameterized variable. Finally, Figure 4.24 shows the distribution of the `ghostProb` score as obtained from Detailed Simulation and as parameterized by the trained GAN-based model. Looking at the histograms in Figure 4.24 we can conclude that the generator succeeds in reproducing the distribution of the `ghostProb`. Since the latter is also obtained through a neural network in the official LHCb reconstruction, this task has similarities with a *knowledged distillation problem*[7], though the dependencies on most of the inputs is, in this case, replaced with random noise shaped by the generator logic, and reproducing accurately also the tails of the response of this multivariate classifier.

4.2.5 Track covariance matrix

The purpose of the tracking reconstruction algorithms applied to either collected or detailed simulated data is to determine the most accurate estimates of the track parameters $\vec{x}$ and

[7]With *knowledge distillation* we refer to the problem of transferring the "knowledge" acquired by a neural network (called *teacher*) by training a second neural network (called *student*) based on the response of the first one.

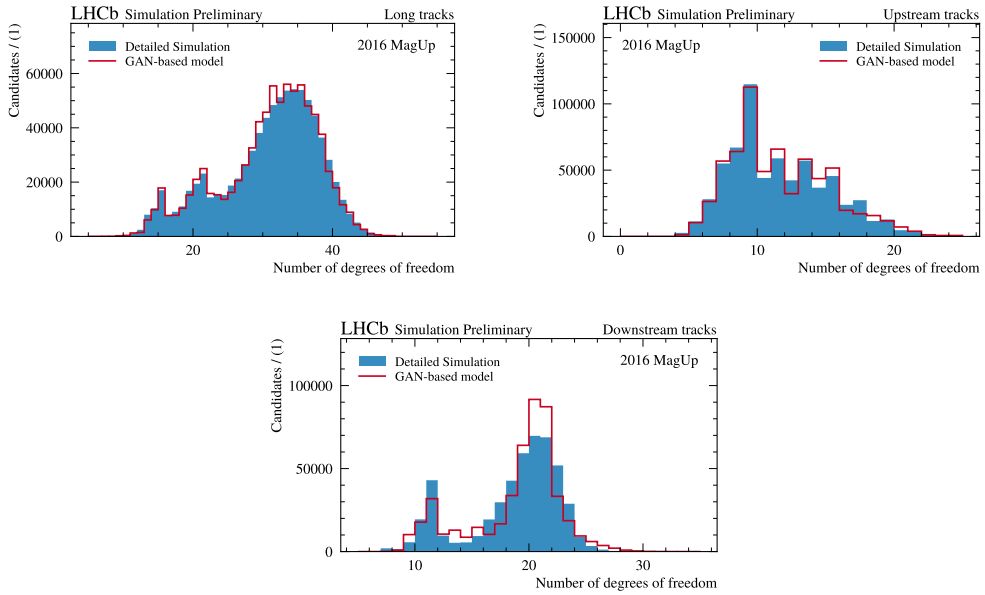


Figure 4.23: Validation plots of the number of track fit degrees of freedom (ndf) for tracks reconstructed as *Long* (upper left), *Upstream* (upper right), or *Downstream* (lower center). The distributions obtained from Detailed Simulation are represented as blue shaded histograms. The results of a GAN model trained to parameterize the high-level response of the LHCb Tracking system, including the track fit output, are shown using red solid-line histograms.

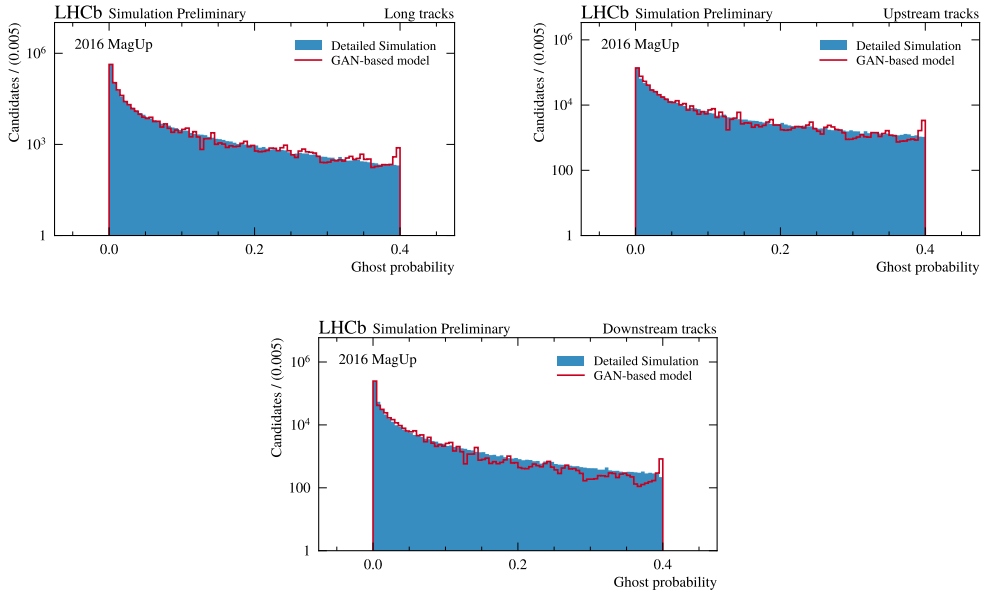


Figure 4.24: Validation plots of the probability that a random combinations of hits not associated with any charged particle (ghost) are reconstructed as *Long* (upper left), *Upstream* (upper right), or *Downstream* tracks. The distributions obtained from Detailed Simulation are represented as blue shaded histograms. The results of a GAN model trained to reproduce the response of the neural network used by LHCb to identify ghost tracks [39] are shown using red solid-line histograms.

the corresponding 5×5 covariances C. Once the event has been completely reconstructed, these estimates can be used to match the tracks with particle identification objects, i.e., RICH rings, calorimeter clusters, and muon candidates. Finally, these high-level quantities can be used in physics analyses to locate the primary and secondary vertices and calculate the invariant mass of particle combinations.

The estimation of the track parameters and the covariance matrix results from an iterative track fit based on a Kalman filter algorithm [38, 163]. The principle at the basis of the Kalman procedure is to add the tracking hits one-by-one to the fit, each time updating the local track state considering the whole collection of hits. The method is driven by the minimization of the χ^2 of the measurements on the track, and therefore it is mathematically equivalent to a least-squares fit. This fit procedure is defined so that accounts for the uncertainties on the track parameters considering, for instance, multiple scattering phenomena, or energy loss due to ionization. The Kalman procedure incorporates all these effects during the update of the covariance matrix.

In LAMARR, the track parameters $\vec{x}$ are provided relying on the GAN-based resolution model discussed in Section (4.2.4). Designed to infer the errors introduced in the detection and reconstruction steps, the latter allows to parameterize the track kinematic properties applying such errors to the MC truth information provided by the physics generators. To finalize the parameterization of the track states, and in particular of the `ClosestToBeam` track state, LAMARR is also equipped with a model for the covariance matrix. Just like for the resolution model, also in this case the underlying phenomena are characterized by a *statistical nature*, which makes two identical tracks potentially associated with two different covariances. Again, to face this non-trivial *regression problem*, LAMARR relies on a GAN-based model to learn, from detailed simulated samples, the *conditioned* probability distributions [141] associated with each element of the covariance matrix.

The GAN model for covariance takes as input the same conditions used for the resolution model. In addition, to build a parameterization for the covariance that properly accounts for the quality of the track fit, such GAN is designed to take as input also the set of track quality variables provided by the resolution model, namely

- the track fit χ^2 per degree of freedom;
- the number of degrees of freedom used in the track fit;
- the `ghostProb` score.

The combination of generator-level kinematic properties, the particle specie and charge, the set of detectors involved in reconstructing tracks, together with a global assessment of the fit procedure are used to parameterize the 5×5 covariance C of the `ClosestToBeam` state. Solving the minimax game at the basis of the target model construction is made difficult by the high correlation between the matrix elements, a factor which requires the introduction of some (reversible) simplifications:

1. as the covariance is symmetric, the GAN was trained to model only the diagonal and lower-triangular elements;
2. since the elements of the diagonal are positive, we can simplify the GAN learning process by taking the logarithm of the diagonal elements;

3. to reduce the correlation between diagonal and off-diagonal elements, the GAN was trained to model correlations instead of the covariance for off-diagonal elements.

Hence, recalling that the correlation matrix ρ derives from the covariance C as follows

$$\rho_{ij} \equiv \text{corr}(x_i, x_j) = \frac{C_{ij}}{\sqrt{C_{ii}C_{jj}}} \quad \text{where} \quad C_{ij} \equiv \text{cov}(x_i, x_j) \tag{4.8}$$

the complete list of matrix elements parameterized by the GAN is reported in the following:

- the logarithm of the covariance diagonal elements, i.e., $\log(C_{ii})$ with $i \in \{1 \dots 5\}$;
- the lower-triangular elements of the correlation matrix, i.e., ρ_{ij} with $i > j$.

A further decorrelation step is achieved by applying different preprocessing strategies to the two classes of matrix elements. In particular, the elements of the covariance diagonal are standardized by removing the mean and scaling to unit the variance, a transformation that leaves unchanged the shape of the probability distributions. On the other hand, the distributions of the off-diagonal elements of the correlation matrix are mapped to a Gaussian by applying a quantile normalization. The preprocessing strategies were implemented using respectively the `StandardScaler` and `QuantileTransformer` classes provided by the scikit-learn Python package [175]

Model design and training

The covariance parameterization relies on a GAN model powered by an 8-layer generator and a 6-layer discriminator. Both neural networks have 128 neurons and Leaky ReLU activation functions in each hidden layer and are equipped with skip connections [111]. To train the generator to learn conditioned probability distributions, the latter takes as input the aforementioned conditions x chained with latent vectors z, sampled from a 128-dimensional normal distribution. The output layer of the generator has 15 neurons, namely the total number of matrix elements to parameterize, and no activation function. As the generator, also the discriminator takes the conditions x as inputs, together with either instances from the training samples y or resulting from the generator $G(z|x)$. The discriminator outputs a single value, passing through a sigmoid function to represent the probability that the input comes from the reference dataset.

The discriminator was trained to correctly classify the input instances according to their origin minimizing the loss function $\mathcal{L}_D$ defined in (4.6). At the same time, the generator was trained to reproduce as accurately as possible the reference sample, trying to fake the discriminator by minimizing the loss function $\mathcal{L}_G$ defined in (4.7). The training of both the discriminator and generator networks was driven by two RMSprop optimizers initialized with learning rates of 2×10^{-4} and 10^{-4}, respectively. The training procedure consisted of 350 epochs during which both the learning rates were exponentially decreased by about two orders of magnitude. Finally, to ensure stable GAN training, the following precautions were adopted [144]:

- setting a label smoothing of 0.1 to both the BCE-based loss functions $\mathcal{L}_D$ and $\mathcal{L}_G$;
- adding to the discriminator input a Gaussian noise with zero mean and a standard deviation of 0.05.

	Generator	*Discriminator*
skip connections [111]	✔	✔
latent space dim [134]	128	-
input shape	(None, 144)	(None, 31)
input preprocessing	✔	✔
n hidden layers	8	6
n hidden neurons	128	128
hidden activation functions	Leaky ReLU	Leaky ReLU
output activation function	linear	sigmoid
output shape	(None, 15)	(None, 1)
output preprocessing	✔	✖
optimizer	RMSprop	RMSprop
learning rate	1×10^{-4}	2×10^{-4}
loss function	BCE-based loss (4.7)	BCE-based loss (4.6)
BCE label smoothing	0.1	0.1
Gaussian noise stddev [144]	0.05	0.05
learning rate scheduling	ExpDecay	ExpDecay
scheduling decay rate	0.1	0.1
scheduling decay steps	200000	150000
batch-size	7500	7500
batches per epoch	~ 900	~ 900
n epochs	350	350

Table 4.4: Hyperparameters list of the GAN-based model for the covariance.

The complete list of hyperparameters adopted to build and train the model for the covariance matrix is reported in Table 4.4.

The GAN-based model for the covariance shares with the resolution parameterization the dataset of $\mathcal{O}(2 \times 10^7)$ reconstructed particles produced via Detailed Simulation. As usual, a fraction of 50% of the dataset was used for the training procedure, 10% for monitoring any evidence of overtraining, and the remaining 40% to perform validation studies. Figure 4.25 reports the learning and metric curves resulting from the covariance GAN training. The evolution of the loss functions $\mathcal{L}_D$ and $\mathcal{L}_G$ is depicted on the left plot. At the beginning of the training, the discriminator had some difficulties learning how to generalize the separation problem: the initial indecision in distinguishing the reference instances from the generated ones is visible from the high variability of the generator loss (in blue) on the test set. After this stalemate, the discriminator starts to accomplish the classification task, giving back fruitful feedback to the generator. The result is a canonical GAN learning process where both the loss functions point asymptotically to the Nash equilibrium, namely $\mathcal{L}_{D/G} = -\ln(0.5) \simeq 0.69$. The right plot highlights the same behavior through the accuracy metric score. Such accuracy measures the performance of the generator in producing data enough faithful to the reference sample so that the discriminator wrongly classifies them as true instances. In this case, the equilibrium point corresponds to the discriminator making a mistake on half of the investigated sample, on average, resulting in an accuracy equal to 50%.

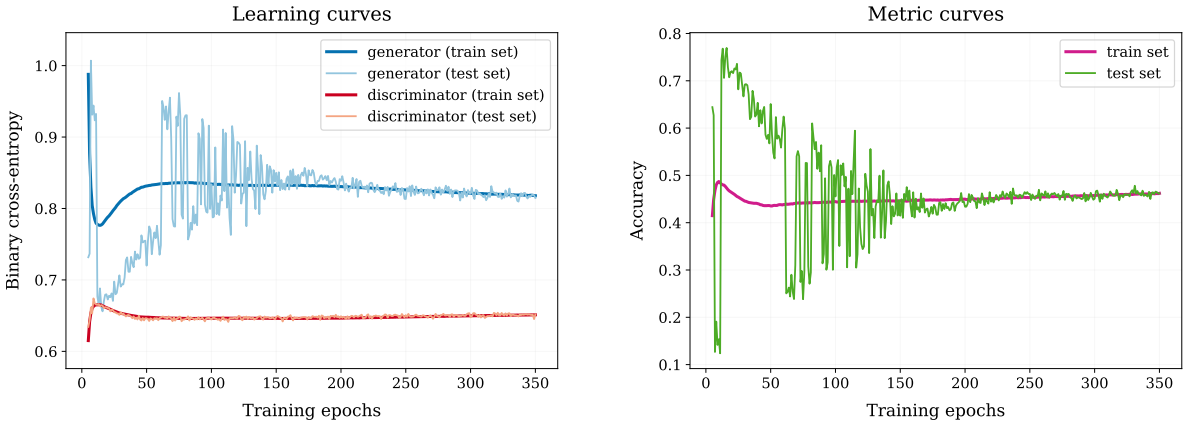


Figure 4.25: Learning and metric curves of a GAN model trained to parameterize the 5×5 covariance matrices describing the track states at LHCb. The left plot reports the competition of the *generator* and *discriminator* networks, whose training is driven by the minimax game. The evolution of the accuracy is depicted in the right plot to show the improvement of the similarity between the reference and generated samples during the training.

Validation studies

The GAN training was favored by the adoption of a series of reversible simplifications intended for avoiding non-physical predictions, like for the case of the parameterization of $\log(C_{ii})$ instead of C_{ii}, and for reducing the correlations between the covariance elements modeling ρ_{ij} in place of C_{ij}. Despite the pursued strategy, the final goal is that the trained generator succeeds in reproducing correctly all the elements of the covariance matrix C, including the correlations between each of its elements. Hence, to assess the effectiveness of the adopted simplifications and the performance achieved by the trained generator, a dataset of about 2 million *reconstructed* particles was retained for validation studies.

As the first step of this validation study, we need to revert the simplifications adopted for stabilizing the training. Hence, both the detailed simulated sample and the generator output were post-processed inverting the preprocessing transformations. In this way, we have back $\log(C_{ii})$ with $i \in \{1 \dots 5\}$ and ρ_{ij} with $i > j$, from which obtaining the covariance elements C_{ij} is trivial. Then, the comparison between the probability distributions expected from Detailed Simulation and the ones induced by the GAN training is available. The histograms along the diagonal of Figure 4.26 report such comparison for the *diagonal* elements of the covariance C resulting from a pure sample of Long tracks. The reference distributions are shown through blue shaded histograms, while the outputs of the trained generator are superimposed by using red solid-line histograms. The lower-triangular portion of Figure 4.26 instead reports several scatter plots showing the correlations between the various C_{ii} elements. The true correlations are described by blue points, while the result of a neural network trained in an adversarial configuration is depicted by using red points. The agreement between the histograms and the non-trivial correlations shown in Figure 4.26 is excellent. The trained generator succeeds in reproducing the expected distributions for the diagonal elements C_{ii}, even on the tails whose parameterization is typically made difficult by the scarce statistics. In addition, the NN-based model was so able to learn the *conditional* probability distributions that also

the correlations between the C_{ii} are well reproduced, even the ones that reveal a linear dependency:

$$C_{xx} \propto C_{yy} \qquad \text{and} \qquad C_{t_x t_x} \propto C_{t_y t_y} \tag{4.9}$$

The covariance model achieves slightly worse performance for Upstream tracks, while the agreement between the correlations expected from the Detailed Simulation and the ones learned by the GAN suffers from a significant drop in performance for Downstream tracks. This is probably related to the reduced number of Downstream instances which represents less than 10% of the total size of the training sample. Nevertheless, the agreement exhibited between the reference and generated histograms is still decent.

Lastly, Figures 4.27 and 4.28 report the performance achieved by the GAN-based model in reproducing a selected set of *off-diagonal* covariance elements C_{ij} for Long tracks. In particular, Figure 4.27 shows the distributions and correlations of $\text{cov}(x, v)$ with $v \in \{y, t_x, t_y, q/p\}$, while the performance achieved for $\text{cov}(q/p, v')$ with $v' \in \{x, y, t_x, t_y\}$ is explored in Figure 4.28. For both the set of elements C_{ij}, the covariance GAN correctly parameterizes the distributions expected from Detailed Simulation. In addition, the model exhibits astonishing performance in reproducing the C_{ij} correlations, even on the non-trivial one shown in Figure 4.27. Also the correlations depicted in Figure 4.28 are well parameterized, even if it should be noticed that the model struggles in reproducing second-order correlation components since their poor statistics. As for the diagonal elements, the performance achieved by the covariance GAN in parameterizing the off-diagonal elements for Upstream tracks is slightly worse than the one for Long tracks, and even worse for Downstream tracks, especially when it comes to reproducing C_{ij} correlations. Despite this, the agreement between the reference and generated histograms is more than decent so as to not compromise the flash-simulation of primary and secondary vertices, and impact parameters aimed by LAMARR.

4.3 Charged particles pipeline: the PID system

Once the hits left by a charged particle traversing the LHCb spectrometer are properly combined into a track candidate, the following step of the reconstruction phase is to complete the track information with a hypothesis on the *particle specie*. To this end, each reconstructed track is typically coupled with the response of one or more of the detectors that compose the *Particle Identification* (PID) system. The latter, operating in a wide momentum range ($2 \div 150$ GeV/c), allows LHCb to distinguish among all the quasi-stable particles by relying on a set of complementary experimental strategies. The momentum measurement combined with the Cherenkov angle provided by two RICH detectors permits the separation of kaons from protons and charged pions. The ECAL detector allows the identification of photons and neutral pions, and to restore the energy lost by electrons for bremsstrahlung radiation finalizing their identification. The response of the HCAL detector is typically employed by the trigger for events selection, while the muons capable of escaping HCAL are tracked by using the five dedicated stations of the MUON system located at the end of the spectrometer.

Given a traversing particle through LHCb, we expect that the response of the PID detectors depends only on the kinematics of the particle, the occupancy of the detectors (which may be different event-to-event and for different particle production mechanisms), and experimental conditions such as alignments, temperature, and gas pressure (which

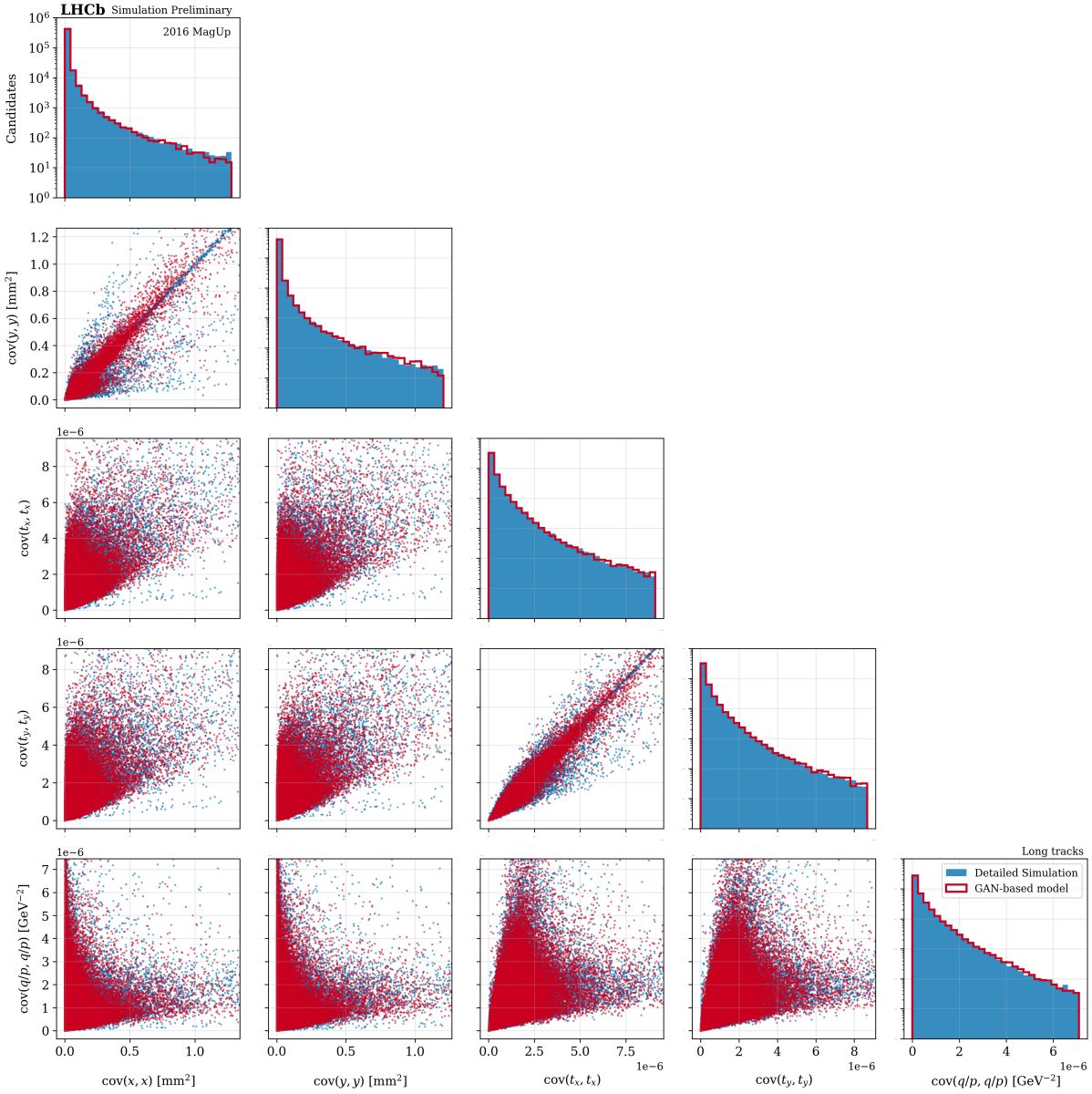


Figure 4.26: Validation plots of the *diagonal* elements of the covariance matrix C associated to the track parameters $\vec{x}$, namely $\mathrm{cov}(v, v)$ with $v \in \{x, y, t_x, t_y, q/p\}$. The distributions of the diagonal elements as obtained from Detailed Simulation are represented via blue shaded histograms along the diagonal of this Figure. The results of a GAN model trained to parameterize the covariance matrix are superimposed using red solid-line histograms. The lower-diagonal portion of this Figure reports several scatter plots used to investigate the ability of the trained model to reproduce also the correlations between the C_{ii} elements. Again, what expected from Detailed Simulation is shown in blue, while the red points result from the trained neural network.

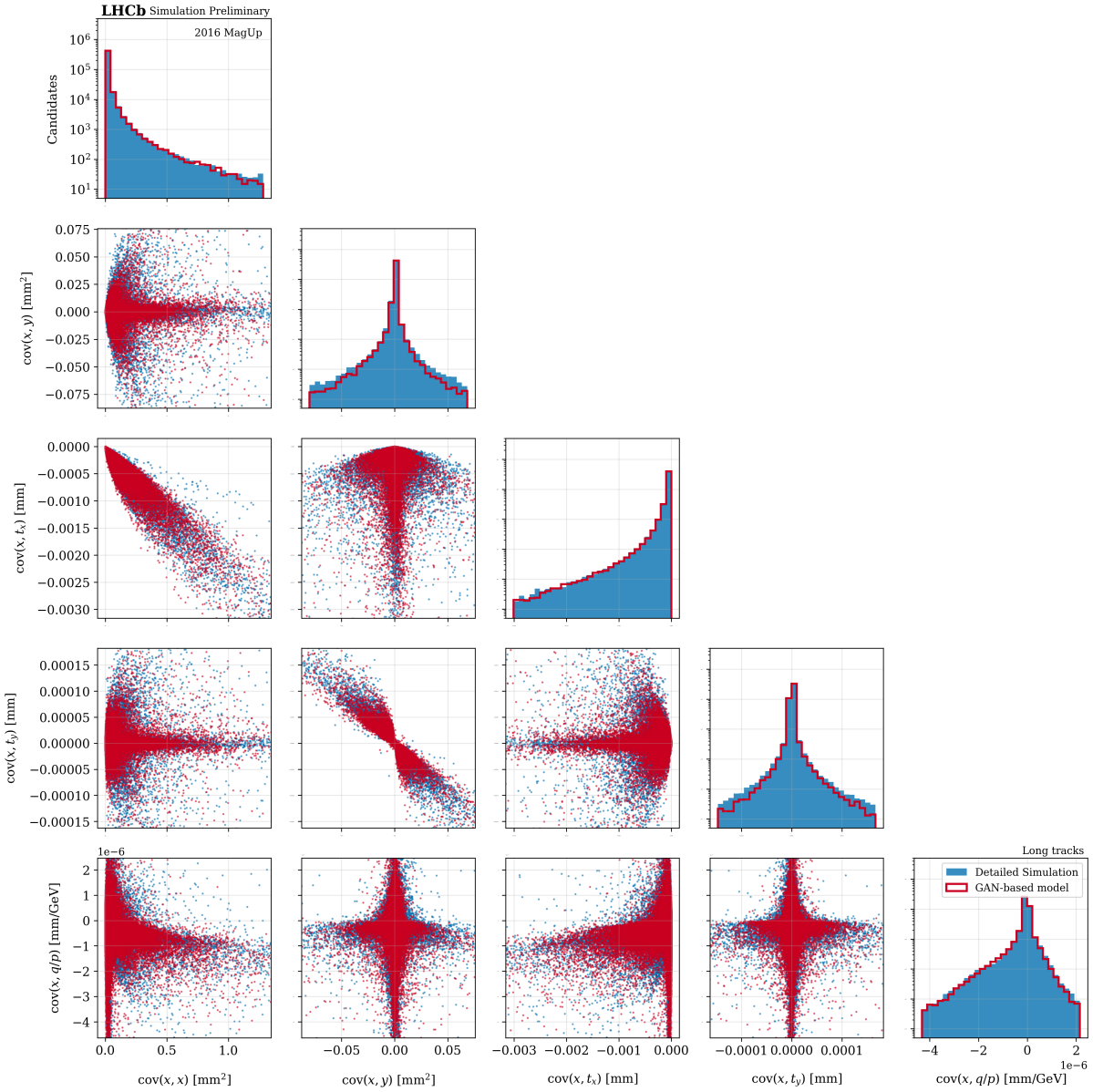


Figure 4.27: Validation plots of a selected set of *off-diagonal* elements of the track covariance matrix C considering, in particular, $\mathrm{cov}(x, v)$ with $v \in \{y, t_x, t_y, q/p\}$. The distributions of such off-diagonal elements as obtained from Detailed Simulation are represented via blue shaded histograms along the diagonal of this Figure. The results of a GAN model trained to parameterize the covariance matrix are superimposed using red solid-line histograms. The lower-diagonal portion of this Figure reports several scatter plots used to investigate the ability of the trained model to reproduce also the correlations between the C_{ij} elements. Again, what expected from Detailed Simulation is shown in blue, while the red points result from the trained neural network.

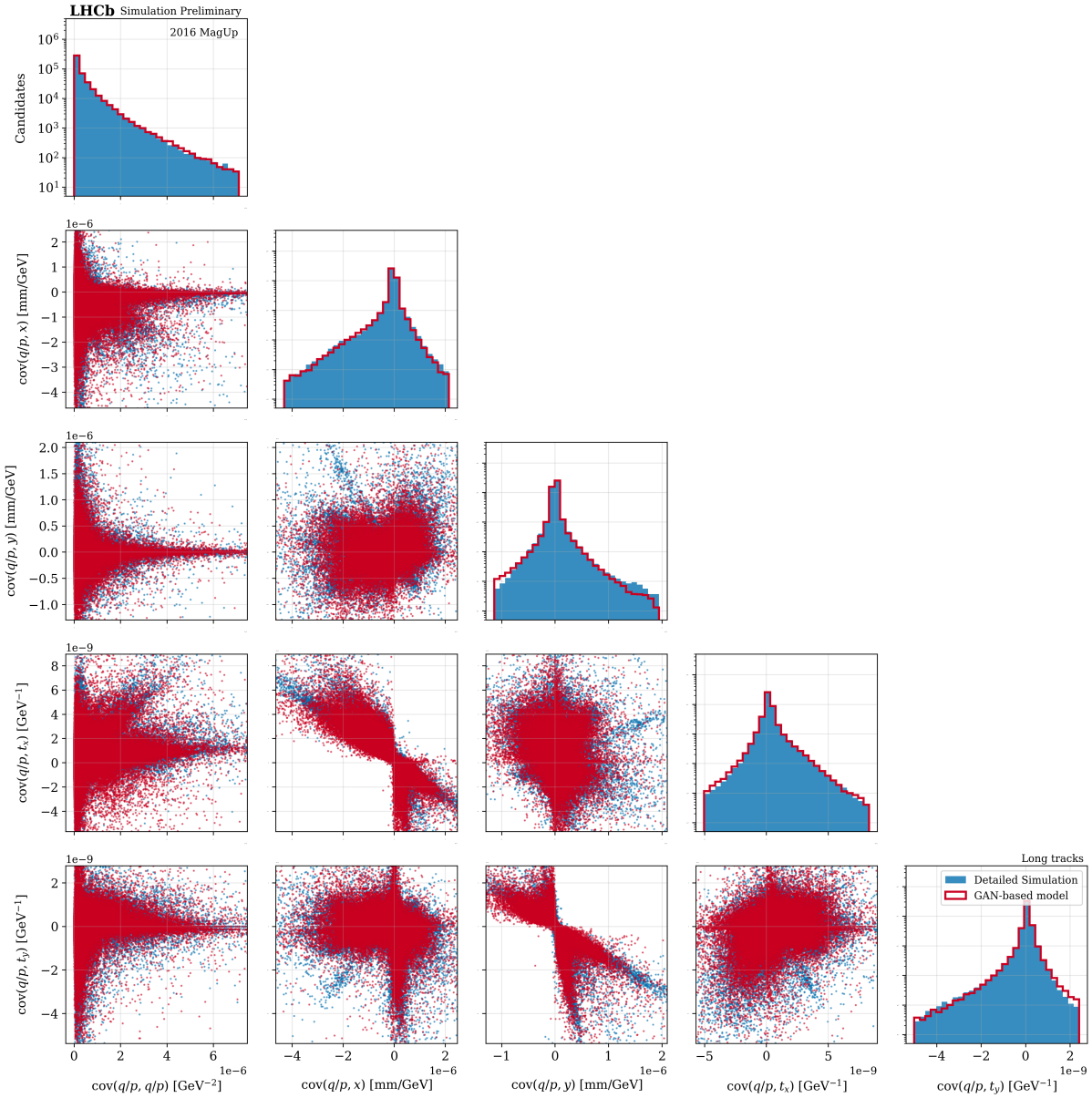


Figure 4.28: Validation plots of a selected set of *off-diagonal* elements of the track covariance matrix C considering, in particular, $\mathrm{cov}(q/p, v)$ with $v \in \{x, y, t_x, t_y\}$. The distributions of such off-diagonal elements as obtained from Detailed Simulation are represented via blue shaded histograms along the diagonal of this Figure. The results of a GAN model trained to parameterize the covariance matrix are superimposed using red solid-line histograms. The lower-diagonal portion of this Figure reports several scatter plots used to investigate the ability of the trained model to reproduce also the correlations between the C_{ij} elements. Again, what expected from Detailed Simulation is shown in blue, while the red points result from the trained neural network.

may modify the response of detectors run-to-run) [49]. LAMARR assumes that the response of a PID variable is fully parameterized by some known set of variables, such as the track momentum p, the pseudorapidity η, and the number of reconstructed tracks (`nTracks`) used for measuring the detector occupancy. By considering the pipeline philosophy of LAMARR, the set of particles actually reconstructed and the corresponding kinematic properties are provided by the tracking models discussed in Section 4.2. On the contrary, to parameterize the detector occupancy, the current version of LAMARR relies on a *parametric function*, whose definition derives by fitting the simulated `nTracks` distribution with a linear combination of Gaussian distributions. Indeed, while counting the number of particles in acceptance has been found to provide a rather reasonable proxy for the number of reconstructed tracks, relying on a parameterization for the detector occupancy enables replacing the PYTHIA8 generator with a *particle-gun* generating only the signal while relying on the parameterizations for all the occupancy-related effects. As first-order approximation, the `nTracks` parameterization succeeds in reproducing the number of reconstructed tracks, but it strongly depends on the *flavour* of the hadron decays that compose the training sample. Hence, to describe the detector occupancy for events including b- or c-hadron decay, LAMARR disposes of two different flavour-specialized `nTracks` models.

The track kinematic properties provided by the LAMARR tracking modules and the number of reconstructed tracks as parameterized for a given event can be used to reproduce the high-level response of the LHCb PID system. In particular, to provide the analysis-level quantities required for the majority of the physics studies, LAMARR relies on the following parameterizations:

- **RICH system.** Model for the high-level response of the RICH system when traversed by muons, pions, kaons, or protons;
- **MUON system.** Model for the high-level response of the MUON system when traversed by muons, pions, kaons, or protons;
- `isMuon` **criterion.** Model for the probability of a track to be associated with a muon candidate parameterizing the `isMuon` criterion for muons, pions, kaons, or protons;
- **Global PID response.** Model for the global high-level response of the PID system, resulting from the combination of the response of the RICH detectors, calorimeters, and MUON system when traversed by muons, pions, kaons, or protons.

At the moment, LAMARR does not provide any parameterizations for the response of the PID system when traversed by electrons, whose energy recovery requires to disposal of a dedicated model for the bremsstrahlung photons, currently under investigation as discussed in Section 4.4.2. Hence, the ECAL detector is not included among the parameterizations for charged particles listed above. Also the HCAL detector is excluded from such a list since it is mostly employed for the trigger. LAMARR has not any dedicated model for the trigger, which is indirectly parameterized by relying on the models for geometrical acceptance and tracking efficiency. Actually, a restricted set of analysis-level variables resulting from the calorimeters is employed to compute the global response of the PID system, either within the combined likelihoods or for evaluating the capability of

a multivariate classifier (ANNPID) to further improve the separation between different particle species. As discussed in Section 4.3.5, GAN-based models succeed in reproducing these global high-level variables without relying on the information from the calorimeters, whose contribution is approximated by relying on the generator *latent space* $\mathcal{Z}$ [1].

The rest of this Section is devoted to describing how *neural networks* and *generative models* can be used to parameterize a set of analysis-level quantities resulting from the LHCb PID system when traversed by muons, pions, kaons, or protons. Section 4.3.1 illustrates the features of a Python package designed and developed during my Ph.D. to simplify the implementation and training of GAN-based models. How to parameterize the high-level response of either the RICH or MUON system by using deep generative models properly conditioned is described in Sections 4.3.2 and 4.3.3, respectively. Section 4.3.4 demonstrates that a neural network trained to solve a classification problem succeeds in modeling the `isMuon` criterion. Finally, the outputs of all the aforementioned parameterizations can be combined by relying on a further generative model to reproduce the global high-level response of the PID system, as detailed in Section 4.3.5.

4.3.1 Ready-to-use GANs in Python

My Ph.D. research activity has been largely dedicated to the design, optimization, and validation of the ML-based models used by LAMARR to reproduce the high-level response of the LHCb detector by only relying on the information from physics generators. As discussed in Section 4.2, neural networks and GANs can fruitfully be employed to describe the response of the Tracking system, and the same applies to the PID system as it will be further detailed in the following Sections. The heavy use of GANs within the parameterizations and the different performance offered by the state-of-the-art algorithms and regularization strategies discussed in Section 3.2 created the necessity of simplifying their implementation and standardizing the training and optimization procedures. Such efforts have rapidly turned into PIDGAN [6], a Python package that, by relying on Keras [167] and TensorFlow [168] as back-ends, simplifies the implementation and training of GAN-based models intended for High Energy Physics (HEP) applications. Originally designed to develop parameterizations to flash-simulate the LHCb PID system, PIDGAN can be used to describe a wide range of LHCb sub-detectors (including the Tracking system) and succeeds in reproducing the high-level response of a generic HEP experiment. The PIDGAN package was publicly presented[8] in November 2023 during the "Fifth ML-INFN Hackathon: Advanced Level", where it was used to parameterize a set of analysis-level quantities as reconstructed by the CMS experiment when traversed by high-energy *jets*[9].

PIDGAN provides ready-to-use Python implementations for several GAN algorithms (listed in Table 4.5) by relying on the TensorFlow and Keras APIs. In particular, GANs are implemented by subclassing the TensorFlow `Model` class and customizing the training procedure that is executed when one calls the `fit()` method. Different GAN *flavours* do not vary only for the loss function used, but also for the regularization strategies eventually adopted. By design, PIDGAN encodes these flavour-specific differences directly within the customized TensorFlow classes and exposes high-level APIs that simplify the GAN implementation. To mitigate the vanishing gradient problem, PIDGAN also provides a set

[8]The complete event agenda is available at `https://agenda.infn.it/event/37650`.

[9]A *jet* is a narrow cone of hadrons and other particles produced by the hadronization of a quark or gluon in a HEP experiment. Read more on `https://en.wikipedia.org/wiki/Jet_(particle_physics)`.

`GAN` Implementation of the GAN algorithm proposed by Ian Goodfellow and others [134]. Both the loss functions discussed in the original paper are available through the `use_original_loss` flag.
`BceGAN` Implementation of the BCE-based GAN algorithm. The loss functions of the discriminator and generator networks are defined in (4.6) and (4.7), respectively.
`LSGAN` Implementation of the LSGAN algorithms [143]. Both the loss functions discussed in the paper are available through the `minimize_pearson_chi2` flag.
`WGAN` Implementation of the WGAN algorithm [145]. The *weights clipping* strategy is used to regularize the Lipschitzianity of the discriminator network.
`WGAN_GP` Implementation of the WGAN-GP algorithm [146]. The *gradient penalty* strategy is used to regularize the Lipschitzianity of the discriminator network.
`CramerGAN` Implementation of the CramerGAN algorithm [148]. The *gradient penalty* strategy is used to regularize the Lipschitzianity of the discriminator network.
`WGAN_ALP` Implementation of the WGAN-ALP algorithm [147]. The *adversarial Lipschitz penalty* strategy is used to regularize the Lipschitzianity of the discriminator network.
`BceGAN_GP` Implementation of the BCE-based GAN algorithm. The *gradient penalty* strategy is used to regularize the Lipschitzianity of the discriminator network [146].
`BceGAN_ALP` Implementation of the BCE-based GAN algorithm. The *adversarial Lipschitz penalty* strategy is used to regularize the Lipschitzianity of the discriminator network [147].

Table 4.5: List of GAN algorithms provided by PIDGAN (`v0.1.3`).

of techniques for training stabilization that can be enabled during the instantiation of the GAN algorithm. Among the techniques available, we can inject Gaussian noise within the discriminator to prevent the disjoint separation between the reference and generated samples, and then ensure valuable feedback for the generator training [144]. Alternatively, to further regularize the discriminator, PIDGAN allows constraining its hidden states by requiring that the latters do not differ too much between the reference and generated samples by following what authors of Ref. [176] called the *features matching* strategy.

PIDGAN simplifies also the design of the generator and discriminator networks that need to operate taking *conditions* (e.g., momentum, pseudorapidity, or `nTracks`) as input [141] to simulate the response of a generic HEP experiment. The two players are implemented by subclassing the TensorFlow `Model` class and customizing the `call()` method to support the processing of conditions or to add automatically the *latent vectors* z as needed by

the generator network. By doing so, PIDGAN enables building and train a GAN model by using about twenty lines of code, as depicted in the following example:

```
from pidgan.players.generators import Generator
from pidgan.players.discriminators import Discriminator
from pidgan.algorithms import GAN

x = ...  # conditions
y = ...  # targets

G = Generator(
    output_dim=y.shape[1],
    latent_dim=64,
    output_activation="linear"
)
D = Discriminator(
    output_dim=1,
    output_activation="sigmoid"
)

model = GAN(generator=G, discriminator=D, use_original_loss=True)

model.compile(
    metrics=["accuracy"],
    generator_optimizer="rmsprop",
    discriminator_optimizer="rmsprop",
)

model.fit(x, y, batch_size=256, epochs=100)
```

The default implementation of the generator and discriminator networks relies on the TensorFlow Sequential model[10] that allows to construct the two players as plain FNNs. Alternatively, PIDGAN also provides implementations based on the TensorFlow Functional API[11] that enable the use of *skip connections* [111] necessary whenever one aims to extend the scale of the models avoiding the consequent vanishing gradient issues. In addition, PIDGAN offers a further implementation for the discriminator that aims to induce physics constraints during the training procedure by relying on a set of *auxiliary features* [177]. The latters, computed at runtime, result from a user-defined combination of the discriminator input features and can be used to indirectly inform the generator on non-trivial correlations among the variables through the GAN minimax game. The benefits of disposing of such an auxiliary system will be more clear in the following Sections where it is deeply used to improve the quality of the probability distributions of variables not directly parameterized by the generator, but instead resulting from a combination of its outputs.

Finally, the PIDGAN's GAN algorithms allow to define a third independent network, called *referee*, and to train it together with the other two players. Just like the discriminator, the referee network is trained to distinguish the reference instances from the generated ones, but without participating in the minimax game. Both the discriminator and referee

[10] `https://www.tensorflow.org/guide/keras/sequential_model`
[11] `https://www.tensorflow.org/guide/keras/functional_api`

networks allow to reduce to one the dimensionality of the classification problem. However, if the discriminator is contaminated by the generator updates that prevent it from having unbiased outputs, the referee can be fruitfully employed to assess the performance achieved by the generator [178]. The referee network provided by PIDGAN can be trained also as a standalone to perform binary or multi-class classification tasks. Hence, by relying on PIDGAN, we can design and train GANs to learn conditional probability distributions, as well as neural networks to parameterize the efficiencies (including the tracking ones discussed in the previous Sections).

Optimization campaigns for PID models

The performance exhibited by GAN-based models is intimately connected to the combination of hyperparameters chosen for their design and training. Despite such generative algorithms have proven to succeed in reproducing the high-level response of the LHCb PID detectors [1, 177, 179–181], using them for simulation production requires a precise parameterization of the variables typically adopted in physics analyses. Hence, PID GANs, more than the tracking models, benefit from the use of *optimization studies*, performed to identify the best-suited set of hyperparameters to train models that reproduce accurately the response of the LHCb spectrometer in the widest possible range of the input conditions (e.g., p, η, and `nTracks`).

During my Ph.D., I had the opportunity to access several GPU instances, provided by on-premises, cloud, and HPC resources, and used them to train ML-based models or run intense optimization campaigns. The stochastic nature of the neural network training makes the path between the chosen hyperparameters and the performance exhibited by the model not deterministic, but instead affected by statistical uncertainties. Bayesian techniques allow the definition of optimization strategies that take into account such uncertainties while searching for the minimum (or maximum) of the function under investigation. Since the Bayesian techniques do not rely on gradient computation, either a single instance or multiple computing nodes *in parallel* can participate to the same optimization study by testing different combinations of hyperparameters to find the optimum. As discussed in Section 2.6.1, disposing of several computing instances from different resource providers laid the foundations for HOPAAS [2], a cloud service designed for coordinating optimization studies across multiple computing instances via HTTP requests.

Since PID GANs strongly rely on hyperparameter optimization studies, PIDGAN is natively integrated with the HOPAAS service. In particular, PIDGAN encodes a set of HOPAAS APIs within a custom TensorFlow `Callback` that can be enabled to update the remote state of the optimization study continuously during the training procedure. Such a mechanism can also be employed to abort the test of non-promising hyperparameters (*pruning*), reducing the waste of computing resources to take a sub-optimal training to an end. In this context, the referee network plays a key role by providing an independent measurement of the quality of the trained generator that can be used to either drive the hyperparameter optimization study or to promptly interrupt non-promising trials.

Despite the valuable role played by the referee network, its performance may vary profoundly trial-to-trial due to the combined effect of the stochastic nature of neural networks training and the high variability of a *dynamic* system like the one established by the GANs training. Hence, to have a clear picture of the optimization achievements, the

use of a *static* score was preferred to drive the studies of the PID models. In particular, as will be detailed in the following Sections, the *Kolmogorov-Smirnov* (KS) distance was selected as the optimization score to measure the differences between the reference sample and the generated one. Since such metric rapidly loses power as the dimensionality increases, a well-known problem referred to as the *curse of dimensionality* [182], the KS distance was computed variable-by-variable restoring a univariate problem. In addition, aiming to reproduce the response of the PID detectors as precisely as possible in a wide range of the input conditions, the target variables were split into bins of momentum, pseudorapidity, and `nTracks` before computing the KS distances. Finally, the maximum value among these mismodeling errors was taken as the optimization score, so that requiring its minimization corresponds to rejecting models introducing large errors in at least some region of the phase-space of the LHCb PID system $(p, \eta, \texttt{nTracks})$

4.3.2 RICH detectors

The primary role of the LHCb RICH system is to distinguish among the charged hadrons (i.e., pions, kaons, and protons), and to contribute to the identification of the charged leptons (i.e., electrons and muons) in combination with the other PID detectors. The discrimination between the various particle species relies on the Cherenkov angle θ_C defined in Eq. (1.6) that, combined with the track momentum, allows to derive a mass hypothesis. The Cherenkov photons emitted by the particles traversing the RICH radiators are conveyed through a system of mirrors and collected forming rings in dedicated photo-detectors. Depending on the momentum of the traversing particle and assuming a mass hypothesis, a test ring for each particle species is compared against the hits in the photo-detectors, and the corresponding likelihood is evaluated [44]. Hence, the high-level response of the RICH system is usually expressed as a *differential log-likelihood* (DLL) between two particle hypotheses h_1 and h_2. Since the most abundant species produced at hadron colliders are pions, it is customary to define DLLs with respect to the pion hypothesis:

$$\texttt{RichDLLh} \equiv \log\left(\frac{\text{RICH likelihood for } h}{\text{RICH likelihood for } \pi}\right) \quad \text{with} \quad h \in \{e, \mu, K, p\} \tag{4.10}$$

Implementing a flash-simulation paradigm, LAMARR aims to reproduce the RICH DLLs without relying on the underlying physics processes, but rather directly parameterizing them only by using the information provided by its pipelines. Since the Cherenkov radiation depends on the velocity of the traversing particles, we expect that the RICH response is strongly related to the momentum. In addition, due to the arrangement of the mirrors system (reported in Figure 1.10), the DLLs are necessary related to the pseudorapidity of the particles. Since the likelihoods measure the match between the hits in the photo-detectors and the test rings, we also expect that the latters depend on the occupancy of the RICH detectors, which may vary the total number of reconstructed hits. Given two traversing particles through the RICH system, even if they are produced with the same specie, kinematic properties, and detector occupancy, the resulting DLLs may be different due to the statistical nature of the radiation-matter interactions causing the detector low-level response. Hence, in searching for an ML-based model for the RICH system we should opt for *deep generative models*, algorithms able to derive from data the underlying probability distributions. The current version of LAMARR relies on conditional

GANs for this purpose, following the path traced by Refs. [1, 179]. The features used to conditionate the RICH models are listed below:

- the reconstructed momentum p of the traversing particles;
- the reconstructed pseudorapidity η of the traversing particles;
- the number of reconstructed tracks `nTracks`;
- the charge q of the traversing particles.

The reconstructed particles and the corresponding momentum and pseudorapidity are available to LAMARR from the tracking models discussed in Section 4.2. The measurement of the occupancy of RICH detectors (i.e., `nTracks`) results by sampling from a parametric probability distribution function modeled to reproduce what expected from Detailed Simulation. Finally, the charge of the traversing particles is directly provided by the physics generators. These features are used to learn the conditional probability distributions of the following likelihoods:

- the RICH DLL of the electron hypothesis versus the pion one `RichDLLe`;
- the RICH DLL of the muon hypothesis versus the pion one `RichDLLmu`;
- the RICH DLL of the kaon hypothesis versus the pion one `RichDLLk`;
- the RICH DLL of the proton hypothesis versus the pion one `RichDLLp`.

A keen eye has surely noticed that the input conditions do not include the *specie* of the traversing particles, despite we expect that this property affects significantly the response of the RICH system by design. The strategy pursued by LAMARR is to provide one specialized parameterization per each of the particle species currently tackled. To this end, four different pure samples of muons, pions, kaons, and protons were prepared simulating a cocktail of b-hadron decays with an official configuration of GAUSS [61] `sim10` (`v56r6`), involving PYTHIA8 [25], EVTGEN [58], and GEANT4 [59, 60]. The resulting raw data was then used to reconstruct particle candidates calculating the tracking and PID information by relying on the BRUNEL and DAVINCI applications [171].

Model design and training

As mentioned in Section 4.3.1, the best combination of hyperparameters and training strategies adopted for parameterizing precisely the RICH system results from intense optimization campaigns that have taken $\mathcal{O}(10^3)$ hours per each of the four specie-specialized GAN models, combining GPUs from on-premises, cloud, and HPC resources. Optimization studies running on different computing instances were coordinated by relying on the HOPAAS [2] service (further described in Section 2.6.1) that allows to find the optimum set of hyperparameters with respect to a target score. In this case, the target score corresponds to the KS-distance between the distributions of the RICH DLLs generated by the trained GANs and the ones resulting from the reference samples. Indeed, the KS-distance was computed in bins of momentum, pseudorapidity, and `nTracks`, and the target score was defined as its maximum value among the whole phase space. Hence, minimizing such a

target score corresponds to requiring the set of hyperparameters that allow to train the model with the lowest mismatch in the whole phase space $(p, \eta, \texttt{nTracks})$.

As expected, the hyperparameters that affect more the performance of the trained models are the initial values of the learning rates for both the generator and the discriminator, and the strategy adopted for their scheduling, such as the decay steps in case of an exponential decrease of the learning rates. Another crucial component is the GAN *flavour* and the corresponding minimax game that drives the generator training. The optimization campaigns have revealed two major GAN algorithms that succeed in parameterizing the RICH response better than others: the BCE-based algorithm already used for the tracking parameterizations and the LSGAN algorithm [143]. By using the PIDGAN package [6], the implementation of such algorithms is straightforward. In particular, the RICH GAN systems rely on 10-layer neural networks for both the generator and the discriminator. Each hidden layer counts 128 neurons and outputs with a Leaky ReLU activation function. Regardless of the GAN algorithm, the generator hidden layers are equipped with skip connections [111], while the discriminator is implemented as a plain FNN in the case of LSGAN since we experienced some training instability problems when using residual blocks with this *flavour*.

To parameterize the RICH response to properly take into account the underlying phase space $(p, \eta, \texttt{nTracks})$, the generator is trained by using as input features the aforementioned conditions x in combination with elements z of the latent space, sampled from a 64-dimensional normal distribution. The generator has four output neurons, one for each of the target likelihoods y, namely $\texttt{RichDLLh}$ with $h \in \{e, \mu, K, p\}$. Since the latters are differential log-likelihoods, their subtraction has a precise physical meaning, such as $\texttt{RichDLLpK} \equiv \texttt{RichDLLpK} - \texttt{RichDLLpK}$ which represents the proton hypothesis versus the kaon one. The separation between protons and kaons is one of the main tasks of the LHCb RICH system, hence it is of primary importance that the generator reproduces accurately also the $\texttt{RichDLLpK}$ distribution. To this end, the discriminator takes as input a set of *auxiliary features* in addition to the conditions x and the DLLs y (real or generated). Inspired by what is proposed in Ref. [177] for adding physics-based constraints to the output of a GAN trained to reproduce the low-level response of the LHCb ECAL detector, the RICH discriminator is fed by additional features to accomplish its classification task:

- the RICH DLL of the muon hypothesis versus the electron one $\texttt{RichDLLmue}$;
- the RICH DLL of the proton hypothesis versus the kaon one $\texttt{RichDLLpK}$.

Thanks to the PIDGAN APIs, the discriminator is able to compute the above auxiliary features at runtime[12] during the training procedure by using either the reference data or the generator output. The last layer of the discriminator counts a single neuron with a sigmoid activation function that describes the probability that the input, including conditions, target likelihoods, and auxiliary features, belongs to the reference sample. By doing so, the generated samples can benefit from such auxiliary physics information by relying on the minimax game and avoiding adding explicitly any other features to the generator.

[12]The preprocessing strategy adopted for the target DLLs should take into account that the *auxiliary features* are computed at runtime. This is physically reasonable only if the preprocessing maintains the shape of the likelihood distributions, such as by using the `MinMaxScaler` or `StandardScaler` transformations of scikit-learn [175].

	Generator	*Discriminator*
skip connections [111]	✔	✖/✔/✔/✔
latent space dim [134]	64	-
n auxiliary features [177]	-	2
input shape	(None, 68)	(None, 10)
input preprocessing	✔	✔
n hidden layers	10	10
n hidden neurons	128	128
hidden activation functions	Leaky ReLU	Leaky ReLU
output activation function	linear	sigmoid
output shape	(None, 4)	(None, 1)
output preprocessing	✔	✖
optimizer	RMSprop	RMSprop
learning rate	$\sim (4/1/5/4) \times 10^{-4}$	$\sim (5/2/7/8) \times 10^{-4}$
loss function	Least squares loss (3.12) BCE-based loss (4.7) BCE-based loss (4.7) BCE-based loss (4.7)	Least squares loss (3.12) BCE-based loss (4.6) BCE-based loss (4.6) BCE-based loss (4.6)
BCE label smoothing	-/0.05/0.05/0.05	-/0.05/0.05/0.05
Gaussian noise stddev [144]	✖/0.02/0.02/0.02	✖/0.02/0.02/0.02
learning rate scheduling	ExpDecay	ExpDecay
scheduling decay rate	0.1	0.1
scheduling decay steps	$(15/12/12/27) \times 10^4$	$(20/17/37/39) \times 10^4$
batch-size	10000	10000
batches per epoch	~ 900	~ 900
n epochs	500	500

Table 4.6: Hyperparameters of the GAN-based models for the RICH system. When multiple hyperparameter values are reported, they refer to the different settings adopted for the definition and training of the models for muons, pions, kaons, and protons, respectively.

To mitigate the vanishing gradient problem typical of GAN systems, the BCE-based algorithm relies on label smoothing and the injection of zero-mean Gaussian noise to the discriminator input [144]. On the contrary, LSGAN does not need any of these customary strategies, since its loss functions are designed to limit the gradient issues [143]. The complete list of hyperparameters used to build and train the four specie-specialized models for the RICH system is reported in Table 4.6.

The training and validation of the GAN models used to parameterize the high-level response of the RICH detectors can count on a dataset of $\mathcal{O}(0.5 \times 10^9)$ of detailed simulated particles, split in four pure samples of reconstructed muons, pions, kaons, and protons. Each of these sub-samples is divided into training (50%), test (10%), and validation (40%) sets. As usual, the test set was used for monitoring any sign of overtraining, while the validation set was adopted for assessing the performance of the trained models by comparing the reference and generated distributions. Figure 4.29 reports the learning and

metric curves resulting from the RICH GANs training. The evolution of the generator (in blue) and discriminator (in red) loss functions for muons, pions, kaons, and protons are depicted in top-down order on the left. Since either BCE-based GAN or LSGAN rely on a discriminator that outputs probabilities, the accuracy is a well-defined metric score for both algorithms. The right plots show the evolution during the training of the generator accuracy, namely the misidentification probability of the discriminator in recognizing instances from the reference sample. Regardless of the GAN algorithm, the desired solution of the minimax competition corresponds to the incapability of the discriminator to distinguish between reference and generated data, namely an accuracy score that asymptotically points to 50%.

Validation studies

The ultimate goal of LAMARR is to provide analysts with flash-simulated samples that reproduce faithfully the response and performance of the LHCb spectrometer [181, 183]. Since the role of the RICH detectors is to distinguish among charged hadrons and to contribute to the identification of charged leptons, in the validation studies we need to test that the trained GANs exhibit the same discrimination performance resulting from Detailed Simulation. To this end, the following Figures report the probability distributions of several RICH differential log-likelihoods $\Delta\text{LL}(h_1 - h_2)$ as expected from pure samples of h particles with $h \in \{\mu, \pi, K, p\}$. The reference distributions are then compared with what results from the trained generators. To further test the performance of the models, the selection efficiency and the misidentification probability plots are also reported, showing the capability of GANs to reproduce faithfully the response of the RICH system [181].

The distributions of `RichDLLmu` for muons and pions are depicted in Figure 4.30 in four bins of momentum. The high overlapping between the two distributions represents the difficulty in separating the two particle species relying only on the RICH response. The distribution of the DLL is well reproduced by the two GANs (specialized for muons and pions) for $p < 50$ GeV/c. The very high momentum bin presents some mismodeling effects, where both the models struggle in accurately reproducing the very narrow structure at `RichDLLmu` $= 0$. This mismodeling appears also in the efficiency plots reported in Figure 4.30 when the loose selection `RichDLLmu` > 0 is investigated: the overestimation of the fraction of particles that pass such cut at high momentum results from the same parameterization drawback. Due to the narrow distribution of `RichDLLmu` at high momentum, the effect of the mismodeling disappears by studying the efficiencies induced by the mild selection `RichDLLmu` > 5.

Figure 4.31 reports the distributions of `RichDLLk` for kaons and pions in four bins of momentum. Differently from the muon case, the RICH system offers an excellent kaon-pion separation in a wide momentum range ($2 \div 50$ GeV/c). The two GANs (specialized for kaons and pions) succeed in reproducing precisely the distributions of such DLL in the whole phase space. This is also demonstrated by the efficiency plots where both the trained generators reproduce the discriminating performance expected from Detailed Simulation within the statistical errors. The ability of the pion-specialized GAN to populate the structure peaked at `RichDLLk` $= 0$ for $p < 10$ GeV/c is noteworthy. This is a detector-dependent effect resulting from the active materials adopted by the RICH2 that makes a limit for the identification of kaons with $p \leq 9.3$ GeV/c [44]. The trained generator succeeds in reproducing such an effect by relying on the minimax game.

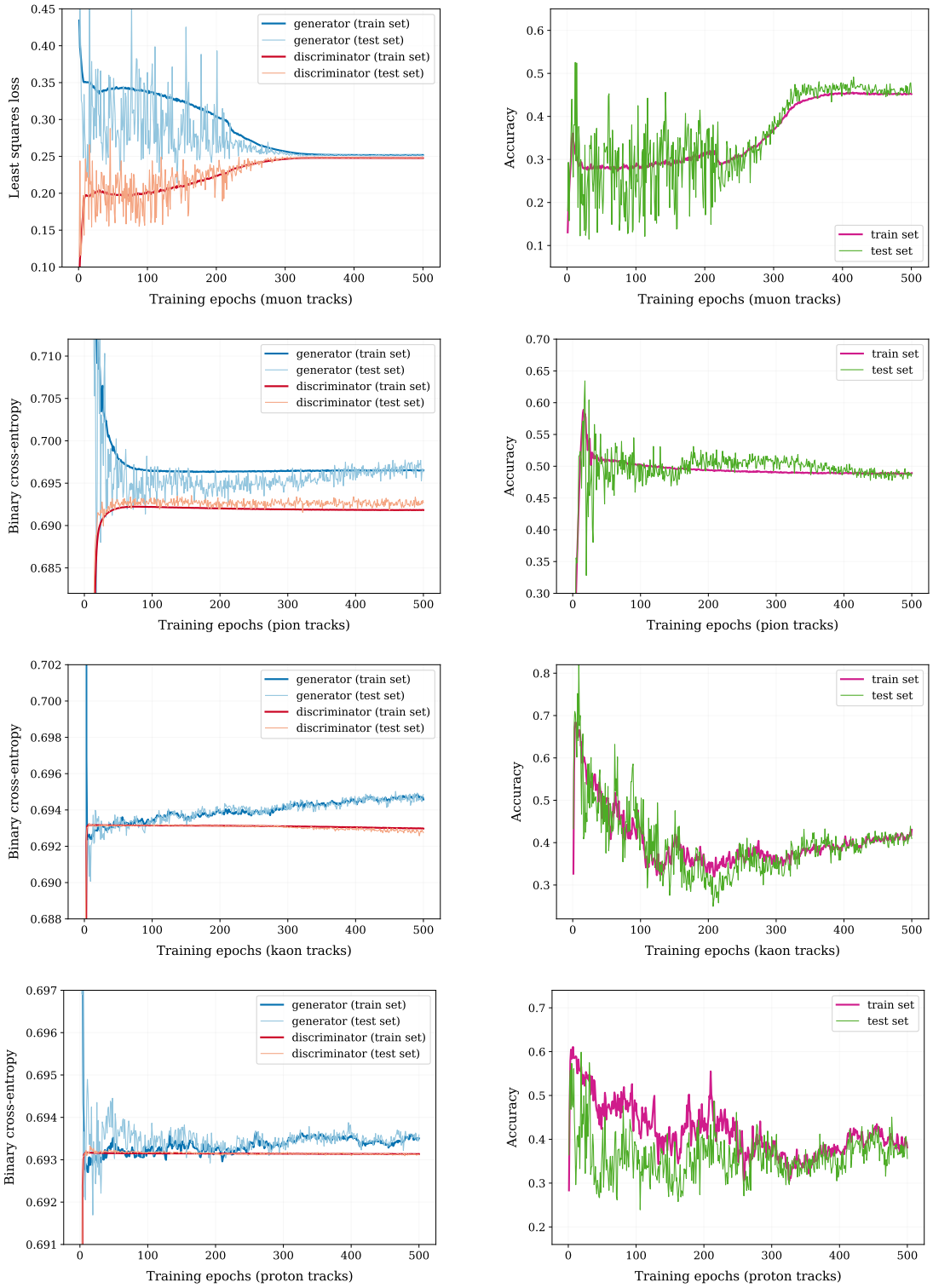


Figure 4.29: Learning and metric curves of the four GAN-based models trained to parameterize the LHCb RICH response when traversed by muons, pions, kaons, and protons (in top-down order). The competition between the *generator* and *discriminator* networks of the various GANs is depicted on the left plots. The evolution during the training of the fraction of generated instances wrongly classified by the discriminator, namely the generator accuracy, is reported on the right plots.

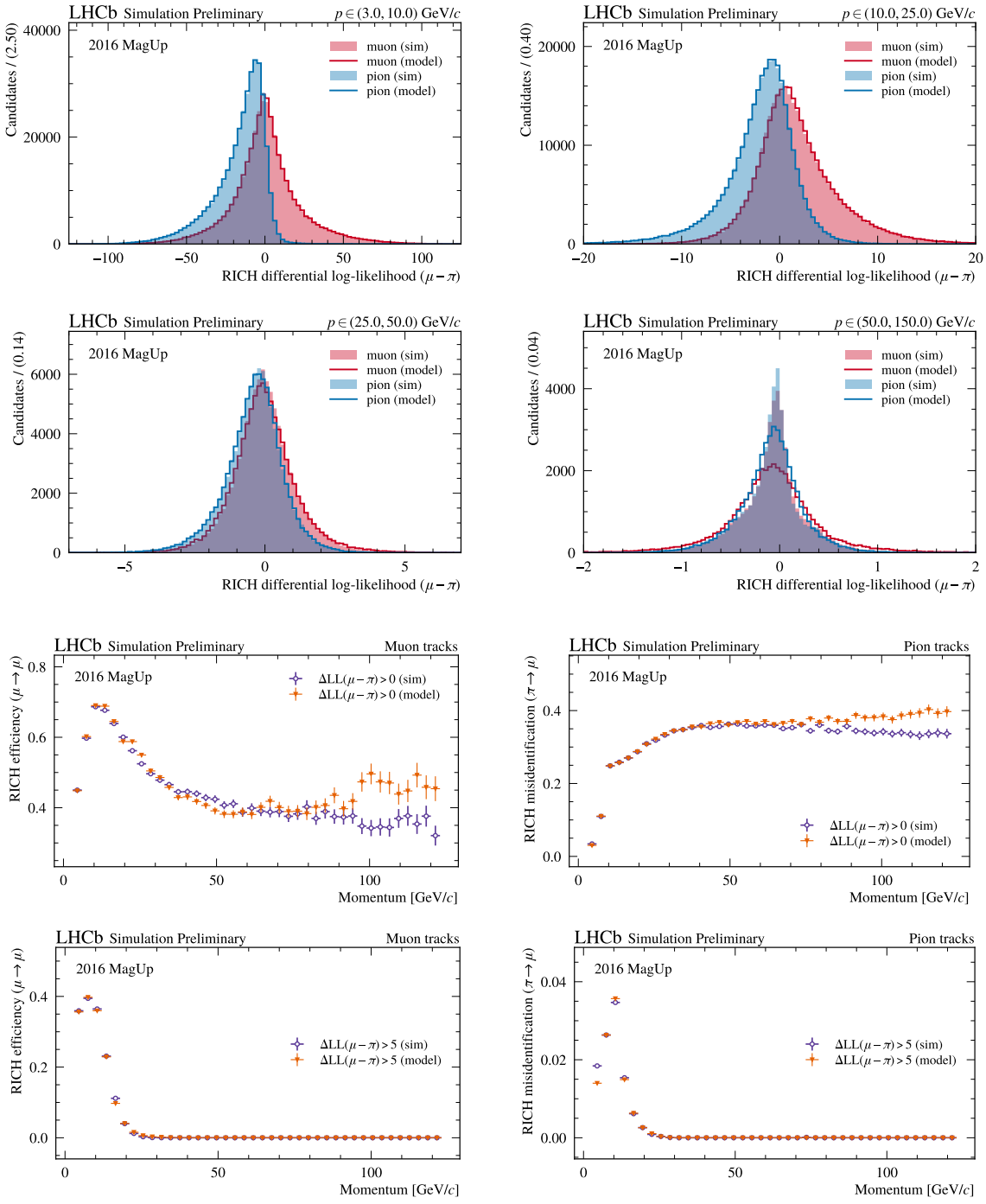


Figure 4.30: Validation plots of the GAN-based models trained to parameterize the *muon-pion separation* offered by the LHCb RICH detectors. The distributions resulting from Detailed Simulation for the `RichDLLmu` variable are represented as filled histograms for muons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the RICH efficiency for the muon identification is reported on the left plots using two selection cuts: `RichDLLmu` > 0 and `RichDLLmu` > 5. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to \mu$) as depicted on the right plots.

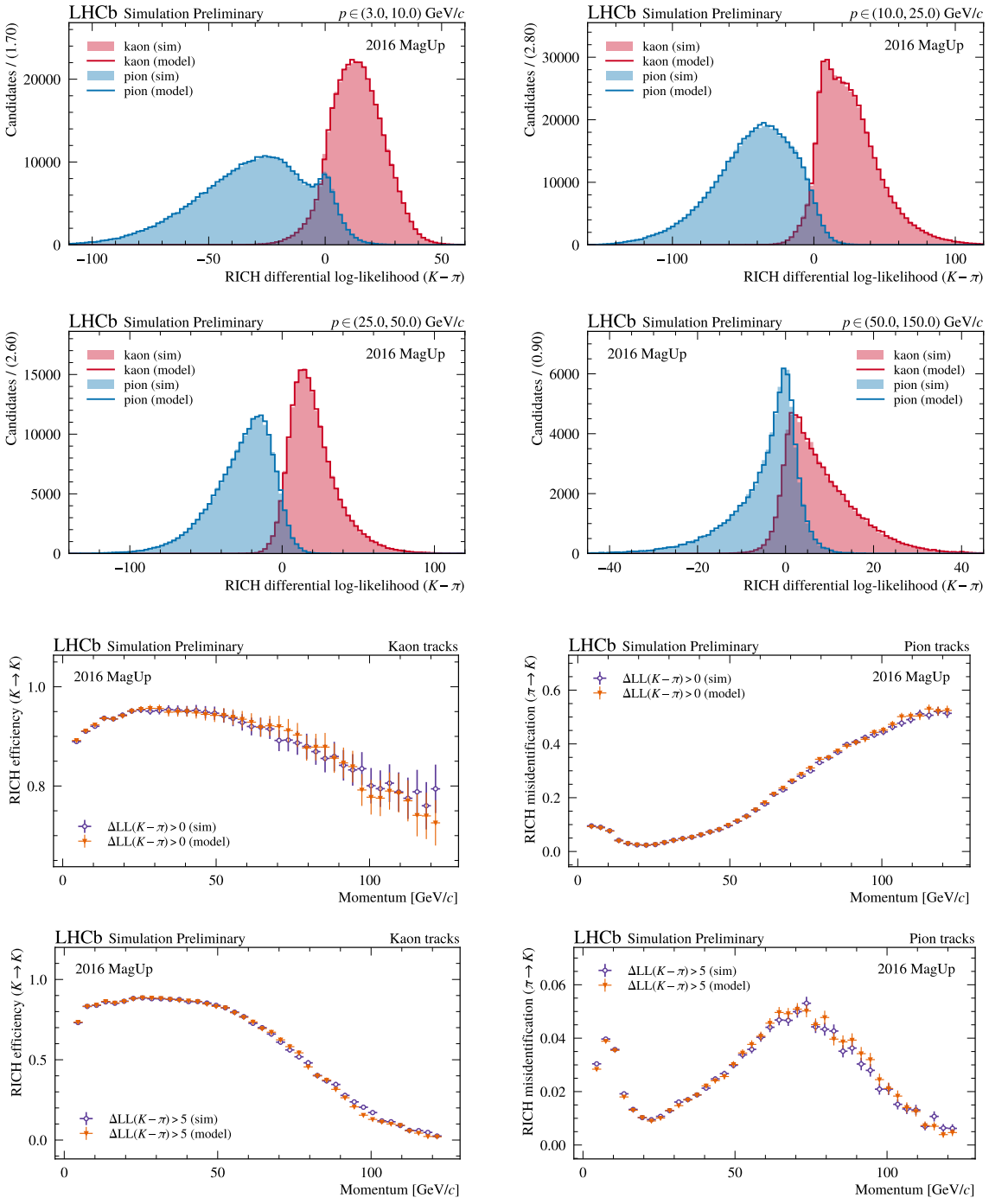


Figure 4.31: Validation plots of the GAN-based models trained to parameterize the *kaon-pion separation* offered by the LHCb RICH detectors. The distributions resulting from Detailed Simulation for the `RichDLLk` variable are represented as filled histograms for kaons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the RICH efficiency for the kaon identification is reported on the left plots using two selection cuts: `RichDLLk` > 0 and `RichDLLk` > 5. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to K$) as depicted on the right plots.

The RichDLLp distributions for protons and pions are depicted in Figure 4.32 in the usual four momentum bins. The LHCb RICH detectors excel in distinguishing protons from pions in a wide momentum range ($2 \div 100$ GeV/c). Also in this case, the two GANs (specialized for protons and pions) exhibit good performance in reproducing the discriminative power of the RICH system, either in terms of probability distributions, as demonstrated by the histograms in Figure 4.32, or in terms of efficiency plots for both the selection cuts of the DLL. The capability of the pion-specialized GAN to reproduce the peak at RichDLLp $= 0$ for $p < 25$ GeV/c is remarkable. Again, this is due to the construction design of the RICH2 detector that prevents the active identification of protons with $p \leq 17.7$ GeV/c [44]. The training procedure allows to correctly parameterize such effect, offering the analysts an accurate description of the RICH performance.

Finally, to show the effect of using auxiliary features during the discriminator training, Figure 4.33 reports the distributions of RichDLLpK for protons and kaons in four bins of momentum. By design, the RICH system exhibits an excellent proton-kaon separation in the momentum range ($20 \div 100$ GeV/c). Despite the DLL is not listed within the output features of the trained generators, the combination of the generated RichDLLp with RichDLLk allows to obtain very good results among the phase space, especially for $p \geq 10$ GeV/c. This is visible looking at the histogram plots in Figure 4.33 where the distributions induced by the two GAN models (specialized for protons and kaons) are in perfect agreement with what is expected from Detailed Simulation. The bin with $p < 10$ GeV/c presents some mismodeling effects due to the difficulty of reproducing a Dirac delta superimposed with poorly populated *resolution* phenomena at RichDLLpK $= 0$. Such peculiar behavior follows from the construction design of the RICH2 detector that limit the identification of either kaons or protons for $p \leq 9.3$ GeV/c and $p \leq 17.7$ GeV/c, respectively. This corresponds to having most of the likelihoods for the proton hypothesis equal to the ones for the kaon hypothesis in the momentum bin with $p < 10$ GeV/c, resulting in a DLL that peaks at zero. The parameterization of this phenomenon by using plain GANs is a non-trivial task since it requires that the generators learn to output identical values for both RichDLLp and RichDLLk when fed by the corresponding conditions. Using RichDLLpK as an auxiliary feature helps in accomplishing such a task, seeding physics constraints through the minimax game, thus allowing to accurately reproduce the efficiencies shown in Figure 4.33.

An alternative solution: Flow-based models

The continue development of deep generative models relying on ever-changing architectures and training strategies, opens the doors to alternative algorithms that find application in HEP and aim to compete with GANs for Fast Detector Simulation. Among these, the promising results recently achieved by the CMS Collaboration make emerge Flow-based models [67, 184] and encourage to investigate them also for the LHCb use cases.

A preliminary study aimed to assess the performance achieved by Flow-based models in parameterizing the high-level response of the RICH system is briefly discussed here. Differently from GANs that only enable the sampling from the learned underlying distribution, Normalizing Flows allows to explicitly compute the probability density functions by relying on a sequence of invertible and differentiable transformations, typically powered by neural networks [149]. However, the intrinsic non-linear nature of the neural network makes defining transformations with such properties non-trivial, and pushes to rely on

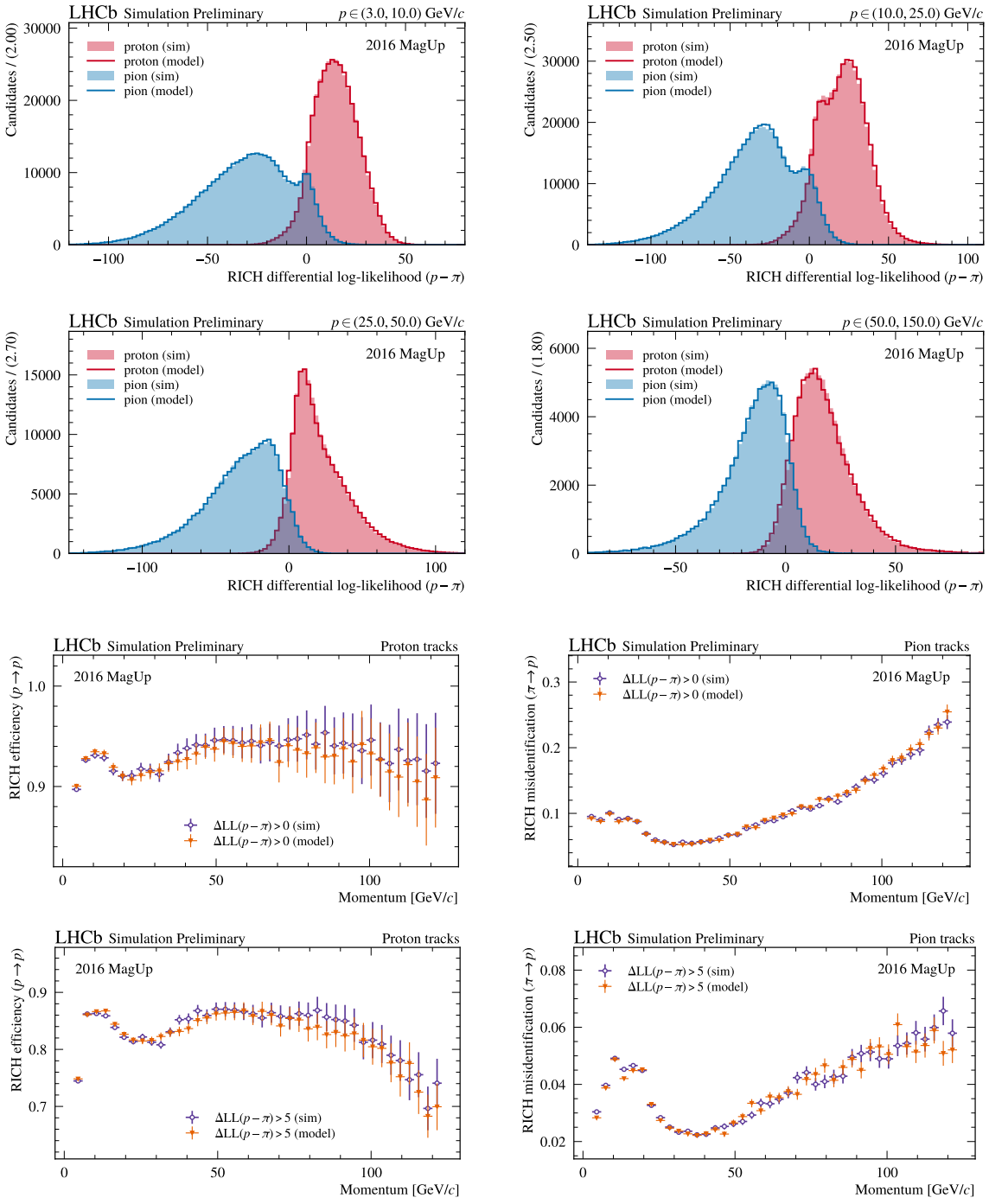


Figure 4.32: Validation plots of the GAN-based models trained to parameterize the *proton-pion separation* offered by the LHCb RICH detectors. The distributions resulting from Detailed Simulation for the `RichDLLp` variable are represented as filled histograms for protons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the RICH efficiency for the proton identification is reported on the left plots using two selection cuts: `RichDLLp` > 0 and `RichDLLp` > 5. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to p$) as depicted on the right plots.

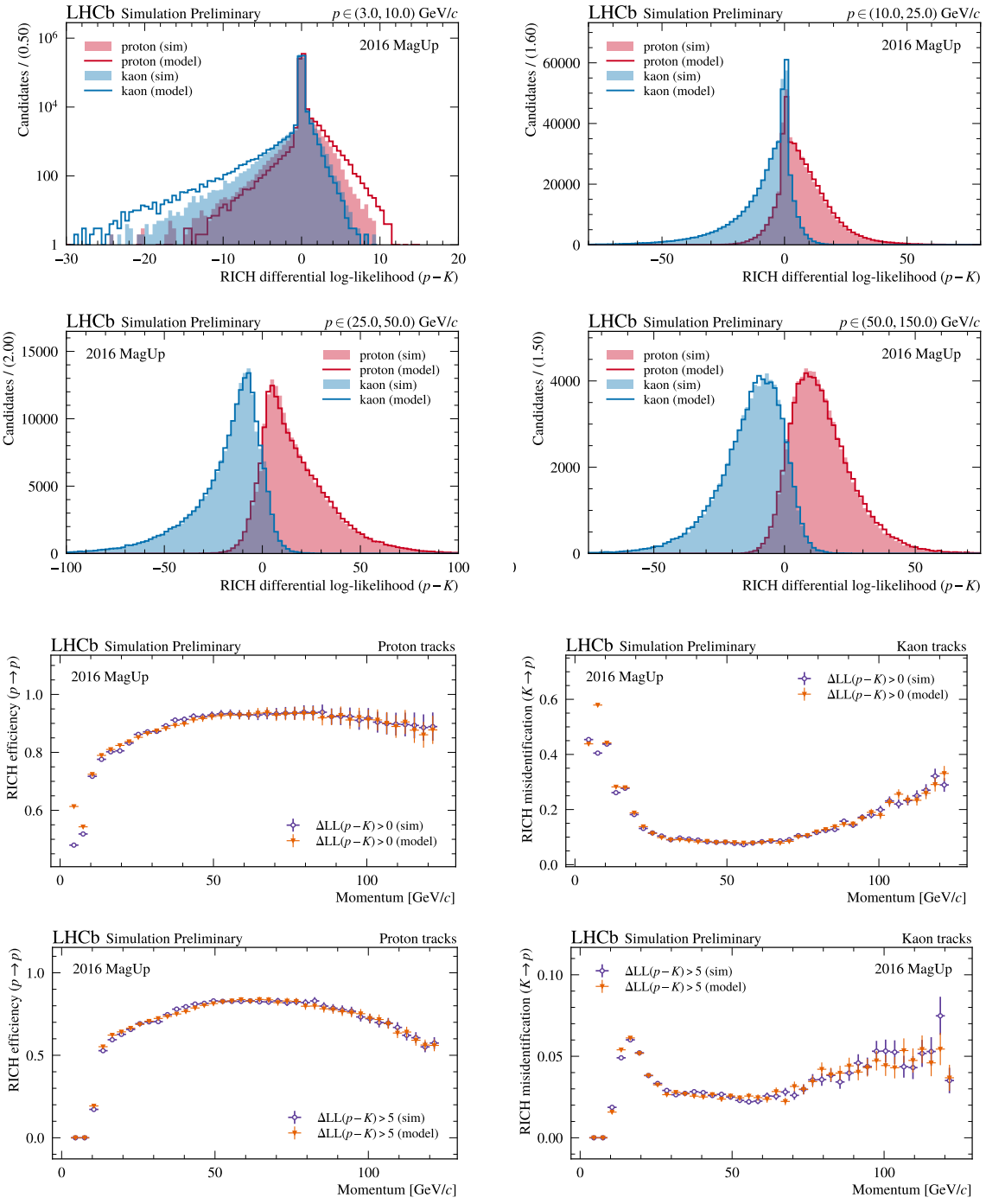


Figure 4.33: Validation plots of the GAN-based models trained to parameterize the *proton-kaon separation* offered by the LHCb RICH detectors. The distributions resulting from Detailed Simulation for the `RichDLLpK` variable are represented as filled histograms for protons (in red) and kaons (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the RICH efficiency for the proton identification is reported on the left plots using two selection cuts: `RichDLLpK` > 0 and `RichDLLpK` > 5. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($K \to p$) as depicted on the right plots.

dedicated architectures designed for a compromise between *expressivity* and *regularity*. In particular, for these preliminary studies, we make use of a *Masked Autoregressive Flows* (MAF), whose definition is depicted in Section 3.3.1, relying on affine functions to transform a 4-dimensional Gaussian distribution into the joint probability density function representing the distribution of the four target RICH DLLs. Following the implementation suggested in Ref. [154], each transformation can be easily implement to rely on a set of additional variables, hence enabling to compute conditional probability density functions. Following the example of the GANs, four specie-specialized Flow-based models are defined and designed to take as input condition the track kinematic information, the detector occupancy, and the charge of the generated particle. The MAF implementation relies on the `nflows` package [185], while the training was performed using PyTorch [186].

The preliminary results are depicted in Figure 4.34, where the ability to reproduce the kaon-proton separation is investigate both for GAN- and Flow-based models. In particular, the distributions of `RichDLLpK` for kaons (left) and protons (right) resulting from the two generative models are compared with what expected from Detailed Simulation in three bins of momentum. As already discussed, `RichDLLpK` is not part of the target variables of the trained generators that succeed in reproducing physically reasonable results relying on the auxiliary feedback provided by the discriminator training. Interistingly, neither the MAF-based models have access to the `RichDLLpK` variable, but succeed in reproducing it even without relying on any auxiliary process. In general, the performance exhibited by the two models are quite similar, despite the presence of minor mismodeling in the MAF case probably due to a non-optimal hyperparameter configuration. Despite these preliminary results are promising and encourage further studies in the future, we do not observe any clear evidence to stop relying on GANs for parameterizing the high-level response of the LHCb experiment, that would also require additional efforts to develop ad-hoc solutions to be integrated within the simulation software stack, as further discussed in Chapter 5.

4.3.3 MUON system

The LHCb MUON system disposes of five stations located at the end of the spectrometer for the selection of high-p_T muons at the trigger level and for the offline identification of muon candidates. The latter relies on the distribution of specialized likelihood functions only computed after the verification of the `isMuon` flag, a binary muon-identification criterion implemented via FPGA [42]. Given the mean square errors along the MUON stations between the detected hits and the linear extrapolation of the reconstructed track D^2 defined in Eq. (1.10), the likelihood for the muon hypothesis is defined as the cumulative of D^2 distribution obtained from true muon candidates. The likelihood for the non-muon hypothesis relies instead on the cumulative of D^2 distribution resulting from protons wrongly identified as muons by the `isMuon` criterion. Similarly to what is done by the rest of the PID detectors at LHCb, these two likelihoods are combined by forming a DLL discriminating variable called muDLL [42]:

$$\text{muDLL} = \log\left(\frac{\text{MUON likelihood for } \mu}{\text{MUON likelihood for } p}\right) \equiv \texttt{MuonMuLL} - \texttt{MuonBgLL} \tag{4.11}$$

Despite the physics processes used for the identification of muons being profoundly different from the ones exploited by the RICH detectors, also, in this case, we expect that

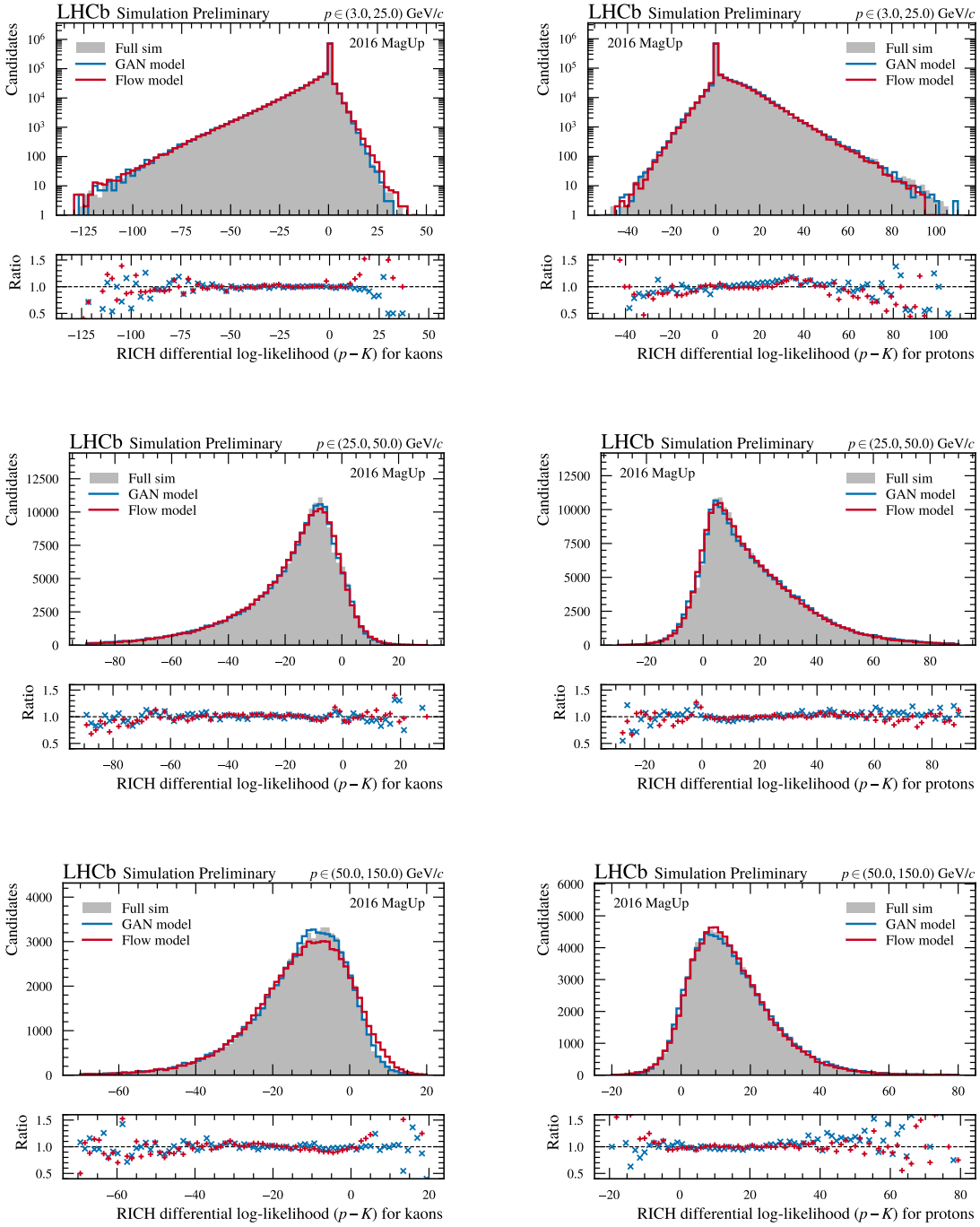


Figure 4.34: Comparison plots between GAN- and Flow-based models trained to reproduce the high-level response of the LHCb RICH system. The distribution of the `RichDLLpK` variable for kaons (left) and protons (right) as result from Detailed Simulation are represented as grey shaded histograms in three bins of momentum. The output of the trained models are reported by using solid line histograms, blue for the GANs and red for the Flows. The ratios between the modeled and reference histogram entries are also reported following the same color labeling.

the response of the MUON system depends on the momentum p and pseudorapidity η of traversing particles, and the total number of reconstructed tracks `nTracks`. To understand the reason behind these dependencies, we should consider that the track momentum is one of the information adopted by `isMuon` to make decisions for the muons identification. The efficiency of `isMuon` as well as the probability of misidentification are then strongly related to p values which may vary the D^2 distributions and the likelihoods. The scheme of the MUON system depicted in Figure 1.14 makes evident why the detector response depends also on the pseudorapidity of the traversing particles. Lastly, considering a low-occupancy event, it stands to reason that the latter is characterized by a clear signature for both `MuonMuLL` and `MuonBgLL` so that they can be effectively separated. On the contrary, in the case of a crowded event, the same likelihoods are affected by *resolution* effects that typically drop the global performance of the MUON system.

LAMARR relies on GANs to parameterize the high-level response of the LHCb MUON system. Such parameterizations are designed to take as input the same set of conditions used for the RICH model discussed in Section 4.3.3. The MUON GANs were trained to learn the conditional probability distributions of the following likelihoods [1, 180]:

- the MUON log-likelihood for the muon hypothesis `MuonMuLL`;
- the MUON log-likelihood for the non-muon hypothesis `MuonBgLL`.

As for the RICH case, for parameterizing the MUON system LAMARR relies on four different specie-specialized models, trained by using as many pure samples of muons, pions, kaons, and protons once they have passed the `isMuon` criterion.

Model design and training

The selection of the best hyperparameters and training strategies to build an effective parameterization of the MUON systems relies on Bayesian optimization campaigns [2] that have required more than $\mathcal{O}(10^3)$ GPU hours. Just like for the RICH models, the optimization procedure was driven by the minimization of a *static* metric measuring the distance between the reference distributions resulting from Detailed Simulation, and the distributions generated by four specie-specialized GANs. The metric chosen for these studies was the KS-distance that, computed in bins of momentum, pseudorapidity, and `nTracks`, aims to minimize the mismodeling errors introduced by the GANs in the whole phase space $(p, \eta, \texttt{nTracks})$.

Besides the BCE-based GAN and LSGAN [143] algorithms which have already proven their effectiveness with the parameterization of the RICH response, the optimization studies for the MUON system highlighted also the potentialities of other GAN *flavours*. The injection of Gaussian noise [144] within the discriminator reveals the capability of the original GAN design proposed by Ian Goodfellow and others [134] to generate faithful synthetic data. On the other hand, the use of Lipschitz-constrained discriminators [145–147] avoids relying on customary techniques to stabilize BCE-based training. In particular, the likelihoods of the MUON system seem to be properly reproduced by using the *adversarial Lipschitz penalty* (ALP) algorithm discussed in Section 3.2.2 to constraint explicitly the Lipschitzianity of the discriminator. Independently of the GAN *flavour*, these adversarial systems rely on 10-layer neural networks for both the generator and the discriminator. As usual, each hidden layer comprises 128 neurons and Leaky ReLUs as

activation functions. The generators can count on 1-D residual blocks [111] to ensure a smooth training procedure. On the contrary, the use of skip connections for the discriminators is discouraged when using LSGANs or the original GAN design. All the GAN algorithms, the regularization strategies, and the network architectures mentioned above can be easily implemented by relying on the PIDGAN APIs [6].

We expect that the response of the MUON system depends on the kinematics of the traversing particles and the detector occupancy, hence such information x is used to conditionate the output of the generator in combination with latent vectors z, which are sampled from a 64-dimensional normal distribution $\mathcal{N}_{64}(\mathbf{0}, \mathbf{1})$. The last layer of the generators $G(z|x)$ counts two neurons with no activation functions that aim to parameterize the likelihoods `MuonMuLL` and `MuonBgLL`. However, the variable used for analyses at LHCb is muDLL, which results by subtraction from the modeled likelihoods. Hence, to ensure that the outputs of the generators are physically reasonable, all the discriminators make use of muDLL as an *auxiliary feature* [177] together with the conditions x and the likelihoods y. This information ends up with a single neuron that is used by the discriminator $D(y, y_{\text{aux}}|x)$ to separate the reference sample from the synthetic one. Table 4.7 reports the complete list of hyperparameters used to build and train the specie-specialized models of the MUON system, which include label smoothing, Lipschitz regularization, and learning rate scheduling.

For training and validating the GAN-based models of the MUON system, a dataset of $\mathcal{O}(4 \times 10^6)$ particles passing the `isMuon` criterion was produced through Detailed Simulation. A fraction of 50% of the dataset was used for training, while another 10% was retained to monitor any signs of overtraining. Finally, the remaining 40% of the dataset was preserved for validation studies. The learning and metric curves of the MUON GANs are depicted in Figure 4.35. The evolution of the competition between the generator and discriminator networks during the training is shown on the left plots for the various GAN *flavours* adopted. Different algorithms have different metrics measuring the proximity to the Nash equilibrium. In particular, when discriminators learn probabilities, the metric chosen is the Jensen-Shannon divergence, while when the discriminators do have not any output activation functions, the Wasserstein distance defined in (3.16) is preferred. The evolution during the training of such metric scores is reported on the right plots of Figure 4.35.

Validation studies

As done for the RICH parameterization discussed in Section 4.3.2, the validation studies of the MUON models aim to test the ability of the trained GANs to reproduce faithfully the discrimination performance of the MUON system, in particular in distinguishing muons from charged hadrons. To this end, Figures 4.36, 4.37, and 4.38 report the muDLL distributions resulting from Detailed Simulation for muons versus pions, kaons, and protons, respectively. The reference distributions are then compared with the ones produced by the trained generators. To further test the performance of the models, the selection efficiency and the misidentification probability plots are also reported, showing the capability of GANs to fully parameterize the MUON system. It should be pointed out that muDLL is not listed within the output features of the trained generators, but instead, it results from the combination of the two target likelihoods, namely `MuonMuLL` and `MuonBgLL`. To help the generators reproduce physically reasonable outputs, a notion

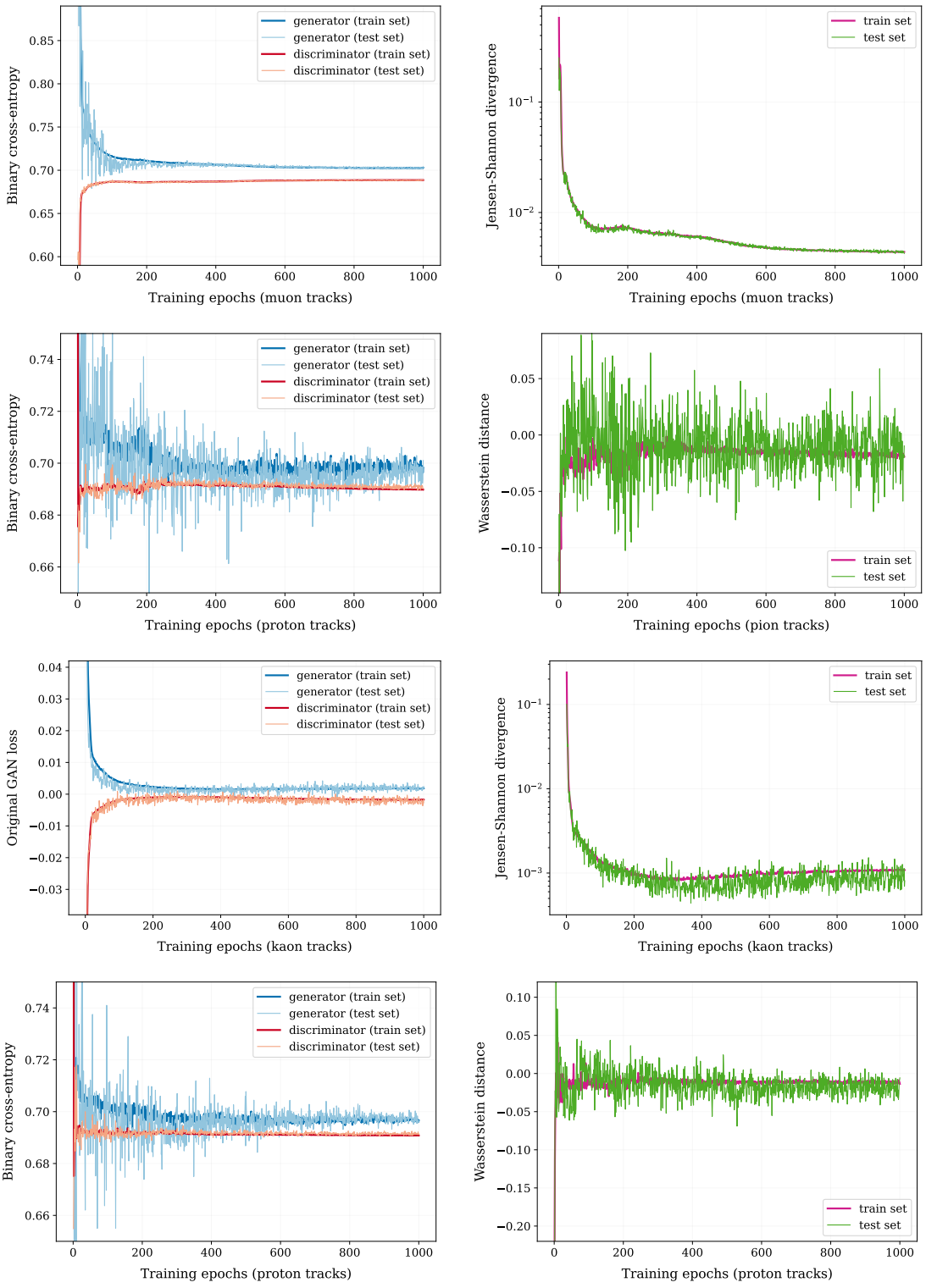


Figure 4.35: Learning and metric curves of the four GAN-based models trained to parameterize the LHCb MUON response when traversed by muons, pions, kaons, and protons (in top-down order). The competition between the *generator* and *discriminator* networks of the various GANs is depicted on the left plots. The evolution during the training of the metric adopted for monitoring the proximity to the Nash equilibrium in each of the GAN *flavours* is reported on the right plots.

of the actual distribution of muDLL is seeded through the minimax game by using it as an auxiliary feature within the discriminator training.

The muDLL distributions for muons and pions are depicted in Figure 4.36 in four bins of momentum. Despite the high overlapping between the two distributions in all the bins, the MUON system offers a very good muon-pion separation that results from the combination of muDLL with the preliminary filtering achieved by the `isMuon` criterion. Despite the non-trivial shape of the DLL distribution, the two GANs (specialized for

	Generator	*Discriminator*
skip connections [111]	✔	✔/✔/✖/✔
latent space dim [134]	64	-
n auxiliary features [177]	-	1
input shape	(None, 68)	(None, 7)
input preprocessing	✔	✔
n hidden layers	10	10
n hidden neurons	128	128
hidden activation functions	Leaky ReLU	Leaky ReLU
output activation function	linear	sigmoid linear sigmoid linear
output shape	(None, 2)	(None, 1)
output preprocessing	✔	✖
optimizer	RMSprop	RMSprop
learning rate	$\sim (3/6/5/7) \times 10^{-4}$	$\sim (5/3/3/4) \times 10^{-4}$
loss function	BCE-based loss (4.7) BCE-based loss (4.7) GAN original loss (3.2) BCE-based loss (4.7)	BCE-based loss (4.6) BCE-based loss (4.6) GAN original loss (3.2) BCE-based loss (4.6)
BCE from logits	False/True/-/True	False/True/-/True
BCE label smoothing	0.05/✖/-/✖	0.05/✖/-/✖
Gaussian noise stddev [144]	0.02/-/0.05/-	0.02/-/0.05/-
Lipschitz regularization [145]	-/ALP/-/ALP	-/ALP/-/ALP
learning rate scheduling	ExpDecay	ExpDecay
scheduling decay rate	0.1	0.1
scheduling decay steps	$(21/13/18/15) \times 10^4$	$(15/25/23/20) \times 10^4$
batch-size	$(22/3/7/3) \times 10^3$	$(22/3/7/3) \times 10^3$
batches per epoch	~ 500	~ 500
n epochs	1000	1000

Table 4.7: Hyperparameters of the GAN-based model for the MUON system. When multiple hyperparameter values are reported, they refer to the different settings adopted for the definition and training of the models for muons, pions, kaons, and protons, respectively.

muons and pions) succeed in parameterizing the expected response of the MUON system, as demonstrated by the histogram plots. To further investigate the performance of the trained models, Figure 4.36 reports also the efficiency plots obtained by applying two cuts to the muDLL variable. Also, in this case, the GANs exhibit a very good ability in reproducing the discrimination performance expected from the MUON system.

Figures 4.37 and 4.38 report the muon-kaon and muon-proton separations exhibited by the muDLL variable in four bins of momentum. The GANs specialized for kaons and protons succeed in parameterizing the MUON high-level response, showing a very good agreement with Detailed Simulation either within the histograms or by comparing the misidentification probability plots.

An alternative solution: Flow-based models

Following what done for the RICH parameterization, also the high-level response of the MUON system is employed as laboratory to study the performance exhibited by Flow-based models on LHCb-tailored use-cases. As previously discussed, we aim to explicitly obtain the probability density function representing the distribution of the MUON likelihoods by relying on MAF-based models powered by affine transformations [149]. The usual input conditions x are used to conditionate the bijective transformations, and hence the flow, in order to obtain a proper description for the target density, representing the distribution of `MuonMuLL` and `MuonBgLL`. Similarly to what done for GANs, four specie-specialized Flow-based models were prepared and trained on data sample containing only particles satisfying the `isMuon` criterion. Also in this case, the MAF implementation relies on the `nflows` package [185], while the training was delivered by relying on PyTorch [186] as back-end.

The preliminary results are reported in Figure 4.39, where the distributions of the muDLL for protons (left) and muons (right) are investigated both for GAN- and Flow-based models. What results from Detailed Simulation is shown through grey shaded histograms in three bins of momentum. The distributions reproduced by GANs (blue) and MAFs (red) are superimposed by using solid line histograms. Also in this case the variable chosen for the performance metric, muDLL, is not part of the target features either for GANs nor Flows. However, if the GANs take advantage from the auxiliary features employed by the discriminator for reproducing the expected distributions, Flows struggle in correctly parameterizing the non-regular shape of the muDLL distributions, that lead to non-negligible mismodeling effects. Despite the obtained results required to be further investigated, they push to postpone in the future similar studies, confirming to rely on GANs for the construction of an end-to-end pipeline of models aimed at parameterizing the high-level response of the LHCb detector.

4.3.4 `isMuon` criterion

To reduce the misidentification of hadrons as muons at the percentage level, LHCb makes use of a dedicated FPGA-based strategy, called `isMuon`, which implements a loose boolean selection of muon candidates. The latter provides high efficiency and is based on the penetration of muons through the calorimeters and the iron filters of the MUON system. The higher momentum a muon has, the greater the expected number of stations traversed by such muon. Hence, requiring to find hits around the track extrapolation in the stations

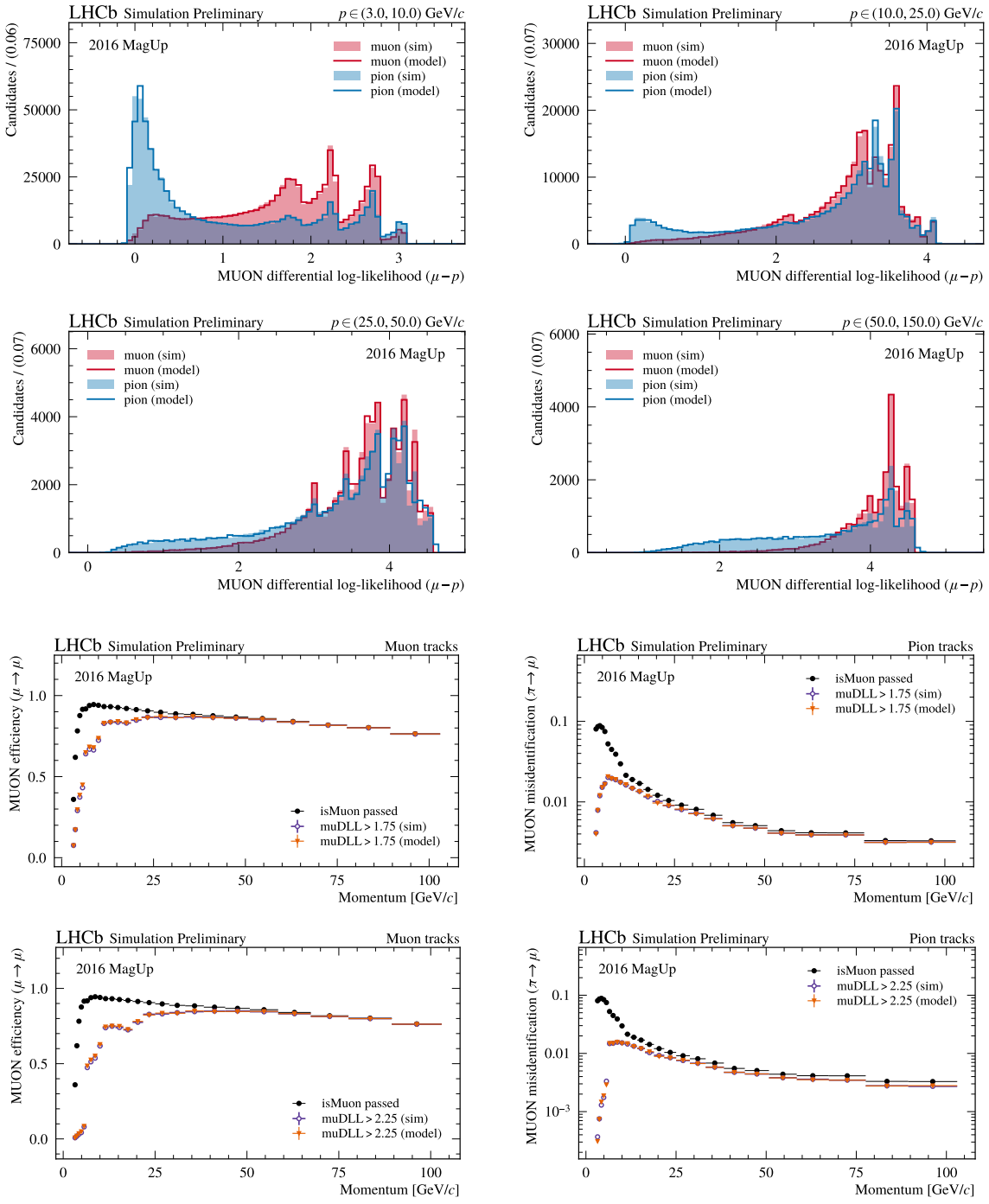


Figure 4.36: Validation plots of the GAN-based models trained to parameterize the *muon-pion separation* offered by the LHCb MUON detectors. The distributions resulting from Detailed Simulation for the muDLL variable are represented as filled histograms for muons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the efficiency of the muDLL criterion, applied on top of the `isMuon` requirement, is reported on the left plots using two selection cuts: muDLL > 1.75 and muDLL > 2.25. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to \mu$) as depicted on the right plots.

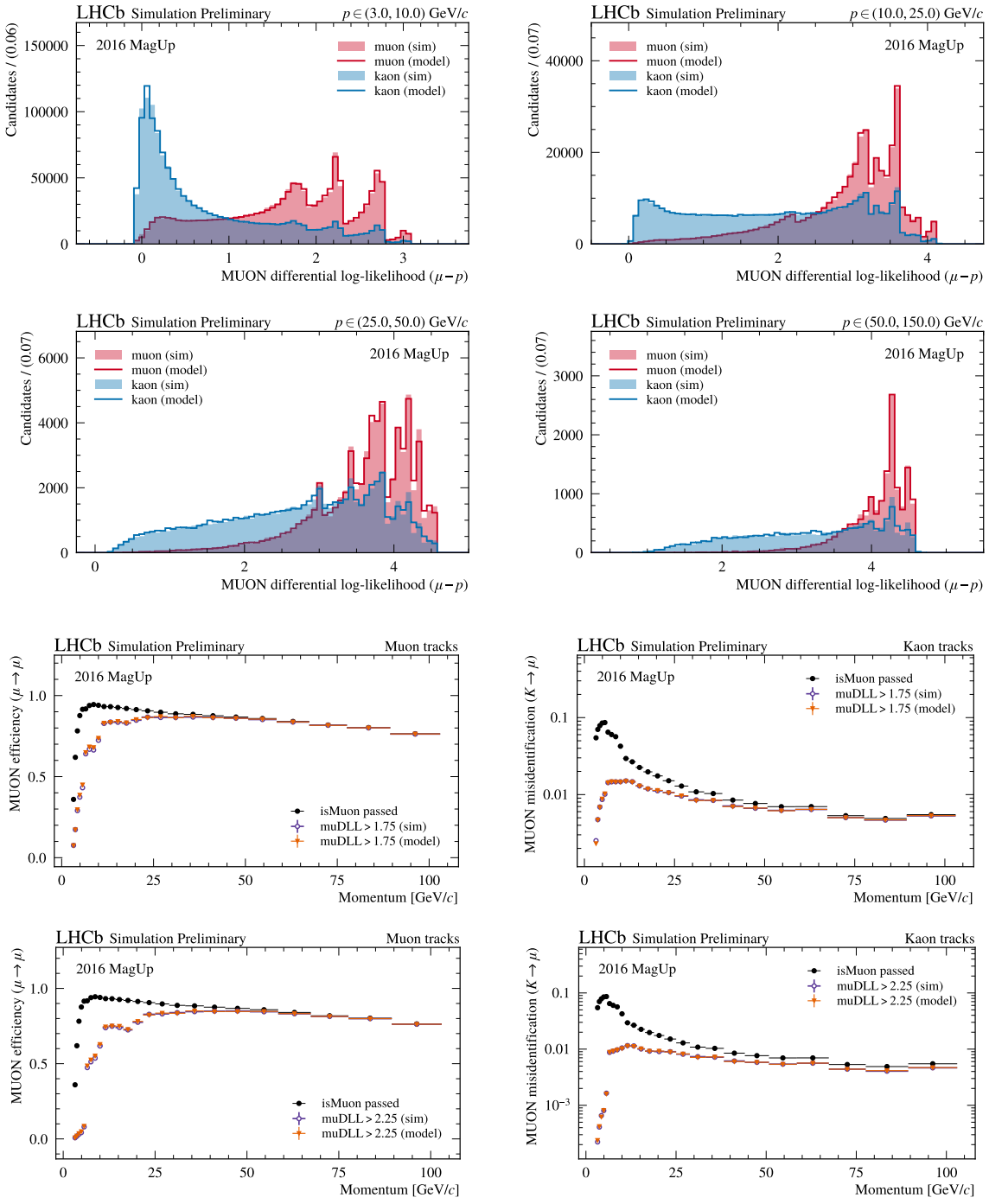


Figure 4.37: Validation plots of the GAN-based models trained to parameterize the *muon-kaon separation* offered by the LHCb MUON detectors. The distributions resulting from Detailed Simulation for the muDLL variable are represented as filled histograms for muons (in red) and kaons (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the efficiency of the muDLL criterion, applied on top of the `isMuon` requirement, is reported on the left plots using two selection cuts: muDLL > 1.75 and muDLL > 2.25. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($K \to \mu$) as depicted on the right plots.

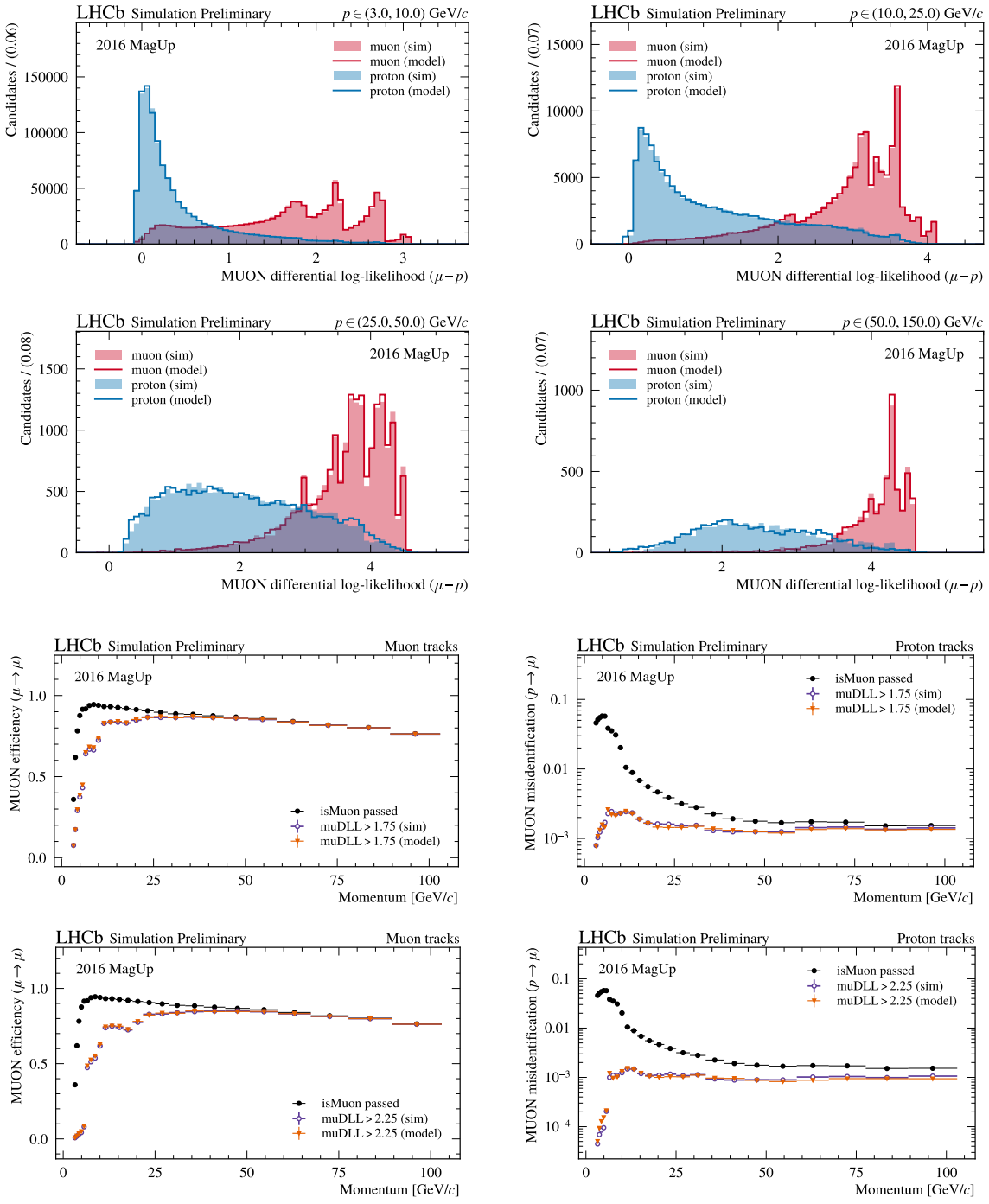


Figure 4.38: Validation plots of the GAN-based models trained to parameterize the *muon-proton separation* offered by the LHCb MUON detectors. The distributions resulting from Detailed Simulation for the muDLL variable are represented as filled histograms for muons (in red) and protons (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the efficiency of the muDLL criterion, applied on top of the `isMuon` requirement, is reported on the left plots using two selection cuts: muDLL > 1.75 and muDLL > 2.25. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($p \to \mu$) as depicted on the right plots.

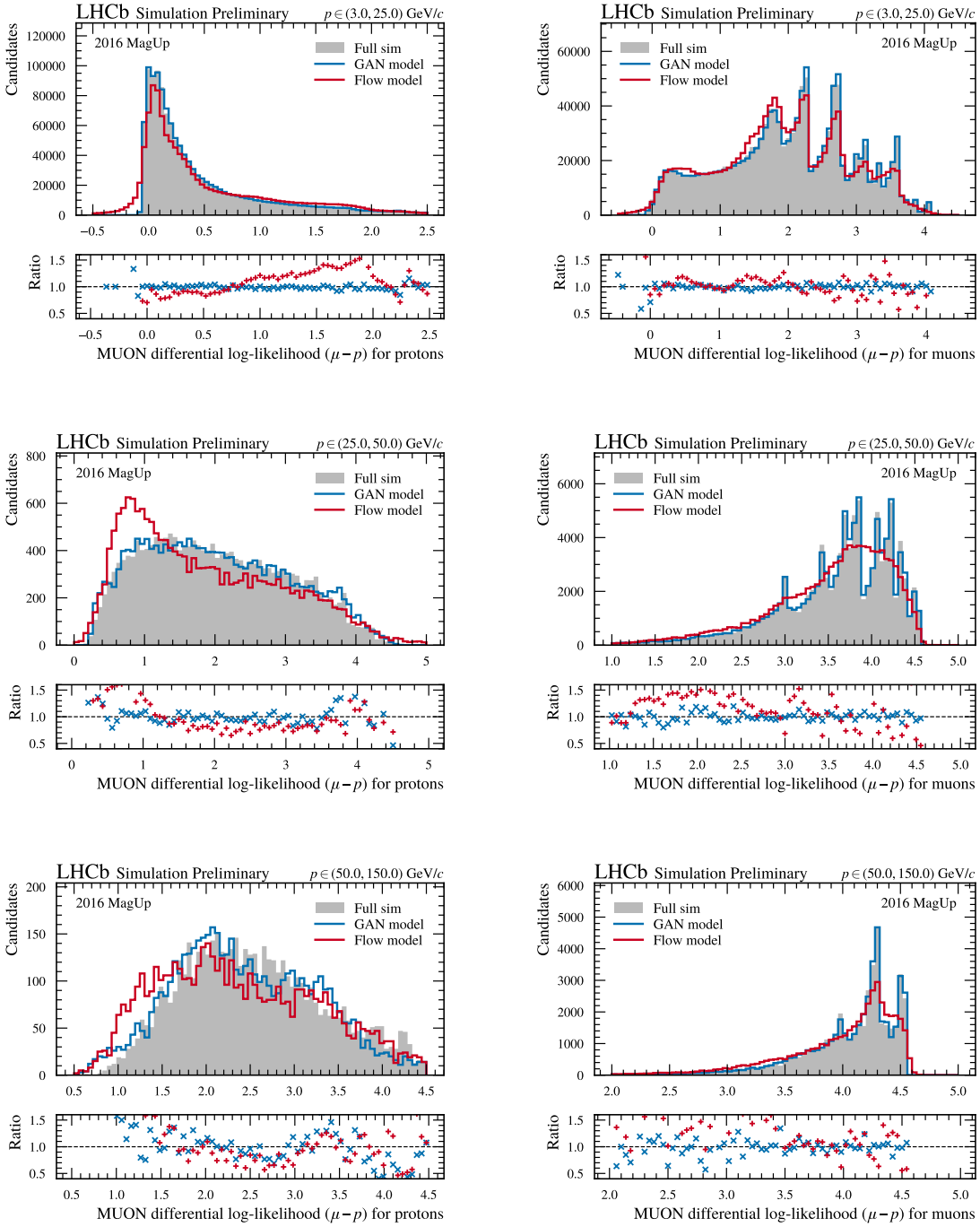


Figure 4.39: Comparison plots between GAN- and Flow-based models trained to reproduce the high-level response of the LHCb MUON system. The distribution of the muDLL variable for protons (left) and muons (right) as result from Detailed Simulation are represented as grey shaded histograms in three bins of momentum. The output of the trained models are reported by using solid line histograms, blue for the GANs and red for the Flows. The ratios between the modeled and reference histogram entries are also reported following the same color labeling.

indicated in Table 1.2, we can derive a first rough estimation of the muon identification [47].

The current version of LAMARR relies on four specie-specialized neural networks for parameterizing the `isMuon` flag when the MUON system is traversed by muons, pions, kaons, or protons. Similarly to what is discussed in Section 4.3.3 for the MUON system, we expect that the output of the `isMuon` criterion depends on the momentum p and pseudorapidity η of traversing particles, and the total number of reconstructed tracks `nTracks`. Hence, four different neural networks were trained to predict the fraction of particles that fulfill the `isMuon` rules based on the following set of features:

- the reconstructed momentum p of the traversing particles;
- the reconstructed pseudorapidity η of the traversing particles;
- the number of reconstructed tracks `nTracks`;
- the charge q of the traversing particles.

To train these neural networks, we relied on the same dataset of $\mathcal{O}(0.5 \times 10^9)$ particles used for training the RICH models, after having substituted the DLLs with the `isMuon` flag. The training of such NN-based approximators aims to predict the fraction of particles that pass the `isMuon` criterion by performing a *binary classification task*. According to this view, the `isMuon` models are formally equivalent to the parameterization of the LHCb geometrical acceptance discussed in Section 4.2.2.

Model design and training

Each of the four neural networks used to parameterize the output of the `isMuon` criterion is constructed with a set of 10 fully connected layers with ReLU activation functions. A final dense layer with a single neuron and a sigmoid activation function follows. To mitigate the gradient vanishing problem, all the NNs are equipped with skip connections [111]. The training procedure was driven by the minimization of the BCE using Adam as optimizer. Similarly to what was done for the acceptance model, the training procedure was split into two phases. At the beginning of the training procedure, the BCE was used with label smoothing set to 0.05, and Adam initialized with a learning rate of 0.05. During the following 200 epochs, such learning rate was exponentially decreased up to 10^{-4}. Then, fixed the learning rate to the last scheduled value and restored to zero the label smoothing, the neural networks were fine-tuned for 100 more epochs. The complete list of hyperparameters used to define and train the four specie-specialized models parameterizing the `isMuon` criterion is reported in Table 4.8. Since the referee implementations provided by PIDGAN [6] can also be trained as standalone, we can rely on them to train and fine-tune the `isMuon` models.

The training dataset was arranged by selecting a fraction of 50% of the $\mathcal{O}(0.5 \times 10^9)$ particles available from Detailed Simulation. As usual, another 10% portion was retained to detect any overtraining effects, while the remaining 40% was preserved for validation studies. Figure 4.40 shows the evolution of the ROC AUC score during the training, which illustrates the performance achieved by the neural networks in accomplishing the binary classification task.

	training	*fine-tuning*
skip connections [111]	✔	✔
input shape	(None, 4)	(None, 4)
input preprocessing	✔	✔
n hidden layers	10	10
n hidden neurons	128	128
hidden activation functions	ReLU	ReLU
hidden kernel regularizer	L2	L2
L2 regularization factor [169]	5×10^{-5}	5×10^{-5}
output activation function	sigmoid	sigmoid
output shape	(None, 1)	(None, 1)
optimizer	Adam	Adam
learning rate	0.05	1×10^{-4}
loss function	BCE	BCE
BCE label smoothing	0.05	✘
learning rate scheduling	ExpDecay	✘
scheduling decay rate	0.1	-
scheduling decay steps	50000	-
batch-size	20000	20000
batches per epoch	~ 600	~ 600
n epochs	200	100

Table 4.8: Hyperparameters of the neural networks used to parameterize the `isMuon` criterion.

Validation studies

The ability of the trained neural networks to parameterize the `isMuon` criterion is assessed by relying on detailed simulated samples that count 5 million of instances for each of the specializing particle species. Aiming to predict the fraction of particles that satisfy the `isMuon` rules [47], the neural networks were trained to output the probability that a particle (i.e., muon, pion, kaon, or proton), with a certain combination of momentum and pseudorapidity and that traverses LHCb in different detector conditions (such as `nTracks`), passes such binary criterion. Hence, to assess the performance of the trained models, we compare the kinematic distributions of the particles that verify `isMuon` with the ones obtained by weighting the kinematic distributions of the *reconstructed* particles with the predicted probabilities.

The validation plots of the `isMuon` model for muons as a function of the momentum p in four bins of pseudorapidity η is depicted in Figure 4.41. The momentum distributions of the reconstructed true muons are shown in grey, while the ones that pass the binary criterion are highlighted by using blue-hatched histograms. The results of the trained muon-specialized neural network are superimposed with a red solid-line. Figures 4.42, 4.43, and 4.44 show the same comparisons for pions, kaons, and protons, respectively. In all the cases, the neural networks succeed in reproducing the fraction of particles that verify `isMuon` as expected from Detailed Simulation, even on the tails of the kinematic distributions.

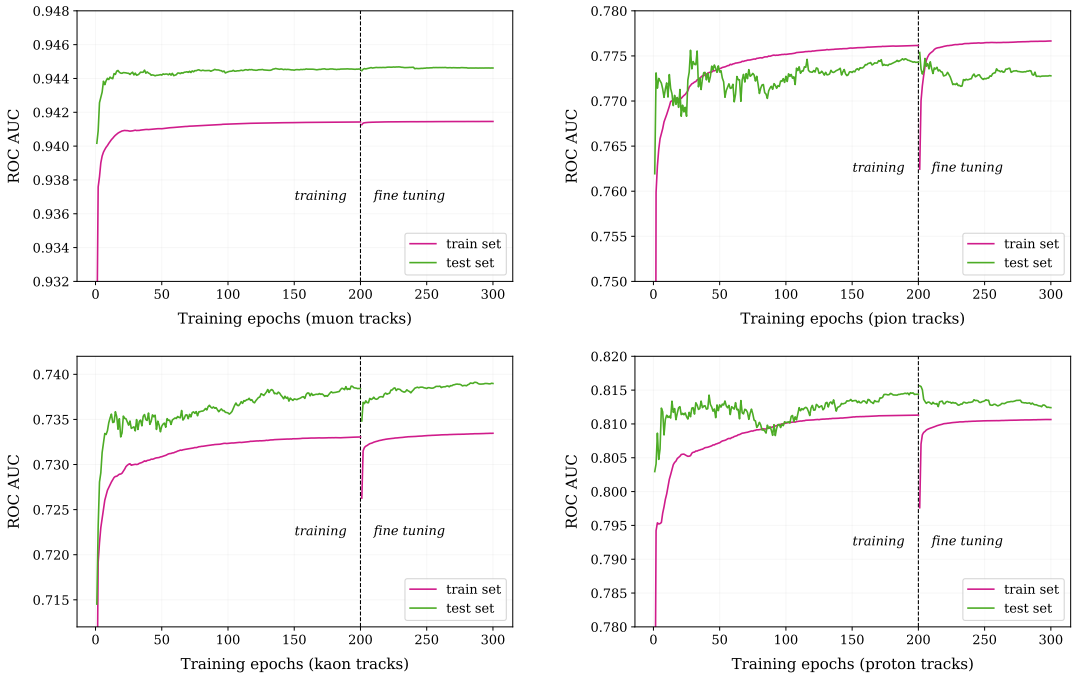


Figure 4.40: Metric curves reporting the *ROC AUC score* of four L2-regularized neural networks [169] trained to model the response of the `isMuon` criterion when the LHCb MUON system is traverse by muons, pions, kaons, or protons. The training procedure consisted of 200 epochs where label smoothing and learning rate scheduling are used. A fine-tuning phase followed counting 100 epochs with label smoothing and learning rate scheduling strategies disabled.

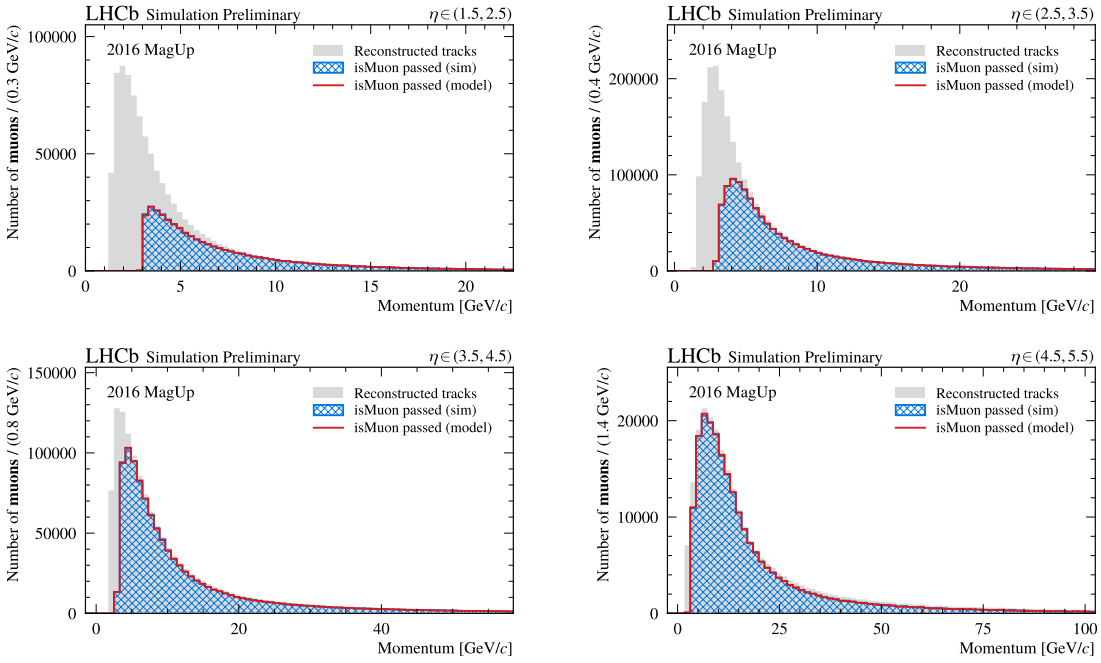


Figure 4.41: Validation plots of the `isMuon` model for *muons* as a function of the momentum p in four bins of pseudorapidity η. The kinematics distributions of the reconstructed muons are represented as light grey shaded histograms. The distributions of muons that pass the `isMuon` criterion are shown through blue-hatched histogram, while their parameterization as modeled by a deep neural network is superimposed using red solid-line histograms.

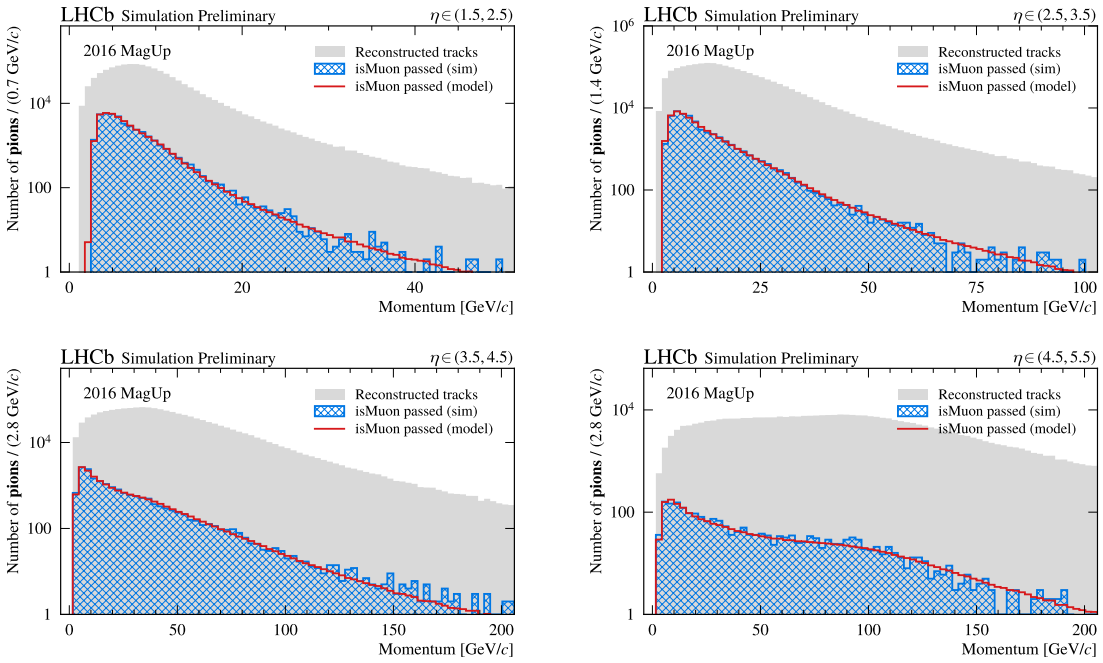


Figure 4.42: Validation plots of the `isMuon` model for *pions* as a function of the momentum p in four bins of pseudorapidity η. The kinematics distributions of the reconstructed pions are represented as light grey shaded histograms. The distributions of pions that pass the `isMuon` criterion are shown through blue-hatched histogram, while their parameterization as modeled by a deep neural network is superimposed using red solid-line histograms.

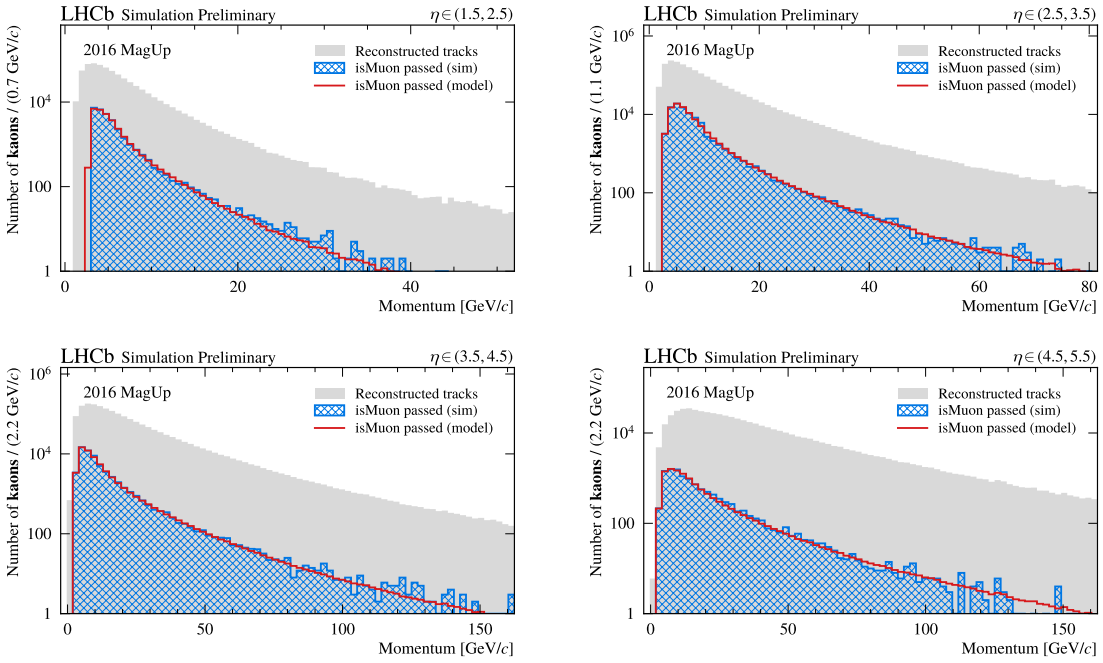


Figure 4.43: Validation plots of the `isMuon` model for *kaons* as a function of the momentum p in four bins of pseudorapidity η. The kinematics distributions of the reconstructed kaons are represented as light grey shaded histograms. The distributions of kaons that pass the `isMuon` criterion are shown through blue-hatched histogram, while their parameterization as modeled by a deep neural network is superimposed using red solid-line histograms.

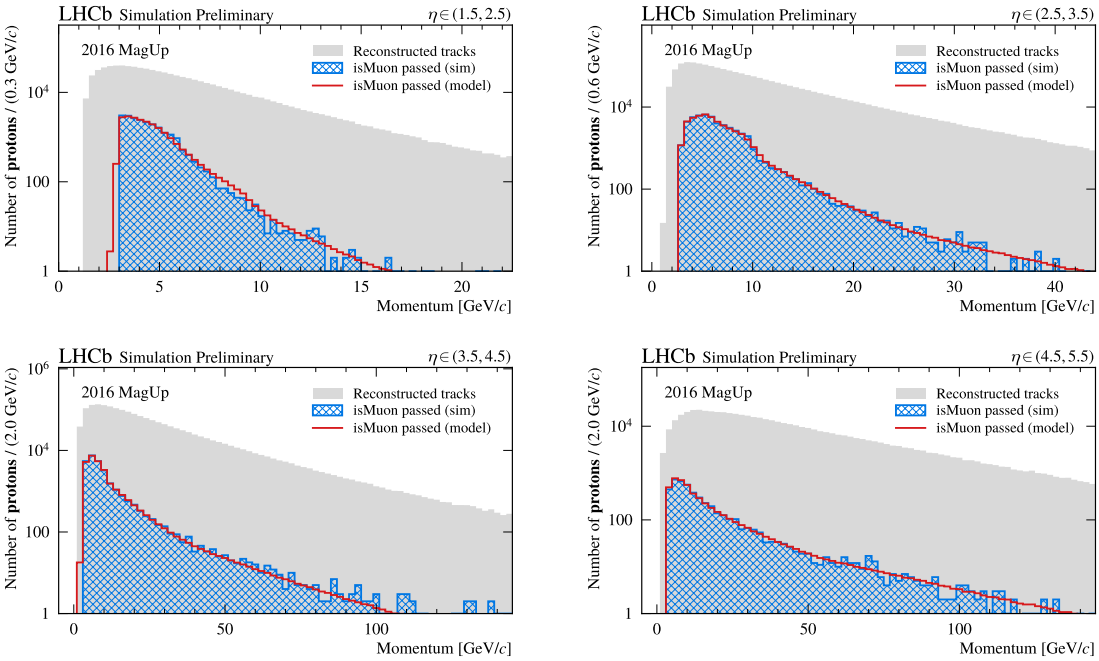


Figure 4.44: Validation plots of the `isMuon` model for *protons* as a function of the momentum p in four bins of pseudorapidity η. The kinematics distributions of the reconstructed protons are represented as light grey shaded histograms. The distributions of protons that pass the `isMuon` criterion are shown through blue-hatched histogram, while their parameterization as modeled by a deep neural network is superimposed using red solid-line histograms.

To further investigate their performance, the parameterizations are also tested in terms of efficiency. In particular, Figure 4.45 reports the `isMuon` efficiency and misidentification probabilities as a function of the momentum. Since we expect that the output of `isMuon` strongly depends on the transverse momentum p_T [47], the efficiencies shown in Figure 4.45 are computed for particles with $p_T \in (1.5, 3.0)$ GeV/c. Also in this case the performance exhibited by the trained models is very good with efficiencies that match the ones drawn from simulations within the statistical errors in almost all the momentum bins. Similar results are shown in Figure 4.46 where the same study is repeated for particles traversing the spectrometer with an occupancy described by `nTracks` $\in (100, 250)$.

4.3.5 Global response of the PID system

Despite the experimental strategies adopted by each of the LHCb PID detectors being very different, each of them allows the computation of a likelihood ratio between two particle hypotheses for each reconstructed track. The RICH detectors provide likelihoods that distinguish muons and hadrons from charged pions by comparing the size of the Cherenkov rings against a specific mass hypothesis. The Calorimeter system provides likelihoods that identify electrons versus hadrons based on the shape of the associated cluster and the energy deposited on each of the calorimeter sub-detectors (i.e., PS, ECAL, and HCAL). Finally, the MUON system provides likelihoods that separate muons from hadrons by exploiting the hits left within five stations located at the end of the spectrometer. To improve the PID performance, the likelihood ratios of the three detector systems are linearly added to obtain *combined differential log-likelihoods* (CombDLLs), which are

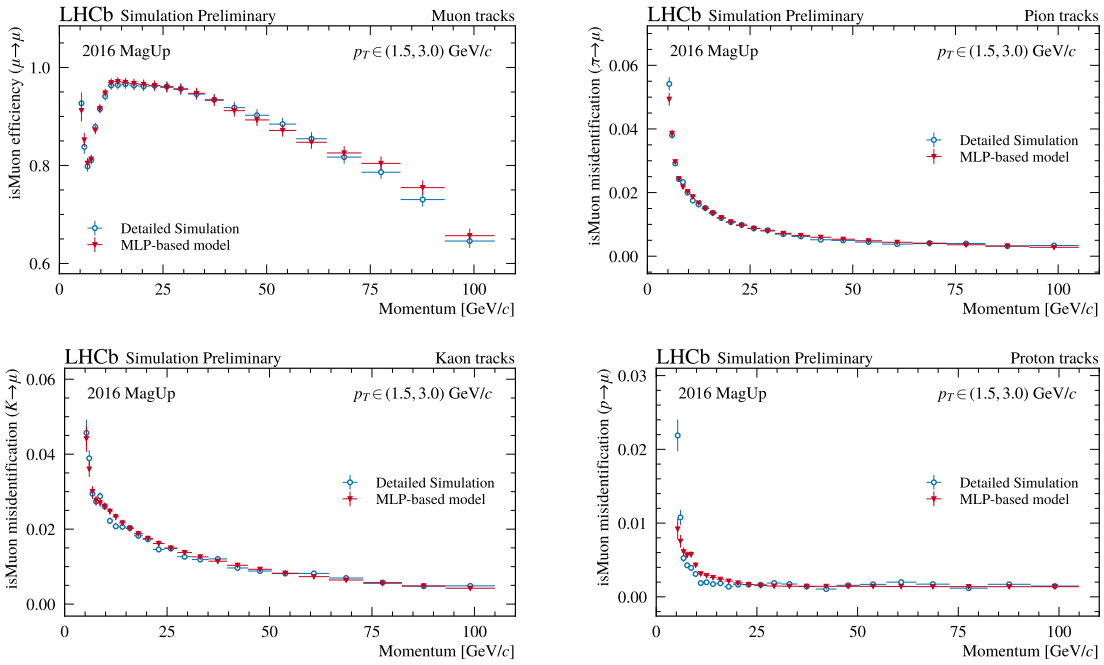


Figure 4.45: `isMuon` *efficiency* and *misidentification probabilities* as a function of the momentum in a specific bin of the transverse momentum p_T. The efficiencies resulting from detailed simulated muons (top left), pions (top right), kaons (bottom left), and protons (bottom left) are reported using blue circle-markers. What results from a deep neural network trained to parameterize the `isMuon` criterion is depicted through red triangular-markers.

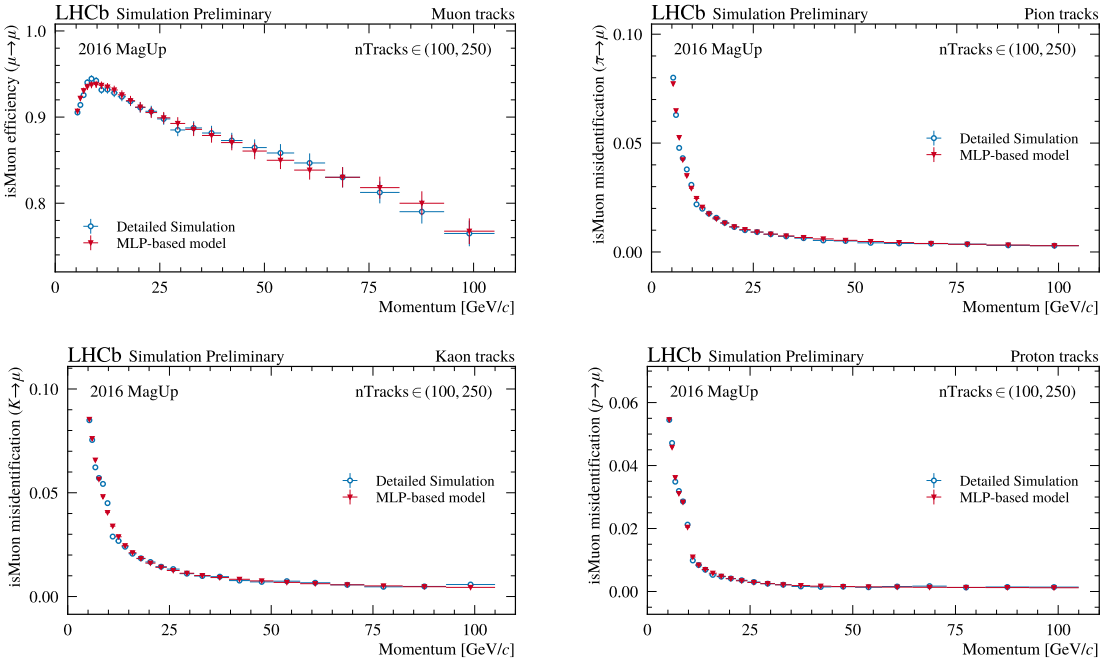


Figure 4.46: `isMuon` *efficiency* and *misidentification probabilities* as a function of the momentum in a specific bin of the number of reconstructed tracks `nTracks`. The efficiencies resulting from detailed simulated muons (top left), pions (top right), kaons (bottom left), and protons (bottom left) are reported using blue circle-markers. What results from a deep neural network trained to parameterize the `isMuon` criterion is depicted through red triangular-markers.

typically used to define the selection criteria for data analyses [34, 49]. Since the most abundant species produced at hadron colliders are pions, it is customary to define such CombDLLs with respect to the pion hypothesis. The second approach used at LHCb for selecting particle candidates relies on a multivariate classifiers, called ANNPIDs designed to combine the aforementioned likelihood ratios with information coming from the Tracking system, and an additional one provided by the PID detectors not entering the likelihood computation, as listed in Table 1.3. These classifiers are structured as MLPs with a single hidden layer and a single output neuron with a sigmoid activation function [49]. Since the ANNPID classifiers are trained to predict probabilities, the latters are typically referred to as `ProbNNh` with $h \in \{e, \mu, \pi, K, p\}$.

Aiming to provide analysts with everything necessary for pursuing the LHCb physics program, LAMARR also disposes of a dedicated set of GAN-based models to parameterize the global response of the PID system. Since both the CombDLLs and ANNPIDs rely on the response of the MUON system only for those tracks that pass the `isMuon` criterion, two different sets of parameterizations have been designed, counting four specie-specialized models each. When the `isMuon` flag is true, the corresponding GAN models make use of the momentum p and pseudorapidity η of the traversing particles, the number or reconstructed tracks `nTracks`, and the RICH and MUON likelihoods to parameterize the global response of the PID system. On the contrary, when `isMuon` is not satisfied, the MUON likelihoods are not included. Hence, the complete list of features used to conditionate the PID models follows:

- the reconstructed momentum p of the traversing particles;
- the reconstructed pseudorapidity η of the traversing particles;
- the number of reconstructed tracks `nTracks`;
- the charge q of the traversing particles;
- the RICH DLL of the electron hypothesis versus the pion one `RichDLLe`;
- the RICH DLL of the muon hypothesis versus the pion one `RichDLLmu`;
- the RICH DLL of the kaon hypothesis versus the pion one `RichDLLk`;
- the RICH DLL of the proton hypothesis versus the pion one `RichDLLp`;
- when `isMuon` $= 1$, the MUON log-likelihood for the muon hypothesis `MuonMuLL`;
- when `isMuon` $= 1$, the MUON log-likelihood for the non-muon hypothesis `MuonBgLL`.

Relying on these input features, LAMARR allows to reproduce the global response of the PID system only using the information provided by the previous modules of the pipeline defined for charged particles. The reconstructed particles and the corresponding momentum and pseudorapidity are available from the tracking models discussed in Section 4.2. The measurement of the detector occupancy results by sampling from a parametric probability distribution modeled on simulations. The charge of the traversing particles is directly provided by the physics generators. Both the RICH and MUON likelihoods results from the GAN models discussed in Sections 4.3.2 and 4.3.3. Finally,

the `isMuon` criterion can be parameterized by relying on the neural networks detailed in the previous Section. The combination of such features allows to effectively parameterize not only the CombDLL variables but also the response of the ANNPID, even if it relies on information not included in the aforementioned list. As demonstrated in Ref. [1], by relying on the *latent space* $\mathcal{Z}$, GANs are able to reproduce the missing information, allowing LAMARR to parameterize successfully the conditional probability distributions of the following variables:

- the PID CombDLL of the electron hypothesis versus the pion one $\Delta\text{LL}_{comb}(e-\pi)$;
- the PID CombDLL of the kaon hypothesis versus the pion one $\Delta\text{LL}_{comb}(K-\pi)$;
- the PID CombDLL of the proton hypothesis versus the pion one $\Delta\text{LL}_{comb}(p-\pi)$;
- the ANNPID probability for electrons `ProbNNe`;
- the ANNPID probability for pions `ProbNNpi`;
- the ANNPID probability for kaons `ProbNNk`;
- the ANNPID probability for protons `ProbNNp`;
- the PID CombDLL of the muon hypothesis versus the pion one $\Delta\text{LL}_{comb}(\mu-\pi)$;
- when `isMuon` $= 1$, the ANNPID probability for muons `ProbNNmu`.

The training and validation of the two sets of GAN-based models require two different datasets resulting from Detailed Simulation. The first one includes pure samples of muons, pions, kaons, and protons passing the `isMuon` criterion, and contains the information from the MUON system and the whole global response of the PID system for studying muon candidates, namely CombDLL$(\mu-\pi)$ and `ProbNNmu`. On the contrary, the second dataset includes four pure samples of particles not passing `isMuon` and does not contain neither the MUON likelihoods nor the `ProbNNmu` variable.

Model design and training

The design of the parameterizations for the global response of the LHCb PID system requires optimization campaigns that have made use of significant computational power (most of which included GPUs), provided by combining on-promises, cloud, and HPC resources. Dozens of the optimization studies were run in parallel and coordinated by the HOPAAS service [2], earning more than $\mathcal{O}(10^3)$ GPU hours to select the set of training strategies listed in Tables 4.9 and 4.10. In particular, the Bayesian optimizer exposed by HOPAAS was used to find the best-suited collection of hyperparameters for training GANs that aim to parameterize faithfully the PID response. To this end, similarly to the campaigns for the RICH and MUON models, also in this case the optimization procedure was designed to minimize the errors introduced by the adoption of flash-simulations with respect to fully simulated samples. To have an *objective measurement*[13] of the mismodeling errors, the metric chosen for the optimization studies was the KS-distance.

[13]Here, with *objective measurement*, we refer to the necessity of evaluating the mismodeling errors with a metric robust against the dynamic evolution of the generator-discriminator learning process.

	Generator	*Discriminator*
skip connections [111]	✔	✔/✔/✔/✖
latent space dim [134]	64	-
n auxiliary features [177]	-	2
input shape	(None, 74)	(None, 21)
input preprocessing	✔	✔
n hidden layers	5	5
n hidden neurons	128	128
hidden activation functions	Leaky ReLU	Leaky ReLU
output activation function	linear	linear sigmoid sigmoid sigmoid
output shape	(None, 9)	(None, 1)
output preprocessing	✔	✖
optimizer	RMSprop	RMSprop
learning rate	$\sim (7/5/7/4) \times 10^{-4}$	$\sim (7/8/10/4) \times 10^{-4}$
loss function	BCE-based loss (4.7) BCE-based loss (4.7) BCE-based loss (4.7) Least squares loss (3.12)	BCE-based loss (4.6) BCE-based loss (4.6) BCE-based loss (4.6) Least squares loss (3.12)
BCE from logits	True/False/False/-	True/False/False/-
BCE label smoothing	✖/0.05/0.05/-	✖/0.05/0.05/-
Gaussian noise stddev [144]	-/0.02/0.02/✖	-/0.02/0.02/✖
Lipschitz regularization [145]	ALP/-/-/-	ALP/-/-/-
learning rate scheduling	ExpDecay	ExpDecay
scheduling decay rate	0.1	0.1
scheduling decay steps	$(15/18/10/14) \times 10^4$	$(21/11/20/20) \times 10^4$
batch-size	$(9/2.5/7.5/3) \times 10^3$	$(9/2.5/7.5/3) \times 10^3$
batches per epoch	~ 500	~ 500
n epochs	1000	1000

Table 4.9: Hyperparameters of the GAN-based models for the PID system for particles passing the `isMuon` criterion. When multiple hyperparameter values are reported, they refer to the different settings adopted for the definition and training of the models for muons, pions, kaons, and protons, respectively.

Such metric was computed in bins of momentum, pseudorapidity, and `nTracks` for each of the analysis-level variables parameterized by the GAN-based models. The maximum error measured was then taken as the optimization score so that minimizing its values corresponds to searching for the best-suited model to parameterize the PID response in the whole phase space $(p, \eta, \texttt{nTracks})$.

The Bayesian optimizer expands the list of *generative models* that can be used to parameterize the PID system by including novel GAN algorithms and regularization

	Generator	*Discriminator*
skip connections [111]	✔	✔/✖/✖/✔
latent space dim [134]	64	-
n auxiliary features [177]	-	2
input shape	(None, 72)	(None, 18)
input preprocessing	✔	✔
n hidden layers	5	5
n hidden neurons	128	128
hidden activation functions	Leaky ReLU	Leaky ReLU
output activation function	linear	sigmoid linear tanh sigmoid
output shape	(None, 8)	(None, 1) (None, 256) (None, 1) (None, 1)
output preprocessing	✔	✖
optimizer	RMSprop	RMSprop
learning rate	$\sim (7/7/8/8) \times 10^{-4}$	$\sim (7/5/6/4) \times 10^{-4}$
loss function	BCE-based loss (4.7) Energy distance (3.26) Wasserstein distance (3.16) BCE-based loss (4.7)	BCE-based loss (4.6) Energy distance (3.26) Wasserstein distance (3.16) BCE-based loss (4.6)
BCE from logits	False/-/-/False	False/-/-/False
BCE label smoothing	0.05/-/-/0.05	0.05/-/-/0.05
Gaussian noise stddev [144]	0.02/-/-/0.02	0.02/-/-/0.02
Lipschitz regularization [145]	-/GP/ALP/-	-/GP/ALP/-
learning rate scheduling	ExpDecay	ExpDecay
scheduling decay rate	0.1	0.1
scheduling decay steps	$(15/14/16/10) \times 10^4$	$(20/7/10/14) \times 10^4$
batch-size	$(5/9.5/9/10) \times 10^3$	$(5/9.5/9/10) \times 10^3$
batches per epoch	~ 500	~ 500
n epochs	1000	1000

Table 4.10: Hyperparameters of the GAN-based models for the PID system for particles not passing the `isMuon` criterion. When multiple hyperparameter values are reported, they refer to the different settings adopted for the definition and training of the models for muons, pions, kaons, and protons, respectively.

strategies for the discriminator. The Wasserstein distance, defined in Eq. (3.16), finds finally a way to enter into such a list, adding WGANs to the list of *flavours* to select from. In particular, the optimization studies reveal that the ALP algorithm proposed in Ref. [147] is the best strategy to induce the Lipschitzianity of the WGAN discriminators. On the contrary, the energy distance defined in Eq. (3.26) and used by CramerGANs relies on the *gradient penalty* (GP) algorithm to implicitly force the discriminator to approximate a Lipschitz function. To accomplish the classification task, CramerGANs generally dispose of a high-dimensional vector space, called *critic space*. BCE-based GANs powered by noise injection [144] or the ALP regularization reconfirm their ability to reproduce conditional probability distributions, as well as LSGANs that exhibit good performance without relying on any customary strategy for training stabilization. Once again, implementing different GAN algorithms and regularization strategies is made easy by using the PIDGAN package [6].

Both sets of PID GANs strongly depend on the input condition x, whose processing is crucial for either the generator or discriminator networks. The generator, implemented via 8-layer (or 5-layer) neural network, couples the x features with the latent vector z, sampled from a 64-dimensional Gaussian distribution. On the other hand, the discriminator, constructed as 6-layer (or 5-layer) neural network, uses the conditions x and the target variables y to distinguish the reference data from the generated one. To reduce the vanishing gradient problem, the generator is equipped with skip connections [111], as well as the discriminator unless it drives the training of LSGAN systems that exhibit instabilities when dealing with residual blocks. To further improve the generator output, the discriminator relies on a set of auxiliary features [177] that are used to constrain the generated CombDLLs to physically reasonable quantities by the minimax game and the *message passing* through the corresponding computational graph. The auxiliary features used for this scope follows:

- the PID CombDLL of the muon hypothesis versus the electron one $\Delta\text{LL}_{comb}(\mu - e)$;
- the PID CombDLL of the proton hypothesis versus the kaon one $\Delta\text{LL}_{comb}(p - K)$.

Whether the PID GANs are specialized for particles passing the `isMuon` criterion or not, the aforementioned CombDLLs exhibit very peculiar characteristics (partially discussed in the following Section), whose accurate parameterization is of primary importance for physics analysts. Hence, disposing of discriminators powered by auxiliary features becomes crucial for producing faithful flash-simulated samples.

To train and validate the two sets of GANs, as many datasets of detailed simulated particles were produced. The first dataset, dedicated to particles satisfying the `isMuon` criterion, counts $\mathcal{O}(4 \times 10^6)$ instances, split into four pure samples of muons, pions, kaons, and protons. The second dataset relies instead on $\mathcal{O}(0.5 \times 10^9)$ particles not passing `isMuon`, again, split into four pure samples of muons, pions, kaons, and protons. Both datasets were divided following the same philosophy: a fraction of 50% for training, another 10% for monitoring overtraining effects, and the remaining 40% for validation studies. The learning curve of the PID GAN models specialized for particles passing or not `isMuon` are depicted in Figures 4.47 and 4.48, respectively. As reported in Tables 4.9 and 4.10, the training of the various GAN *flavours* is driven by different loss functions and regularization strategies, that strongly affect the evolution of the competition between the generator and discriminator networks shown in the following figures.

Validation studies

Before to assemble the pipeline of parameterizations for the charged particles described along Sections 4.2 and 4.3, we need to demonstrate that GANs can model the global response of the PID system and reproduce faithfully the performance offered by LHCb in distinguishing different particle species as exhibited in Detailed Simulation. To this end, the following figures report the probability distributions of several CombDLL($h_1 - h_2$) and `ProbNNh` as expected from pure samples of h particles with $h \in \{\mu, \pi, K, p\}$. The reference distributions are then compared with what results from the trained generators. To further test the performance of the models, plots for the selection efficiency and misidentification probability are also reported with the aim of demonstrating that GANs succeed in reproducing the global response of the LHCb PID system.

The parameterization of the muon-pion separation is investigated in Figures 4.49 and 4.50 in four bins of momentum by relying on the distributions of the combined $\Delta\text{LL}(\mu - \pi)$ and `ProbNNmu`, respectively. Aiming to select muon candidates by rejecting pions, the reference distributions are compared versus the output of two GANs specialized for muons and pions that have passed the `isMuon` criterion. Thanks to the additional separation power provided by the global classifiers represented in the histograms, the LHCb PID system offers a very good muon-pion separation that results from the combined action of discriminating variables and the `isMuon` filter, as highlighted in the efficiency plots depicted in Figures 4.49 and 4.50. By relying on the DLLs from the RICH and MUON systems, the trained generators show good performance in reproducing both the CombDLL and the response of the ANNPID for muon identification, as shown in the histogram plots.

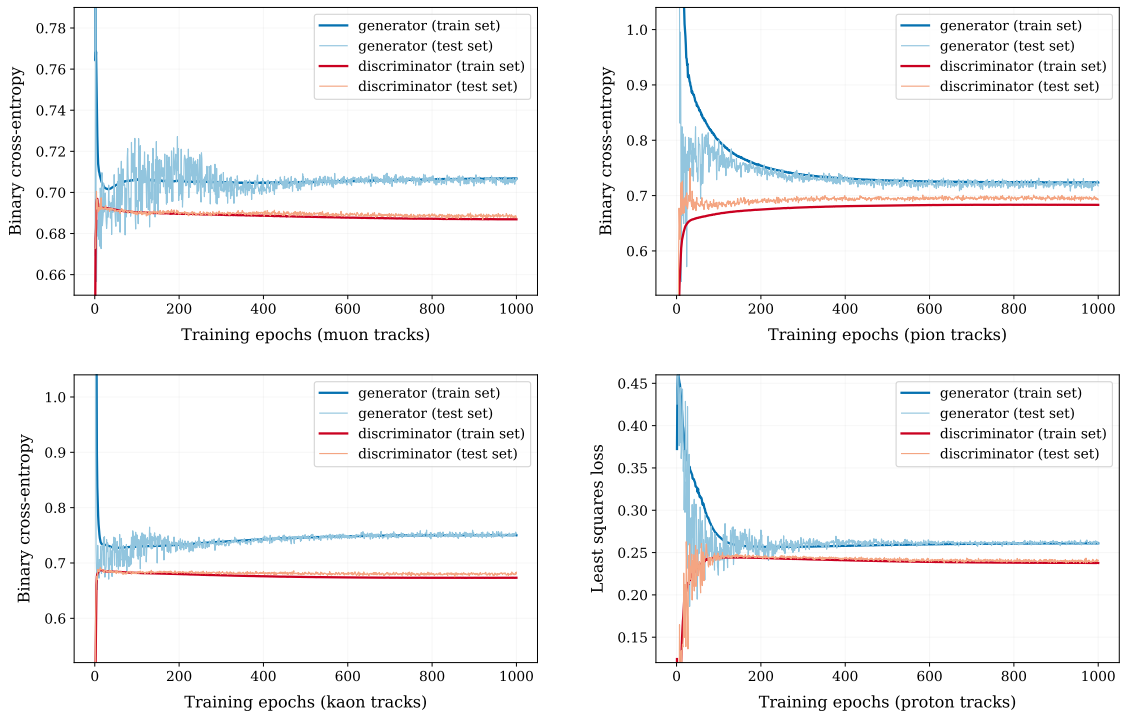


Figure 4.47: Learning curves of the GANs trained to parameterize the global response of the LHCb PID system when traversed by muons, pions, kaons, and protons *passing* the `isMuon` criterion. The loss functions used by the various GAN models are listed in Table 4.9.

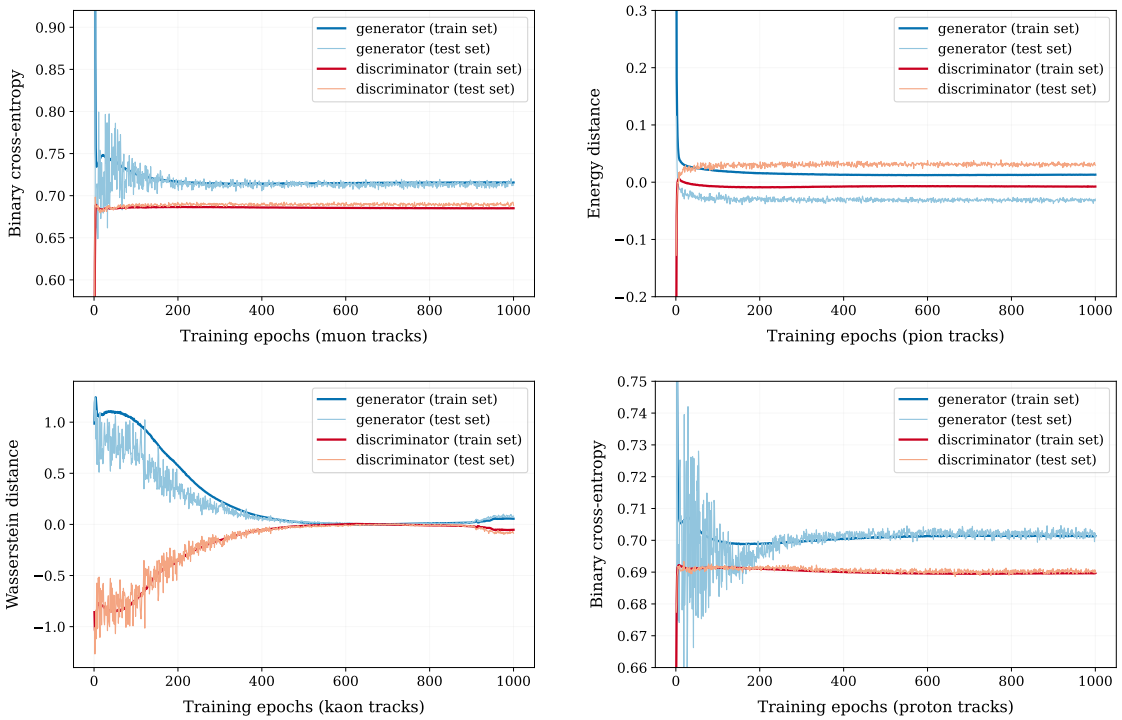


Figure 4.48: Learning curves of the GANs trained to parameterize the global response of the LHCb PID system when traversed by muons, pions, kaons, and protons *not passing* the `isMuon` criterion. The loss functions used by the various GAN models are listed in Table 4.10.

As a further investigation, we apply selection cuts on both the generated variables and obtain as a result efficiencies and misidentification probabilities that match, within the statistical uncertainties, with what is expected from Detailed Simulation as shown in Figures 4.49 and 4.50.

The distributions of $\Delta\text{LL}_{comb}(K-\pi)$ and `ProbNNk` for kaons and pions are depicted in Figures 4.51 and 4.52 in four bins of momentum. Such a study aims to test the ability of GANs specialized for kaons and pions to model the LHCb discrimination performance for these charged mesons. In LAMARR, the response of the PID system to particles that satisfy or not the `isMuon` criterion is, by design, parameterized by two different sets of GANs. Here, for simplicity, both the reference and generated distributions are obtained for kaons and pions not passing `isMuon`, allowing us to have a clear picture to investigate the performance achieved by generators coming from one of the two sets of parameterizations. By doing so, the response of the MUON system is not included in the computation of the CombDLLs, whose major contribution is then the response of the RICH detectors. As a consequence, looking at the histograms in Figure 4.51, we can observe a structure at $\Delta\text{LL}_{comb}(K-\pi) = 0$ for $p < 10$ GeV/c. As discussed in Section 4.3.2, this effect is due to the construction design of the RICH2 detector, whose active materials prevent identifying kaons with $p \leq 9.3$ GeV/c [44]. Disposing of the RICH DLLs among the input conditions, it is quite easy for the trained generators to accomplish the simulation task, and to reproduce successfully the distributions of the CombDLLs resulting from Detailed Simulation. Figure 4.52 shows that the combination of the input conditions with samplings from the *latent space* $\mathcal{Z}$ is sufficient to parameterize the response of the ANNPID [1], reconfirming that GANs also succeed in accomplishing knowledge distillation

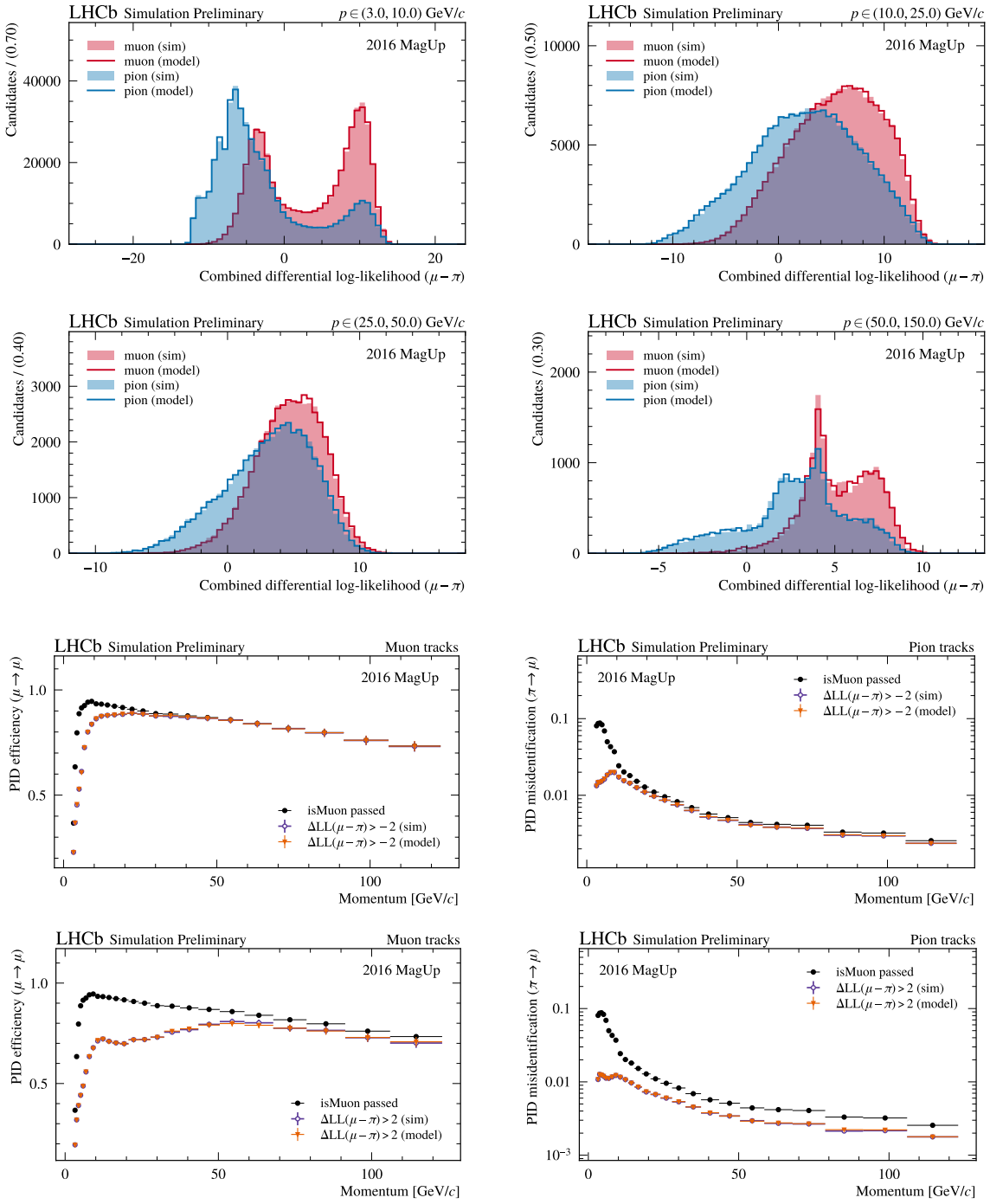


Figure 4.49: Validation plots of the GAN-based models trained to parameterize the *muon-pion separation* offered by LHCb by relying on the response of the whole PID system. The distributions of the corresponding combined DLL as modeled by Detailed Simulation are represented as filled histograms for muons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the efficiency for the muon identification based on the PID criterion, applied on top of the `isMuon` requirement, is reported on the left plots using two selection cuts: $\Delta LL > -2$ and $\Delta LL > 2$. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to \mu$) as depicted on the right plots.

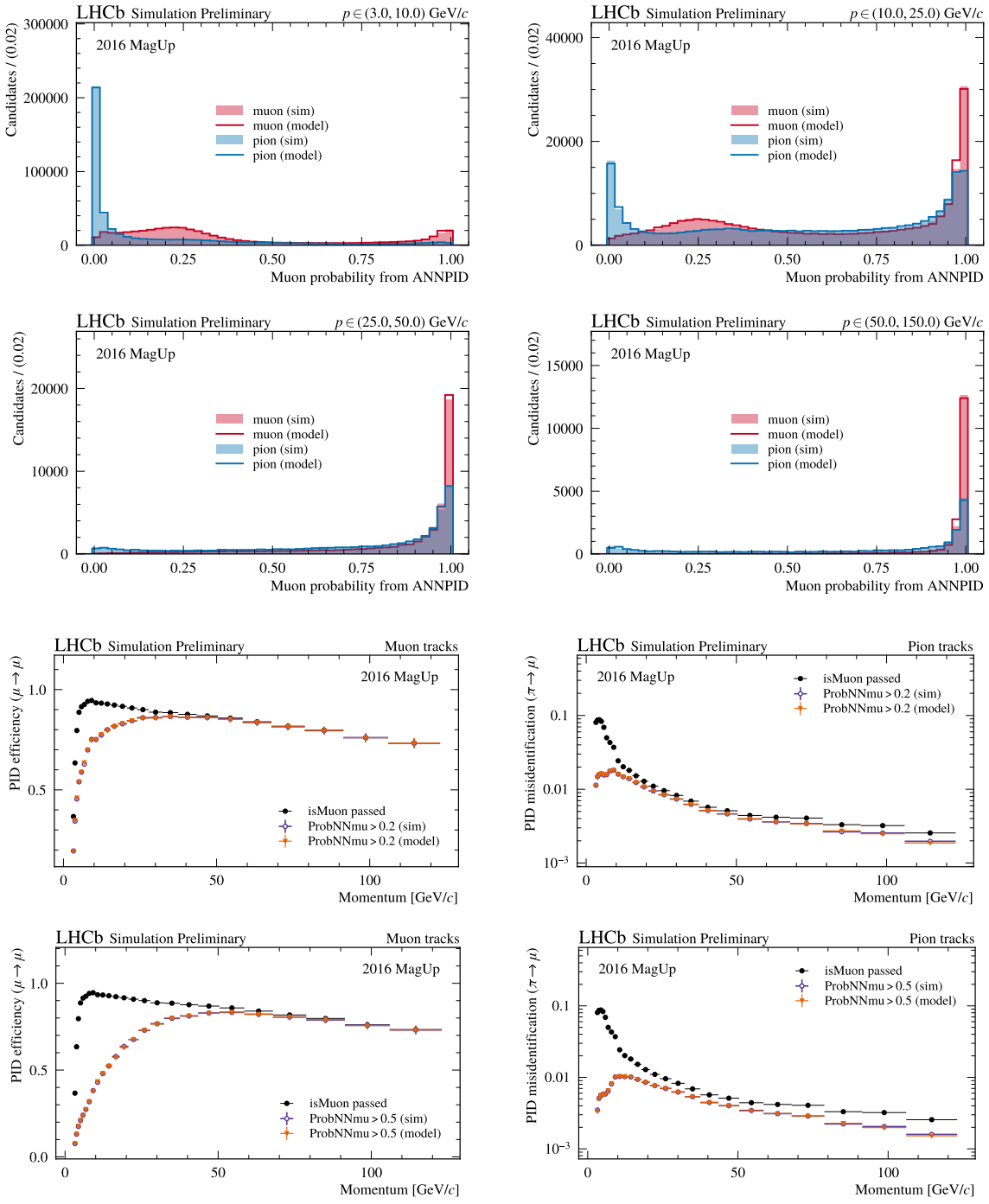


Figure 4.50: Validation plots of the GAN-based models trained to parameterize the *muon-pion separation* offered by LHCb by relying on multivariate techniques. The distributions resulting from Detailed Simulation for the `ProbNNmu` variable are represented as filled histograms for muons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the PID efficiency for the muon identification is reported on the left plots using two selection cuts: `ProbNNmu` > 0.2 and `ProbNNmu` > 0.5. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to \mu$) as depicted on the right plots.

problems, as discussed in Section 4.2.4 for the parameterization of the `ghostProb` score.

Figures 4.53 and 4.54 report the distributions of $\Delta\text{LL}_{comb}(p-\pi)$ and `ProbNNp` for protons and pions in the usual four momentum bins. Also in this case, to have a clear validation picture, both the reference and generated distributions result from protons and pions not passing the `isMuon` criterion. Similarly to the kaon case, the CombDLL variable presents structures at $\Delta\text{LL}_{comb}(p-\pi)=0$ in the histograms with $p<25$ GeV/c due to the RICH materials that make the proton discrimination difficult for $p\leq 17.7$ GeV/c [44]. Directly relying on the RICH likelihoods, the two GANs (specialized for protons and pions) parameterize correctly such phenomenon, and also succeed in reproducing the output of the ANNPID for proton identification depicted in Figure 4.53. To further investigate the performance of the trained models, proton candidates are selected by applying cuts on the generated variables, and the corresponding efficiencies (or misidentification probabilities) are evaluated. The results, added to Figures 4.53 and 4.54 for CombDLL and `ProbNNp`, respectively, match with what is expected from Detailed Simulation within the statistical errors.

Finally, the GAN-based models for the proton-kaon separation are investigated in Figures 4.55 and 4.56 in four bins of momentum by relying on $\Delta\text{LL}_{comb}(p-K)$ and `ProbNNp`, respectively. It should be pointed out that such CombDLL is not listed within the output features of the trained generators, but instead, it results from the combination by subtraction of the CombDLLs for the proton and kaon hypotheses versus the pion one. To constrain the generator to physically reasonable outputs, the actual distribution of $\Delta\text{LL}_{comb}(p-K)$ is used by the discriminator as an auxiliary feature. Since we are comparing protons and kaons not satisfying `isMuon`, the histograms in Figure 4.55 show the expected narrow peaks at $\Delta\text{LL}_{comb}(p-K)=0$ for $p<25$ GeV/c. Again, this is due to the combination of identification problems for kaons and protons with $p\leq 9.3$ GeV/c and $p\leq 17.7$ GeV/c, respectively. Also the ANNPID classifier is affected by the same problem as demonstrated by the large overlapping between the proton and kaon distributions shown in Figure 4.56 for $p<25$ GeV/c. Driven by the RICH likelihoods and the auxiliary features, the two GANs (specialized for protons and kaons) reproduce quite faithfully $\Delta\text{LL}_{comb}(p-K)$. However, parameterizing the aforementioned narrow peaks is still a non-trivial task, that leads to a mismodeling of proton candidates at low momentum when we apply loose cuts like the top example depicted in the efficiency plots in Figure 4.55. On the contrary, the proton-kaon separation provided by the `ProbNNp` variable is well reproduced, joining the set of analysis-level variables successfully parameterized by GAN-based models, and further confirming the validity of the flash-simulation strategy pursued by LAMARR.

4.4 Neutral particles pipeline: the ECAL detector

As part of the PID system, the main role of the LHCb calorimeters is to enable the separation of photons from π^0 candidates and to contribute to the identification of electrons. The reconstruction algorithms allow to distinguish neutral from charged particles by studying the absence (or presence) of tracks in front of the energy deposits collected within the Calorimeter system by forming *clusters*. In addition to the (in)consistency of the reconstructed tracks with the barycenters of the energy deposits, also the shape of the clusters is employed for particle identification, either to separate electrons from charged

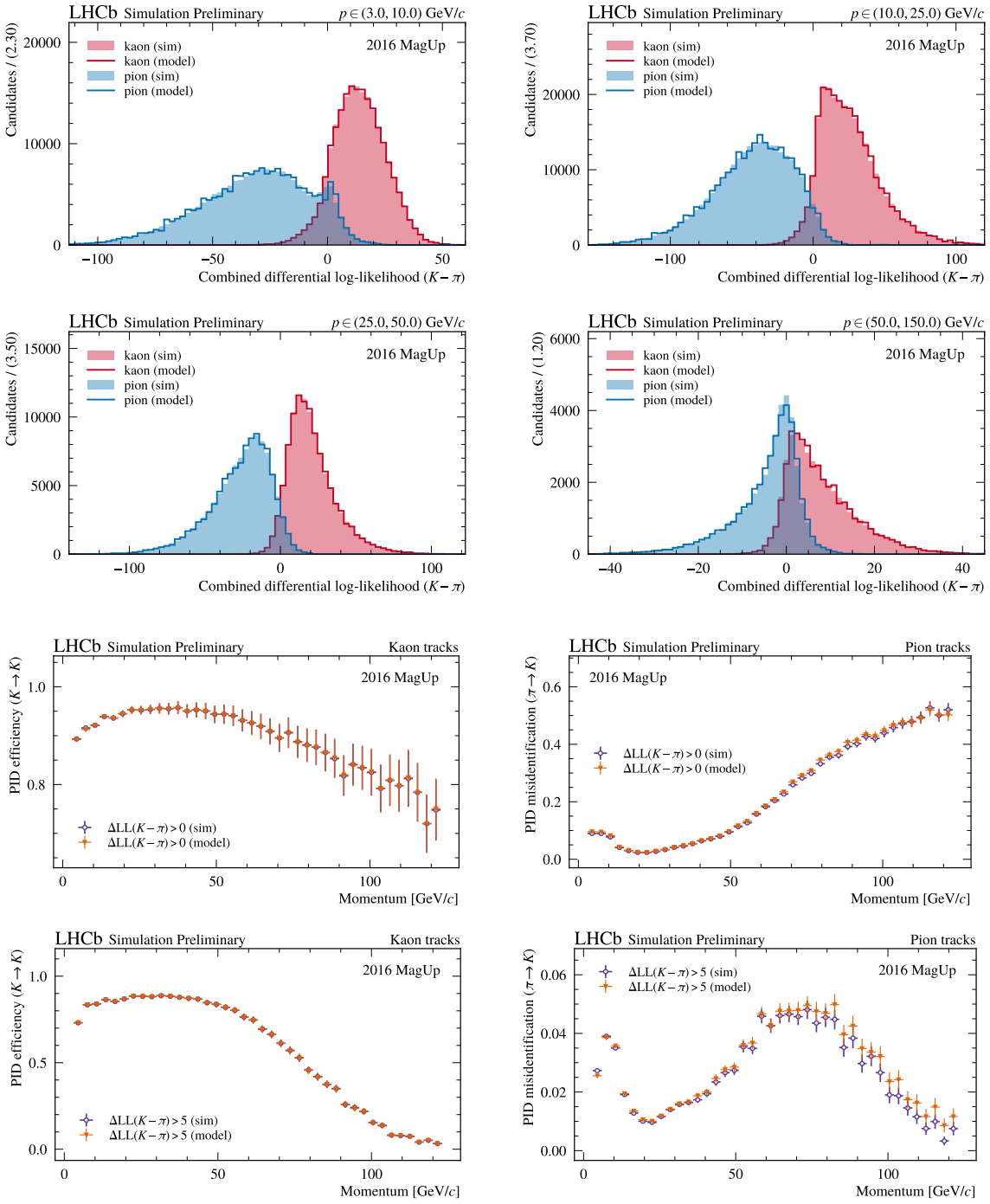


Figure 4.51: Validation plots of the GAN-based models trained to parameterize the *kaon-pion separation* offered by LHCb by relying on the response of the whole PID system. The distributions of the corresponding combined DLL as modeled by Detailed Simulation are represented as filled histograms for kaons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the efficiency for the kaon identification based on the PID criterion, applied on top of the `isMuon` requirement, is reported on the left plots using two selection cuts: $\Delta\text{LL} > 0$ and $\Delta\text{LL} > 5$. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to K$) as depicted on the right plots.

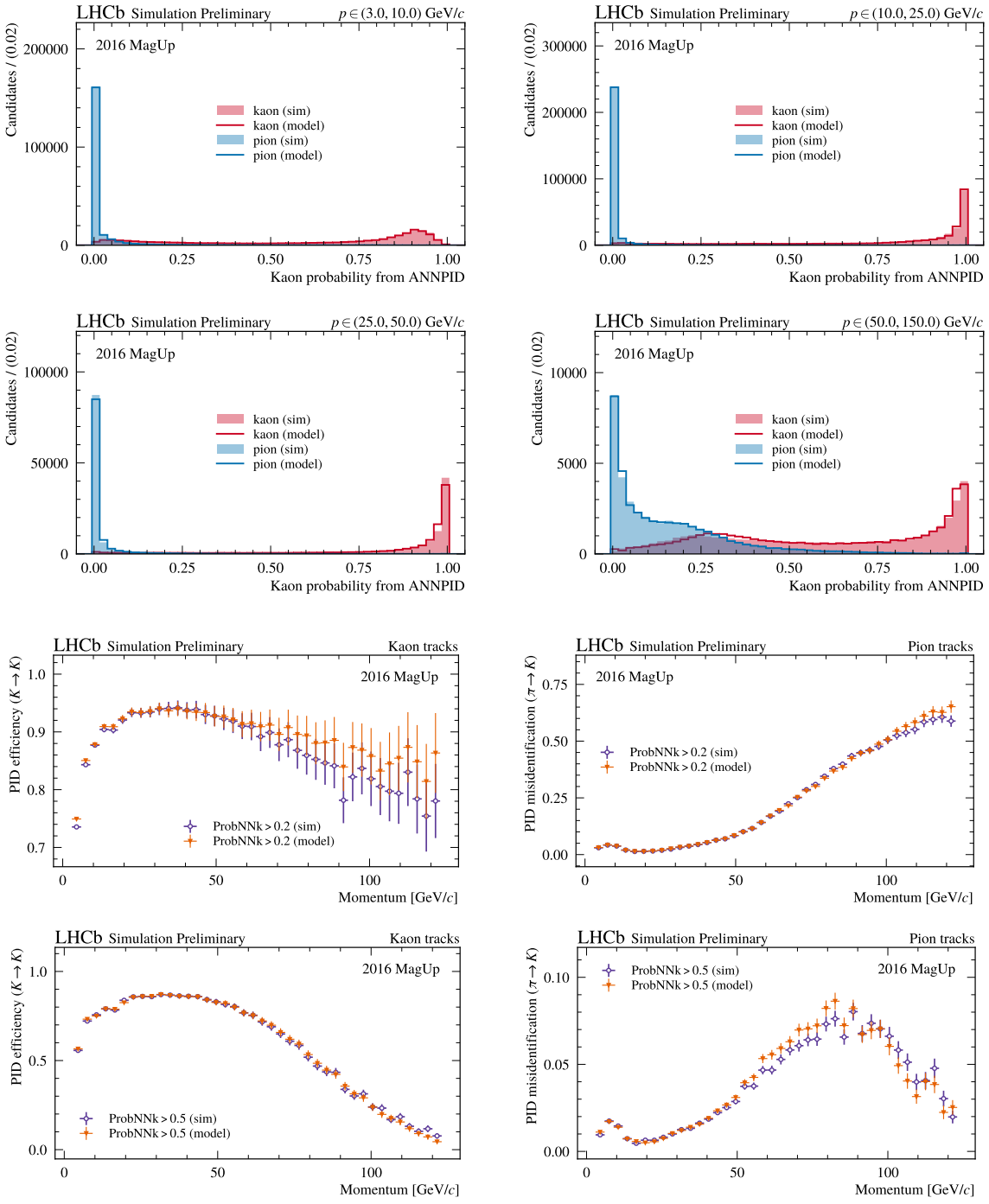


Figure 4.52: Validation plots of the GAN-based models trained to parameterize the *kaon-pion separation* offered by LHCb by relying on multivariate techniques. The distributions resulting from Detailed Simulation for the `ProbNNk` variable are represented as filled histograms for kaons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the PID efficiency for the kaon identification is reported on the left plots using two selection cuts: `ProbNNk` > 0.2 and `ProbNNk` > 0.5. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to K$) as depicted on the right plots.

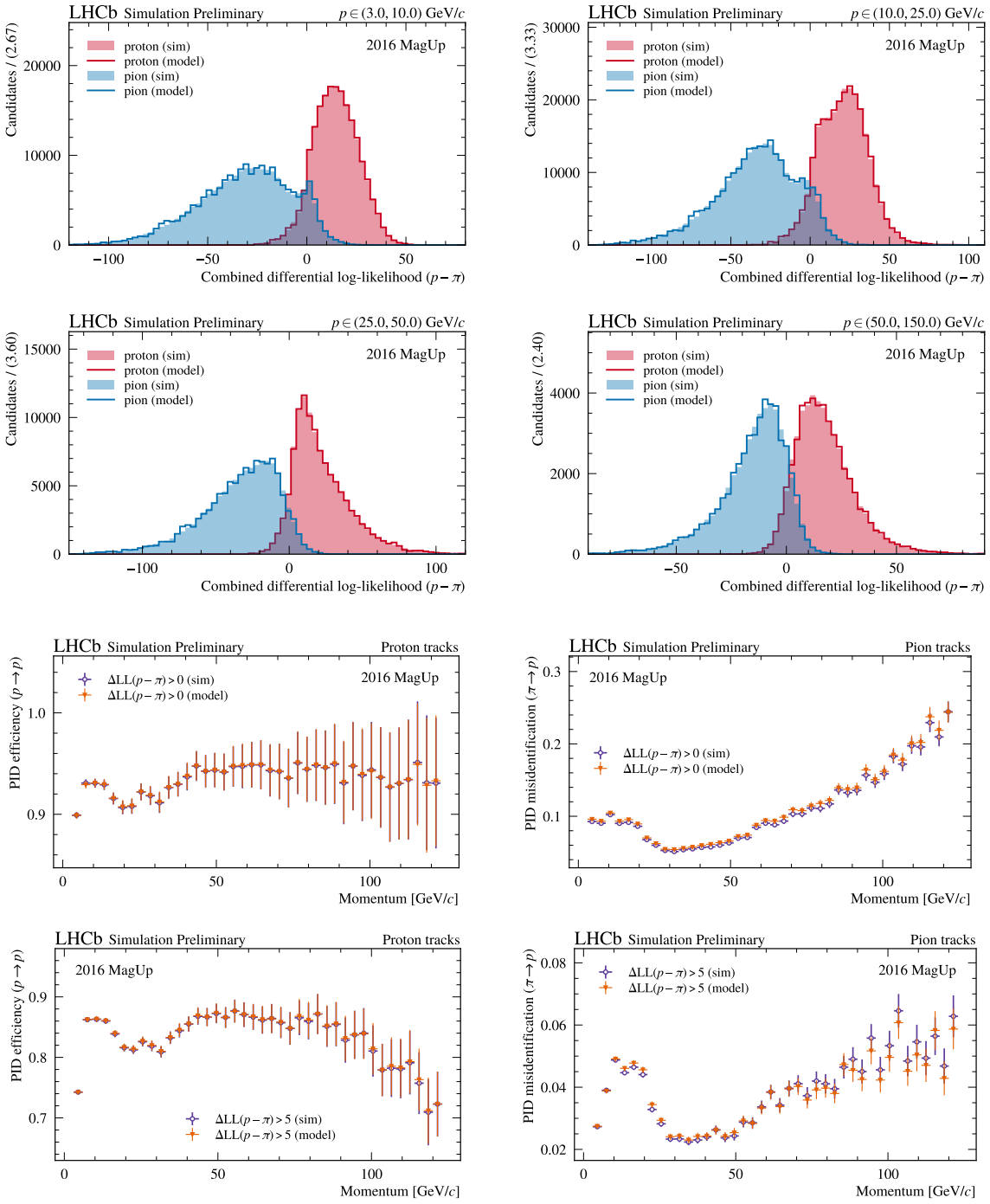


Figure 4.53: Validation plots of the GAN-based models trained to parameterize the *proton-pion separation* offered by LHCb by relying on the response of the whole PID system. The distributions of the corresponding combined DLL as modeled by Detailed Simulation are represented as filled histograms for protons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the efficiency for the proton identification based on the PID criterion, applied on top of the `isMuon` requirement, is reported on the left plots using two selection cuts: $\Delta\mathrm{LL} > 0$ and $\Delta\mathrm{LL} > 5$. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to p$) as depicted on the right plots.

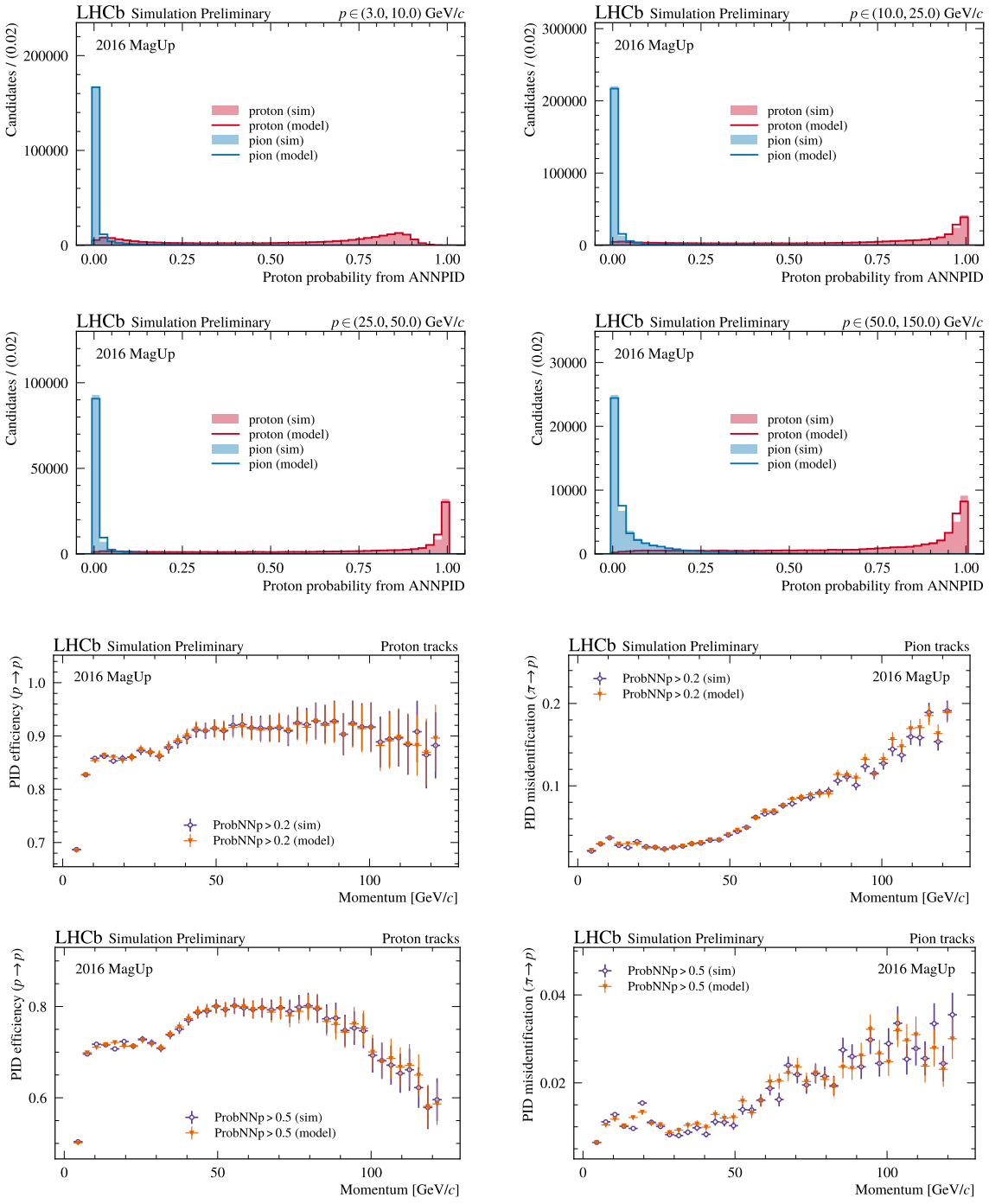


Figure 4.54: Validation plots of the GAN-based models trained to parameterize the *proton-pion separation* offered by LHCb by relying on multivariate techniques. The distributions resulting from Detailed Simulation for the `ProbNNp` variable are represented as filled histograms for protons (in red) and pions (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the PID efficiency for the proton identification is reported on the left plots using two selection cuts: `ProbNNp` > 0.2 and `ProbNNp` > 0.5. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($\pi \to p$) as depicted on the right plots.

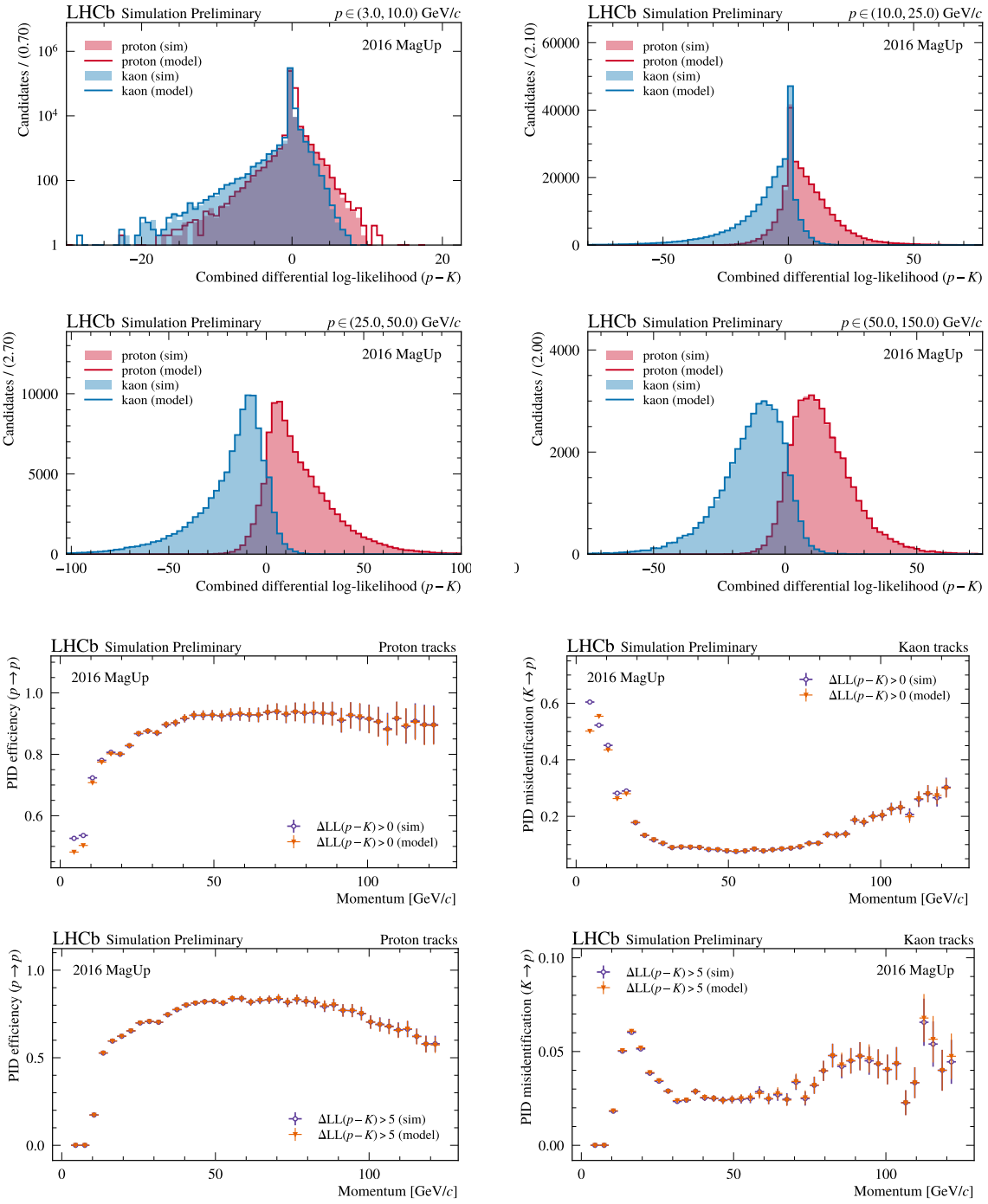


Figure 4.55: Validation plots of the GAN-based models trained to parameterize the *proton-kaon separation* offered by LHCb by relying on the response of the whole PID system. The distributions of the corresponding combined DLL as modeled by Detailed Simulation are represented as filled histograms for protons (in red) and kaons (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the efficiency for the proton identification based on the PID criterion, applied on top of the `isMuon` requirement, is reported on the left plots using two selection cuts: $\Delta\mathrm{LL} > 0$ and $\Delta\mathrm{LL} > 5$. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($K \to p$) as depicted on the right plots.

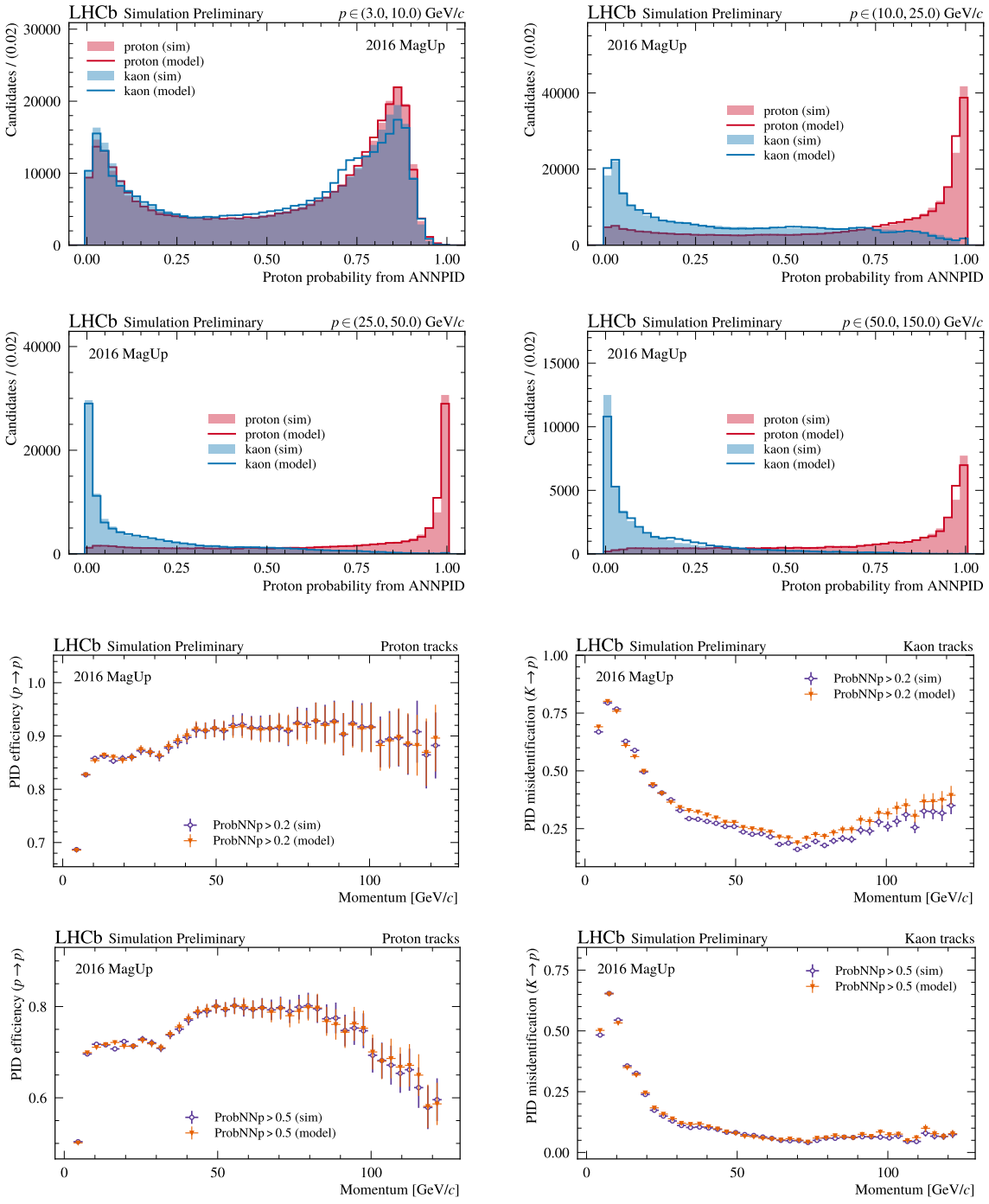


Figure 4.56: Validation plots of the GAN-based models trained to parameterize the *proton-kaon separation* offered by LHCb by relying on multivariate techniques. The distributions resulting from Detailed Simulation for the `ProbNNp` variable are represented as filled histograms for protons (in red) and kaons (in blue) in four bins of momentum p. The output of the trained generators is superimposed through solid-line histograms. To further investigate the quality of the generated distributions, the corresponding *efficiency plots* are added to the figure. In particular, the performance achieved by the GANs in reproducing the PID efficiency for the proton identification is reported on the left plots using two selection cuts: `ProbNNp` > 0.2 and `ProbNNp` > 0.5. The same cuts are used to investigate the ability of the GANs to model the misidentification probability ($K \to p$) as depicted on the right plots.

hadrons or to distinguish between photons and π^0 candidates. When evaluating the photon likelihoods, the reconstruction algorithms take into account the possibility that photons convert into electron-positron pairs when interacting with the detector material upstream of the calorimeter. In addition, since a large fraction of pairs of photons coming from the decay[14] of high energy π^0 cannot be resolved due to the ECAL granularity, specialized procedures are used at LHCb to reconstruct such π^0 by disentangling a potential pair of photons *merged* into single clusters [34, 46].

For measuring the total energy of a traversing particle, a generic calorimeter system relies on the energy deposited by a cascade of secondary particles produced by the interaction of the target particle with the detector materials. The large number of particles that come into play together with the corresponding radiation-matter interactions make the detailed simulation of the *particle shower* computationally expensive, as already observed for the LHCb case in Figure 4.1. It represents a shared problem across the HEP community that is investing great efforts in developing alternative strategies to reduce the CPU cost for calorimeter simulations, as demonstrated by the "CaloChallenge" initiative[15]. Since the energy deposited along the calorimeter layers can be represented as two-dimensional images whose pixels correspond to the calorimetric cells (as shown in Figure 4.57), the progress achieved in Computer Vision for the *image generation* problem can help in accelerating the detailed simulations. In particular, deep generative models (e.g., GAN, Normalizing Flows, and Diffusion Models) continue to be investigated and tuned by the HEP community to parameterize the energy deposited within the calorimeter active volume [187–189]. Despite the apparent simplicity of the calorimetric clusters depicted in Figure 4.57, parameterizing the energy deposits so that the underlying physics processes are taken into account correctly and the reconstruction algorithms reproduce reasonable results is a non-trivial task.

The LHCb Collaboration participates in this joint effort investigating solutions based on deep generative models [162, 177, 190] or parametric functions [160, 161]. Since such techniques aim to reduce the CPU cost of calorimeter simulations by providing models that reproduce the energy deposits without relying on the underlying physics processes, they are generally referred to as *fast-simulations*. Inspired by the pioneering work of Ref. [187] with CaloGAN, Viktoria Chekalina and others [162] proposed a WGAN-based model implemented via 2-D convolution layers to parameterize the particle showers as produced within the LHCb ECAL detector. Despite the good agreement shown in Figure 4.57 between clusters resulting from Geant4-based simulations (top row) and the ones produced with WGANs (bottom row), using the latters to derive, from the traditional reconstruction algorithms, high-level quantities with distributions physically reasonable is hard. It is demonstrated by the mismatch exhibited by WGANs with respect to Geant4 reported in Figure 4.58 (left) for a shape property of the electromagnetic cluster. A viable solution is to inform the generator network of the physics constraints underlying the simulation task by relying on the *auxiliary system* proposed in Ref. [177]. The discriminator network disposes of a differentiable parameterization of the reconstruction algorithms that is used to compute (at runtime) a set of high-level quantities either from the real clusters or the generated ones during the training. Any mismodeling of the physics

[14]The branching ratios of the two main decay modes of π^0 are $\mathcal{B}(\pi^0 \to \gamma\gamma) = (98.823 \pm 0.034)\%$ and $\mathcal{B}(\pi^0 \to e^-e^+\gamma) = (1.174 \pm 0.035)\%$ [10].

[15]The challenge concluded in May 2023 with a final workshop whose complete agenda is available at `https://agenda.infn.it/event/34036`.

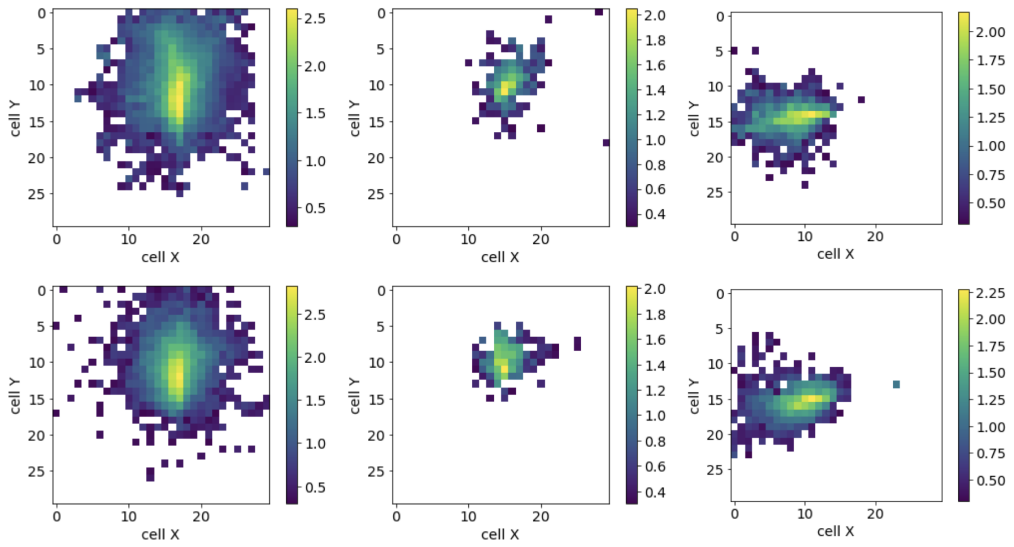


Figure 4.57: Electron showers produced within an electromagnetic calorimeter inspired by the LHCb detector as simulated by GEANT4 (top row) or by a WGAN-based model (bottom row) for three different sets of input conditions. Figure reproduced from Ref. [162].

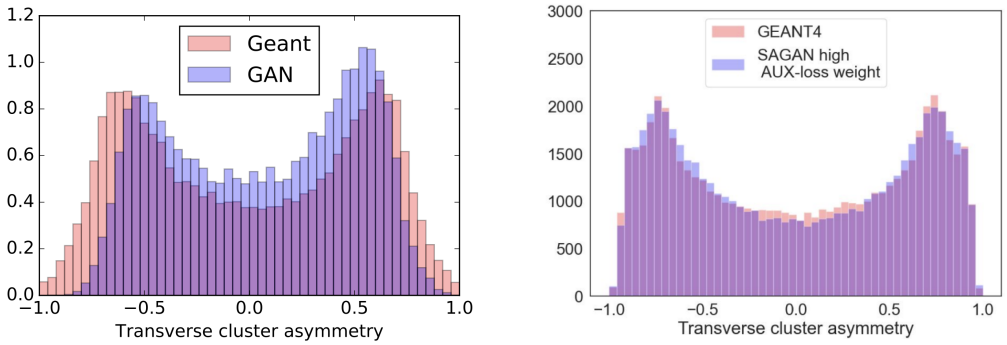


Figure 4.58: Performance in reproducing the distribution of a high-level cluster variable computed by relying on the particle shower generated with a plain GAN-based model (left), or with a GAN powered by the *auxiliary system* (right). Figures reproduced from Refs. [162, 177].

properties exhibited by the generator is used by the discriminator as an auxiliary feature to accomplish the classification task. Then, through the minimax game, the generator receives feedback on how to simulate clusters complying with the physics constraints, improving its capability to reproduce the distributions of high-level variables, like the one depicted in Figure 4.58 (right).

During the preparatory work for providing LHCb with a flash-simulation option [160], that is when LAMARR did not yet exist and the Simulation Group still relied on DELPHES [70], a parameterization describing the energy deposits in terms of *Molière radii*[16] was designed to offer useful insights for detector upgrade studies. In particular, given a generated particle extrapolated to the ECAL face, the first step of such parameter-

[16]The *Molière radius* is a characteristic constant of a material giving the scale of the transverse dimension of the fully contained electromagnetic showers initiated by an incident high energy electron or photon. Read more on `https://en.wikipedia.org/wiki/Moliere_radius`.

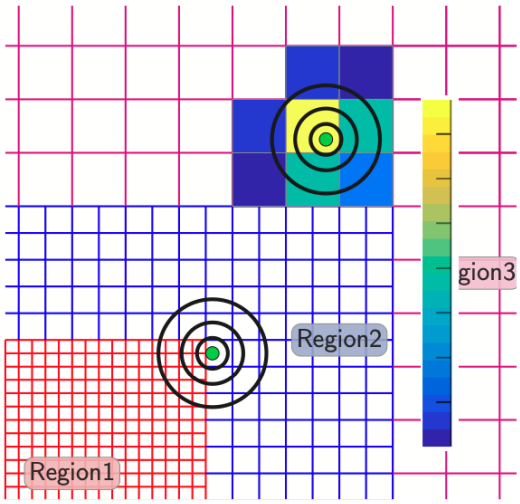


Figure 4.59: Schematic representation of the three granularity regions of the LHCb ECAL detector. Two traversing photons are reported using green points surrounded by three concentric circles with radii equal to 1, 2, and 3.5 R_M (Molière radius). The calorimetric cluster computed by relying on R_M is also shown for one of the two photons. Figure reproduced from Ref. [160].

ization was to apply customary resolution effects as provided by the dedicated DELPHES module. Then, the computation of the energy deposited in each of the calorimetric cells relied on R_M (Molière radius) that, by definition, represents the radius of a cylinder containing on average 90% of the shower's energy deposition (for the LHCb ECAL detector, $R_M = 3.5$ cm [46]). By taking the smeared coordinates of the incident point of the generated particle, it was used as the center for drawing three concentric circles in correspondence of 1, 2, and 3.5 R_M, which delimits the deposition of 90%, 95%, and 99% of the total energy. The smeared energy was uniformly distributed according to these percentages and then discretized by considering the granularity of the ECAL detector, as schematically represented in Figure 4.59. Despite some preliminary promising results [160], such parameterization was not developed to be put in production due to the presence of intrinsic limitations that would have granted its use only for detector studies. But if on the one hand, the fast-simulation nature of the model just described makes it difficult to reproduce analysis-level quantities, on the other it still requires running reconstruction algorithms that, in case of high-multiplicity events, may become rather CPU expensive.

Just like for the rest of the LHCb spectrometer, the solution proposed by LAMARR is to move to the *flash-simulation* paradigm and rely on an agnostic description either of the ECAL detector (no notion of calorimetric cells or granularity) or the underlying physics processes. However, differently from what is done for the Tracking and charged PID systems described in Sections 4.2 and 4.3, reproducing the calorimeter high-level response is a non-trivial task since *canonical* generative models rely on the assumption that an unambiguous relation between the generated particle and the reconstructed object exists[17]. Instead, the presence of electrons emitting bremsstrahlung radiation, converted photons, or merged π^0 may lead to having n generated particles responsible for m reconstructed objects (in general with $n \neq m$). This *particle-to-particle correlation problem* binds us to use techniques like the ones discussed so far (i.e., NN-based classifiers or GANs) only

[17]To a first approximation, the response of the Tracking and charged PID systems satisfy this condition.

under certain assumptions and requires instead more sophisticated solutions for facing directly the n-to-m problem [5]. The third year of my Ph.D. has been dedicated to investigating strategies able to model the response of the LHCb ECAL detector when traversed by photons. Two different approaches have been identified, whose characteristics are summarized in the following:

- **Signal photons.** When the decay modes under study include photons, to provide a clear and accurate parameterization of the ECAL detector, its response is described in terms of reconstruction efficiency and resolution effects. To this end, an unambiguous relation between photons and clusters is enforced by considering reconstructed only those clusters that *match* with at least one generated photon. Back to the k-to-k case, we can use NN-based classifiers and GANs to model efficiency and resolution, as repeatedly done in the LAMARR pipeline for charged particles.

- **Seq2seq approach.** When the photons are produced by secondary processes, such as the bremsstrahlung radiation of electrons, avoiding the ambiguity is impossible and the objective becomes the correct parameterization of the n-to-m problem. To this end, we can describe the calorimeter simulation as a sort of *translation problem* where we aim to transform sequences of n generated particles into sequences of m reconstructed clusters. Advanced sequence-to-sequence (Seq2seq) algorithms can be used effectively to face this problem [5].

The rest of this Section is devoted to detailing the aforementioned strategies to parameterize the high-level response of the LHCb ECAL when traversed by photons. In particular, Section 4.4.1 describes how to use multivariate classifiers and GANs to model the *efficiency* and *resolution* of the ECAL detector when interested by "signal photons". The Seq2seq algorithms under investigation to tackle the *particle-to-particle correlation problem* are discussed in Section 4.4.2.

4.4.1 The flash-simulation of signal photons

The underlying hypothesis shared across all the (non-)parametric functions discussed so far to model the high-level response of the LHCb spectrometer is that each generated particle is associated with one and only one *proto-particle*, namely the high-level object collecting all the reconstructed information of a particle candidate. As long as such an assumption is valid, we can describe the response of any detectors in terms of *efficiency*, modeling when to exclude particles since not reconstructed, and *resolution*, describing, in general, how to use the generator-level information to reproduce high-level quantities. In the previous Sections, we have assumed true the k-to-k relation for both the Tracking and charged PID systems, neglecting any particle-to-particle correlation effect. In the case of the Tracking system, this results in the incapability of LAMARR to parameterize the *ghost tracks*, namely random combinations of hits produced by different charged particles and wrongly reconstructed as tracks. On the other hand, for the charged PID system, such an assumption prevents modeling the electrons, whose propagation through the detector is characterized by the emission of bremsstrahlung photons, hence requiring an n-to-m treatment of the problem. To provide LAMARR with a parameterization for the electrons, we first need to build a model able to reproduce the LHCb response to traversing photons, namely to the ECAL detector.

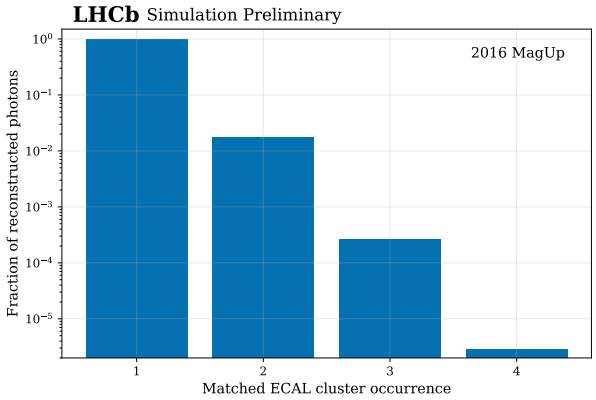


Figure 4.60: The *reconstruction criterion* admits that different generated photons may be reconstructed with the same cluster, producing clones. The clone multiplicity with respect to the total amount of reconstructed photons is reported in the bar plot on the left.

Whenever the photons are not only produced by secondary processes but are also part of the decay channel under study, modeling faithfully the ECAL response to such photons is of primary importance. As discussed in the previous Sections, NN-based classifiers and generative models succeed in describing the LHCb detectors in terms of efficiency and resolution but require that an unambiguous relation between generated particles and reconstructed objects exists. Hence, to pursue the same strategy adopted for the Tracking and charged PID systems, a dedicated parameterization for *signal photons* has been investigated by enforcing a k-to-k relation. To this end, a simplified reconstruction criterion was designed to combine the generated photons with *matching* reconstructed clusters. The first step of this criterion requires that photons and clusters have a geometrical match:

$$\sqrt{(x_{\text{photon}} - x_{\text{cluster}})^2 + (y_{\text{photon}} - y_{\text{cluster}})^2} < R_M \tag{4.12}$$

where (x, y) are the coordinates of the photons or cluster barycenters extrapolated to the ECAL face, while R_M is the Molière radius. It is a loose cut that includes multiple random photon-cluster matches. Hence, the reconstruction criterion also requires that photons and clusters have an energetic match:

$$|E_{\text{photon}} - E_{\text{cluster}}| < 2\,\sigma_E \tag{4.13}$$

where E is the energy of the photons or cluster, while σ_E is the ECAL energy resolution reported in Eq. (1.8). This second requirement reduces significantly the number of random matches, allowing the criterion to provide a reasonable approximation of the ECAL efficiency performance. Given a generated photon, if at least one cluster is found to match with it geometrically and energetically, then such a photon is considered *reconstructed*. It is worth noticing that the simplified criterion just described admits that different photons may be reconstructed with the same cluster, resulting in a collection of *cloned clusters*. As shown in Figure 4.60, such a side effect interests less than 2% of the reconstructed photons and may produce a slight overestimation of the efficiency. Despite the presence of these biases, the reconstruction criterion offers a reasonable approximation of the k-to-k relation, providing as necessary to investigate the description of the high-level ECAL response in terms of efficiency and resolution.

The reconstruction criterion was applied to a sample of $\mathcal{O}(10^7)$ photons produced by simulating a cocktail of b-hadron decays with an official configuration of Gauss [61] `sim10` (`v56r4`), involving Pythia8 [25], EvtGen [58], and Geant4 [59, 60]. The resulting raw

data was then used to reconstruct $\mathcal{O}(10^7)$ clusters calculating the barycenter coordinates, the energy, and a set of neutral PID variables by relying on the BRUNEL and DAVINCI applications [171]. Among the generated photons, a rough 30% passed the reconstruction criterion, providing the missing information to train either the ECAL efficiency model or the resolution one.

Calorimeter efficiency

After having built the calorimetric cluster by collecting the energy deposited within a 3×3 cell pattern around local energy maxima, the ECAL reconstruction algorithms allow the classification of such a cluster into a charged or neutral particle candidate based on the extrapolation of tracks to the calorimeter. For this study, we have only considered neutral clusters that in LHCb are further classified as *single* or *split* photons, where the second label refers to the pair of photons extracted from the merged clusters typically produced by π^0 [191]. To parameterize the action of the reconstruction algorithms in labeling the neutral clusters, we rely on a neural network trained to perform a *multi-class classification task*. The aim is to predict whether a generated photon will find a geometrical and energetic matching cluster, and hence be reconstructed. In case of a positive answer, the parameterization should also indicate the reconstruction procedure employed for defining the proto-particle, forecasting the probability that such a cluster will be reconstructed as a single or split photon. To this end, a 10-layer neural network equipped with skip connections [111] was trained by relying on the following set of generator-level information:

- the position (x, y) where the generated photon enters the ECAL detector;
- the logarithm of the momentum $\log(p)$ of the generated photon;
- the slopes t_x and t_y of the generated photon;
- the origin vertex position (x, y, z) of the generated photon;
- a boolean flag indicating if the generated photon results from a π^0 decay.

The training of the NN-based classifier was performed by using the same set of precautions adopted for the Tracking and charged PID efficiencies, such as an L2 kernel regularizer [169], the learning rate scheduling, or the CCE label smoothing, that was then removed during the neural network fine-tuning phase (refer to Section 4.2.2 for all the details). The neural network was designed and trained by relying on the PIDGAN APIs [6].

The ECAL efficiency model has been validated by applying the parameterization to a dataset never used during the training procedure and that counts about 2 million simulated photons. The neural network was trained to predict the fraction of generated photons matching with a cluster, and which of those are reconstructed as a *single* or *split* photon. To assess the performance of the trained model, we compare the distribution of the *reconstructed* photons with the ones obtained by weighting the kinematic distributions of the *generated* photons with the probability predicted for the various cluster classes (i.e., not reconstructed, single photon, or split photon).

The validation plots of the efficiency model are depicted in Figure 4.61 for a cocktail of photons produced in b-hadron decays as a function of the pseudorapidity η in four bins of transverse momentum p_T. The kinematic distributions of the generated photons

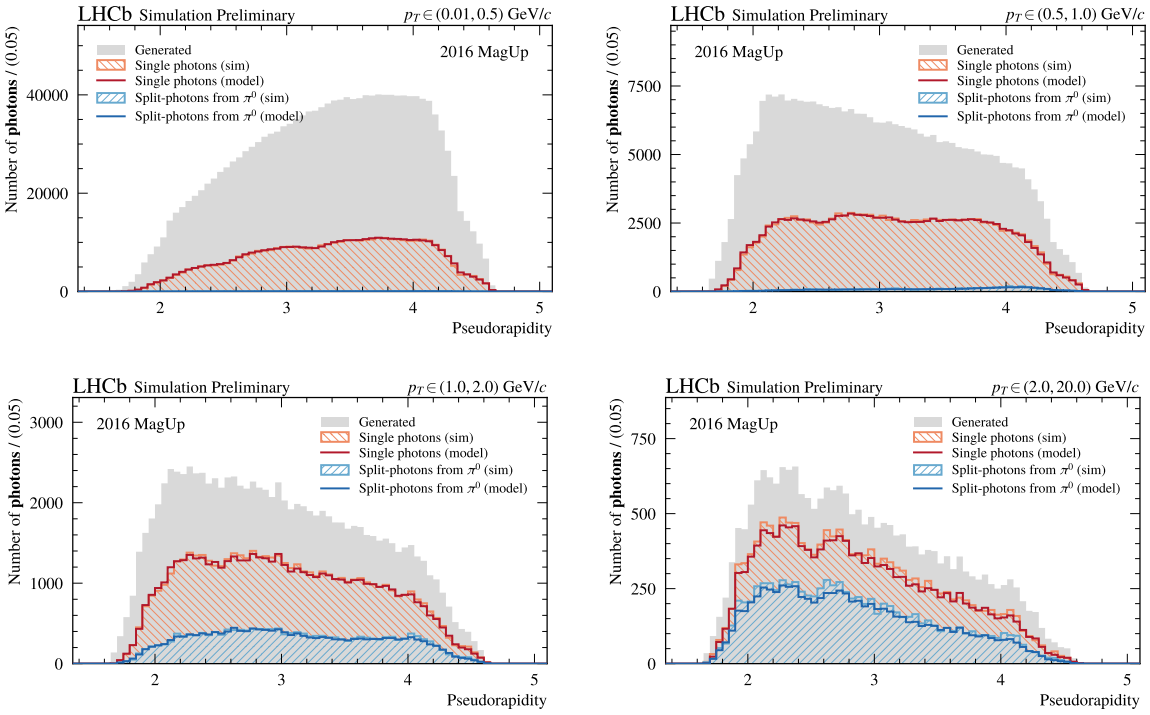


Figure 4.61: Validation plots of the ECAL efficiency model for *photons* as a function of the pseudorapidity η in four bins of the transverse momentum p_T. The kinematic distributions of the generated photons are represented as light grey shaded histograms. The distributions of clusters reconstructed as single (in red) or split (in blue) photons, are shown through stacked histograms (in hatched fill). The parameterization of the reconstructed class of the ECAL clusters as modeled by a deep neural network is superimposed using solid-line stacked histograms.

are reported in grey, while stacked histograms (in hatched fill) are used to highlight if a matching cluster is reconstructed as a single (in red) or split (in blue) photon. The output of the neural network trained to perform this multi-class classification is superimposed with solid-line histograms. The trained model exhibits good performance in parameterizing the ECAL reconstruction algorithms, succeeding in reproducing the expected distributions for single and split photons. In particular, the neural network has correctly inferred that one has the major contribution to the split photon class for $p_T > 2$ GeV/c, since the reconstruction algorithms are designed to search for merged clusters, hence resulting in split photons, for π^0 with high p_T [46].

Calorimeter resolution and photon identification

The efficiency model allows to reduce the ECAL n-to-m problem to a k-to-k relation where each of the k generated photons is associated with one and only one of the k reconstructed clusters[18]. Satisfying the hypothesis of having an unambiguous relation between generated particles and reconstructed objects, the high-level response of the ECAL detector can be described in terms of resolution effects. Hence, we aim to reproduce cluster-level variables only relying on the generator-level photon information and a model parameterizing the

[18]Due to the presence of the cloned clusters, this relation is not a *bijective* function, but only *surjective*.

errors introduced during the detection and reconstruction steps. The previous Sections have demonstrated that GANs can be fruitfully used to parameterize such errors and that the latters, powered by a stochastic component (i.e., the *latent space*), allow reproducing different detector responses even if the same generator-level information is employed.

For parameterizing the errors introduced in the detection and reconstruction of photons using the LHCb detector, we have investigated the performance achieved by a GAN-based model trained in conditional mode [141]. Due to the physics processes underlying the ECAL detection, we expect that the cluster-level information, including the reconstruction errors, is strongly related to the kinematic properties of the traversing photons. Hence, the GAN resolution model is designed to take as *input conditions* the same list of features used for the efficiency model. To these features, the reconstruction class of the matching clusters (i.e., single or split photons) is added, since we expect that different ECAL reconstruction procedures can introduce different error sources. By combining such information, the GAN model was trained to derive the (x, y)-position of the cluster barycenters in terms of reconstruction errors with respect to the coordinates of the point where the generated photon enters the ECAL detector. The same adversarial system was also trained to learn the errors introduced by the detection and reconstruction steps in measuring the total energy deposited by a traversing photon. Lastly, stressing a bit the definition of *resolution*, the GAN model was also trained to reproduce the conditional probability distributions of a set of neutral PID variables that, in a way, described the errors introduced for the photon identification. In particular, the resolution model is designed to parameterize the neutral PID sequence that assigns a confidence level (DLL) to the reconstructed clusters being produced by a single photon, a quantity called `PhotonID` [191]. The response of two independent NN-based classifiers employed at LHCb to distinguish photons from electrons (IsNotE) and hadrons (IsNotH) [46] is also included in the GAN output. The complete list of features parameterized by the resolution model is reported in the following:

- the reconstruction errors on the barycenter position $(\delta x, \delta y)$ of the matched cluster;
- the relative reconstruction error on the energy $\delta E/E$;
- the confidence level of the single photon hypothesis (`PhotonID`) [191];
- ANN-based probability of photon versus electron (IsNotE) [46];
- ANN-based probability of photon versus hadron (IsNotH) [46].

Both the generator and discriminator networks of the GAN system are implemented via 10-layer neural networks equipped with skip connections [111]. The two players were trained by using the BCE-based loss functions $(\mathcal{L}_D, \mathcal{L}_G)$ defined in (4.6) and (4.7), respectively. To avoid training instability while ensuring a faithful description of the ECAL response, we adopted the same strategies used for the Tracking and charged PID resolution models, such as the BCE label smoothing, the Gaussian noise injection [144], or the learning rate scheduling (refer to Section 4.2.4 for all the details. The ECAL GAN model was designed and trained by relying on the PIDGAN APIs [6]. To train the resolution model, a dataset of $\mathcal{O}(3 \times 10^6)$ detailed simulated photons with the corresponding matched clusters was prepared. The training procedure was performed on a fraction of 50% of the dataset. An independent 10% portion of the sample was used to monitor any evidence of overtraining, while the remaining 40% of the dataset was retained for validation studies.

The output of the trained generator can be split into two main categories: from one side, we have the reconstruction errors on the cluster barycenter position and energy, while, on the other hand, we have the set of photon identification variables (i.e., `PhotonID`, IsNotE, and IsNotH). For the reconstruction errors, we are mostly interested in modeling the widths of the distributions, namely the resolution, and correctly parameterizing how it changes as a function of the photon kinematics and the cluster reconstructed class. Figure 4.62 investigates these dependencies in kinematic bins for the spatial (top row) and energy (bottom row) resolutions exhibited by clusters reconstructed as *single photons*. The plots reported in this Figure demonstrate that the GAN-based model succeeds in parameterizing the reconstruction errors, inferring the correct dependency between the spatial (energy) resolution and $1/p_T$ (p_T). It should be pointed out that neither p_T nor $1/p_T$ are part of the input conditions, and hence the generator is able to derive these dependencies from the true energy E and the slopes (t_x, t_y). As shown in the bottom right plot of Figure 4.62, the trained generator is also able to reproduce accurately the relative energy resolution as a function of the photon energy expected from the Detailed Simulation and consistent with the one reported in the LHCb performance paper of Ref. [46]. Hence, the GAN model can be employed to reconstruct the cluster barycenter position and energy by only relying on the generator-level photon information.

The second category of variables provided by the GAN-based resolution model is the neutral PID quantities. To assess the performance of the GAN in parameterizing the PID information, Figures 4.63 and 4.64 report the comparison between the probability distributions of the modeled quantities resulting from Detailed Simulation and the ones reproduced by the trained generator. In particular, the distributions of `PhotonID` for clusters reconstructed as single photons are depicted in Figure 4.63 in four bins of transverse momentum p_T. The GAN model exhibits good performance in reproducing the distributions resulting from Detailed Simulation, even if some minor mismodeling effect appears for $p_T > 1$ GeV/c probably due to the lower populated bins. The same occurs in Figure 4.64 where the distributions of IsNotE (top quartet) and IsNotH (bottom quartet) are depicted in four bins of transverse momentum p_T. Also in this case, the limited statistics available for the training produces some minor mismodeling for $p_T > 1$ GeV/c. Nevertheless, the GAN resolution model succeeds in parameterizing also the photon identification variables, offering LAMARR a viable solution to provide analysts with the calorimeter information for *signal photons*.

4.4.2 Seq2seq for particle-to-particle correlations

The reconstruction of photons from the energy deposits in the calorimeter is based on the idea that the electromagnetic shower induced by the interaction of the photon with the high-Z material composing (part of) the calorimeter is geometrically localized around the position of the interaction. From the *clusters* of energy deposits in that localized area, however, it may be difficult to recognize the number of photons close-by photons concurring to the energy deposition. As mentioned above, the photons produced in the decay of a high-momentum π^0 are almost superposed making their energy deposit in the calorimeter almost impossible to distinguish from that of a single photon. The LHCb calorimeter has been designed to identify energy deposits produced by multiple photons studying the very first part of the shower development where the contributions from multiple photons are still separated. Unfortunately, back-scattering phenomena and secondary interactions in

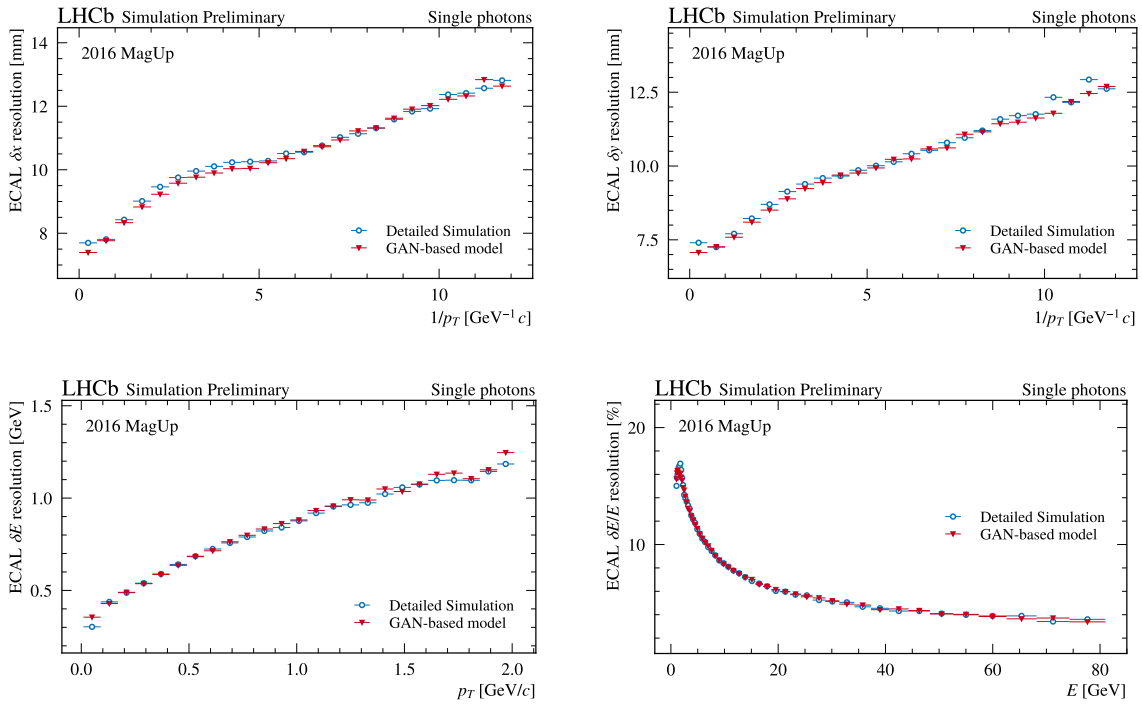


Figure 4.62: The top plots report the resolution of the cluster barycenter (x, y)-coordinates as a function of the reciprocal of the photon transverse momentum p_T. The bottom plots shows the resolution of the cluster energy as a function of the photon transverse momentum (bottom left) and the relative energy resolution as a function of the photon energy E (bottom right). The results of simulated matching clusters reconstructed as *single photons* is reported using blue circle-markers. The output of a GAN-based model trained to parameterize the ECAL resolution is depicted through red triangular-markers. It is worth noticing that neither $1/p_T$ nor p_T are part of the input variables to the GAN system, which succeeds in inferring it from MC energy and slopes.

the detectors upstream of the calorimeter can mimic multiple-photon clusters, making it frequent for the reconstruction algorithm to split the energy deposited by a single photon into two or more clusters believed to be due to different neutral particles. In addition, aiming to provide LAMARR with a flash-simulation model for electrons, it is of primary importance to reconstruct the dynamics of electron tracks that, emitting bremsstrahlung radiation while interacting with the tracking stations upstream the magnet, require to correct a posteriori the measurement of the momentum using the energy deposited within the Calorimeter system by the emitted photons, introducing important correlation effects between different particles. In these cases there is no way to simplify the n-to-m problem, since we are interested in reproducing the non-trivial correlations between the generated particles (e.g., electrons) and the reconstructed objects (e.g., electron tracks, electron clusters, photon clusters). Hence, we need a model capable of processing n-length sequences of input data to produce m-length sequences of target features. Such models belong to the family of Seq2seq approaches and are generally employed for Natural Language Processing (NLP) problems. Recently, the Seq2seq architectures have become one of the major centers of development for the Artificial Intelligence (AI) community. The *attention mechanism* proposed in Ref. [115] plays a key role in this respect, having allowed large

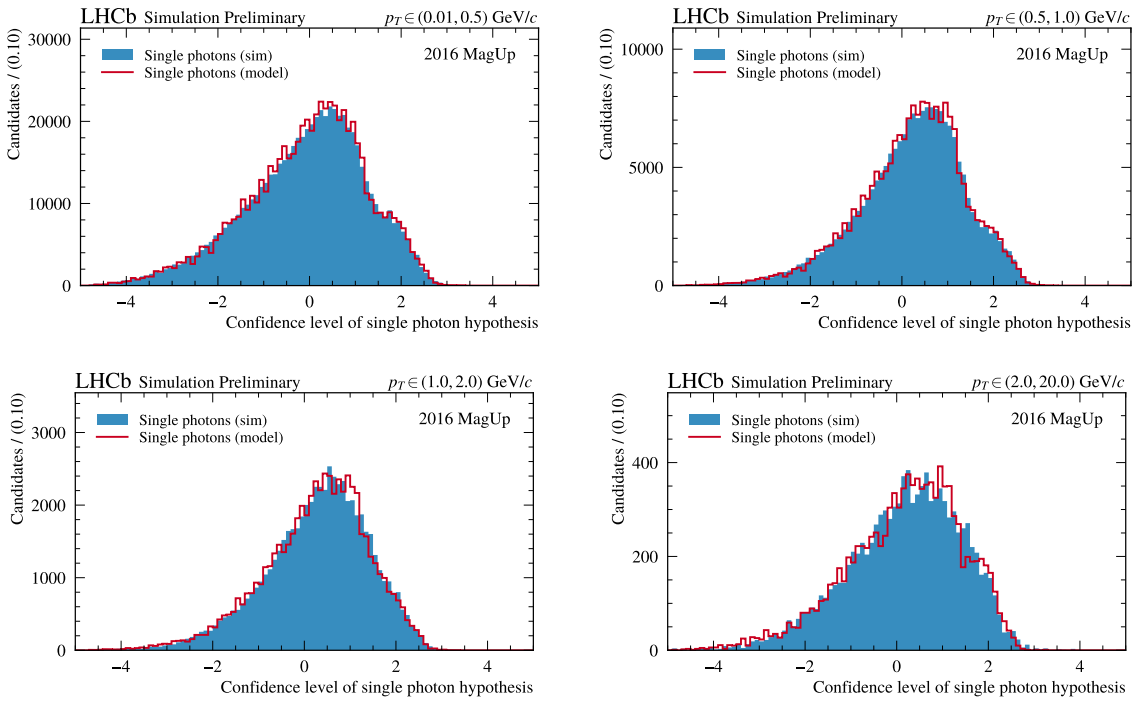


Figure 4.63: Validation plots of the GAN-based model trained to parameterize the neutral PID sequence used at LHCb to assign the *confidence level* (DLL) for single photons. The distributions obtained from Detailed Simulation for clusters reconstructed as *single photons* are represented as blue shaded histograms in four bins of transverse momentum p_T. The output of the trained generator is superimposed through red solid-line histograms.

Seq2seq models to achieve important results in the NLP context and having given rise to a real proliferation of Large Language Models (LLMs) [99–101, 104–106].

Reformulating the calorimeter simulation as a *translation problem*, we can rely on advanced Seq2seq architectures to transform a "sentence" of generated particles into a "sentence" of reconstructed objects by leaving to the attention mechanism the task of modeling all the correlations between each element of the two "sentences". Differently from the parameterizations adopted for the k-to-k problem that are allowed, by design, to treat each particle independently, the Seq2seq models aim to provide an *event-level* description of the ECAL response to enable the possibility that each (proto-)particle is correlated to all the others, in principle. Referring to this formulation, we have investigated the performance achieved by *Transformer* [115] and *Graph Neural Network* (GNN) [113, 192] models to reproduce the response of the LHCb spectrometer to traversing photons [5]. It represents a preliminary study to test the ability of Seq2seq or Graph2graph models to face the particle-to-particle correlation problem inherent in the calorimeter flash-simulation. Further developments are planned for the future to extend these studies to the electrons, whose dynamics may require to involve also the Tracking system within the parameterization of the n-to-m problem.

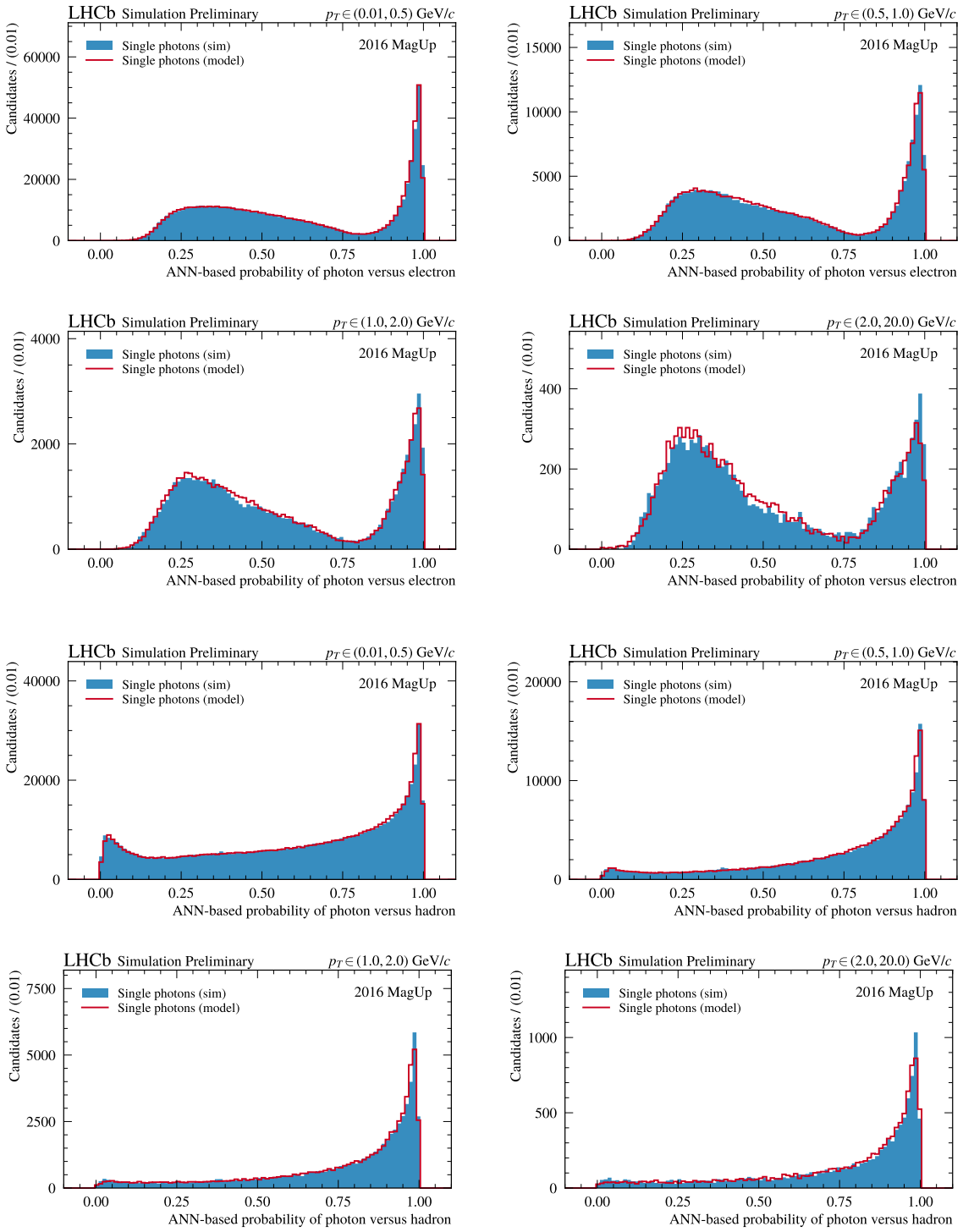


Figure 4.64: Validation plots of the GAN-based model trained to parameterize the two *multivariate classifiers* employed at LHCb to distinguish the photons from electrons (with IsNotE, shown on the top quartet of plots), or from hadrons (with IsNotH, shown on the bottom quartet of plots). The distributions obtained from Detailed Simulation for clusters reconstructed as *single photons* are represented as blue shaded histograms in four bins of transverse momentum p_T. The output of the trained generator is superimposed through red solid-line histograms.

Transformer-based calorimeter model

The Transformer model relies on an encoder-decoder architecture powered by the attention mechanism proposed in Figure 1 of Ref. [115]. The *encoder* is designed to processes the source sequence (i.e., the generated photons) and aims to parameterize the photon-to-photon correlations by producing a sequence of embeddings that describes the comprehensive event-level contribution of the generated photons to the ECAL response. On the other hand, the purpose of the *decoder* is to retrieve the reconstructed information by processing the target sequence (i.e., the reconstructed clusters) and modeling both the cluster-to-cluster and photon-to-cluster correlations. In particular, the latters result from the repeated application of the attention mechanism ($\times N$ in Figure 1 of Ref. [115]) aiming to combine the photon embeddings with the cluster features available to the decoder.

The Transformer-based ECAL model currently investigated is inspired to the architecture originally proposed by Ashish Vaswani and others [115], while the design of the skip connections follows what adopted by Alexey Dosovitskiy and others [193] for implementing the Vision Transformer (ViT). In addition, contrary to the architectures just mentioned, the ECAL Transformer (also referred to as CALOTRON) has no activation function in output and is trained to perform a *regression task*. By design, CALOTRON has no notion of the detector geometry, the underlying physics processes employed for the particle detection, nor the physics constraints defined by the Laws of Nature, such as the law of conservation of energy. Hence, training a model able to reproduce physically reasonable results is a non-trivial task, since it requires that the Transformer disposes of a faithful description of all the photon-cluster correlations. Obviously, as one might guess, the longer the source and target sequences are, more complex inferring physics constraints from data becomes. A similar problem affects also the LLMs, whose major improvements often involve the capability of processing a larger number of words, typically represented by *tokens* [99–101].

To test the validity of the CALOTRON model we decided to limit the complexity of the physics problem, leaving to future developments the design of specialized architectures able to face the sequence-length problem. In particular, we have opted to reduce the maximum length of both the source and target sequences to ease the parameterization of the photon-cluster correlations performed by the attention mechanism. To ensure the selection of the most *important* photons and clusters, each sequence was order by decreasing energy and the first 32 (16) most energetic photons (clusters) were retained. Each photon entry is described by a collection of kinematic properties, such as the (x, y)-position on the ECAL face, the momentum, the slopes t_x and t_y, and the (x, y, z)-position of the origin vertex. As a proof-of-concept, such information aims to be used for inferring a few cluster-level reconstructed features, like the barycenter (x, y)-position and the total energy collected. Despite the adoption of such precautions, let the Transformer converging to physical results remains challenging. Hence, to improve the quality of the CALOTRON output, the *canonical* training of a regression model was enhanced with the addition of an adversarial component. Similarly to what occurs for GANs, the CALOTRON system relies on the competition between two players: from one side, we have the Transformer that is trained to produce a faithful sequence of reconstructed clusters, while, on the other side, there is a binary classifier (a discriminator in the GAN jargon) that processes sequences of clusters trying to distinguish real sequences from the fake ones. However, contrary to the GAN case, the discriminator cannot be implemented via a plain FNN but requires

instead an architecture able to process sequences of features (i.e., 2-rank data) to predict probabilities. Decoder-only Transformers offer a feasible architecture to implement such a sequence-classifier, but for these preliminary studies we have opted for *Deep Sets* [194], a model already investigated by the ATLAS Collaboration for jet flavour tagging [195]. Given an m-length sequence of features $\vec{x}$, the Deep Sets architecture applies a first neural network φ to each sequence entry, sums over the entries, and then applies additional processing on the summed representation with a second neural network f, as described in the following equation:

$$y = f\left(\sum_{i=1}^{m} \varphi(\vec{x}_i)\right) \tag{4.14}$$

Letting y be the probability of being a real sequence of clusters, and $\vec{x}$ the target set of cluster features, we can rely on a Deep Sets model to improve the quality of the CALOTRON output.

To validate the performance achieved by the CALOTRON model, the kinematic distributions produced by the ECAL Transformer are compared with what results from Detailed Simulation. Since we are interested in the global performance of the model, both the CALOTRON output and the reference sequences are unrolled along the event (length) dimension, resulting into flat arrays of features. The distributions of the cluster barycenter (x, y)-position and total energy collected are depicted in Figure 4.65. Although not perfect, the preliminary results are encouraging and leave room for further developments aimed to improve the Transformer architecture, employ specialized training strategies, or perform hyperparameter optimization studies. Anyway, the most surprising result is shown in Figure 4.66, where the geometrical information and the energy signature either of the simulated clusters or resulting from the CALOTRON output are summarized in a single 2-D histogram. Despite CALOTRON has no notion of the detector geometry nor the underlying physics processes, it succeeds in reproducing the absence of active volume in the middle of the calorimeter, as well as it tends to populate more the region adjacent to the calorimeter hole, as expected from Detailed Simulation [5]. In addition to improve the quality of the predicted cluster features, future developments point to implement CALOTRON as a generative model in order to make it able to parameterize the statistical nature of the physics processes underlying the ECAL response. A possible solution is to rely on a Seq2seq GAN-based model, like the ViTGAN [196] or GigaGAN [103] models, where both the generator and discriminator are implemented via Transformer architectures.

Graph-based calorimeter model

Despite the analogy between the calorimeter simulation and the translation problem, formally speaking, the correlations between the generated photons and the reconstructed clusters are better described by a *graph-to-graph* (Graph2graph) approach, since the use of sequences assume that a notion of order among photons or clusters exists. Actually, if we consider the response of the LHCb calorimeter, this assumption is not satisfied, but rather it has been enforced to ease the parameterization of the physics constraints, like the law of conservation of energy, and to enable the use of Transformer-based models to face the particle-to-particle correlations problem. On the other hand, representing photons and clusters as the nodes of a graph, and describing their correlations in terms of links between nodes allows to reformulate the topology of the problem so that GNNs can be

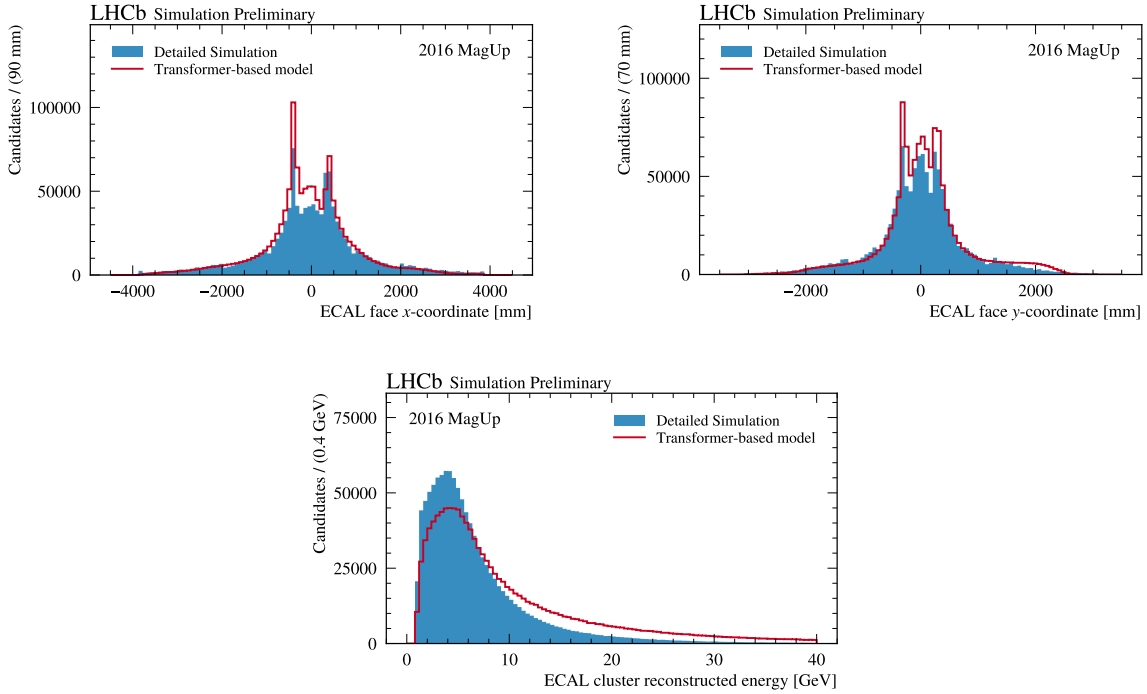


Figure 4.65: Distributions of the cluster barycenter (x, y)-coordinates (top) and the cluster energy as measured by the LHCb ECAL detector. What obtained from Detailed Simulation is reported with blue shaded histograms. The results of a Transformer-based model trained to parameterize the event-level response of ECAL when traversed by photons are shown using red solid-line histograms. To improve the quality of the Transformer output, it is trained in combination with a Deep Sets model specialized in distinguishing real events from the fake ones (*adversarial training*).

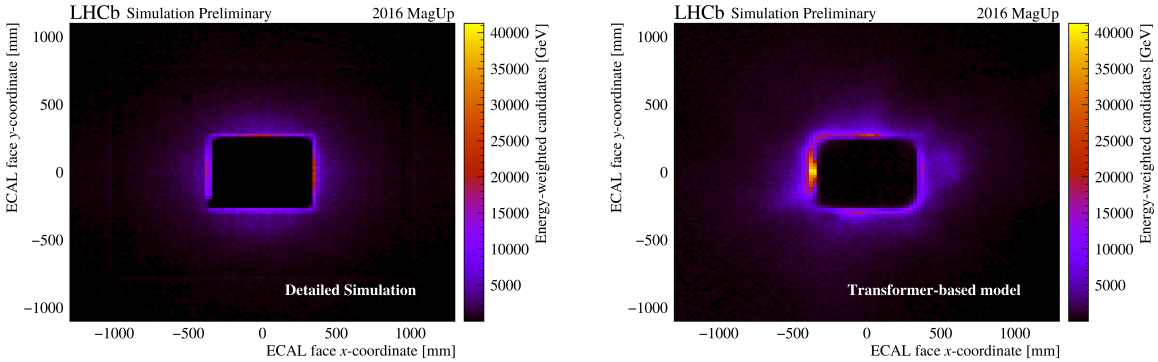


Figure 4.66: Distributions of the (x, y)-position of the reconstructed clusters on the LHCb ECAL face for a 2600×2000 mm^2 frame placed around the center. The geometrical information is combined with the energy signature by weighting each bin entry with the energy of the corresponding reconstructed cluster. What obtained from Detailed Simulation is reported on the left 2-D histogram, while the predictions of an adversarial trained Transformer model is shown on the right.

used to tackle it.

The solution investigated to parameterize the event-level response of the ECAL detector relies on a *heterogeneous graph* composed of two families of nodes:

- **Photon nodes.** Nodes representing the generated photons (input);
- **Cluster nodes.** Nodes representing the reconstructed clusters (output).

The generated photons are described by the same collection of kinematic properties adopted for CALOTRON, namely the (x, y)-position on the ECAL face, the momentum, the slopes t_x and t_y, and the (x, y, z)-position of the origin vertex. Aiming to provide a proof-of-concept that GNNs can be fruitfully used to parameterize the particle-to-particle problem intrinsic of the calorimeter simulation, we limited to a small set the cluster features to be predicted by the graph model. Hence, the output nodes are represented by relying on the cluster barycenter (x, y)-position and the total energy collected. In addition, each node has an associated *hidden state*, representing the analogue for GNNs of what *hidden layers* are for FNNs.

The edges among the photon nodes were built by following a nearest neighbourhood criteria based on the Euclidean distance between nodes in the (x, y, E)-space, and requiring that each photon node was connected to at least other two nodes. The connections among cluster nodes and between cluster and photon nodes were randomly assigned, but requiring that each cluster node was linked to at least other two cluster nodes and exactly two photon nodes. The GNN is designed so that the features of the photon nodes cannot be modified during the training procedure, although the associated *hidden states* can be updated to propagate information through the graph. On the other hand, the features of the cluster nodes are designed to be modified during the training by relying on the corresponding hidden states that, in turn, are updated using the information from the photon and the other cluster nodes connected. The functions that map node features into hidden state and vice-versa were implemented via plain FNNs, called `MapFeatures`. Based on the hidden states and by considering the connections among the nodes, the training procedure of a GNN consists of an iterative process where the information hosted in each node are propagated to the first neighbors by relying on MLPs. This *message passing* procedure can be enhanced with the attention mechanism by relying on the *Graph Attention Network* (GAT) architecture proposed in Ref. [197] architecture. The ECAL GNN-based model investigated was then implemented via by using an input `MapFeatures` to map the node features into hidden states, four GATv2 layers to update the hidden states through the message passing process, and an output `MapFeatures` to map-back the cluster hidden states into the predicted target features. Similarly to what discussed for CALOTRON, to improve the quality of the GNN output, the training of the regression model was combined with the one of a discriminator specialized in distinguishing real graphs from the fake ones. Also in this case, the discriminator was implemented via Deep Sets [194], since Eq. (4.14) does not assume any order to the input sequences of features. To provide the discriminator with information about the geometrical and physical correlations, pairs of photons and pairs of clusters are used as input entries for the Deep Set.

To validate the performance achieved by the GNN-based model, the distributions of the features predicted by the cluster nodes are compared with what results from Detailed Simulation. Since we are interested in the global performance of the model,

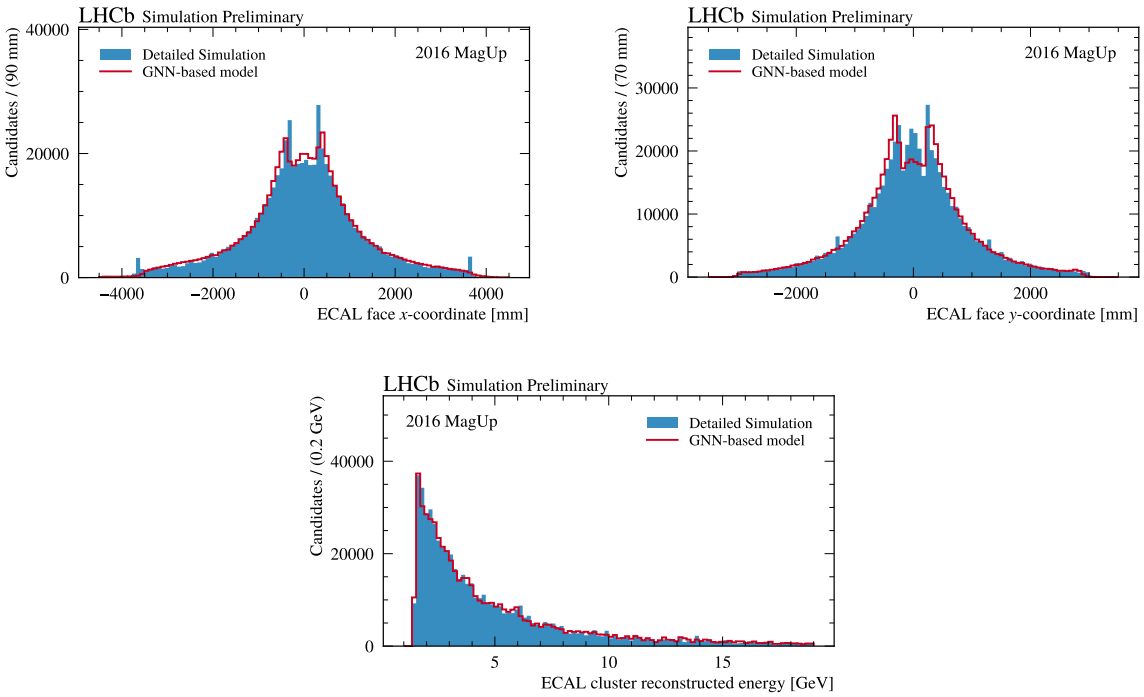


Figure 4.67: Distributions of the cluster barycenter (x, y)-coordinates (top) and the cluster energy as measured by the LHCb ECAL detector. What obtained from Detailed Simulation is reported with blue shaded histograms. The results of a GNN-based model trained to parameterize the event-level response of ECAL when traversed by photons are shown using red solid-line histograms. To improve the quality of the GNN output, it is trained in combination with a Deep Sets model specialized in distinguishing real events from the fake ones (*adversarial training*).

either the features of the cluster nodes and of the reference sample are unrolled along the event dimension, resulting into flat arrays. The distributions of the cluster barycenter (x, y)-position and the total energy collected are depicted in Figure 4.67. The agreement exhibited with the reference distributions is quite impressive, also considering that, contrary to the CALOTRON case, in this case no constraint to the maximum dimension of the graph has been applied. This demonstrates the potential of GNNs that, operating in the same topology space of the data, gains architecture-based advantages with respect to the Transformer-based models. Figure 4.68 shows that, as well as CALOTRON, also the GNN succeeds in reproducing the absence of active volume in the middle of the ECAL detector, even if graph model seems to struggle in populating more the region adjacent to the calorimeter. Further studies are planned in the future to include also electrons in the graph representation, and to implement the GNN-based model as a generative model.

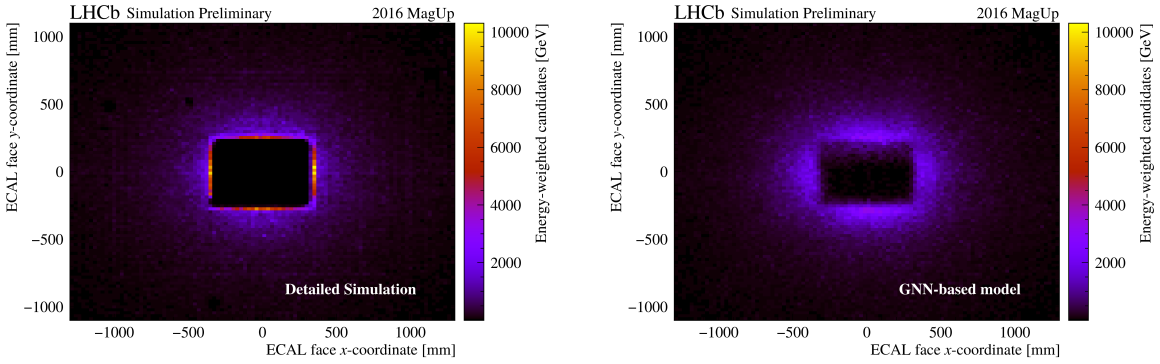


Figure 4.68: Distributions of the (x, y)-position of the reconstructed clusters on the LHCb ECAL face for a 2600×2000 mm^2 frame placed around the center. The geometrical information is combined with the energy signature by weighting each bin entry with the energy of the corresponding reconstructed cluster. What obtained from Detailed Simulation is reported on the left 2-D histogram, while the predictions of an adversarial trained GNN model is shown on the right.

5 Integration, validation, and future of the Lamarr project

The software stack developed for High Energy Physics is composed of experiment-independent libraries implementing the basic building blocks to include custom data structures, such as ROOT trees, and physics-motivated models, such as those made available by Pythia and EvtGen, used by different experiments to implement their own frameworks based on the different specification. This chapter investigates the deployment of machine learning models, trained with libraries such as Keras and TensorFlow, in the LHCb software stack, focusing on the solution identified for the flash simulation, presenting the additional requirement of offloading tasks to neural-network models with the lowest possible latency. The validation of the resulting software project, combining Pythia, EvtGen and Lamarr is also presented taking as an example the simulation of $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ *decays.*

5.1 Deploying neural networks in HEP

The application of machine learning algorithms in the field of High Energy Physics (HEP) has a long and glorious tradition for what concerns data analysis and statistical inference [198], while attempts of employing such techniques to speed up the code execution are more recent and, as discussed in Chapter 1 and longly demonstrated in Chapter 4, mainly flourishing in the field of *Fast Detector Simulation*. Traditionally, simulating the response of large and complex detectors to traversing particles is extremely expensive in terms of computing resources since it requires to reproduce accurately the quantum interactions of each particle with the material the detectors are built with, which include the generation of new particles, and results in cascade phenomena known as particle showers. Fast-simulation approaches aim to reduce the computational cost by replacing parts of the computation with statistical models, possibly defined as machine learning algorithms trained on the detailed simulated samples or by relying on acquired calibration data [199]. A more radical approach is the one proposed by flash-simulation techniques that provide parameterizations encoding in one go both the detection and reconstruction steps typically performed by Detailed Simulation softwares, hence offering the best performance in terms of CPU cost reduction.

The deployment of machine learning (ML) techniques in such computational intensive,

highly branched scenario, however, is not a trivial task. Different particles in the same collision event, for example, may undergo different physics processes and may require the evaluation of different models to provide a parameterized simulation, either based on fast or flash approach. On the other hand, modern machine learning frameworks are designed for high-level languages, such as Python, and achieve CPU performance by defining batches of data where the same sequence of operations is performed on multiple data entries, which is rarely the case in the context of HEP simulation (see Section 1.3).

A further challenge is related to the slow development cycle of the large applications used for Detector Simulation, and their distribution through the computing grids, such as WLCG, where they are typically executed. On the other hand, the continuous development and tuning of ML-based parameterizations of (parts of) the detector requires the ability to plug new models in the Simulation at runtime, without depending on a new release of the full software stack. This can be hardly achieved if the machine learning models are compiled together with the main application, even if retaining the possibility of loading optimized parameters from files.

Lastly, long dependency chains are considered dangerous for HEP software projects that are intended to serve large communities of researchers for several decades. However, the rapid evolution of machine learning algorithms in terms of both architectures and training strategies moves developers to adopt the ever-changing high-level packages [167, 168, 175, 186, 200, 201], whose life cycles are strongly connected with the latest results achieved by the Artificial Intelligence (AI) community.

Hence, during my Ph.D., I have contributed to the development of a simple tool designed to simplify the deployment of models implemented either via scikit-learn [175] or Keras [167] APIs. The tool, called `scikinC` [7], allows to *transpile*[1] machine learning models trained in Python, in plain C functions by taking as input a single data entry. The transpiled functions can then be compiled as shared objects and dynamically linked to the main applications, with negligible overhead. The transpiled functions are designed to run in the same thread where they are called and, being stateless, they are thread-safe by design.

The rest of this Section is dedicated to describe the `scikinC` tool and provide a simple example usage for Fast Detector Simulation. In particular, Section 5.1.1 reports a brief review on the state-of-the-art and related projects that face the problem of machine learning models deployment. A detailed description of the strategy adopted by the `scikinC` tool to tackle the same problem is depicted in Section 5.1.2. Finally, Section 5.1.3 presents a simplified application of `scikinC` for Fast Detector Simulation.

5.1.1 State-of-the-art and related projects

The to-go deployment option for machine learning is the ONNX runtime [202]. ONNX, originally an open format for neural network exchange, provides today extensions to deploy models trained with several frameworks, including scikit-learn [175] or Keras [167]. Since the focus of ONNX is on interoperability and performance, it provides a runtime optimized for various platforms including multiple CPUs, GPUs and other hardware accelerators. The ONNX runtime has APIs for several languages including C and C++, but unfortunately

[1] A *transpiler* is a source-to-source compiler which takes the source code of a program written in one programming language as its input and produces the equivalent source code in another programming language. Read more on `https://en.wikipedia.org/wiki/Source-to-source_compiler`.

the need for a runtime designed to get the most from parallel computation on multiple threads introduces a small overhead and sets some minor thread-safety issues. These considerations make ONNX ideal for a number of applications in HEP as long as the model is complex enough or the batches can be large enough to make the overhead negligible, this is especially true for reconstruction or analysis tasks, and for some special task in Fast Detector Simulation, such as simulating the energy deposits in a calorimeter in a collision event.

To avoid the need for a runtime and aiming at a significant reduction of the amount of external dependencies for long-term scientific applications, the HEP community developed the LWTNN (*Lightweight Trained Neural Network*) project [203], providing a format to exchange machine learning algorithms alternative to ONNX, which can then be interpreted and executed as part of the main application. LWTNN is a quite popular in the HEP community because of its simplicity.

A more drastic approach to reduce the complexity of the dependency tree is to separate the applications in charge of running the main task and the machine learning inference infrastructure in two different processes communicating via sockets. With this approach, the inference application acts as a server specialized in machine learning applications, providing optimal access to hardware accelerators and building batches, possibly combining the requests from several concurrent instances of the main application. The server application does not even need to run on the same machine and can be written in Python in the exact same environment as used for training. *Machine Learning as a Service* (MLaaS) [204, 205] is a fascinating opportunity for extremely computing-intensive tasks, where the large overhead due to the inter-process communication becomes negligible in front of the speed-up obtained offloading the inference to specialized hardware accelerators.

At the opposite end, the HEP community is developing tools to compile machine learning models to optimize the inference of small models on tiny batches, which still cover an impressing amount of use-cases in our simulation, reconstruction and analysis applications. Compiling the models also helps to reduce the memory footprint of the inference routines, which is especially important when multiple instances of the application run in parallel as different threads or processes on the same machine. The most advanced transpiler for machine learning models is being developed by the CERN ROOT team as part of the *Toolkit for Multivariate data Analysis* (TMVA) [48] and is called SOFIE (*System for Optimized Fast Inference code Emit*) [206]. SOFIE extends TMVA by introducing the ability to parse ONNX files and convert them into C++ code. The generated code can be compiled with a C++ compiler with some BLAS[2] libraries as the only dependency. While an extremely promising tool providing conversion templates for a large variety of ONNX operators, SOFIE comes as part of the ROOT framework, which sets serious limitations in terms of portability, especially when targeting the cloud-based Python environments commonly used for training.

Outside of the HEP community, there are several active projects to develop compiler for deep neural networks built on top of LLVM[3], for example TVM [207], nGraph [208],

[2]Basic Linear Algebra Subprograms (BLAS) is a specification that prescribes a set of low-level routines for performing common linear algebra operations, such as vector addition or matrix multiplication. Read more on `https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms`.

[3]LLVM provides compilation technologies to develop a front-end for any programming language and a back-end for any instruction set architecture. More on `https://en.wikipedia.org/wiki/LLVM`.

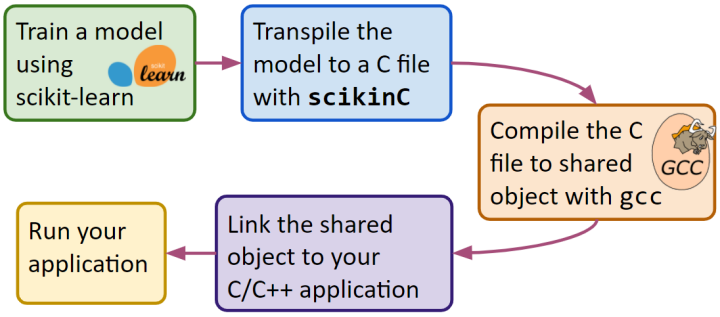


Figure 5.1: Sketch of the working principle of `scikinC`. Figure reproduced from Ref. [7].

Tensor Comprehensions [209], Glow [210] and Extended Linear Algebra (XLA) [211]. A comprehensive review of the different approaches adopted in the various projects can be found in Ref. [212]. Each of these projects produces compiled and optimized representations of deep neural networks that can be linked to applications developed in several languages, which will then depend on runtimes or, at least, on external kernels implementing the actual computation.

Interesting experiences were reported in the field of machine learning deployment on microcontrollers and real-time processing, requiring extremely low latency and being usually designed for tiny batches. We report in particular on two packages, `emlearn` [213] and `keras2c` [214], providing transpilers for several scikit-learn and Keras models to C. Conceptually, these recent packages are similar to `scikinC`, with minor differences in the implementation choices rather than in the design principles. In particular, `keras2c` is more mature and complete than `scikinC` on the Keras models, which is more focused on scikit-learn models and their distribution, as discussed in the next Section.

5.1.2 The `scikinC` implementation

As mentioned above, `scikinC` was primarily intended for offering a simple solution to integrate small to medium-sized machine learning models within a Fast Detector Simulation framework.

The models, preprocessed and trained relying on scikit-learn and/or Keras APIs, are converted to C code using the parser made available by `scikinC`. The generated source code can be immediately compiled into a *shared object* using a C compiler, such as GCC, and then dynamically linked to the main application using standard C libraries. In the transpiling process, `scikinC` encodes directly in the C files the weights of the models, hence avoiding dependencies on configuration files. Dynamic linking enables selecting the model to use for inference at runtime via the option files or the command-line arguments used to configure the main application. This feature gives access to frequent updates of the models (for example for validation purpose), without the need of a new release of the whole software stack in production. The procedure to follow for transpiling a scikit-learn model in C, and then compiling and linking it in a C/C++ application is schematically depicted in Figure 5.1.

The transpiling procedure consists of filling template C implementations of the forward pass (i.e., the *inference*) of the supported models implemented in scikit-learn or Keras by accessing the attributes and properties exposes by the class APIs. The `scikinC` modular

Class	*Module*	*Implementation*	*Test*
`MinMaxScaler`	preprocessing	✔	✔
`StandardScaler`	preprocessing	✔	✔
`QuantileTransformer`	preprocessing	✔	✔
`FunctionTransformer`	preprocessing	✔	✔
`ColumnTransformer`	preprocessing	✔	✔
`GradientBoostingClassifier`	ensemble	✔	✔
`Pipeline`	pipeline	✔	◌

Table 5.1: List of the scikit-learn models that `scikinC` can transpile to C files.

Class	*Module*	*Implementation*	*Test*
`Sequential`	models	✔	✔
`Functional`	models	◌	✘
`Dense`	layers	✔	✔
`Dropout`	layers	✔	✔
`Leaky ReLU`	layers	✔	✔
`PReLU`	layers	✔	✔
`Softmax`	layers	✔	✔
`sigmoid`	activations	✔	✔
`tanh`	activations	✔	✔
`relu`	activations	✔	✔
`linear`	activations	✔	✔

Table 5.2: List of the Keras models that `scikinC` can transpile to C files.

design, involving a separate template for each model, simplifies the extension of the package to support additional models.

The `scikinC` template implementations currently provided for scikit-learn and Keras models are reported in Tables 5.1 and 5.2, respectively. Most of the models are completely implemented and successfully tested. The sole exceptions are the scikit-learn's `Pipeline` whose deployment breaks when one relies on pipelines of pipelines, and the Keras Functional API that is implemented only to deploy "feed-forward" neural networks equipped with skip connections [111].

All the functions defined by `scikinC` are intended for single entries as no support is provided for batch inference, and share the same signature:

```
float *functionname (float* output, const float* input);
```

where `input` and `output` are arrays containing the input features of the model and the

predictions obtained. The pointer to the output array is also returned to ease inline operations on the result of the computation. The number of input features and produced output variables is fixed by the model and is supposedly known by the client application. Passing the size of the arrays as additional arguments is therefore unnecessary.

During code generation, `scikinC` disables the name mangling of the linker symbols, assigning to the functions representing the entry point user-defined names and symbols for the evaluation of each model. Combining the standard function signature and user-defined linker symbol, the function can be accessed easily by using the function `dlopen` which is part of standard C libraries.

During the development of `scikinC`, we became aware of two ongoing projects with similar, but not completely overlapping, goals: `emlearn` targeting microcontrollers and `keras2c` targeting real-time data processing. The latter introduces a more general signature for the transpiled C functions, including the shape of the input and output tensors. In the future, `scikinC` will share the same signature in order to make switching from a `keras2c` model to a `scikinC` model transparent for the client application. On the same line, we will consider offloading the conversion of Keras models to `keras2c` focusing on models trained with scikit-learn, aiming at a complete toolbox of C transpilers for machine learning models with the lowest latency. The `scikinC` codebase is released under MIT license and its Git repository is publicly available on GitHub[4].

5.1.3 Example application to a simplified fast-simulation

To show an end-to-end real application of `scikinC`, we propose a (simplified) example on Fast Detector Simulation. Considering to have access to a (small) sample of simulated data from some experiment, we aim to model the response of the detector in terms of *efficiency* and *resolution*. Once trained, the models should be integrated in the C++ framework this hypothetical experiment is equipped with for searching new physics effects, so enabling quick checks on whether the experiment would be sensitive to the Physics Model to test.

We consider the decay of 180 GeV/c neutral kaons into three pions, namely

$$K^0 \to \pi^+\pi^-\pi^0$$

The experiment relies on a calorimeter to reconstruct the π^0, but this requires neutral pions with a momentum higher than 70 GeV/c to reject some background. For simplicity, we assume that the efficiency of this requirement is 100% for neutral pions above 70 GeV/c and zero otherwise. This requirement introduces an efficiency term on the Dalitz plot[5] that might compromise the sensitivity of the experiment to the Physics Model, which will be further investigated by studying the invariant mass of the two charged pions: $m(\pi^+\pi^-)$. The latter variable is reconstructed with a rough spectrometer reconstructing the momentum of the charged pions with an error

$$\frac{\delta_p}{p} \approx 1\,\mathrm{GeV}^{-1} \cdot p$$

For simplicity, we assume the same error on the momentum also for the neutral pion.

[4] `https://github.com/landerlini/scikinC`

[5] The *Dalitz plot* is a two-dimensional plot used to characterize the kinematics of a three-body decay aiming at representing the relative frequency of different combination of its products. Read more on `https://en.wikipedia.org/wiki/Dalitz_plot`

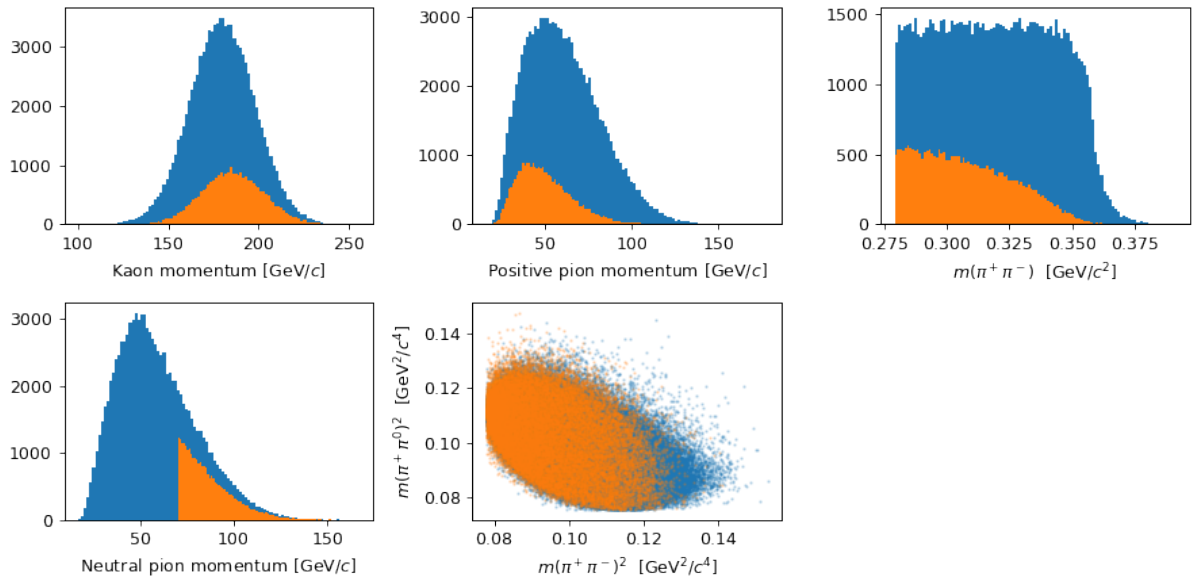


Figure 5.2: Representation of simplified simulated detector effects that we aim to parameterize with machine learning algorithms. The generated events prior of any reconstruction and selection are reported through blue filled histograms. The reconstructed and selected events are depicted by using orange filled histograms. The Dalitz plot is also reported via scatter plot following the same color labeling. Figure reproduced from Ref. [7].

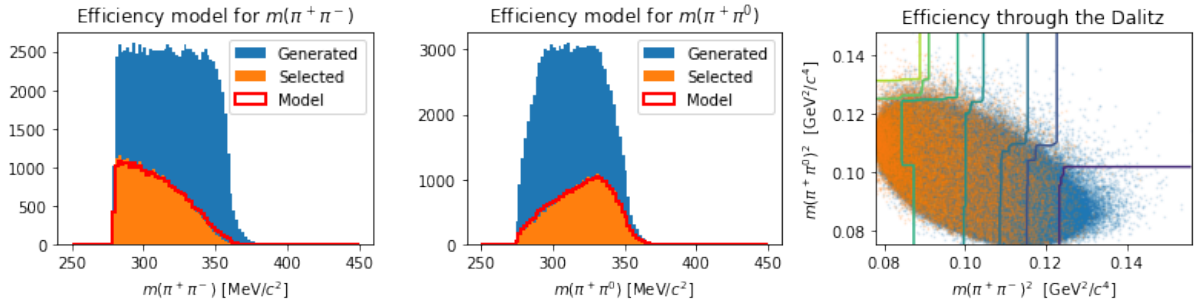


Figure 5.3: Representation of the detector *efficiency* parameterization as modeled by a GBDT-based classifier implemented and trained using the scikit-learn's `GradientBoostingClassifier` model. Figure reproduced from Ref. [7].

The effect of the requirement on the momentum of the neutral pion on the other variables of the system is depicted in Figure 5.2: the suppression of higher values of $m(\pi^+\pi^-)$ is clearly visible in the Dalitz plot (orange scatter plot).

To model the efficiency in a quick and reliable way, we train a *Gradient Boosted Decision Tree* (GBDT) [215] to predict the probability that an event will be reconstructed given the coordinates of the Dalitz plot: $m(\pi^+\pi^-) \perp m(\pi^+\pi^0)$. Conceptually, this is equivalent to what is discussed, for instance, in Section 4.2.2 where the geometrical acceptance of the LHCb spectrometer is parameterized by a neural network trained to perform a binary classification task. In this simplified example, we rely instead on a GBDT-based classifier. The performance of the trained model is reported in Figure 5.3.

Similarly, the resolution function associated to each entry in the Dalitz plot can be

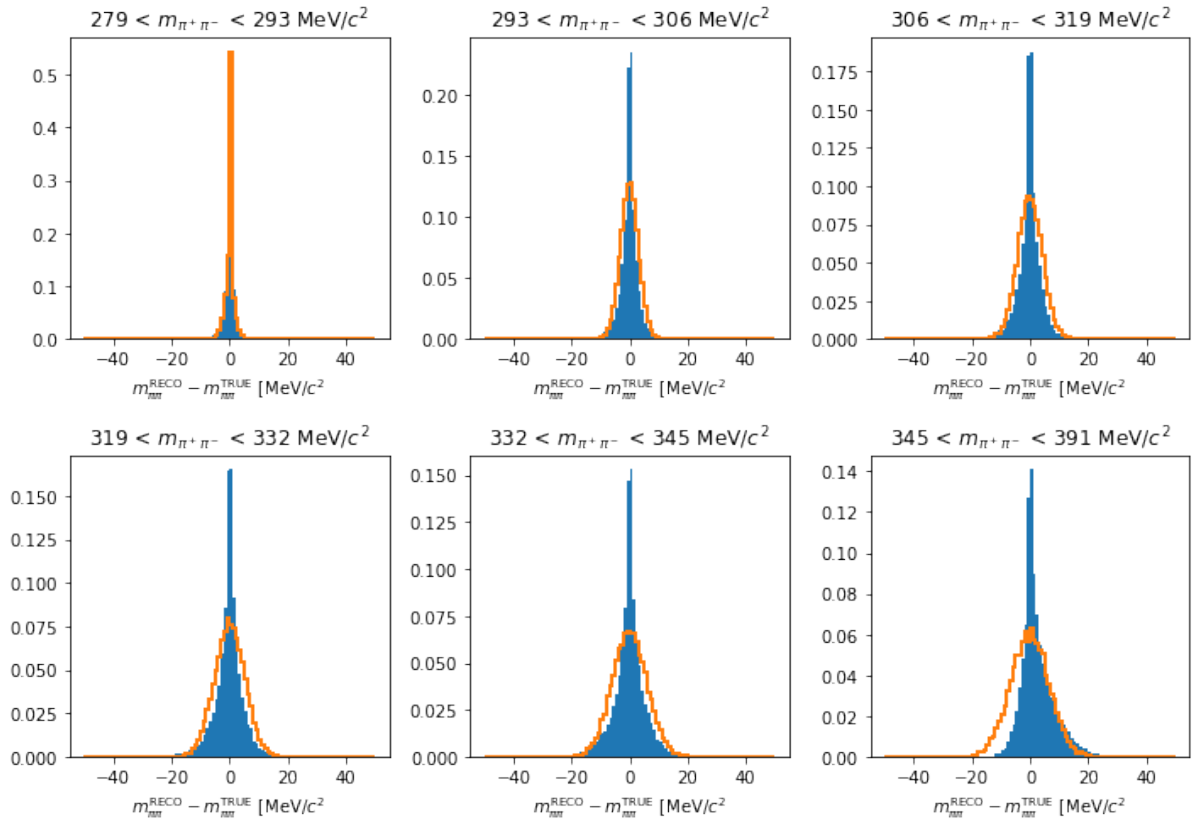


Figure 5.4: Representation of the detector *resolution* parameterization as modeled by a FNN predicting the standard deviation of the reconstruction error (i.e., $\Delta x = x_{\mathrm{reco}} - x_{\mathrm{true}}$) distributions for the $m(\pi\pi)$ variables. The ground-truth resolution model for the $m(\pi^+\pi^-)$ variable is represented as a blue filled histogram, while the FNN-based Gaussian prediction is shown as an orange solid-line. Figure reproduced from Ref. [7].

parameterized with an neural network. Since the momentum uncertainty depends on the momentum itself, we expect that the uncertainty on the invariant masses varies through the Dalitz plot, and it is interesting to model. To simplify the setup, a minimal (2-layer) feed-forward neural network (FNN) is trained to predict the standard deviation of the experimental error distributions on the reconstructed $m(\pi\pi)$ as a function of the Dalitz plot coordinates. To ease the training of the FNN, the input variables are preprocessed using the scikit-learn's `MinMaxScaler` class. The result of this oversimplified parameterization is depicted in Figure 5.4.

In real experiments, *deep generative models*, like Generative Adversarial Networks (GAN), are typically employed to parameterize resolution functions with neural networks. The training of such more advanced algorithms, however, is beyond the scope of this example, since their deployment would happen exactly in the same way as discussed below for the simplified model just described.

Once the GBDT-based efficiency model, the preprocessing step, and the FNN-based resolution parameterizations are defined, `scikinC` can transpile them to C file as follows:

```
1 import scikinC
2
3 converted_code = scikinC.convert(
```

```
    {
        "efficiency_model": efficiency_model,
        "resolution_prepr": preprocessing,
        "resolution_model": resolution_model,
    }
)
```

Here, the dictionary keys, placed within the quotes, will be used as linker symbols in the compiled shared library, while the dictionary values, placed at right of the colon signs, are the scikit-learn and Keras objects defining the trained models. The content of `converted_code` is C code containing the instructions for the evaluation of the converted models. Saving `converted_code` to a file, for example `converted.C`, it can be immediately compiled to a shared object as

```
gcc -o converted.so --shared -fPIC -Ofast converted.C
```

The binary version of the models representing the efficiency and resolution of the simulated detector are now stored in a shared library, named `converted.so`, that can be dynamically linked to other applications, for example using the `dlopen` function. A minimal C application employing the parameterization for detector efficiency and resolution as modeled with machine learning algorithms is reported in the following. It should be noticed that both the name of the shared object file and of the linker symbol are strings that can be taken as inputs at runtime. Hence, changing the parameterization of the efficiency from a GBDT to a FNN, for example, could be achieved without recompiling the main application.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

// Import the dlfcn header
#include  <dlfcn.h>

// Define the signature for scikinC functions and name it mlfunc
typedef float *(*mlfunc)(float *, const float*);

int main (int argc, char* argv[])
{
    float m_pipi = atof(argv[1]);
    float m_pipi0 = atof(argv[2]);

    // Open the converted.so shared object
    void *handle = dlopen ("./converted.so", RTLD_LAZY);

    // Load the functions from the shared object based on their names
    mlfunc efficiency_model = (mlfunc) dlsym ( handle              ,
                                               "efficiency_model" );
    mlfunc resolution_model = (mlfunc) dlsym ( handle              ,
                                               "resolution_model" );
    mlfunc resolution_prepr = (mlfunc) dlsym ( handle              ,
                                               "resolution_prepr" );

    float efficiency [1];
    float input[]  = {m_pipi, m_pipi0};
```

```
    // Once loaded, trained models can be called as normal functions
    efficiency_model (efficiency, input);
    printf ("Efficiency: %.3f\n", efficiency[0]);

    float preprocessed [2], resolution [2];
    resolution_prepr (preprocessed, input);
    resolution_model (resolution, preprocessed);

    // ...

    return 0;
}
```

5.2 Integration and validation of LAMARR

The detailed simulation of the proton-proton collisions and, in particular, the computation of the physics processes occurring within the detector material when traversed by particles, is the major consumer of CPU resources at LHCb [216], having used more than 90% of the total computing budget during LHC Run 2.

The LHCb simulation software stack GAUSS [61], is based on GAUDI [53] and combines MC generators simulating the proton-proton collisions, such as PYTHIA8 [25, 158], with EVTGEN [58], specialized in modeling heavy hadron decays, and GEANT4 [59, 60], taking care of the detailed simulation of radiation-matter interactions in the detector material to compute the energy released by the traversing particles. The energy deposited in the active volumes is then converted into electronic signals by using the BOOLE application for a seamless integration with the reconstruction algorithms applied to the acquired data as implemented in BRUNEL [171].

Among the many solutions that the LHCb Collaboration is investigating to reduce the pressure on CPU resources [64, 90, 161, 162, 177, 190], techniques implementing the flash-simulation paradigm represent the most radical approach, inferring the results of the detector simulation, digitization, and event reconstruction relying on ML-based models or simpler parameterizations.

LAMARR [3–5] is the *flash-simulation* option for LHCb, designed to provide the experiment with the fastest solution for simulation production. As deeply discussed in Chapter 4, LAMARR relies on a set of neural-network-based modules to parameterize the experimental errors introduced by the quantum interactions of particles with the detector materials and by the reconstruction algorithms employed to build analysis-level quantities, like track momenta or PID likelihoods. The LAMARR parameterizations are arranged into two main pipelines (represented in Figure 4.4) that, taking as input particles resulting from the MC physics generators, reproduce the high-level response of the LHCb experiment when traversed by charged (e.g., muons, pions, kaons, or protons) and neutral particles (like photons or π^0). While the latter case is currently under investigation to find the best-suited model to face the particle-to-particle correlation problem discussed in Section 4.4, the parameterizations of the pipeline for charged particles are at a more mature stage and are ready to be further validated, assessing the performance achieved once the information provided by the physics generators flows throughout the pipeline up to compute analysis-level quantities.

Recalling that LAMARR has no notion of the geometry of the spectrometer nor of

the physics processes underlying the detection strategies, the charged particle pipeline consists of a series of consecutive parameterizations aimed at particle selection, kinematic correction due to resolution effects, and computation of higher-level variables produced by the reconstruction algorithms. Following the formalism employed at LHCb, the various modules are split into two macro-categories, referred to as *Tracking* and *Particle Identification* models. Thus, given a set of generated charged particles, the selection of which of those cross the active volumes of the detector and are then correctly reconstructed as tracks, relies on two neural networks whose performance have been discussed in Sections 4.2.2 and 4.2.3, respectively. The selected subset of reconstructed tracks is still described by the generator-level information, that can be *corrupted* as expected from the resolution effects by relying on two GANs trained to parameterize the tracking reconstruction procedure and detailed in Section 4.2.4 and 4.2.5. Disposing of analysis-level kinematic information, namely the high-level response of the LHCb Tracking system, one more set of parameterizations, either implemented via GANs or plain neural networks, can be employed to couple each flash-simulated track with the corresponding PID information, as it has been discussed in Section 4.3. Hence, the pipeline for charged particles provided by LAMARR converges at the *persistency* step, where the parameterized variables are converted into a data format as similar as possible to the one adopted for reconstructed data, namely the one produced by the BRUNEL application.

To validate such a pipeline, the integration of LAMARR within GAUSS is required. This allows LAMARR to be interfaced with standard MC physics generators, to further process (decay reconstruction) the flash-simulated samples with the analysis applications in use by the Collaboration, and to distribute these computations through the WLCG computing nodes. On the other hand, ensuring a fast development cycle is a key feature for any ML-based framework, since ever-changing architectures and training strategies can easily outperform the predecessors. Consequently, the AI community is extremely versatile in terms of software technologies, in total contrast with the decades tradition typical of software stacks in HEP. Hence, to take the best of both worlds, LAMARR relies on the transcompilation approach offered by `scikinC` [7] which, as discussed in Section 5.1.2, enables a low-latency inference for models trained in scikit-learn and Keras. Notably, all the parameterization employed within the charged particle pipeline rely on preprocessing strategies available in `scikinC`, and are implemented via either Keras Sequential models or FNNs equipped with skip connections [111], both provided by the C converter, too. The shared library resulting from the compilation procedure can be dynamically linked to the main application, GAUSS in this case, and easily distributed through the WLCG nodes by using `cvmfs` [97]. Interestingly, this approach gives the way to a CI/CD practice where each of the parameterizations can be replaced with an updated one, whatever is the architecture or the training strategy, as long as `scikinC` implements it, and the number of input and output variables remains the same.

The flash-simulation philosophy at the base of the LAMARR framework has been preliminary validated by comparing the distributions resulting from a pipeline of ML-based models and the ones obtained with standard simulation strategies. In particular, we discuss here the validation studies performed using simulated $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays with $\Lambda_c^+ \to pK^-\pi^+$. The semileptonic nature of the Λ_b^0 decay requires to dispose of a LHCb-tuned physics generator specialized for heavy hadron decays, such as EVTGEN, hence highlighting the necessity of having a simulation framework integrated within GAUSS. In addition, as part of the *calibration samples* used for unbiased studies of the

LHCb PID system performance [49], the decay channel has been deeply studied by the Collaboration and, interestingly, it includes the four charged particle species currently parameterized within LAMARR (i.e., muons, pions, kaons, and protons) in its final states. Lastly, it should be pointed out that the training of the ML-based models was performed with an independent dataset obtained by combining samples from the Detailed Simulation of several heavy hadron decays, including a negligible fraction of $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays. A description of the datasets involved in the comparison follows:

1. A sample obtained from Detailed Simulation generating 1 million Λ_b^0 with PYTHIA8, simulating their decay with EVTGEN, and processing the daughter particles with GEANT4, BOOLE, BRUNEL and BENDER [217]. This sample is part of the official and centralized LHCb productions and serves as a reference.

2. A sample obtained generating 1 million Λ_b^0 with PYTHIA8, simulating their decay with EVTGEN, and processing the daughter particles with LAMARR and BENDER.

3. A sample obtained generating 100 million Λ_b^0 single particles with a *particle-gun*, with parameterized momentum and rapidity spectra and decaying it with EVTGEN, and processing the daughter particles with LAMARR and BENDER.

All samples are produced for the 2016 LHCb data taking conditions with the magnet polarity pointing upward. The configuration of the BENDER application is exactly the same for the processing of the first two datasets, while a simplified truth-matching procedure has been adopted to speed-up the analysis of the large datasets obtained with the particle-gun.

The validation campaign discussed below dates back to July 2022 [3], and hence does not rely on the latest version of the ML-based parameterizations detailed along Sections 4.2 and 4.3, but rather on a set of (sub-optimal) models briefly described in Ref. [218]. However, although the lower performance, the models succeed in describing the high-level response of the LHCb detector when traversed by the charged particles resulting from $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays with $\Lambda_c^+ \to p K^- \pi^+$, as reported in the following Sections. Among the major difference with respect to the parameterizations discussed in Chapter 4, there are the use of GBDT in place of the neural networks for the efficiencies (i.e., geometrical acceptance, tracking efficiency, and `isMuon` criterion), the use of only Keras Sequential models (no skip connections) for the GAN players, and the absence of neither particle species nor track classes within the Tracking parameterizations. The latter difference violates the terms for the adoption of a CI/CD practice, as the latest models have a different number of input and/or output variables.

Most of the third year of my Ph.D. has been spent investigating the particle-to-particle correlation problem needed for the flash-simulation of the LHCb response to neutrals, and contributing to the development of SQLAMARR (which will be discussed in Section 5.3), the stand-alone version of the LAMARR framework. Aiming at moving new validation studies on the frameworks under development, and including also the parameterization of the ECAL detector for the photons, no other validation campaigns has been performed so far. At the time of writing, both the new logic of the flash-simulation framework (i.e., SQLAMARR) and the calorimeter models are no mature enough to be presented here, and will be instead reported in future publications.

5.2.1 Validation studies

As mentioned above, the decay channel chosen for the validation study is part of the calibration samples, namely datasets collected by LHCb where the kinematics of the decay candidates is so that allows an unambiguous identification of one of the daughters, without the use of the PID information, hence providing an unbiased sample for assessing the performance exhibited by the experiment in identifying such a particle [49]. Notably, $\Lambda_c^+ \to pK^-\pi^+$ is the decay channel employed for the performance studies of the protons. Although the comparison plots rely on detailed simulated samples, hence no needs for an unbiased analysis should be necessary, along the validation plots we will mainly focus on the distributions of the protons in consistency with the sample employed.

Kinematics and invariant masses

To test the ability of the Tracking models to reproduce the resolution effects introduced by the detection and reconstruction steps, Figure 5.5 reports the distributions of the momentum p (top) and the pseudorapidity η (bottom) for protons produced via $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays with $\Lambda_c^+ \to pK^-\pi^+$. The reference distributions, represented as cyan shaded histograms, result following the standard Detailed Simulation strategies, in particular relying on GEANT4 for computing the radiation-matter interactions (e.g., multiple scattering) occurring within the detector material. On the other hand, the pipeline defined in LAMARR is employed to transform the generator-level kinematic information provided by either PYTHIA8 and EVTGEN (orange markers), or a particle-gun model (purple markers). Interestingly, since the effects on the detector performance due to the occupancy are enclosed in the parameterizations, very similar results are produced by LAMARR running on these two configurations. In both the case, the agreement exhibited with respect to the reference distributions is very good, demonstrating the validity of the LAMARR models to *reconstruct* tracks and to parameterize the *errors* introduced by the resolution effects.

While looking at the kinematic distributions (p, η) we can assess the quality of the variables directly provided by LAMARR, evaluating the models performance on quantities that require further (decay-level) reconstruction allows to challenge the flash-simulation framework. In particular, Figures 5.6 and 5.7 report the distributions for the invariant mass of Λ_c^+ and $\Lambda_c^+ \mu^-$ candidates, respectively. The excellent agreement exhibited between the distributions in both the configurations (PYTHIA8 or particle-gun) and the reference distributions confirms the quality of the resolution parameterizations and, in addition, validates the use of flash simulated samples within the analysis applications, such as BENDER, employed at LHCb.

Tracking uncertainties

As discussed in Section 4.2.5, the uncertainty on the reconstructed trajectories of particles is represented in the LHCb Event Model as the covariance matrix obtained from the fit procedure by minimizing the χ^2 of the track passing through the hits in the tracking detectors. The correctness of the uncertainty on the reconstructed quantities of the tracks is of primary importance to assess the consistency of a track with a vertex. For example, when reconstructing a b-hadron from its decay to multiple hadrons it is very effective to reject all the charged particles that are consistent with being produced in the primary interaction. Aiming at assessing the performance achieved by LAMARR in

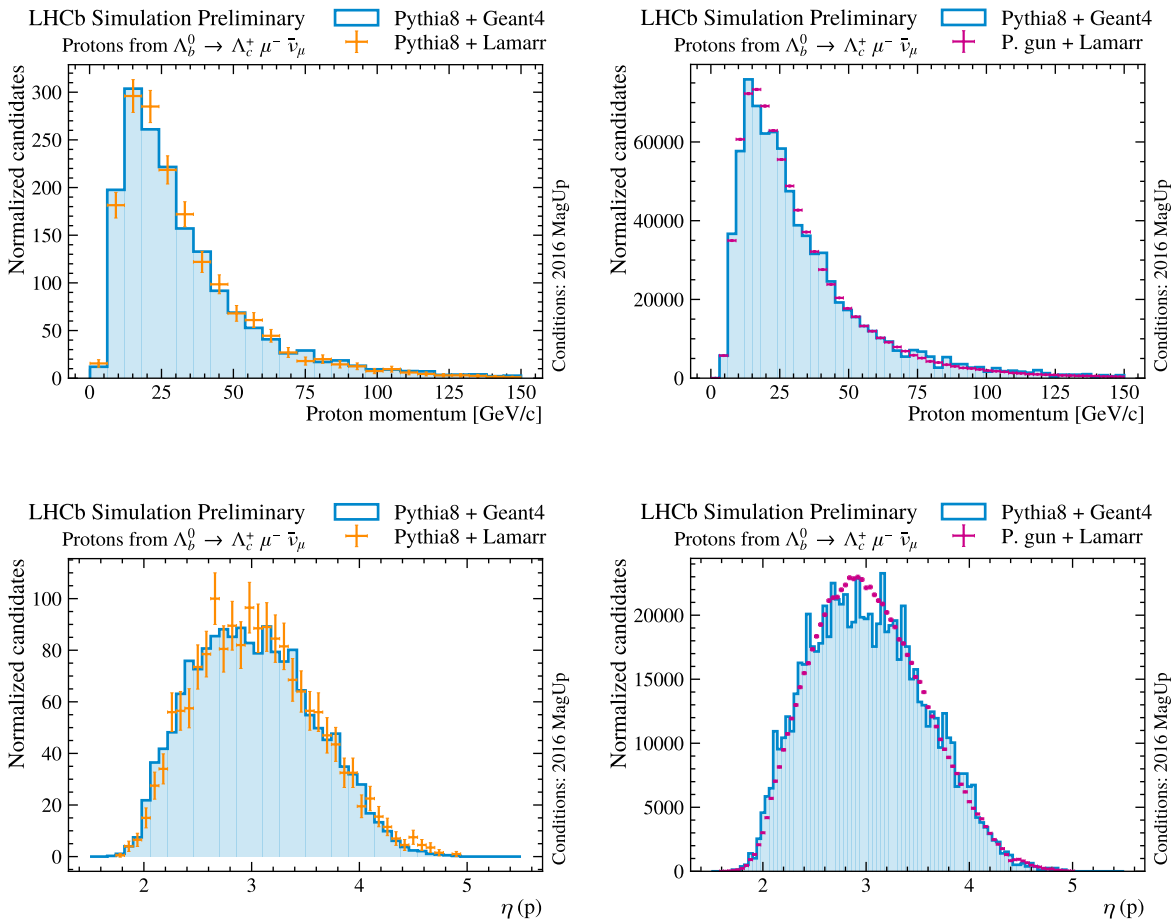


Figure 5.5: Kinematic variables of protons produced in $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays with $\Lambda_c^+ \to pK^-\pi^+$. The Λ_b^0 baryons are generated either with PYTHIA8 (left) or particle-gun (right), and the decay model is implemented with EVTGEN including feed-down modes. The momentum is shown on the first row for protons processed by LAMARR after PYTHIA8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). The second row displays the pseudorapidity for protons processed by LAMARR after PYTHIA8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In all cases, what results from LAMARR is superposed to reference samples made of Λ_b^0 generated with PYTHIA8, decays implemented with EVTGEN, and interactions with the detector processed by GEANT4 (cyan shaded histogram). Figure reproduced from Ref. [218].

reproducing the covariance matrix, validation studies involving the uncertainties on the *impact parameter* (IP), namely the minimum distance of a track to a primary vertex, and on the *primary vertex* (PV) are depicted in Figures 5.8 and 5.9. Notably, to test the performance of the flash-simulation framework with variables actually used in physics analyses, the smeared values of the track kinematic parameters and the corresponding uncertainties in terms of covariance matrix elements are further processed by relying on the BENDER application. The compatibility of the proton track with the primary vertex, namely the IP χ^2 is reported in Figure 5.8, while the vertex quality expressed in terms of χ^2_{vtx} of the Λ_c^+ is depicted in Figure 5.9. In both the cases, the output of the LAMARR pipeline of the two configurations (PYTHIA8 or particle-gun) is compared with what results

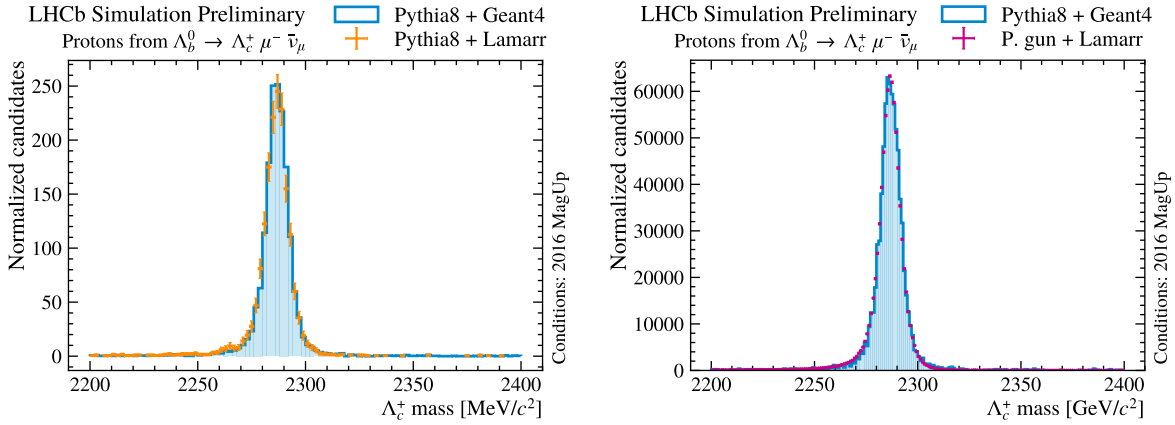


Figure 5.6: Invariant mass distribution of Λ_c^+ candidates produced in $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays with $\Lambda_c^+ \to pK^-\pi^+$. The Λ_b^0 baryons are generated either with PYTHIA8 or particle-gun, and the decay model is implemented with EVTGEN including feed-down modes. The invariant mass is computed for combinations of proton, kaon and pion from the Λ_c^+ decay simulated by LAMARR after PYTHIA8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In both cases, what obtained with LAMARR is superposed to reference sample resulting from Λ_b^0 generated with PYTHIA8, with a decay model described by EVTGEN, and interactions with the detector obtained from GEANT4 (cyan shaded histogram) and reconstructed with BRUNEL. Figure reproduced from Ref. [218].

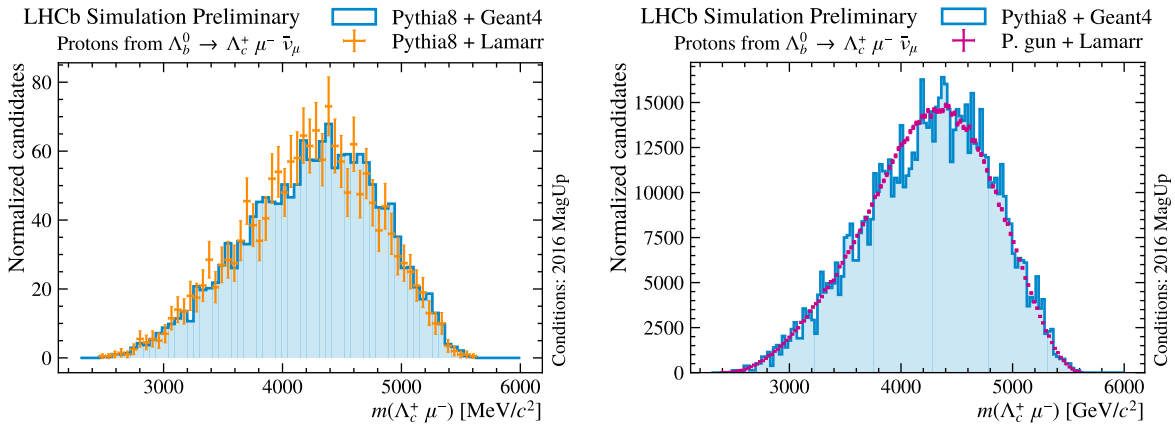


Figure 5.7: Invariant mass distribution of $\Lambda_c^+ \mu^-$ candidates produced in $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays with $\Lambda_c^+ \to pK^-\pi^+$. The Λ_b^0 baryons are generated either with PYTHIA8 or particle-gun, and the decay model is implemented with EVTGEN including feed-down modes. The invariant mass is computed for the combinations of proton, kaon, pion and muon simulated by LAMARR after PYTHIA8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In both cases, what obtained with LAMARR is superposed to reference sample resulting from Λ_b^0 generated with PYTHIA8, with a decay model described by EVTGEN, and interactions with the detector obtained from GEANT4 (cyan shaded histogram) and reconstructed with BRUNEL. Figure reproduced from Ref. [218].

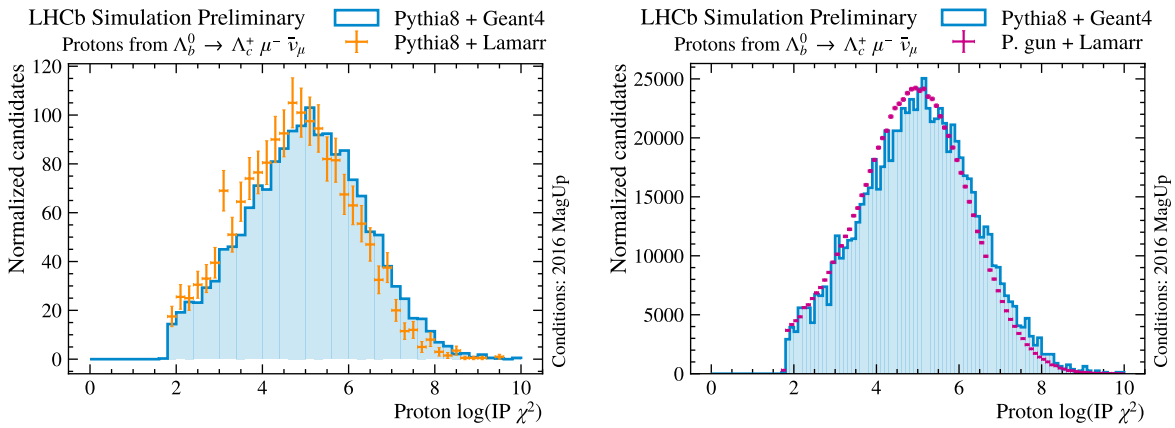


Figure 5.8: Distribution of the impact parameter χ^2 for protons produced in $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays with $\Lambda_c^+ \to pK^-\pi^+$. The Λ_b^0 baryons are generated either with Pythia8 or particle-gun, and the decay model is implemented with EvtGen including feed-down modes. The impact parameter χ^2 is reported for protons simulated by Lamarr after Pythia8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In both cases, what obtained with Lamarr is superposed to reference sample resulting from Λ_b^0 generated with Pythia8, with a decay model described by EvtGen, and interactions with the detector obtained from Geant4 (cyan shaded histogram) and reconstructed with Brunel. In both cases, the position of the primary vertex and its uncertainty are computed using the smearing algorithm developed by LHCb for particle-gun simulations. Figure reproduced from Ref. [218].

from the detailed simulation of the detection and reconstruction steps. The IP χ^2 for the proton tracks and the χ^2_{vtx} of the Λ_c^+ candidates are then computed by relying on a common configuration of the Bender application. Although the presence of some mismodeling effect clearly visible in the distributions, we are here mainly interested in validating the capabilities of Lamarr to reproduce reliable physical quantities progressively querying its ML-base parameterizations and further processing the outcomes with the standard analysis software in use at LHCb. In this respect, the results obtained are more than satisfactory, while an improvement on the agreement between the flash and detailed simulated variables is expected once the latest models will be employed.

The ultimate comparison with which challenges the ability of Lamarr to reproduce the uncertainty on the reconstructed trajectories rely on the complete reconstruction of the decay chain based on the `DecayTreeFitter` (DTF) algorithm [219] in use at LHCb. The χ^2_{DTF} of Λ_c^+ candidates produced in $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays with $\Lambda_c^+ \to pK^-\pi^+$ is reported in Figure 5.10. Notably, two configurations are investigated for assessing the Lamarr performance: in the first one, the fit of the decay chain is designed to ignore the PV (top row), while, in the second case, the fit is configured to constraint the Λ_c^+ trajectory to match with the PV (bottom row). Interestingly, the agreement exhibited in the bottom plots between the reference distributions and the ones resulting from Lamarr (Pythia8 and particle-gun) is another, more effective proof of the validity of the flash-simulation philosophy at the base of the novel framework, and of the ability of the employed models in reproducing so well the high-level response of the LHCb detector that even particle-gun samples are transformed in faithful datasets ready for the analysis.

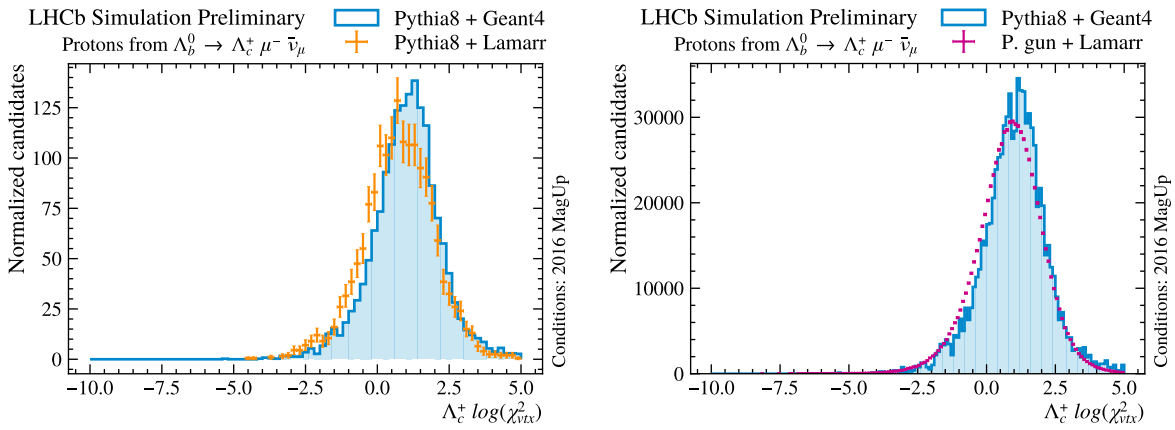


Figure 5.9: Distribution of the decay vertex χ^2 for the decay $\Lambda_c^+ \to pK^-\pi^+$ of Λ_c^+ baryons produced in the semileptonic decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$. The Λ_b^0 baryons are generated either with Pythia8 or particle-gun, and the decay model is implemented with EvtGen including feed-down modes. The decay vertex χ^2 is shown for Λ_c^+ resulting from the combination of protons, kaons and pions simulated by Lamarr after Pythia8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In both cases, what obtained with Lamarr is superposed to reference samples resulting from Λ_b^0 generated with Pythia8, with a decay model described by EvtGen, and interactions with the detector obtained from Geant4 (cyan shaded histogram) and reconstructed with Brunel. Figure reproduced from Ref. [218].

Proton Identification

To finalize the validation of the Lamarr pipeline for charged particles, we need to assess the performance achieved by the PID models once they are evaluated by consecutively querying each module of the pipeline to obtain the input for the following step. In particular, once the Lamarr Tracking modules have provided the kinematic variables as expected from the high-level response of the LHCb spectrometer, the differential log-likelihoods (DLL) of the RICH system can be computed by relying on GAN-based models similar to the ones discussed in Section 4.3.2. At the same time, for those reconstructed (flash-simulated) tracks that have passed the `isMuon` efficiency model (implemented via GBDTs in this validation study), also the likelihoods of the MUON system can be computed with GANs, whose training strategies are described in Section 4.3.3. Having access to the DLLs both from the RICH and MUON detectors, and disposing of the kinematic parameters from the Tracking modules, the global response of the PID system can be computed by relying on one more, last set of GANs that, as discussed in Section 4.3.5, have demonstrated to succeed in parameterazing both the combined differential log-likelihoods (CombDLL) and the output of the ANNPID classifier employed at LHCb. Actually, all the PID models rely also on the detector occupancy, that is typically modeled with the number of reconstructed tracks `nTracks`. For this validation study, a simplified parameterization for the detector occupancy has been implemented, based on samplings from a `nTracks` histogram resulting from Detailed Simulation.

According to this non-trivial pipeline, a set of proton identification variables have been produced and used for *proton selection*. The resulting efficiency and misidentification probability are then compared to what expected from Detailed Simulation by relying

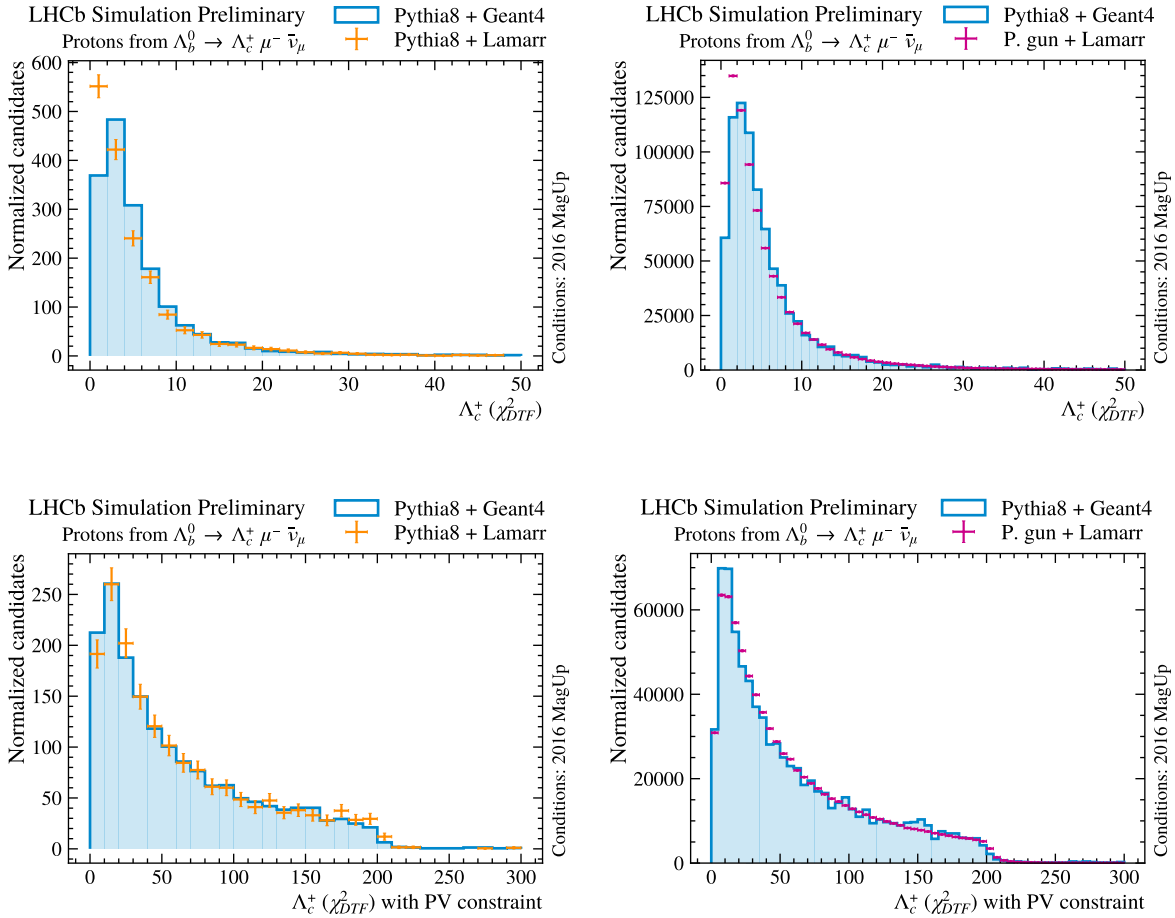


Figure 5.10: Distribution of the χ^2 of a fit to the decay $\Lambda_c^+ \to pK^-\pi^+$ of Λ_c^+ baryons produced in the semileptonic decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$. The fit is part of the `DecayTreeFitter` algorithm [219] configured to ignore the primary vertex (top row) or to constrain the Λ_c^+ trajectory to its position (bottom row). The Λ_b^0 baryons are generated either with Pythia8 or particle-gun, and the decay model is implemented with EvtGen including feed-down modes. The decay vertex χ^2 is shown for Λ_c^+ resulting from the combination of protons, kaons and pions simulated by Lamarr after Pythia8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In both cases, what obtained with Lamarr is superposed to reference samples resulting from Λ_b^0 generated with Pythia8, with a decay model described by EvtGen, and interactions with the detector obtained from Geant4 (cyan shaded histogram) and reconstructed with Brunel. In both cases, the position of the primary vertex and its uncertainty are computed using the smearing algorithm developed by LHCb for particle-gun simulations. Figure reproduced from Ref. [218].

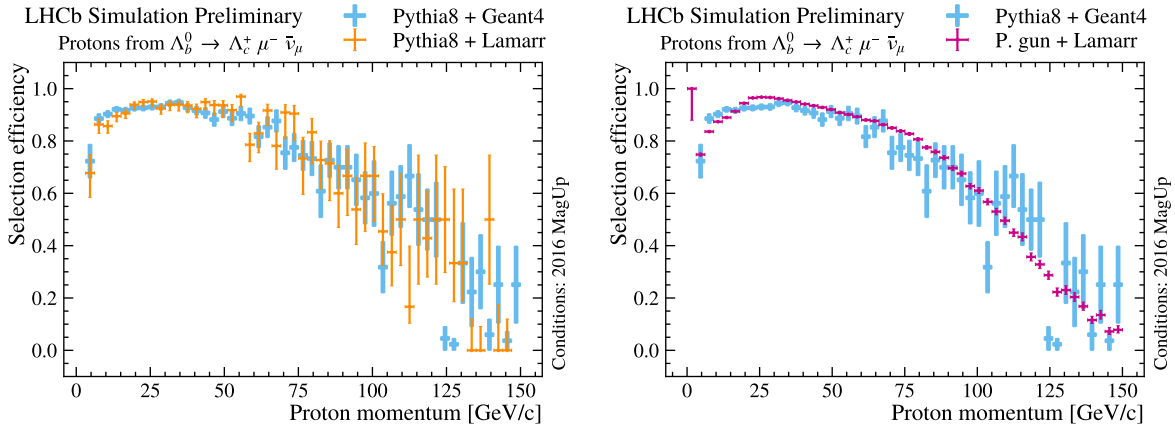


Figure 5.11: Efficiency of a tight requirement on high value of the CombDLL($p - \pi$) on protons (proton identification) tagged though the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to pK^-\pi^+$. The Λ_b^0 baryons are generated either with Pythia8 or particle-gun, and the decay model is implemented with EvtGen including feed-down modes. The efficiency is reported in bins of momentum for protons simulated by Lamarr after Pythia8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In both cases, what obtained with Lamarr is superposed to reference efficiencies resulting from Λ_b^0 generated with Pythia8, with a decay model described by EvtGen, and interactions with the detector obtained from Geant4 (cyan shaded histogram) and reconstructed with Brunel. Figure reproduced from Ref. [218].

on the same selection cuts. Figures 5.11 and 5.12 report the efficiency of the proton identification as a function of the momentum by relying on a tight requirement on the CombDLL($p - \pi$) variable and the ANNPID output for proton identification, respectively. On the other hand, Figures 5.13 and 5.14 report the misidentification probability $p \to K$ as a function of the momentum by using a tight requirement on the CombDLL($K - \pi$) variable and the ANNPID output for kaon identification, respectively. Despite the presence of some mismodeling effect probably due to sub-optimal PID models, the performance exhibited by the Lamarr pipeline in both the configurations (Pythia8 and particle-gun) is extremely promising, pushing forward further validation campaigns relying on the latest models aiming at a soon adoption for analysis purpose.

5.2.2 Preliminary timing studies

Together with the comparison between the distributions obtained from standard detailed simulation strategies and the ones resulting from Lamarr once the pipeline is injected with Λ_b^0 produced with Pythia8 or particle-gun and their decays described with EvtGen, the validation campaign of July 2022 [3] has also performed preliminary timing studies. As expected, the detailed simulation of the response of the LHCb detector interested by $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ decays with $\Lambda_c^+ \to pK^-\pi^+$ is extremely expensive in terms of CPU cost, with Geant4 responsible for the majority of the about 2.5 kHS06 $\cdot$ s required to simulate each event. Replacing Geant4 with Lamarr allows to reduce the computational cost to about 0.5 kHS06 $\cdot$ s per event, moving the title of major CPU consumer to Pythia8 that spends the large fraction of the computing resources for simulating the non-trivial

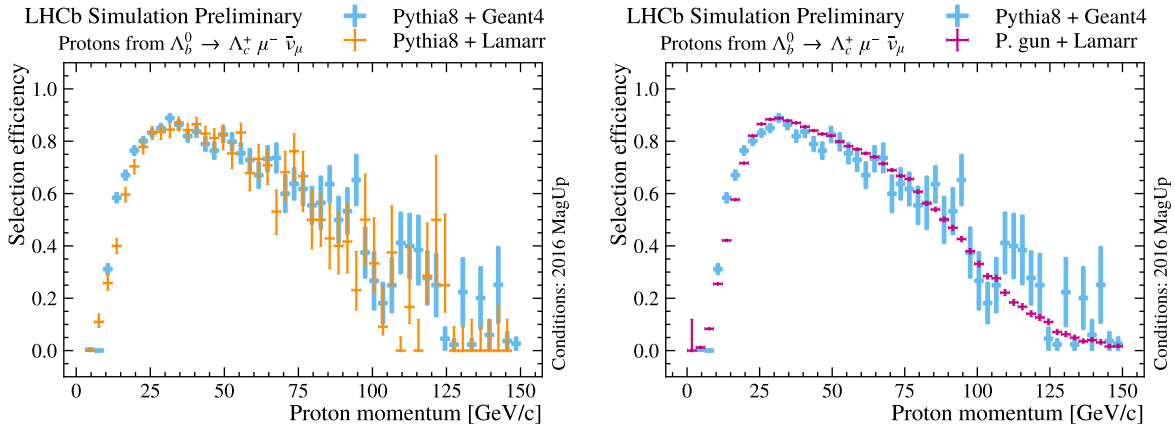


Figure 5.12: Efficiency of a tight requirement on the output of `ProbNNp`, namely a neural network trained to identify protons as evaluated on protons (proton identification) tagged though the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to pK^-\pi^+$. The Λ_b^0 baryons are generated either with Pythia8 or particle-gun, and the decay model is implemented with EvtGen including feed-down modes. The efficiency is reported in bins of momentum for protons simulated by Lamarr after Pythia8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In both cases, what obtained with Lamarr is superposed to reference efficiencies resulting from Λ_b^0 generated with Pythia8, with a decay model described by EvtGen, and interactions with the detector obtained from Geant4 (cyan shaded histogram) and reconstructed with Brunel. Figure reproduced from Ref. [218].

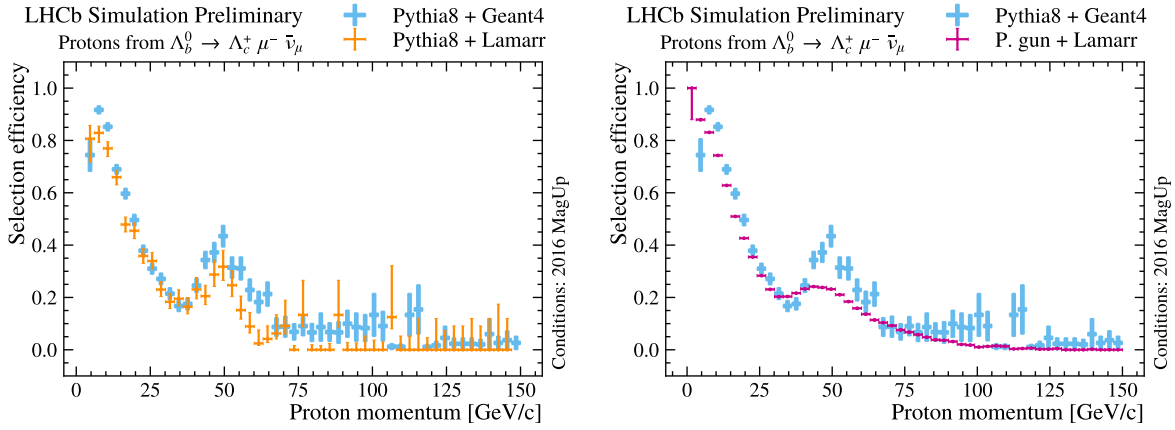


Figure 5.13: Efficiency of a tight requirement on high value of the CombDLL$(k-\pi)$ on protons (proton misidentification as kaon) tagged though the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to pK^-\pi^+$. The Λ_b^0 baryons are generated either with Pythia8 or particle-gun, and the decay model is implemented with EvtGen including feed-down modes. The efficiency is reported in bins of momentum for protons for protons simulated by Lamarr after Pythia8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In both cases, what obtained with Lamarr is superposed to reference efficiencies resulting from Λ_b^0 generated with Pythia8, with a decay model described by EvtGen, and interactions with the detector obtained from Geant4 (cyan shaded histogram) and reconstructed with Brunel. Figure reproduced from Ref. [218].

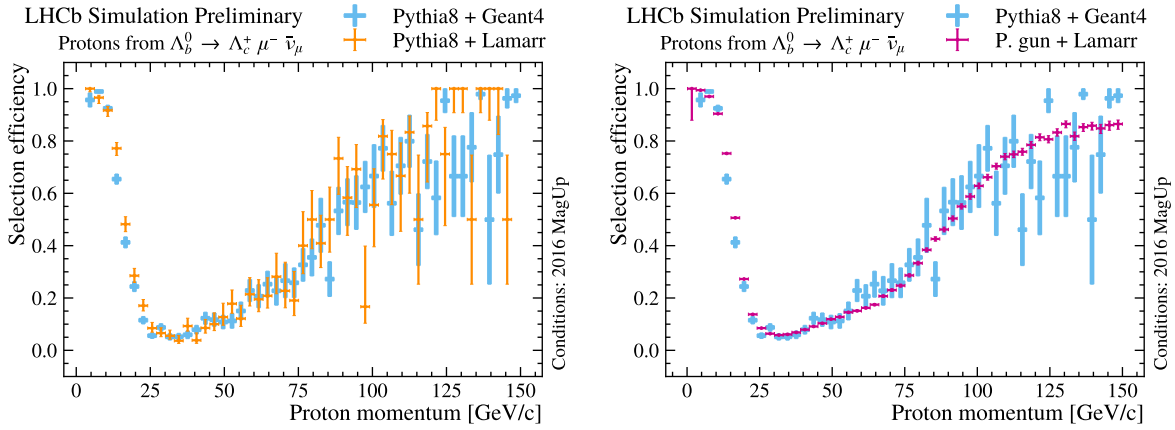


Figure 5.14: Efficiency of a tight requirement on the output of `ProbNNk`, namely a neural network trained to identify kaons as evaluated on protons (proton misidentification as kaon) tagged though the decay $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$ with $\Lambda_c^+ \to pK^-\pi^+$. The Λ_b^0 baryons are generated either with Pythia8 or particle-gun, and the decay model is implemented with EvtGen including feed-down modes. The efficiency is reported in bins of momentum for protons simulated by Lamarr after Pythia8-based generation (orange markers on the left), and for Λ_b^0 generated through particle-gun (purple markers on the right). In both cases, what obtained with Lamarr is superposed to reference efficiencies resulting from Λ_b^0 generated with Pythia8, with a decay model described by EvtGen, and interactions with the detector obtained from Geant4 (cyan shaded histogram) and reconstructed with Brunel. Figure reproduced from Ref. [218].

dynamics of the Λ_b^0 semileptonic decay. Hence, the major speed-up in terms of computation follows from replacing Pythia8 with particle-gun that drops significantly the cost for events simulation up to about 1 HS06 $\cdot$ s. The high-quality distributions obtained with the employ of a particle-gun approach, discussed in the previous Section, allows to take the most, in terms of computing performance, from the use of the flash-simulation paradigm that exhibits up to three orders of magnitude speed-up with respect to Detailed Simulation for the decay channel investigated.

5.3 A stand-alone flash-simulation option

The original attempt of providing LHCb with a flash-simulation option [160] relied on Delphes [70], a stand-alone framework designed to provide simulated samples for phenomenological studies at concentric experiments (such as ATLAS and CMS). Relying on configurable parameterizations, Delphes encodes the errors introduced in the detection and reconstruction steps within a pipeline of modules, called *cards*, offering a simplified description of the tracking, calorimeter, and muon systems of a generic multipurpose (4π) experiment. Easily customizable by each Collaboration, as long as the corresponding experiment has a concentric shape, by default Delphes has not explicit dependency from any collaborations' software stacks, offering a valuable tools for theorists to investigate the sensitivity of the various experiments to New Physics (NP) effects without relying on any proprietary softwares.

Despite inspired by the modular layout of Delphes, contrary to the latter, Lamarr renounces to any "geometrical constraints" describing the response of the LHCb experi-

ment with a generic (and agnostic) pipeline of parameterizations. This, in combination with the transcompilation approach offered by `scikinC` [7], enables a variety of studies and developments on the single parameterizations, providing a unique and shared infrastructure for validation and performance measurements. Designed to offer to LHCb the fastest solution for simulation production, the integration of LAMARR with GAUDI and GAUSS is mandatory to have easily access to the tuned version of physics generators, to enable centralized production relying on the WLCG resources, and to be natively integrated with the analysis software employed at LHCb. On the other hand, these strong dependencies makes the adoptions of LAMARR unappealing for researchers outside of the LHCb community, preventing them from approaching the detector simulation to evaluate the experiment sensitivity to NP phenomena or studying the recently-released LHCb Open Data [220].

These limitations have pointed to SQLAMARR[6], a package aimed to decoupling LAMARR from GAUDI providing a *stand-alone* application with minimal dependencies that can be easily set up and run in any Linux machine. To replace the concept of LHCb Event Model, hence avoiding any dependency from GAUDI, SQLAMARR relies on SQLite[7], a C library with minimal dependencies, enabling vectorized processing of batches of events, and providing a full-features SQL dialect to interact with data. Thus, SQLAMARR provides a set of classes and functions for loading data from physics generators and defining pipelines from models compiled as shared libraries. Finally, to avoid any dependency from ROOT, also the *persistency* is handled by relying on SQLite, writing the reconstructed (or intermediate) quantities in the form of SQLite databases. It should be noticed that it does not prevent from using ROOT data format, as converting a SQLite table to a ROOT `nTuple` requires a few lines of Python:

```
import sqlite3, uproot, pandas

with sqlite3.connect("SomeInput.db") as db:
    file = uproot.open("SomeFile.root", "RECREATE")
    file["myTree"] = pandas.read_sql_table("myTable", db)
```

A similar strategy, aimed at defining a pure Python simulation framework powered by cross-table relations, were investigated in the past and is briefly discussed in Ref. [1].

The minimal dependencies, the thread-safe database engine offered by SQLite, and the design choices for the pipeline configuration make SQLAMARR the perfect candidate to deliver the flash-simulation option within the newer version of GAUSS[8] (`sim11`), that will have access to multi-threading technologies by relying on the novel experiment-independent framework GAUSSINO[9] [64, 90]. In this respect, the development of SQLAMARR moves in two different and complementary directions. From one side, offering an integration of the LAMARR framework within GAUSS-ON-GAUSSINO that takes the most from the unlocked multi-threading capabilities. From the other side, providing a stand-alone flash-simulation framework that, powered by a pipeline of ML-based parameterizations, succeeds in reproducing the high-level response of the LHCb detector even relying on a set of particles generated through particle-gun.

To take full advantage from this generic simulation framework and aiming at making it

[6]Visit `https://lamarrsim.github.io/SQLamarr` for additional details.
[7]`https://www.sqlite.org`
[8]Visit `https://lhcb-gauss.docs.cern.ch/master` for additional details.
[9]Visit `https://gaussino.docs.cern.ch` for additional details.

as agnostic as possible with respect to the software technologies employed by state-of-the-art machine learning algorithms, a pure Python package describing the LHCb simulation pipeline has been developed. The latter, called PyLamarr[10], relies on SQLamarr as back-end and is designed to support pipeline of parameterizations defined either as compiled shared libraries or generic Python objects, such as Keras *custom* models or other architecture-specialized libraries (e.g., `nflows` [185], or TF-GNN [221]). Despite not mature enough at the time of writing, PyLamarr will play a key role for validating the performance of a wide range of parameterizations, whose implementation in `scikinC` would require a significant a priori effort hardly justifiable otherwise. The validation of PID Flow-based models investigated in Sections 4.3.2 and 4.3.3, and the Seq2seq and Graph2graph approaches discussed in Section 4.4.2 is planned and will be tackled by relying on PyLamarr for a first assessment of the quality of these parameterizations when deployed in the global pipeline operated in realistic conditions.

[10]Visit `https://lamarrsim.github.io/PyLamarr` for additional details.

Conclusions

This Ph.D. Thesis presents the first implementation of a parametric simulation of a High Energy Physics experiment obtained by concatenating in a pipeline multiple *Generative Deep Neural Networks*.

To make the discussion concrete, I present the development of such a *flash-simulation* for the LHCb experiment, whose experimental program urgently requires a technological shift to drastically reduce the average cost of a simulated collision event.

In the development of this document, I focused on the three major challenges in the development of such a *flash-simulation*: the training of the generative models to represent the effect of the particle detection and event reconstruction on analysis-level quantities; the modeling of correlations between different particles in the same event; and the deployment of the trained parameterizations within the software stack and the computing model of a major experiment at the LHC.

Among the generative models developed within the Artificial Intelligence technology, I identified *Generative Adversarial Networks* (GANs) as the most promising candidate to encode parameterizations for simulating a particle physics detector. Once trained, the parameterizations are represented as simple feed-forward neural networks, whose computation can benefit from superscalar operations in the modern processors while easily fitting in the in-thread computing model adopted by the software frameworks in Experimental High Energy Physics. Nonetheless, training GANs might result in a frustrating and ineffective exercise without a deep understanding of the two-player dynamics of the training procedure around the Nash equilibrium. The Machine Learning community made great progress on the subject during the years of my Ph.D. research, enabling the development of high-quality parameterizations for seven sets of variables (tracking resolution and reconstruction uncertainties, RICH and MUON system response, global hadron identification, global muon identification, and photon resolution and reconstruction uncertainties).

In addition, the concept of GANs may be extended to more complex architectures capable of processing multiple particles at once, and therefore encoding in the parameterizations effects of correlation between particles. I have explored two neural network architectures providing multi-particle parameterizations: *Transformers*, representing sets of particles as energy-sorted sequences; and *Graph Neural Networks* (GNNs), representing particles as nodes in a graph and defining edges to identify geometrical correlations. I

have used these multi-particle parameterizations to describe the response of the electromagnetic calorimeter of the LHCb experiment to the impinging photons. Indeed, the reconstruction algorithms associating calorimetric energy deposits to photons are prone to break the one-to-one relation between the simulated photon and the reconstructed cluster, by identifying adjacent clusters as originated by a single photon or by erroneously splitting the energy deposit of a single photon into two separate clusters. The global distribution quantities obtained from the trained models are in reasonable agreement with those obtained with production-grade simulation techniques. A more complete validation embedding the parameterization in the global pipeline to assess whether the achieved quality is sufficient for physics analysis is ongoing.

Finally, the idea that the parameterization of the whole detector can be described as a pipeline of subsequent generative models deserves careful validation. I contributed to the first validation of the LAMARR framework by studying the semileptonic decays $\Lambda_b^0 \to \Lambda_c^+ \mu^- \bar{\nu}_\mu$, to compare the distributions of the reconstructed quantities obtained with the flash-simulation to those provided by the production-grade LHCb simulation. The level of agreement was found acceptable also on derived quantities, obtained by running analysis-dependent algorithms taking as inputs multiple parameterized features. For example, the distribution of the χ^2 describing the quality of the global fit to the Λ_b^0 decay tree was found well reproduced. This represents an important closure test since the χ^2 of a decay tree candidate is directly sensitive to the parameterizations of the errors and uncertainties on the tracking and indirectly to those of the particle identification introducing efficiency effects depending on the kinematics.

During the development of this research, I led or provided major contributions to the development of three Python projects: HOPAAS, a software package to orchestrate hyperparameter optimization campaigns through multi-cloud opportunistic resources; PIDGAN, a Python library simplifying the implementation of several different flavors of Generative Adversarial Networks and organizing in a consistent set of APIs the regularization techniques identified in the literature as stabilizing the training procedure; and `scikinC` a simple application to transpile simple Machine Learning models trained using scikit-learn and Keras into C functions ready to be compiled as dynamically linkable libraries. The three packages are available as Free and Open Source Software on GitHub.

The future developments will focus on the validation of the flash-simulation paradigm and of the LAMARR software application to simulate a larger variety of heavy hadrons, decay modes, and for multiple data-taking conditions.

In conclusion, the work presented in this Thesis paves the way towards the adoption of LAMARR, an application rethinking the simulation of High Energy Physics detectors from its fundamentals, and ready to contribute to a significant reduction of the average cost of simulating a collision event for the LHCb experiment.

Acronyms

AI Artificial intelligence. 35, 43, 59, 176, 186, 195

ALEPH Apparatus for LEP Physics. 30

ALICE A Large Ion Collider Experiment. 6, 7

ALP Adversarial Lipschitz penalty. 69, 120, 136, 139, 153–155

Amazon S3 Amazon Simple Storage Service. 42

ANN Artificial neural network. 46–48, 174

ANNPID Artificial Neural Network for Particle Identification. 19, 119, 151, 152, 156, 157, 160, 201, 203

API Application programming interface. ix, 35, 37, 52–56, 119, 121, 122, 125, 137, 172, 174, 186, 188, 189, 210

ARM Advanced RISC Machines. 35

ATLAS A Toroidal LHC Apparatus. vii, 2, 6, 7, 26, 29, 180, 205

AUC Area under the curve. 89, 90, 145, 147

AWS Amazon Web Service. 34, 56

BCE Binary cross-entropy. 88–90, 102, 103, 111, 112, 120, 125–127, 136, 139, 145, 146, 153–155, 174

BLAS Basic Linear Algebra Subprogram. 187

BO Bayesian optimization. 51, 55

BSM Beyond Standard Model. 4, 6

CaaS Container as a Service. 41

CCE Categorical cross-entropy. 93, 94, 172

CD continuous delivery. 195, 196

CERN *Conseil Europèen pour la Recherche Nuclèaire*. vii, viii, 4–7, 33, 42, 56, 187

CI Continuous integration. 195, 196

CKM Cabibbo–Kobayashi–Maskawa. 3, 4

CMS Compact Muon Solenoid. vii, x, 2, 6, 7, 26, 29, 30, 119, 131, 205

CombDLL Combined differential log-likelihood. 19, 149, 151, 152, 155–157, 160, 201, 203, 204

CPU Central processing unit. 25, 26, 29, 35, 37, 39, 43, 56, 77, 78, 80, 81, 83, 167, 169, 185, 186, 194, 203

CSS Cascading Style Sheets. 55

cvmfs CERN Virtual Machine File System. 42, 195

DGM Deep Generative Modeling. 59

DL Deep learning. 46

DLL Differential log-likelihood. 14, 16, 123–125, 127, 131, 134, 139, 145, 151, 156–158, 161, 163, 165, 174, 177, 201

DTF DecayTreeFitter algorithm. 200

ECAL Electromagnetic calorimeter. 14, 17, 29, 80, 81, 114, 118, 125, 149, 167–174, 176, 177, 179–184, 196

FCNC Flavor-Changing Neutral Currents. 5, 6

FLOP Floating-point operation. 34, 35

FLOPS Floating-point operation per second. 57

FNN Feed-forward neural network. 19, 48, 62, 67, 70, 75, 101, 121, 125, 179, 182, 192, 193, 195

FOI Field of Interest. 18

FPGA Field-programmable gate array. 18, 20, 35, 39, 134, 140

GAN Generative Adversarial Networks. 56, 60–68, 70–72, 100–117, 119–122, 124–144, 151–170, 174–180, 192, 195, 196, 201, 209

GAT Graph Attention Network. 182

GBDT Gradient Boosted Decision Tree. 191–193, 196, 201

GCC GNU Compiler Collection. 188

GCP Google Cloud Platform. 34, 56

GD Gradient Descent. 49, 50

GEM Gas Electron Multipliers. 18

GNN Graph Neural Network. 177, 180, 182–184, 209

GP Gradient penalty. 68, 120, 154, 155

GPU Graphic processing unit. 35, 39, 40, 43, 56, 57, 122, 124, 136, 152, 186

Graph2graph Graph-to-graph. 177, 180, 207

HCAL Hadronic calorimeter. 14, 17, 18, 29, 114, 118, 149

HEP High Energy Physics. viii, 5, 23, 25, 26, 33–35, 37, 42, 56, 119, 120, 131, 167, 185–187, 195

HL-LHC High Luminosity Large Hadron Collider. 25

HLT High-Level Trigger. 21

HPC High-performance computing. 39, 52, 53, 56, 57, 122, 124, 152

HPDs Hybrid photodetector. 14

HPO Hyperparameter optimization. 52, 57

HS06 HEP-SPEC06. 25, 203, 205

HTC High-throughput computing. 34, 37, 39, 40

HTML Hypertext markup language. 55

HTT Hyper-Threading Technology. 37, 39

HTTP Hypertext transfer protocol. 53, 54, 56, 122

HTTPS Hypertext transfer protocol secure. 55

I/O Input/Output. 38

IaaS Infrastructure as a Service. 41

IAM Identity and Access Management. 56

INFN *Istituto Nazionale di Fisica Nucleare*. vii, ix, 40, 41, 52, 55–57

IP Impact parameter. 84, 100, 104, 105, 198, 200

IT Inner Tracker. 12

JSD Jensen-Shannon divergence. 64–67, 102–104

KaaS Kubernetes as a Service. 41

KS Kolmogorov-Smirnov. 123, 124, 136, 152

L0 Level 0. 20, 21

LEP Large Electron-Positron collider. 5, 33

LHC Large Hadron Collider. vii, viii, x, 2, 5–7, 10, 11, 20–26, 34, 40, 42, 56, 57, 77–79, 84, 194, 209

LHCb LHC beauty. vii–ix, 1, 4, 7–15, 17, 18, 20–27, 31, 39, 52, 56, 77–81, 83–90, 93, 95, 96, 100, 101, 104, 105, 107–109, 113, 114, 118, 119, 122, 123, 125, 127–138, 140–147, 149, 151, 152, 156–172, 174, 175, 177, 178, 180, 181, 183–185, 191, 194–197, 200–203, 205–207, 209, 210

LHCbPR LHCb Performance and Regression. 78

LLM Large language model. 44, 177, 179

LSGAN Least Squares Generative Adversarial Networks. 64, 65, 120, 125–127, 136, 137, 155

LWTNN Lightweight Trained Neural Network. 187

MAF Masked Autoregressive Flows. 134, 140

MC Monte Carlo. 28, 30, 80, 82, 88, 101, 106, 110, 176, 194, 195

MKL Math kernel library. 35

ML Machine learning. 43, 44, 51, 52, 54, 56, 119, 122, 123, 185, 186, 194–196, 200, 206

MLaaS Machine Learning as a Service. 187

MLP Multilayer perceptron. 48, 151, 182

MWPC Multi-wire Proportional Chambers. 18

NFL No Free Lunch. 46, 52

NLP Natural language processing. 43, 176, 177

NN Neural network. 48, 87–90, 94, 96, 101, 113, 145, 169–172, 174

NP New Physics. 4–6, 205, 206

OAuth Open Authorization. 55

OCI Open Container Initiative. 34

ONNX Open Neural Network Exchange. 186, 187

OOP Object-oriented programming. 39

OpenMP Open Multi-Processing. 37

OT Outer Tracker. 12

PaaS Platform as a Service. 41

PDG Particle Data Group. 3

Ph.D. Doctor of Philosophy. vii, ix, x, 52, 56, 79, 119, 122, 170, 186, 196, 209

PID Particle Identification. 8, 13, 16, 19, 21, 77, 79–82, 114, 118, 119, 122–124, 134, 149, 151–166, 169–172, 174, 175, 177, 194–197, 201, 203, 207

PReLU Parametric rectified linear unit. 49

PS PreShower. 14, 17, 149

PV Primary vertex. 100, 104, 198, 200

QCD Quantum chromodynamics. 2, 3, 7

QGP Quark-Gluon plasma. 6

RAM Random access memory. 39, 57

ReLU Rectified linear units. 49, 88, 89, 93, 94, 102, 103, 111, 112, 125, 126, 136, 139, 145, 146, 153, 154

REST Representational state transfer. 52–55

RICH Ring Imaging Cherenkov. 8, 13–16, 19, 22, 25, 78, 82, 83, 92, 110, 114, 118, 119, 123–137, 140, 145, 149, 151, 152, 156, 157, 160, 201, 209

ROC Receiver operating characteristic. 89, 90, 145, 147

ROE Rest of the event. 28, 29

SaaS Software as a Service. 41

Seq2seq Sequence-to-sequence. 170, 176, 177, 180, 207

SGD Stochastic Gradient Descent. 50

SIMD Single instruction, multiple data. 39

SM Standard Model. 1–5

SOFIE System for Optimized Fast Inference code Emit. 187

SPD Scintillating Pad Detector. 14, 17

SQL Structured Query Language. 55, 206

Tcl Tool command language. 30

TLU Threshold logic unit. 47, 48

TMVA Toolkit for Multivariate data Analysis. 20, 187

TOSCA Topology and Orchestration Specification for Cloud Applications. 41

TT Trigger Tracker. 9, 11, 12, 90, 92, 93, 96

VAE Variational autoencoder. 61

VELO Vertex Locator. 9–12, 84, 90, 92, 93, 96, 100, 105

ViT Vision Transformer. 179, 180

VPN Virtual private network. 36

WebDAV Web Distributed Authoring and Versioning. 42

WGAN Wasserstein Generative Adversarial Networks. 67–69, 120, 155, 167, 168

WLCG Worldwide LHC Computing Grid. vii, 22, 25, 34, 36, 37, 40–42, 186, 195, 206

XLA Extended Linear Algebra. 188

Bibliography

[1] M. Barbetti, *Techniques for parametric simulation with deep neural networks and implementation for the LHCb experiment at CERN and its future upgrades*, Master's thesis, University of Firenze, 2020. `https://cds.cern.ch/record/2826210`.

[2] M. Barbetti and L. Anderlini, *Hyperparameter Optimization as a Service on INFN Cloud*, in *21th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT)*, 2023, `arXiv:2301.05522`.

[3] L. Anderlini *et al.*, *Lamarr: the ultra-fast simulation option for the LHCb experiment*, PoS **ICHEP2022** (2022) 233.

[4] M. Barbetti, *Lamarr: LHCb ultra-fast simulation based on machine learning models deployed within Gauss*, in *21th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT)*, 2023, `arXiv:2303.11428`.

[5] LHCb Simulation Project collaboration, L. Anderlini *et al.*, *The LHCb ultra-fast simulation option, Lamarr design and validation*, EPJ Web Conf. **295** (2024) 03040, `arXiv:2309.13213`.

[6] M. Barbetti, *PIDGAN: GAN-based models to flash-simulate the LHCb PID detectors*, 2023. doi: 10.5281/zenodo.10463728.

[7] L. Anderlini and M. Barbetti, *scikinC: a tool for deploying machine learning as binaries*, PoS **CompTools2021** (2022) 034.

[8] ATLAS collaboration, G. Aad *et al.*, *Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC*, Phys. Lett. B **716** (2012) 1, `arXiv:1207.7214`.

[9] CMS collaboration, S. Chatrchyan *et al.*, *Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC*, Phys. Lett. B **716** (2012) 30, `arXiv:1207.7235`.

[10] Particle Data Group, R. L. Workman *et al.*, *Review of particle physics*, Prog. Theor. Exp. Phys. **2022** (2022) 083C01.

[11] LHCb collaboration, R. Aaij *et al.*, *Observation of CP violation in charm decays*, Phys. Rev. Lett. **122** (2019) 211803, arXiv:1903.08726.

[12] LHCb collaboration, R. Aaij *et al.*, *Measurement of CP violation in $B^0 \to \psi(\to \ell^+\ell^-)K^0_S(\to \pi^+\pi^-)$ decays*, Phys. Rev. Lett. **132** (2024) 021801, arXiv:2309.09728.

[13] LHCb collaboration, R. Aaij *et al.*, *Test of lepton universality in beauty-quark decays*, Nature Physics **18** (2022) 277, arXiv:2103.11769.

[14] LHCb collaboration, R. Aaij *et al.*, *Test of lepton universality in $b \to s\ell^+\ell^-$ decays*, Phys. Rev. Lett. **131** (2023) 051803, arXiv:2212.09152.

[15] LHCb collaboration, R. Aaij *et al.*, *Measurement of lepton universality parameters in $B^+ \to K^+\ell^+\ell^-$ and $B^0 \to K^{*0}\ell^+\ell^-$ decays*, Phys. Rev. **D108** (2023) 032002, arXiv:2212.09153.

[16] LHCb collaboration, R. Aaij *et al.*, *Test of lepton flavour universality using $B^0 \to D^{*-}\tau^+\nu_\tau$ decays, with hadronic τ channels*, Phys. Rev. **D108** (2023) 012018, arXiv:2305.01463.

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

[18] L. Evans and P. Bryant, *LHC Machine*, JINST **3** (2008) S08001.

[19] G. Papotti *et al.*, *Observations of beam-beam effects at the LHC*, in *ICFA Mini-Workshop on Beam-Beam Effects in Hadron Colliders*, 1–5, 2014, arXiv:1409.5208.

[20] C. Barschel, *Precision luminosity measurement at LHCb with beam-gas imaging*, PhD thesis, RWTH Aachen University, 2014, https://cds.cern.ch/record/1693671.

[21] CMS collaboration, S. Chatrchyan *et al.*, *The CMS Experiment at the CERN LHC*, JINST **3** (2008) S08004.

[22] ATLAS collaboration, G. Aad *et al.*, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3** (2008) S08003.

[23] ALICE collaboration, K. Aamodt *et al.*, *The ALICE experiment at the CERN LHC*, JINST **3** (2008) S08002.

[24] LHCb collaboration, A. A. Alves Jr. *et al.*, *The LHCb detector at the LHC*, JINST **3** (2008) S08005.

[25] T. Sjöstrand, S. Mrenna, and P. Skands, *A brief introduction to PYTHIA 8.1*, Comput. Phys. Commun. **178** (2008) 852, arXiv:0710.3820.

[26] LHCb collaboration, C. Elsässer, *$b\bar{b}$ production angle plots*, https://lhcb.web.cern.ch/lhcb/speakersbureau/html/bb_ProductionAngles.html.

[27] LHCb collaboration, *LHCb magnet: Technical Design Report*, CERN-LHCC-2000-007, 2000.

[28] LHCb collaboration, *LHCb VELO (VErtex LOcator): Technical Design Report*, CERN-LHCC-2001-011, 2001.

[29] R. Aaij *et al.*, *Performance of the LHCb Vertex Locator*, JINST **9** (2014) P09007, arXiv:1405.7808.

[30] LHCb collaboration, *LHCb reoptimized detector design and performance: Technical Design Report*, CERN-LHCC-2003-030, 2003.

[31] LHCb collaboration, *LHCb inner tracker: Technical Design Report*, CERN-LHCC-2002-029, 2002.

[32] LHCb collaboration, *LHCb outer tracker: Technical Design Report*, CERN-LHCC-2001-024, 2001.

[33] R. Arink *et al.*, *Performance of the LHCb Outer Tracker*, JINST **9** (2014) P01002, arXiv:1311.3893.

[34] LHCb collaboration, R. Aaij *et al.*, *LHCb detector performance*, Int. J. Mod. Phys. **A30** (2015) 1530022, arXiv:1412.6352.

[35] LHCb collaboration, *Tracking and alignment plots for conferences*, https://twiki.cern.ch/twiki/bin/view/LHCb/ConferencePlots.

[36] LHCb collaboration, L. M. Garcia, L. Henry, B. Kishor, and A. Oyanguren, *Tracking performance for long-lived particles at LHCb*, J. Phys. Conf. Ser. **1525** (2020) 012095, arXiv:1910.06171.

[37] M. De Cian, *Track Reconstruction Efficiency and Analysis of $B^0 \to K^{*0}\mu^+\mu^-$ at the LHCb Experiment*, PhD thesis, University of Zurich, 2013, https://cds.cern.ch/record/1605179.

[38] R. E. Kalman, *A New Approach to Linear Filtering and Prediction Problems*, J. Basic Eng. **82** (1960) 35.

[39] M. De Cian, S. Farry, P. Seyfert, and S. Stahl, *Fast neural-net based fake track rejection in the LHCb reconstruction*, LHCb-PUB-2017-011, 2017.

[40] LHCb collaboration, *LHCb RICH: Technical Design Report*, CERN-LHCC-2000-037, 2000.

[41] LHCb collaboration, *LHCb calorimeters: Technical Design Report*, CERN-LHCC-2000-036, 2000.

[42] LHCb collaboration, *LHCb muon system: Technical Design Report*, CERN-LHCC-2001-010, 2001.

[43] Particle Data Group, R. L. Workman *et al.*, *Review of Particle Physics*, PTEP **2022** (2022) 083C01.

[44] M. Adinolfi *et al.*, *Performance of the LHCb RICH detector at the LHC*, Eur. Phys. J. **C73** (2013) 2431, arXiv:1211.6759.

[45] LHCb collaboration, E. Picatoste Olloqui, *LHCb preshower(PS) and scintillating pad detector (SPD): Commissioning, calibration, and monitoring*, J. Phys. Conf. Ser. **160** (2009) 012046.

[46] C. Abellan Beteta *et al.*, *Calibration and performance of the LHCb calorimeters in Run 1 and 2 at the LHC*, arXiv:2008.11556, submitted to JINST.

[47] F. Archilli *et al.*, *Performance of the muon identification at LHCb*, JINST **8** (2013) P10020, arXiv:1306.0249.

[48] A. Hoecker *et al.*, *TMVA 4 — Toolkit for Multivariate Data Analysis with ROOT. Users Guide.*, arXiv:physics/0703039.

[49] R. Aaij *et al.*, *Selection and processing of calibration samples to measure the particle identification performance of the LHCb experiment in Run 2*, Eur. Phys. J. Tech. Instr. **6** (2019) 1, arXiv:1803.00824.

[50] LHCb collaboration, *LHCb online system, data acquisition and experiment control: Technical Design Report*, CERN-LHCC-2001-040, 2001.

[51] LHCb collaboration, *LHCb trigger system: Technical Design Report*, CERN-LHCC-2003-031, 2003.

[52] LHCb collaboration, *LHCb computing: Technical Design Report*, CERN-LHCC-2005-019, 2005.

[53] G. Barrand *et al.*, *GAUDI - A software architecture and framework for building HEP data processing applications*, Comput. Phys. Commun. **140** (2001) 45.

[54] LHCb collaboration, *LHCb starterkit*, https://lhcb.github.io/starterkit-lessons/first-analysis-steps/dataflow.html.

[55] LHCb collaboration, *RTA and DPA dataflow diagrams for Run 1, Run 2, and the upgraded LHCb detector*, LHCB-FIGURE-2020-016, 2020.

[56] *LHC computing grid web site*, https://wlcg.web.cern.ch.

[57] G. Dujany and B. Storaci, *Real-time alignment and calibration of the LHCb Detector in Run II*, J. Phys. Conf. Ser. **664** (2015) 082010.

[58] D. J. Lange, *The EvtGen particle decay simulation package*, Nucl. Instrum. Meth. **A462** (2001) 152.

[59] Geant4 collaboration, S. Agostinelli *et al.*, *Geant4: A simulation toolkit*, Nucl. Instrum. Meth. **A506** (2003) 250.

[60] Geant4 collaboration, J. Allison *et al.*, *Geant4 developments and applications*, IEEE Trans. Nucl. Sci. **53** (2006) 270.

[61] M. Clemencic *et al.*, *The LHCb simulation application, Gauss: Design, evolution and experience*, J. Phys. Conf. Ser. **331** (2011) 032023.

[62] LHCb collaboration, B. G. Siddi and D. Müller, *Gaussino - a Gaudi-Based Core Simulation Framework*, in *2019 IEEE Nuclear Science Symposium (NSS) and Medical Imaging Conference (MIC)*, 1–4, IEEE, 2019.

[63] Key4hep collaboration, E. Brondolin *et al.*, *Key4hep: Progress Report on Integrations*, in *26th International Conference on Computing in High Energy & Nuclear Physics*, 2023, arXiv:2312.08152.

[64] M. Mazurek, M. Clemencic, and G. Corti, *Gauss and Gaussino: the LHCb simulation software and its new experiment agnostic core framework*, PoS **ICHEP2022** (2022) 225.

[65] ATLAS collaboration, *ATLAS Software and Computing HL-LHC Roadmap*, LHCC-G-182, CERN, 2022.

[66] CMS Offline Software and Computing, *CMS Phase-2 Computing Model: Update Document*, CMS-NOTE-2022-008, CERN, 2022.

[67] CMS collaboration, F. Vaselli, A. Rizzi, F. Cattafesta, and G. Cicconofri, *FlashSim prototype: an end-to-end fast simulation using Normalizing Flow*, CMS-NOTE-2023-003, 2023.

[68] 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.

[69] LHCb collaboration, *LHCb Upgrade Software and Computing*, CERN-LHCC-2018-007, 2018.

[70] 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.

[71] R. Brun and F. Rademakers, *ROOT: An object oriented data analysis framework*, Nucl. Instrum. Meth. A **389** (1997) 81.

[72] 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.

[73] CERN Courier, *Computers at CERN*, https://cerncourier.com/a/computers-at-cern, 1964.

[74] T. Berners-Lee, *Information management: A proposal*, https://www.w3.org/History/1989/proposal.html, 1989.

[75] CERN Courier, *The LHC's worldwide computer*, https://cerncourier.com/a/the-lhcs-worldwide-computer, 2013.

[76] T. N. Theis and H. S. P. Wong, *The end of moore's law: A new beginning for information technology*, Computing in Science & Engineering **19** (2017) 41.

[77] L. Clissa, M. Lassnig, and L. Rinaldi, *How big is Big Data? A comprehensive survey of data production, storage, and streaming in science and industry*, Front. Big Data **6** (2023) .

[78] D. Merkel, *Docker: lightweight linux containers for consistent development and deployment*, Linux Journal **2014** (2014) 2.

[79] Linux Foundation, *Open container initiative*, `https://opencontainers.org/faq`, 2015.

[80] C. Jones and P. Gartung, *CMSSW Scaling Limits on Many-Core Machines*, EPJ Web Conf. **295** (2024) 03008, `arXiv:2310.02872`.

[81] M. Iorio *et al.*, *Computing without borders: The way towards liquid computing*, IEEE Transactions on Cloud Computing (2022) 1.

[82] D. Thain, T. Tannenbaum, and M. Livny, *Distributed computing in practice: the condor experience.*, Concurrency - Practice and Experience **17** (2005) 323.

[83] M. A. Jette and T. Wickberg, *Architecture of the Slurm Workload Manager*, LNCS **14283** (2023).

[84] F. Molder *et al.*, *Sustainable data analysis with Snakemake*, F1000Research **10** (2021) 33.

[85] D. T. Marr *et al.*, *Hyper-Threading Technology Architecture and Microarchitecture*, Intel Technology Journal **Q1** (2002).

[86] N. D. Matsakis and F. S. Klock II, *The rust language*, in *ACM SIGAda Ada Letters*, **34**, 103–104, ACM, 2014.

[87] CMS collaboration, C. D. Jones *et al.*, *Using the CMS Threaded Framework In A Production Environment*, J. Phys. Conf. Ser. **664** (2015) 072026.

[88] ATLAS collaboration, G. A. Stewart *et al.*, *Multi-threaded software framework development for the ATLAS experiment*, J. Phys. Conf. Ser. **762** (2016) 012024.

[89] M. Clemencic, B. Hegner, and C. Leggett, *Gaudi evolution for future challenges*, J. Phys. Conf. Ser. **898** (2017) 042044.

[90] M. Mazurek, G. Corti, and D. Muller, *New simulation software technologies at the LHCb Experiment at CERN*, Comput. Inform. **40** (2021) 815, `arXiv:2112.04789`.

[91] S. Farrell *et al.*, *Multi-threaded Geant4 on the Xeon-Phi with Complex High-Energy Physics Geometry*, in *2015 IEEE Nuclear Science Symposium and Medical Imaging Conference*, 7581868, 2016, `arXiv:1605.08371`.

[92] D. Nuzman, I. Rosen, and A. Zaks, *Auto-vectorization of interleaved data for simd*, in *27th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 132–143, 2006.

[93] J. Shin, M. Hall, and J. Chame, *Superword-level parallelism in the presence of control flow*, in *2005 International Symposium on Code Generation and Optimization (CGO)*, 165–175, 2005.

[94] G. Amadio *et al.*, *Electromagnetic physics vectorization in the GeantV transport framework*, EPJ Web Conf. **214** (2019) 02031.

[95] R. Aaij *et al.*, *Allen: A high level trigger on GPUs for LHCb*, Comput. Softw. Big Sci. **4** (2020) 7, arXiv:1912.09161.

[96] D. Castro *et al.*, *Apache Spark usage and deployment models for scientific computing*, EPJ Web Conf. **214** (2019) 07020.

[97] P. Buncic *et al.*, *CernVM: A virtual software appliance for LHC applications*, J. Phys. Conf. Ser. **219** (2010) 042003.

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

[99] H. Touvron *et al.*, *LLaMA: Open and Efficient Foundation Language Models*, arXiv:2302.13971.

[100] OpenAI team, *GPT-4 Technical Report*, arXiv:2303.08774.

[101] Gemini team, *Gemini: A Family of Highly Capable Multimodal Models*, arXiv:2312.11805.

[102] A. Ramesh *et al.*, *Hierarchical Text-Conditional Image Generation with CLIP Latents*, arXiv:2204.06125.

[103] M. Kang *et al.*, *Scaling up gans for text-to-image synthesis*, in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10124–10134, 2023.

[104] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, in *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019, arXiv:1810.04805.

[105] T. B. Brown *et al.*, *Language models are few-shot learners*, in *33rd International Conference on Neural Information Processing Systems (NeurIPS)*, 2020, arXiv:2005.14165.

[106] A. Chowdhery *et al.*, *PaLM: Scaling Language Modeling with Pathways*, JMLR **24** (2022) 1, arXiv:2204.02311.

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

[108] 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.

[109] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature **512** (2015) 436.

[110] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*, Neural Computation **9** (1997) 1735.

[111] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, `arXiv:1512.03385`.

[112] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun, *Unsupervised learning of invariant feature hierarchies with applications to object recognition*, in *2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[113] F. Scarselli *et al.*, *The Graph Neural Network Model*, IEEE Trans. Neural Networks **20** (2009) 61.

[114] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in *18th International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, **9351** 234–241, 2015.

[115] A. Vaswani *et al.*, *Attention Is All You Need*, in *31st International Conference on Neural Information Processing Systems (NeurIPS)*, 2017, `arXiv:1706.03762`.

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

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

[118] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural Networks **2** (1989) 359.

[119] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, in *3rd International Conference on Learning Representations (ICLR)*, 2015, `arXiv:1412.6980`.

[120] T. Tieleman and G. Hinton, *Lecture 6.5 - RMSProp: Divide the Gradient by a Running Average of Its Recent Magnitude*, 2012. COURSERA: Neural Networks for Machine Learning.

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

[122] J. G. Richens, C. M. Lee, and S. Johri, *Improving the accuracy of medical diagnosis with causal machine learning*, Nat. Comm. **11** (2020) 3923.

[123] G. B. Orr and K. R. Müller, *Neural Networks: Tricks of the Trade*, Springer Berlin Heidelberg, 2003.

[124] J. Bergstra *et al.*, *Algorithms for hyper-parameter optimization*, in *24th International Conference on Neural Information Processing Systems (NeurIPS)*, 2546–2554, 2011.

[125] J. Bergstra, D. Yamins, and D. Cox, *Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures*, in *30th International Conference on Machine Learning (PMLR)*, 115–123, 2013, arXiv:1209.5111.

[126] D. Golovin *et al.*, *Google Vizier: A Service for Black-Box Optimization*, in *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1487–1495, 2017.

[127] R. Liaw *et al.*, *Tune: A Research Platform for Distributed Model Selection and Training*, arXiv:1807.05118.

[128] T. Akiba *et al.*, *Optuna: A Next-generation Hyperparameter Optimization Framework*, in *25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2623–2631, 2019, arXiv:1907.10902.

[129] T. Head *et al.*, *scikit-optimize/scikit-optimize*, 2021. doi: 10.5281/zenodo.5565057.

[130] X. Song *et al.*, *Open Source Vizier: Distributed Infrastructure and API for Reliable and Flexible Black-box Optimization*, in *1st International Conference on Automated Machine Learning (AutoML)*, 2022, arXiv:2207.13676.

[131] M. Barbetti and L. Anderlini, *Reference implementation for Hopaas RestAPIs client in Python*, 2022. doi: 10.5281/zenodo.7528502.

[132] L. Tani, D. Rand, C. Veelken, and M. Kadastik, *Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics*, Eur. Phys. J. C **81** (2021) 170, arXiv:2011.04434.

[133] D. Spiga *et al.*, *Dynamic integration of distributed, Cloud-based HPC and HTC resources using JSON Web Tokens and the INDIGO IAM Service*, EPJ Web Conf. **245** (2020) 07020.

[134] I. J. Goodfellow *et al.*, *Generative Adversarial Networks*, in *27th International Conference on Neural Information Processing Systems (NeurIPS)*, 2014, arXiv:1406.2661.

[135] M. Devouassoux *et al.*, *CloudBank for Europe*, EPJ Web Conf. **251** (2021) 02025.

[136] M. Barbetti, *OptunAPI: API to distribute hyperparameters optimization through HTTP requests*, 2021. doi: 10.5281/zenodo.5539487.

[137] M. Mariotti, D. Spiga, and T. Boccali, *A possible solution for HEP processing on network secluded Computing Nodes*, PoS **ISGC2021** (2021) 002.

[138] L. Ruthotto and E. Haber, *An introduction to deep generative modeling*, arXiv:2103.05180.

[139] L. Weng, *From Autoencoder to Beta-VAE*, https://lilianweng.github.io/posts/2018-08-12-vae, 2018.

[140] L. Weng, *What are diffusion models?*, https://lilianweng.github.io/posts/2021-07-11-diffusion-models, 2021.

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

[142] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, *Unrolled Generative Adversarial Networks*, in *5th International Conference on Learning Representations (ICLR)*, 2017, arXiv:1611.02163.

[143] X. Mao *et al.*, *Least Squares Generative Adversarial Networks*, in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, arXiv:1611.04076.

[144] M. Arjovsky and L. Bottou, *Towards Principled Methods for Training Generative Adversarial Networks*, in *5th International Conference on Learning Representations (ICLR)*, 2017, arXiv:1701.04862.

[145] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein GAN*, in *34th International Conference on Machine Learning (ICML)*, 2017, arXiv:1701.07875.

[146] I. Gulrajani *et al.*, *Improved Training of Wasserstein GANs*, in *31st International Conference on Neural Information Processing Systems (NeurIPS)*, 2017, arXiv:1704.00028.

[147] D. Terjék, *Adversarial Lipschitz Regularization*, in *8th International Conference on Learning Representations (ICLR)*, 2019, arXiv:1907.05681.

[148] M. G. Bellemare *et al.*, *The Cramer Distance as a Solution to Biased Wasserstein Gradients*, in *6th International Conference on Learning Representations (ICLR)*, 2018, arXiv:1705.10743.

[149] G. Papamakarios *et al.*, *Normalizing Flows for Probabilistic Modeling and Inference*, JMLR **22** (2021) 1.

[150] L. Weng, *Flow-based Deep Generative Models*, https://lilianweng.github.io/posts/2018-10-13-flow-models, 2018.

[151] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, *Neural autoregressive flows*, in *35th International Conference on Machine Learning (PMLR)*, **80**, 2078–2087, 2018.

[152] P. Jaini, K. A. Selby, and Y. Yu, *Sum-of-squares polynomial flow*, in *36th International Conference on Machine Learning (PMLR)*, **97**, 3009–3018, 2019.

[153] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, *Neural Spline Flows*, 2019, arXiv:1906.04032.

[154] M. Germain, K. Gregor, I. Murray, and H. Larochelle, *Made: Masked autoencoder for distribution estimation*, in *32nd International Conference on Machine Learning*, **37**, 881–889, 2015.

[155] LHCb collaboration, R. Aaij *et al.*, *The LHCb Upgrade I*, JINST **19** (2024) P05065, arXiv:2305.10515.

[156] LHCb collaboration, D. Popov, *Testing and verification of the LHCb Simulation*, EPJ Web Conf. **214** (2019) 02043.

[157] M. Cacciari, M. Greco, and P. Nason, *The p_T spectrum in heavy-flavour hadroproduction.*, JHEP **05** (1998) 007, arXiv:hep-ph/9803400.

[158] I. Belyaev *et al.*, *Handling of the generation of primary events in Gauss, the LHCb simulation framework*, J. Phys. Conf. Ser. **331** (2011) 032047.

[159] LHCb collaboration, R. Aaij *et al.*, *Measurement of the electron reconstruction efficiency at LHCb*, JINST **14** (2019) P11023, arXiv:1909.02957.

[160] B. G. Siddi, *A fully parametric option in the LHCb simulation framework*, EPJ Web Conf. **214** (2019) 02024.

[161] LHCb collaboration, M. Rama and G. Vitali, *Calorimeter fast simulation based on hit libraries LHCb Gauss framework*, EPJ Web Conf. **214** (2019) 02040.

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

[163] J. Van Tilburg, *Track simulation and reconstruction in LHCb*, PhD thesis, Vrije Universiteit Amsterdam, 2005, https://cds.cern.ch/record/885750.

[164] E. Bos, *Reconstruction of charged particles in the LHCb experiment*, PhD thesis, University of Amsterdam, 2010, https://cds.cern.ch/record/1255878.

[165] LHCb collaboration, W. Baldini *et al.*, *Overview of LHCb alignment*, in *1st LHC Detection Alignment Workshop*, 197–222, 2006.

[166] S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, in *32nd International Conference on International Conference on Machine Learning (ICML)*, 2015, arXiv:1502.03167.

[167] F. Chollet *et al.*, *Keras*, 2015. Software available from https://keras.io.

[168] M. Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from https://tensorflow.org.

[169] C. Cortes, M. Mohri, and A. Rostamizadeh, *L2 Regularization for Learning Kernels*, in *25th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2009, arXiv:1205.2653.

[170] LHCb collaboration, R. Aaij *et al.*, *Measurement of the track reconstruction efficiency at LHCb*, JINST **10** (2015) P02007, arXiv:1408.1251.

[171] G. Corti *et al.*, *Software for the LHCb experiment*, IEEE Trans. Nucl. Sci. **53** (2006) 1323.

[172] LHCb collaboration, *Machine-Learnt parametrizations for the Ultra-Fast Simulation of the LHCb detector*, LHCB-FIGURE-2022-004, 2022.

[173] M. Tobin, *Performance of the LHCb Tracking Detectors*, PoS **Vertex2012** (2013) 047.

[174] LHCb collaboration, F. Dordei, *LHCb detector and trigger performance in Run II*, EPJ Web Conf. **164** (2017) 01016.

[175] F. Pedregosa *et al.*, *Scikit-learn: Machine learning in Python*, J. Machine Learning Res. **12** (2011) 2825, arXiv:1201.0490, and online at http://scikit-learn.org/stable/.

[176] T. Salimans *et al.*, *Improved Techniques for Training GANs*, in *29th International Conference on Neural Information Processing Systems (NeurIPS)*, 2016, arXiv:1606.03498.

[177] A. Rogachev and F. Ratnikov, *GAN with an Auxiliary Regressor for the Fast Simulation of the Electromagnetic Calorimeter Response*, J. Phys. Conf. Ser. **2438** (2023) 012086, arXiv:2207.06329.

[178] C. Weisser and M. Williams, *Machine learning and multivariate goodness of fit*, arXiv:1612.07186.

[179] LHCb collaboration, A. Maevskiy *et al.*, *Fast Data-Driven Simulation of Cherenkov Detectors Using Generative Adversarial Networks*, J. Phys. Conf. Ser. **1525** (2020) 012097, arXiv:1905.11825.

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

[181] LHCb collaboration, L. Anderlini *et al.*, *Towards Reliable Neural Generative Modeling of Detectors*, J. Phys. Conf. Ser. **2438** (2023) 012130, arXiv:2204.09947.

[182] R. E. Bellman, *Adaptive Control Processes*, Princeton University Press, 1961.

[183] F. Ratnikov *et al.*, *A full detector description using neural network driven simulation*, Nucl. Instrum. Meth. A **1046** (2023) 167591.

[184] CMS collaboration, F. Vaselli, *FlashSim: accelerating HEP simulation with an end-to-end Machine Learning framework*, CMS-CR-2023-090, 2023.

[185] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, *nflows: normalizing flows in PyTorch*, 2020. doi: 10.5281/zenodo.4296287.

[186] A. Paszke *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, in *33rd International Conference on Neural Information Processing Systems (NeurIPS)*, 8024–8035, 2019, arXiv:1912.01703.

[187] 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.

[188] C. Krause and D. Shih, *Fast and accurate simulations of calorimeter showers with normalizing flows*, Phys. Rev. D **107** (2023) 113003, arXiv:2106.05285.

[189] O. Amram and K. Pedro, *Denoising diffusion models with geometry adaptation for high fidelity calorimeter simulation*, arXiv:2308.03876.

[190] F. Ratnikov and A. Rogachev, *Fast simulation of the electromagnetic calorimeter response using Self-Attention Generative Adversarial Networks*, EPJ Web Conf. **251** (2021) 03043.

[191] J. Albrecht *et al.*, *Upgrade trigger & reconstruction strategy: 2017 milestone*, LHCb-PUB-2018-005, CERN, Geneva, 2018.

[192] M. Gori, G. Monfardini, and F. Scarselli, *A new model for learning in graph domains*, in *2005 IEEE International Joint Conference on Neural Networks (IJCNN)*, **2** 729–734, 2005.

[193] A. Dosovitskiy *et al.*, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, arXiv:2010.11929.

[194] M. Zaheer *et al.*, *Deep Sets*, in *31st International Conference on Neural Information Processing Systems (NeurIPS)*, 2017, arXiv:1703.06114.

[195] ATLAS collaboration, *Deep Sets based Neural Networks for Impact Parameter Flavour Tagging in ATLAS*, ATL-PHYS-PUB-2020-014, CERN, Geneva, 2020.

[196] K. Lee *et al.*, *ViTGAN: Training GANs with Vision Transformers*, in *10th International Conference on Learning Representations (ICLR)*, 2022, arXiv:2107.04589.

[197] S. Brody, U. Alon, and E. Yahav, *How Attentive are Graph Attention Networks?*, in *10th International Conference on Learning Representations (ICLR)*, 2022, arXiv:2105.14491.

[198] K. Albertsson *et al.*, *Machine Learning in High Energy Physics Community White Paper*, J. Phys. Conf. Ser. **1085** (2018) 022008, arXiv:1807.02876.

[199] LHCb collaboration, L. Anderlini, *Machine Learning for the LHCb Simulation*, arXiv:2110.07925.

[200] N. M. developers, *NVIDIA Modulus: A Framework for Developing Physics Machine Learning Neural Network Models*, 2023. Software available from https://developer.nvidia.com/modulus.

[201] J. Bradbury *et al.*, *JAX: composable transformations of Python+NumPy programs*, 2018. Software available from http://github.com/google/jax.

[202] ONNX Runtime developers, *ONNX Runtime*, 2021. Software available from https://onnxruntime.ai.

[203] D. H. Guest *et al.*, *Lightweight Trained Neural Network*, 2019. doi: 10.5281/zenodo.3366463.

[204] V. Kuznetsov, L. Giommi, and D. Bonacorsi, *MLaaS4HEP: Machine Learning as a Service for HEP*, Comput. Softw. Big Sci. **5** (2021) 17, arXiv:2007.14781.

[205] L. Giommi, D. Spiga, V. Kuznetsov, and D. Bonacorsi, *Prototype of a cloud native solution of Machine Learning as Service for HEP*, PoS **ICHEP2022** (2022) 968.

[206] S. An and L. Moneta, *C++ Code Generation for Fast Inference of Deep Learning Models in ROOT/TMVA*, EPJ Web Conf. **251** (2021) 03040.

[207] T. Chen *et al.*, *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*, in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018, arXiv:1802.04799.

[208] S. Cyphers *et al.*, *Intel nGraph: An Intermediate Representation, Compiler, and Executor for Deep Learning*, arXiv:1801.08058.

[209] N. Vasilache *et al.*, *Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions*, arXiv:1802.04730.

[210] N. Rotem *et al.*, *Glow: Graph lowering compiler techniques for neural networks*, arXiv:1805.00907.

[211] Google XLA team, *Xla - tensorflow, compiled*, https://developers.googleblog.com/2017/03/xla-tensorflow-compiled.html.

[212] M. Li *et al.*, *The deep learning compiler: A comprehensive survey*, IEEE TPDS **32** (2021) 708, arXiv:2002.03794.

[213] J. Nordby, *emlearn: Machine Learning inference engine for Microcontrollers and Embedded Devices*, 2019. doi: 10.5281/zenodo.2589394.

[214] R. Conlin, K. Erickson, J. Abbate, and E. Kolemen, *Keras2c: A library for converting Keras neural networks to real-time compatible C*, Eng. Appl. Artif. Intell. **100** (2021) 104182.

[215] J. H. Friedman, *Greedy function approximation: A gradient boosting machine*, The Annals of Statistics **29** (2001) 1189.

[216] C. Bozzi, *LHCb Computing Resource usage in 2022*, LHCb-PUB-2023-002, CERN, 2023.

[217] I. Belyaev *et al.*, *PYTHON-based Physics Analysis Environment for LHCb*, LHCb-PROC-2004-021, 2004.

[218] LHCb collaboration, *Validation of the Lamarr framework with $\Lambda_b^0 \to \Lambda_c^+ \mu^- X$ decays*, LHCB-FIGURE-2022-014, 2022.

[219] W. D. Hulsbergen, *Decay chain fitting with a Kalman filter*, Nucl. Instrum. Meth. **A552** (2005) 566, arXiv:physics/0503191.

[220] LHCb collaboration, *LHCb data released to the public*, https://lhcb-outreach.web.cern.ch/2022/12/02/lhcb-data-released-to-the-public.

[221] S. Li, J. Pfeifer, B. Perozzi, and D. Yarrington, *Introducing TensorFlow Graph Neural Networks*, https://blog.tensorflow.org/2021/11/introducing-tensorflow-gnn.html.