# University of Padova

Department of Physics and Astronomy

*Master Thesis in Physics of Data*

# Optimization of the PID algorithms at the Belle II Experiment

*Supervisor*
Prof. Alessandro Gaz
University of Padova

*Master Candidate*
Ali Bavarchee

*Student ID*
1219425

*Academic Year*
2022-2023

"Dedication"

— For Hana-Chan

# Abstract

**Particle identification in the Belle II experiment** involves utilizing information from various sub-detectors to classify six different species of charged particles: electrons, muons, charged pions, charged kaons, protons, and deuterons. Previous studies have demonstrated that directly adding log-likelihoods from each detector for each hypothesis is not an optimal use of available information since poorly calibrated detectors can hurt overall particle identification performance. To address these issues, we study different approaches that involve assigning to the individual contributions different weights, depending on the region of the phase space under study. Machine learning tools are employed in order to optimize the weights and study the possible improvements in the performance.

# Contents

# Listing of figures

# Listing of tables

# Listing of acronyms

**NP** ..... New Physics

**CP** ..... Charge-Parity

**SM** ..... Standard Model

**PXD** ..... PiXel Detector

**PDF** ..... Probability Density Function

**ADC** ..... Analog-to-Digital Converter

**SVD** ..... Silicon Vertex Detector

**CDC** ..... Central Drift Chamber

**TOP** ..... Time-Of-Propagation

**ARICH** ..... Aerogel RICH

**RICH** ..... Ring-Imaging CHerenkov

**ECL** ..... Electromagnetic Calorimeter

**KLM** ..... $K_L$ - Muon Detector

**LER** ..... Low Energy Ring

**HER** ..... High Energy Ring

**HEP** ..... High Energy Physics

**ML** ..... Machine Learning

**NN** ..... Neural Network

**CNN** ..... Convolutional Neural Network

**RF** ..... Random Forest

**ROC** ..... Receiver Operating Characteristic

# 1 Introduction

## 1.1 Belle II Experiment

Belle II is a next-generation flavour factory designed to search for new physics (NP) in the flavour sector at the intensity frontier and improve the precision of Standard Model (SM) measurements [2]. The SuperKEKB facility collides electrons and positrons at center-of-mass energies near the $\Upsilon$ resonances, and the asymmetry in beam energies provides a boost to the center-of-mass system for time-dependent charge-parity (CP) symmetry violation measurements. SuperKEKB has a design luminosity of $6 \times 10^{35}\text{cm}^{-2}\text{s}^{-1}$, about 30 times larger than KEKB's recorded peak and about 80 times KEKB's design luminosity. The first collision data-taking run was undertaken in 2018 for commissioning. The SM is the best-tested theory of nature, but it does not explain many fundamental questions, such as the hierarchy in fermion masses, the number of fermion generations, the Higgs boson accounting for neutrino masses, and the matter-antimatter asymmetry in the universe. Many NP scenarios have been proposed, and experiments in high-energy physics aim to address these questions. At the energy frontier, the LHC experiments aim to discover new particles produced in proton-proton collisions, while at the intensity frontier, experiments like Belle II can observe signatures of new particles or processes through measurements of suppressed flavour physics reactions or from deviations from SM predictions. Belle II and SuperKEKB will exploit their strengths at the intensity frontier by characterizing NP through over-constraining measurements in several related flavour

physics reactions.

## 1.2 SuperKEKB

KEKB was upgraded to SuperKEKB to increase instantaneous luminosity using the Nano-Beam scheme and doubled beam current, and the beam energy asymmetry was reduced to mitigate the Touschek effect, resulting in a reduced Lorentz boost. In June 2010, KEKB was closed down and upgraded to SuperKEKB with the aim of increasing the instantaneous luminosity to a level 30 times higher than the maximum instantaneous luminosity achieved by KEKB. To achieve this, the Nano-Beam scheme- invented by P. Raimondi, was adopted, which requires a larger crossing angle to fit the final focusing magnets and therefore, the beam size at the collision point was reduced by a factor of 20 and the currents were increased by a factor of 1.5 compared to KEKB values. Additionally, the beam current in both HER and LER rings was doubled, and the beam energy asymmetry was reduced to counteract the shortened beam lifetime caused by the Touschek effect. As a result, the Lorentz boost was decreased to ($\langle\beta\rangle \approx 0.284$). A summary of the relevant machine parameters for both KEKB and SuperKEKB is presented in Table 1.1.

| Machine Parameter | KEKB HER($e^-$) | KEKB LER($e^+$) | SuperKEKB HER($e^-$) | SuperKEKB LER($e^+$) |
|---|---|---|---|---|
| Beam current (A) | 1.64 | 1.19 | 3.60 | 2.61 |
| Energy (GeV) ($E_{HER}/E_{LER}$) | 8.0 | 3.5 | 7.0 | 4.0 |
| $\beta_y^*$ (mm) | 5.9 | 5.9 | 0.27 | 0.41 |
| Crossing angle (mrad) | 22 | 22 | 83 | 83 |
| Beam lifetime (min) | 200 | 150 | 10 | 10 |
| Luminosity ($10^{34}$ cm$^{-2}$ s$^{-1}$) | 2.11 | 2.11 | 80 | 80 |

**Table 1.1:** Achieved parameters of KEKB and design parameters of SuperKEK [5] [2]

As it is mentioned in Table 1.1, the collision angle was increased to 83 mrad, and a slightly lower beam energy asymmetry was chosen to reduce beam losses due to Touschek scattering.

This reduction will lead to a decrease in the spatial separation between B-mesons, but it will improve the solid angle acceptance for missing energy decays.

## 1.3 Data-Taking Status

Belle II's physics program focuses on studying rare decays and CP asymmetries in $B$ decays, along with a range of other topics including $b$-quarks, $c$-quarks, $\tau$-leptons, two-photon physics, quarkonium, and exotic physics. Recent emphasis has been on understanding QCD's role in 4 and 5-quark states and searching for a dark sector. Extended run periods at various energy regions, including $\Upsilon(1S)$, $\Upsilon(2S)$, $\Upsilon(3S)$, $\Upsilon(5S)$, and near $\Upsilon(6S)$, along with fine energy scans, will address open questions. Additionally, measurements at $\Upsilon(5S)$ will provide insights into $B_s$ decays. Data-taking at SuperKEKB is performed in two main phases [2]:

- the first collision data taking phase: SuperKEKB and the interaction region was commissioned before the installation of the sensitive silicon inner detectors. The peak luminosity delivered by SuperKEKB reached $0.5 \times 10^{34}/\text{cm}^2/\text{s}$, and a data set of order $0.5\ \text{fb}^{-1}$ was collected at the $\Upsilon(4S)$ resonance. This small data set has been used for searches of dark sectors that were previously limited by a lack of efficient triggers.
- The second collision phase had been seen the full detector and allowing for the full flavour program to commence.

The Belle II experiment has gathered an extensive amount of data, exceeding $400\ \text{fb}^{-1}$. The majority of these data were obtained at an energy level corresponding to the mass of the $\Upsilon(4S)$ particle which is reported in Fig. 1.1.

## 1.4 Belle II Detector

The Belle II detector is a significant upgrade to its predecessor, Belle, designed to enhance the particle identification performance and the decay vertex reconstruction capabilities in harsher beam background conditions. It consists of various sub-detectors arranged cylindrically around the collision point of the $e^+e^-$ beams, which are enclosed by a beryllium beam pipe with a radius of 1 cm.

The innermost part of the detector is the PiXel Detector (PXD), which consists of two layers of DEpleted P-channel Field Effect Transistors (DEPFET). The first layer includes 16 sensor modules arranged on eight ladders, while the second layer has four sensor modules mounted

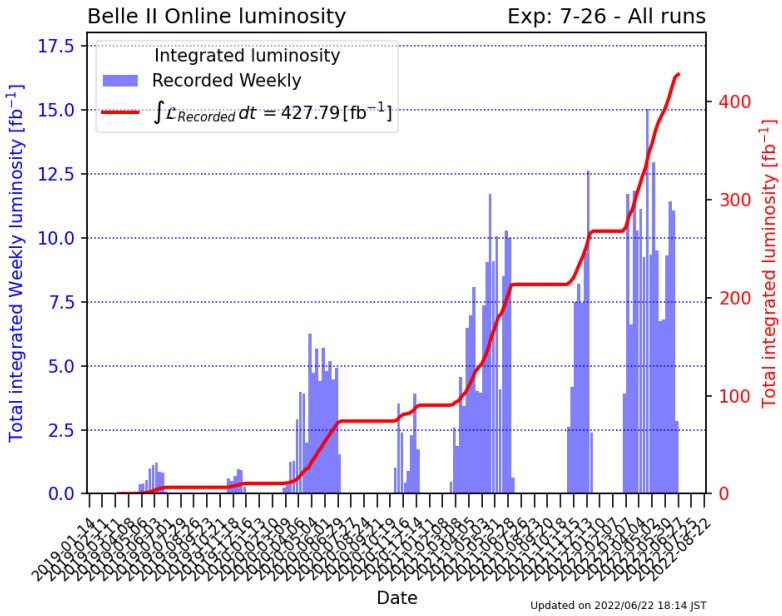


**Figure 1.1:** Total ingrediented weekly data taking progression from Jan 2019 to Aug 2022

only on two ladders, due to construction-related limitations, and the installation of the complete PXD is currently underway. The PXD is surrounded by the Silicon Vertex Detector (SVD), which is a four-layer double-sided silicon strip detector arranged cylindrically around the interaction point (IP). The first layer is straight, and the second to fourth layers are slanted to reduce the material budget and the number of sensors. Both PXD and SVD offer the necessary radiation hardness required to operate at the high luminosity of SuperKEKB. Together, PXD and SVD form the silicon-based vertexing (VXD) inner detector.

The VXD is enclosed by the Central Drift Chamber (CDC), which is filled with a He (50%) and $C_2H_6$ (50%) gas mixture. The CDC has 56 layers of 14,336 wires of axial or stereo orientation, grouped in cells of nine, with one "sense" wire to which HV is applied surrounded by eight grounded "field" wires. The wires are grouped into nine superlayers, and by combining the information of axial and stereo wires, the full three-dimensional trajectory of a charged particle can be reconstructed, providing a measurement of the mean ionization energy loss, $dE/dx$.

The Belle II detector is designed to operate at a 30 times higher luminosity compared to Belle and with background rates higher by a factor of 10 to 20. Therefore, higher event rates require modifications to the trigger scheme, data acquisition system, and computing compared to the previous experiment. In addition, the Belle II detector needs to improve hadron identification and maintain hermeticity at least as good as the original Belle detector. The detector's critical

issue is mitigating the effects of higher background levels, which lead to an increase in occupancy and radiation damage, as well as to fake hits and pile-up noise in the electromagnetic calorimeter, and to neutron-induced hits in the muon detection system. The trigger and DAQ are also adapted to support a broader low-multiplicity (dark sector) physics analysis program.

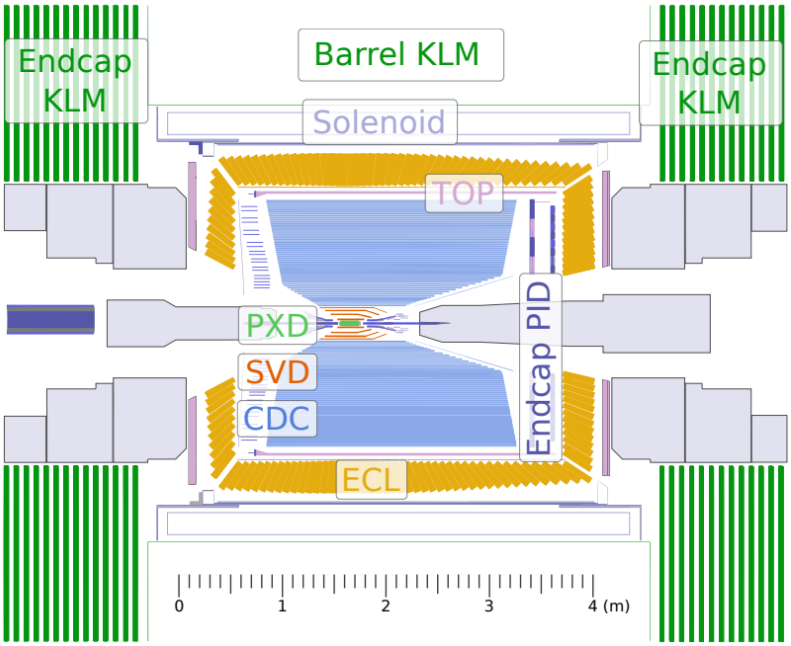


**Figure 1.2:** Belle II detector side view [1]

Figure 1.3 illustrates the design of the Belle II detector, which consists of various components for detecting different types of particles. The detector is made up of two layers of pixelated silicon sensors (PXD) and four layers of silicon strip sensors (SVD) closest to the beam pipe to reconstruct decay vertices. The central drift chamber (CDC) fills the larger outer radius of the tracking volume. There is also a time of propagation system in the barrel region (TOP) and a ring-imaging Cherenkov detector in the forward endcap (ARICH) specifically designed for identifying hadrons. A CsI(Tl) electromagnetic calorimeter (ECL) is used to measure the energy of photons and electrons, and it has a longitudinal size of 16.2 $X_0$ in units of radiation length. Finally, the KLM detector is composed of scintillator strips and resistive plate chambers, which are used for $K_L^0$ meson and muon identification.

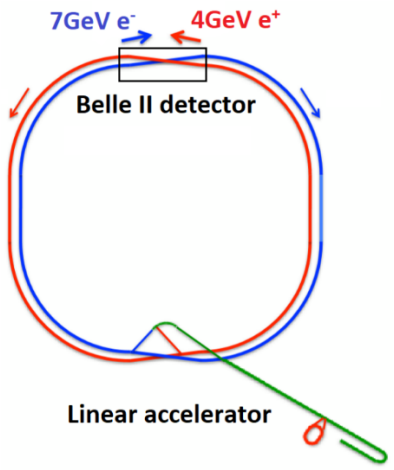


**Figure 1.3:** SuperKEKB and the Belle II detector top view [2]

### 1.4.1 Vertex detector (VXD)

The vertex detector is made up of two devices, the silicon Pixel Detector (PXD) and Silicon Vertex Detector (SVD), with a total of six layers around a 10 mm radius beam pipe. The first two layers at radii of 14 mm and 22 mm use pixelated sensors of the DEPFET type, while the remaining four layers at radii of 38 mm, 80 mm, 115 mm, and 140 mm are equipped with double-sided silicon strip sensors. The outermost layer of the new detector is at a considerably larger radius compared to the Belle vertex detector, which is expected to result in significant improvements in momentum resolution and reconstruction efficiency for certain types of decays.

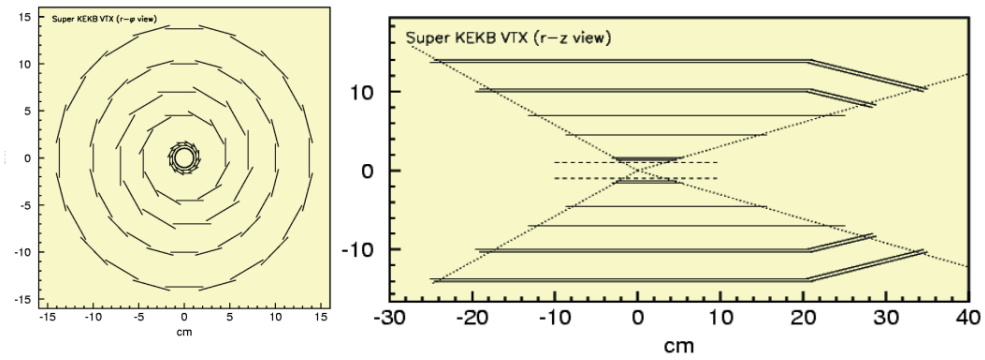


**Figure 1.4:** A schematic view of the Belle II vertex detector [2].

It is possible to distinguish between different mass hypotheses by combining the track momentum ($p_{\text{lab}}$) and the mean energy loss by ionisation ($dE/dx$) in the SVD strips of a charged

particle, which is related to its velocity through the Bethe-Bloch formula, and plotting them in the $(dE/dx, p_{\text{lab}})$ space [6].

Although excluding the SVD from the LID likelihood model [6] has a small impact on performance overall, adding this information improves the identification of soft leptons that curl back inside the CDC, particularly in terms of reducing the misidentification rate of pions.

To extract the probability density function (PDF) for the electron hypothesis, one can reconstruct photon conversion candidates ($\gamma \to e^+e^-$) that occur within the material of the PXD and SVD inner tracking systems. A converted photon candidate is identified by combining two oppositely charged tracks fitted with the electron mass hypothesis. The sample purity is enhanced by applying a dielectron invariant mass selection in a $3 < M_{ee} < 15$ MeV/c$^2$ window, and by performing a vertex fit, where only photon candidates with a fit $\chi^2 > 0.001$ are kept. The $|z|$ coordinate of the photon production vertex is then restricted to be less than 8 cm to reduce secondary electrons. More details and description of $\chi$ and the concept of the purity are mentioned in the next sections [6].

The sPlot technique [6] is used to subtract the residual background, and the resulting two-dimensional distribution of $dE/dx$ vs. $p_{\text{lab}}$ is used to calculate the likelihood. The impact of including SVD likelihoods in the PID model on electron identification is evaluated in data using electrons from $\gamma \to e^+e^-$, and pions and kaons from a $D^{*+} \to D^0(\to K^-\pi^+)\pi^+$ sample. For this analysis, only tracks within the SVD detector fiducial region with $p_{\text{lab}} < 1$ GeV/c are considered. The recent study [6] finds that the electron efficiency increases from around 80% to 90% for a fixed 20% $\pi \to e$ misidentification probability, and from around 94% to 98% for a fixed 40% $K \to e$ misidentification probability.

### 1.4.2 Central Drift Chamber (CDC)

The Belle II spectrometer's central tracking device, the CDC, is a large volume drift chamber with 14,336 sense wires arranged in 56 layers. It has been designed to handle higher event rates and increased background levels, and has smaller drift cells than the one used in Belle. The CDC extends to a larger radius (1130 mm compared to 880 mm) and contains both axial and stereo layers that allow for 3D helix track reconstruction. The chamber gas is a He-$C_2H_6$ 50:50 mixture with an average drift velocity of 3.3 cm/$\mu$s and a maximum drift time of about 350 ns and it has been commissioned with cosmic rays.

For each reconstructed track in the CDC, specific ionisation ($dE/dx$) measurements are obtained. Signal pulses on each sense wire of a cell are digitised with 31.75 MSPS flash ADCs,

and values over a nominal threshold are summed to yield the raw ADC readout. Geometrical corrections are applied for track length in each drift cell ($dx$) based on the polar angle of the track and the track geometry in the cell. Variations in gas gain between data taking runs due to variations in pressure and temperature as well as gas composition are calibrated with a large sample of $e^+e^- \to e^+e^-\gamma$ radiative Bhabha events. Wire-to-wire gain variations due to variations in electronics and cell characteristics are also calibrated.

Samples of $e$, $\mu$, $\pi$, $K$, and $p$ from dedicated control samples are used to determine gas gain saturation effects for tracks with varying intrinsic ionisation (i.e., saturation relative to electrons). The same control samples are also used to parametrise the mean expected $dE/dx$ as a function of $\beta\gamma$. Corrected ADC values from each hit (i.e., sense wire) on a given track are used to calculate a truncated mean by discarding the lowest 5%— and highest 25%— of measurements on a given track and averaging the remainder. Truncation yields a nearly Gaussian average from a highly skewed initial ADC distribution.

Distributions of the measured truncated mean are used to calculate a normalised deviation for each charged particle hypothesis using the formula [6]:

$$\chi_h = \frac{\Delta_{dE/dx}}{\sigma_{pred}} = \frac{dE/dx_{\text{meas}} - dE/dx_{\text{pred}}}{\sigma_{\text{pred}}}. \tag{1.1}$$

The expected spread $\Delta_{dE/dx}$, where $dE/dx_{\text{meas}}$ ($dE/dx_{\text{pred}}$) is the measured (predicted) truncated mean, is parametrised as a function of track polar angle, the number of hits on track, and the $dE/dx$ value itself. The factor $\sigma_{\text{pred}}$ at the denominator is the predicted resolution on $\Delta_{dE/dx}$, as obtained from a fit with a Gaussian function. The $\chi_h$ distributions are converted to likelihoods, assuming their PDF is well described by Gaussian functions.

The discrepancy between data and MC on the high-side tail for pions is due to imperfections in the cos $\theta$-dependent gas-gain saturation corrections.

### 1.4.3 TOP and ARICH

The Belle II spectrometer uses a combination of advanced technologies for particle identification. The time-of-propagation (TOP) counter and the Aerogel Ring Imaging Cherenkov (ARICH) detector are key components. The TOP counter uses a specialized type of Cherenkov detector while the ARICH detector employs aerogel as a Cherenkov radiator. The radiator re-

fractive index ($n$) can be determined from the Cherenkov angle resolution, which is given by [6]

$$\cos\theta_C = \frac{1}{n\beta}, \tag{1.2}$$

where $\beta = p/E$ and $p$ is the particle momentum, $m$ is the mass, and $c$ is the speed of light. Using the tracking detectors, the particle mass and identity can be determined from this relation. The TOP and ARICH detectors provide important information for particle identification, particularly for charged hadrons like pions, kaons, and protons, and also contribute to electron identification at low momenta.

The TOP counter measures the time and position where internally-reflected Cherenkov photons hit an array of photo-multipliers at the end of a fused silica bar. The time measurement can be decomposed into two parts: $t_{ph} = t_{tof} + t_{prop}$, where $t_{tof}$ is the time of flight of the particle from the IP, proportional to $1/\beta$, and $t_{prop}$ is the time spent by the Cherenkov photon inside the quartz bar, which is a function of the particle incident angle, position, and $\beta$. To achieve good particle identification, a coarse segmentation of the PMT is sufficient, while the readout timing resolution must be around 100 ps. The chosen sensors are micro-channel plate photo-multipliers (MCP-PMT) with a pixel size of about $5.5 \times 5.5$ mm$^2$ and a transit time spread of approximately 38 ps, providing a low-resolution measurement of the photon positions and a very precise measurement of their detection time. The readout is performed by the IRSX chip, capable of reaching 30 ps of timing resolution. Particle identification information is obtained by comparing the distribution of the time of arrival of the photons in each of the 512 channels with the expected probability density functions (PDFs) for the six standard charged particle hypotheses. The ratios of the six corresponding likelihood values are then used to assign identification probabilities. However, the separation power decreases as the momentum increases, as the TOP PID information is ultimately based on a measurement of the particle velocity [6].

The ARICH detector uses aerogel as a Cherenkov radiator to separate the charged particles. The aerogel produces Cherenkov photons when a charged particle passes through it. The photons are then detected by the hybrid avalanche photon detector (HAPD), a single photon sensitive high granularity sensor. The RICH uses a novel method to increase the number of detected Cherenkov photons without degrading the Cherenkov angle resolution. The information from both TOP and ARICH is crucial for the particle identification of charged hadrons and contributes to electron identification at low momenta [2].

### 1.4.4 Electromagnetic Calorimeter (ECL)

The Belle II electromagnetic calorimeter (ECL) is made up of thallium-doped caesium iodide crystals and is highly segmented. It has a total of 8736 crystals covering about 90 percent of the solid angle in the center-of-mass system [2]. The crystals were reused from Belle, and the readout electronics and reconstruction software were upgraded to mitigate the overlapping of pulses from neighboring events. Scintillator photo-sensors equipped with waveform-sampling read-out electronics were used. Radiation damage and high background rates in the forward region could degrade the performance. Hence, a replacement of CsI(Tl) with pure CsI is under study. The baseline method for charged particle identification relies on the $E/p$ ratio. Templates of $E/p$ are generated from simulated single-particle Monte Carlo samples for each charged particle hypothesis, and PDFs are extracted via adaptive Gaussian Kernel Density Estimation fits [2]. Independent fits in 18 orthogonal bins [6] are performed to account for variations in the PDF shapes as a function of polar angle, momentum, and charge. Angular and momentum binning are defined in accordance with the calorimeter geometry, and charge dependency is introduced to account for differences in ECL energy deposition patterns due to charge asymmetry in hadronic interactions. The method's performance is currently being studied and evaluated.

### 1.4.5 $K_L$- Muon Detector (KLM)

The $K^0_L$ and muon detector (KLM) in Belle II consists of iron plates and active detector elements for detecting $K^0_L$ mesons and muons. In Belle II, large background rates are expected in some KLM detector areas, which will reduce detection efficiency [2]. The muon identification algorithm proceeds in two steps: track extrapolation and likelihood extraction for each of the six charged particle hypotheses. Track extrapolation starts at the outermost point of the reconstructed track's trajectory and ends when the kinetic energy falls below a defined threshold, the track curls inwards, or the track escapes from the KLM geometrical boundaries. Likelihood values per hypothesis are obtained from precalculated probability density functions for individual particle hypotheses, charge, and extrapolation ending outcome. The longitudinal profile PDF values are sampled according to the pattern of all KLM layers crossed during the extrapolation, and the total longitudinal likelihood is the product over all layers crossed by the

extrapolated track [6].

$$\mathcal{L}^L = \prod_{k=1} \mathcal{L}_k^L \tag{1.3}$$

The transverse scattering probability density function per KLM region and particle hypothesis is sampled according to the measurement of $\chi^2$ of the Kalman filter and the number of degrees of freedom. To mitigate this problem, RPCs have been replaced by layers of scintillator strips with SiPMs as light sensors. The high neutron background will also cause damage to the SiPMs, but irradiation tests have shown that the detector system can still be reliably operated.

## 1.5 Trigger System and Data Acquisition System

The trigger system of Belle II [2] plays a crucial role in identifying events of interest during data collection. The system must work efficiently in the presence of high background rates expected from SuperKEKB and meet the limitations of the data acquisition system. A well-designed trigger system unlocks a broad range of topics not explored in previous B-factories, such as dark sector searches and axion-like particle searches. The main beam background sources, such as Touschek effect, beam-gas scattering, synchrotron radiation, radiative Bhabha process, two-photon process, and beam-beam effects, are discussed in detail. Most of these processes have topologies that are problematic for low-multiplicity production modes. The flagship measurements for Belle II in B- and D- flavour physics are expected to be highly robust to trigger implementation. Events will be easily identified from the presence of at least 3 tracks in the CDC trigger and a large deposition of energy in the ECL, similar to Belle.

The Belle II experiment uses a two-level trigger system [2] consisting of a low-level trigger (L1) and a high-level trigger (HLT) to cope with the high event rate and background levels. The L1 trigger has a latency of 5 $\mu$s and maximum trigger output rate of 30 kHz, and has been upgraded with 3D tracking algorithms, improved online reconstruction techniques for calorimeter, global reconstruction logic, and a new trigger menu. The HLT reconstructs the event with offline reconstruction algorithms and reduces online event rates to 10 kHz for offline storage. It has a trigger menu in development and employs a total of 6000 CPU cores to process at nominal 30 kHz, which is the required rate for nominal instantaneous luminosity. Dark matter searches pose a big challenge for the trigger due to the presence of only one energetic photon in the final state, and loose and tight trigger conditions are applied to suppress dominant background processes such as Bhabha scattering. The high-level trigger farm uses

data from all sub-detectors to reconstruct the event and perform a physics-level event selection. The raw event rate written to storage is anticipated to be between 6 kHz and 10 kHz. The HLT involves many clusters of Linux-based personal computers and runs the Belle II Analysis Software Framework (basf2).

## 1.6 Data Production and Reconstruction

### 1.6.1 Belle II analysis software framework (basf2)

The basf2 [7] is used for online and offline data handling. It is designed to allow independent processing modules to perform small tasks linearly within a defined path. Modules communicate by passing information to and from a common object store. The framework is important due to the enormous data output rate at Belle II, and data sets are processed in several phases, with each phase reducing and enhancing the data. High-level objects are constructed from detector information, and the event size is reduced by a factor of approximately 40. The reduced information is then used to determine particle-level information. The performance of reconstruction algorithms is given for basf2 software.

### 1.6.2 Offline Reconstruction and Monte Carlo Production

The basf2 [7] can be used for offline reconstruction, Monte Carlo production, and physics analysis in particle physics experiments such as Belle II. It is used to process the raw data obtained from the detector after machine-dependent calibration parameters have been determined. The reconstructed data is then stored in ROOT-based mDST files. Monte Carlo production and reconstruction are distributed to data centers worldwide for further analysis. Overall, basf2 is an essential tool for particle physicists to analyze the data obtained from Belle II experiment. It also uses in other simulation modules, specifically the GEANT4 toolkit, to simulate particle interactions and decays. This involves generating particles with given momentum and Particle Data Group (PDG) codes, and simulating their life story through interactions and decay processes using probabilistic relations and random number selection. MC simulations like GEANT4 are highly accurate and have found applications in various fields such as space science, medical physics, radiation protection, and nuclear medicine. Efforts are being made to further improve their accuracy as experimental data provides feedback and their physics capabilities are extended.

### 1.6.3 Belle to Belle II dataset

The Belle II dataset undergoes four levels of data processing, namely online reconstruction, offline reconstruction, mDST analysis, and n-tuple analysis. The online reconstruction involves reading the detector and trigger system, producing the raw data (DST files). The offline reconstruction involves cluster reconstruction, track finding, track fitting, and producing the mDST data. The mDST analysis involves creating final state particle hypotheses, reconstructing intermediate particle candidates and vertex fitting, and then producing flat n-tuples. Finally, the n-tuple analysis involves fitting theoretical predictions to extract interesting observables and producing scientific papers. Converting the raw data from Belle to Belle II is a challenging task due to the differences in detectors and expected backgrounds. However, converting Belle mDST data to the new mDST format used by basf2 allows for the validation of the Belle II analysis software and reproduction of Belle measurements using improved software. Figure 1.5 provides an overview of basf2, including the conversion path presented in this work. By comparing the original Belle results, the Belle results obtained from converted data in basf2, and Belle II sensitivity studies on Belle II Monte Carlo, it is possible to identify improvements in sensitivity and inconsistencies in the analysis and reconstruction algorithms separately.

### 1.6.4 Tracking

The primary goal of tracking is to reconstruct charged particles that come from the primary and secondary decay points. This involves identifying VXD and CDC hits caused by ionization from a specific charged particle amidst other background hits from detector noise, machine background, or other particles. Then, a trajectory is obtained by fitting the hit positions. The majority of tracks originate inside the beam pipe, with the exception of charged decay products from long-lived $V^0$-like particles (such as $K^0_S$, $\Lambda$, and converted photons) that are produced outside the beam pipe. The tracking algorithms are responsible for these tasks. They must identify and match the two decay products with opposite charges of $K^0_S$, $\Lambda$, and photons that undergo decay within the tracking volume.

The trajectories of reconstructed particles are utilized to align the detector. Having a detector that is optimally aligned is crucial for carrying out precise and reliable measurements of flavor quantities with time dependence. Additionally, when fitting decay chains, the knowledge of the spatial distribution of primary interactions (beamspot) can be utilized as a potent constraint that is dependent on the run.

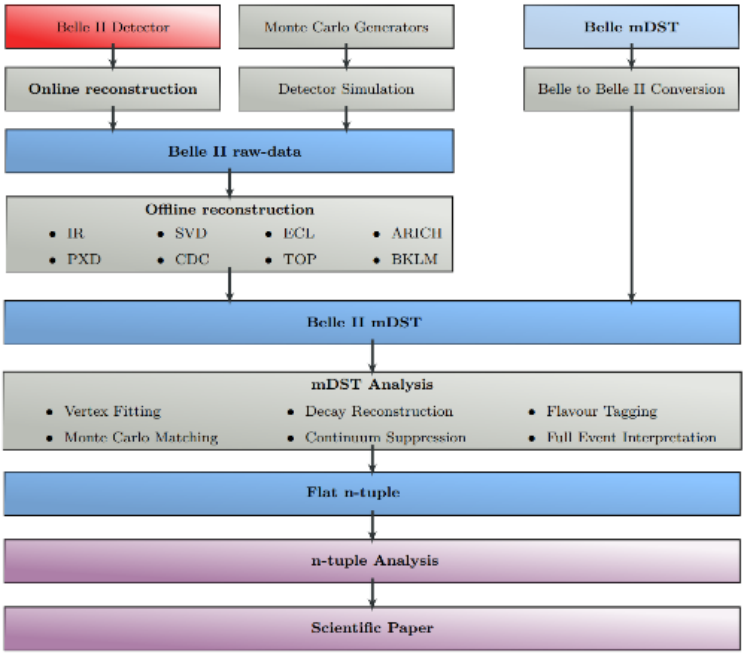


**Figure 1.5:** Schematic overview of the data-flow in the Belle II environment. Data is provided by the Belle II detector (red); the MC generators; or Belle mDST files. Basf2 is responsible for MC generation, detector simulation, online reconstruction, offline reconstruction, mDST analysis and the Belle to Belle II conversion (gray), as well as writing and reading the different data-formats (blue). Analysis-specific user-code is only required during the ntuple analysis, which extracts the desired physics observables (purple) [1] .

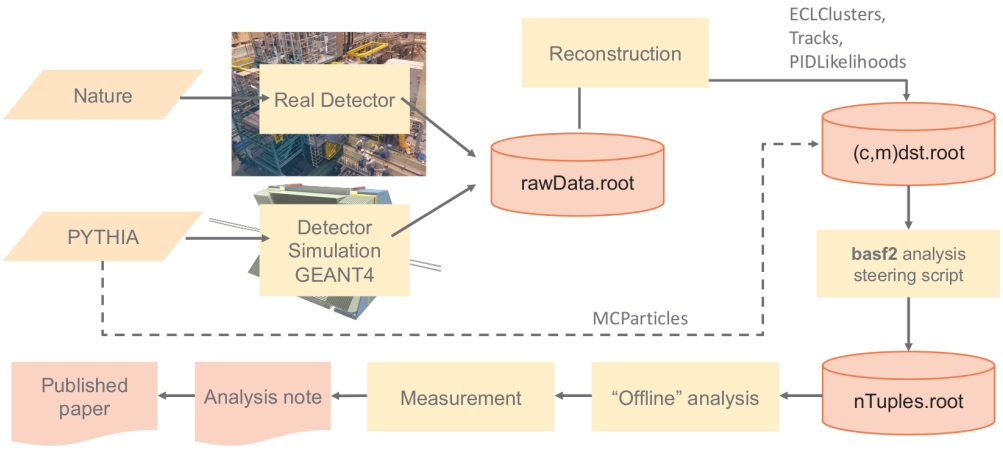


**Figure 1.6:** Belle II analysis: direct information from the real or simulated detector is saved in raw data files [3]

### 1.6.5 Reconstruction of Charged Particles

The reconstruction of charged particles involves tracking software that generates lists of charged particle paths based on a given mass hypothesis. During analysis, a track is represented by $(\vec{x}_0, \vec{p})$, where $\vec{x}_0$ is the closest point of approach to the origin of the coordinate system, and $\vec{p}$ is the particle momentum at $\vec{x}$. The detector hits associated with the track are not carried forward after tracking to reduce the size of analysis files (mDST), although additional information, such as the number of hits in each detector layer of the VXD and CDC used to fit the track, is saved for the analyst. This information is useful for selecting high-quality tracks during analysis. Charged particle reconstruction is composed of two main stages: track finding, which groups together detector hits that belong to a single track into a track candidate, and track fitting, which determines the trajectory of the track by fitting the track candidate.

In track finding, a Hopfield network is used with a quality indicator to generate non-overlapping track candidates [2]. Multiple track finder runs with different sector maps can be used for different momentum regions. Two complementary algorithms, a global and local track finder, are used for CDC track finding. The global track finder applies a Hough transformation and can handle cases with missing hits, while the local track finder uses cellular automaton to search for segments and tracks. The track candidates from VXD and CDC are merged based on the distance between the extrapolated VXD and CDC track candidates. In the future, cross-detector searches may be conducted to add hits from VXD to CDC track candidates and vice versa.

In track fitting, the tracking software uses five parameters to describe the helix-like trajectory of a charged particle in a magnetic field: $d_0$, $z_0$, $\varphi_0$, $\tan\lambda$, and $\omega$. Due to interactions with detector material and a non-uniform magnetic field, the trajectories are not perfect helices, and particle mass is used as a hypothesis to account for such interactions. The deterministic annealing filter is used as the primary track fitting algorithm, which is based on the Kalman filter (KF) but is more robust against false hit assignments and incorrect assumptions about wire passage.

# 2

# Particle Identification (PID)

## 2.1 Introduction

Particle identification (PID) is a process in particle physics that aims to determine the type of particle that has been detected. Identifying particles is crucial to understanding the physics processes that occur in high-energy experiments and for distinguishing signals from background. There are several methods used for particle identification. In the barrel region, TOP has this duty, while in the forward endcap region, ARICH, a Cherenkov ring imaging detector, by utilizing aerogel as its Cherenkov radiator, is vested upon to identify charged particles. Also, ECL measures the energy deposited by a particle as it passes through a detector. Different particles deposit different amounts of energy, and so the energy deposition can be used to identify the type of particle. In particular, electromagnetic calorimeters can distinguish between electrons and photons by measuring the pattern of energy deposition.

In order to improve the particle identification (PID) system and cope with a higher background environment, as well as to increase the spectrometer's capability to distinguish between kaons and pions, an upgrade to the PID system is necessary. Furthermore, to enhance the calorimeter response to electromagnetic particles, it is desirable to reduce the amount of material and make it more uniform. In the barrel region of the spectrometer, the current time-of-flight and aerogel Cherenkov counters have been replaced by a Time-Of-Propagation (TOP) counter. The TOP counter measures the time of propagation of Cherenkov photons

internally reflected inside a quartz radiator, and reconstructs the Cherenkov image from the 3D information provided by two coordinates and precise timing, determined by micro-channel plate PMTs. The quartz bar array surrounds the outer wall of the CDC and is divided into 16 modules in $\varphi$ in the baseline geometry. In order to reduce the impact of the chromatic effect, a focusing mechanism has been implemented that separates the ring image based on the wavelength of Cherenkov photons. This division occurs as the photons are focused onto various sections of the PMT array, depending on their respective wavelengths or energies.

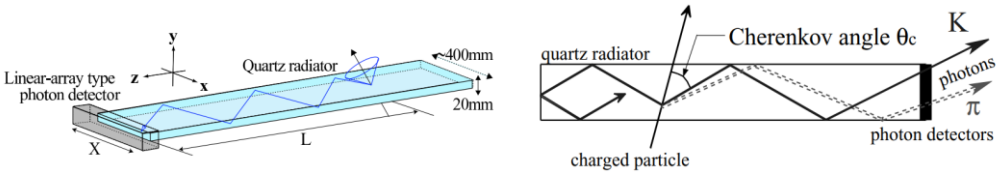


**Figure 2.1:** Conceptual overview of TOP counter (left), Schematic side-view of TOP (right) [2].

## 2.2 TOP Simulation

To explore the optimal performance of the TOP detector under different configurations, it is necessary to obtain simulation output. In order to replicate the output from a real detector after generating the four-vectors of the event. The actual detector measures the interaction between the particles and the detector's material using various processes such as bremsstrahlung, Cherenkov radiation, ionisation, and scintillation, among others. All these processes can be simulated using software, and Geant4 is the most well-known simulation software. Geant4 takes the four-vectors and simulates their interaction with a virtual Belle II detector, and the result is the deposited energy and particles produced by the interactions in each sub-detector. Once this is done, the custom software converts the output from Geant4 into signals, which are then used to determine the pixels that were fired. The simulation of the entire detector is a costly process and can take up to a second for Belle II and even minutes for experiments like ATLAS and CMS due to the high number of produced particles and the high energy involved [8].

## 2.3 TOP Reconstruction

Mainly the studies have focused on separating K and $\pi$ particles. In order to determine the expected rates of true detections and false positives, the detected photons for each track are

compared against probability distribution functions (PDFs) specific to each particle hypothesis ($P_K(x,t)$ and $P_\pi(x,t)$). The origin of the PDFs varies depending on the simulation group [2].

Based on the probability distribution functions (PDFs), a likelihood is calculated for a simulated primary charged particle could be [9]:

$$\mathcal{L}_{K,\pi} = \prod_i P_{K,\pi}\left(x,t\right)_i \tag{2.1}$$

where the index $i$ runs over each detected photon in the event. Typically the log of the likelihood is used as it is more computationally stable. The difference between the likelihood under each hypothesis is calculated [9] [6].

$$\Delta \log \mathcal{L} = \log \mathcal{L}_K - \log \mathcal{L}_\pi = \log\left(\frac{1}{P_{(\frac{\pi}{K})}} - 1\right) \tag{2.2}$$

Where $P_{(\frac{\pi}{K})}$ is the pion binary PID (LID) which is defined in equation 2.7 of section2.6.2. The log-likelihood difference is used to determine whether the detected particle is a pion or a kaon, with positive values indicating a pion and negative values indicating a kaon.
By comparing the likelihood-based classifications to the known simulated particle species, the fractions of correctly and incorrectly identified particles can be determined.

The methods use PDFs from Monte Carlo simulations, which require significant computational resources to generate for a single combination of track parameters, limiting the number of track parameters that can be evaluated for expected efficiencies and fake rates. As the Monte Carlo-based method is not practical for the final detector configuration, efforts are ongoing to develop suitable reconstruction methods, such as adapting the analytical method to the Geant-based Monte Carlo data.

°Previously, the methods so called Nagoya and Hawaii methods, the PDFs are derived from a large number of events involving single tracks with a specific momentum, impact position, and angle on the quartz bar. On the other hand, another method as known as Ljubljana reconstruction, uses the known properties and distributions of Cherenkov radiation to create PDFs through analytical calculations.

°The Ljubljana method uses analytically calculated PDFs, allowing for much finer sampling of performance across the parameter space for potential charged tracks with less computational resources. However, this method may lack some details seen under realistic operating conditions.

## 2.4 Particle Identification: forward End-cap

One of the fundamental needs for Belle II is to identify charged particles over the full kinematic range. To address this requirement, ARICH has been designed for the forward endcap. ARICH can differentiate between kaons and pions over a significant portion of their momentum spectrum and can distinguish between pions, muons, and electrons below 1 GeV/c.

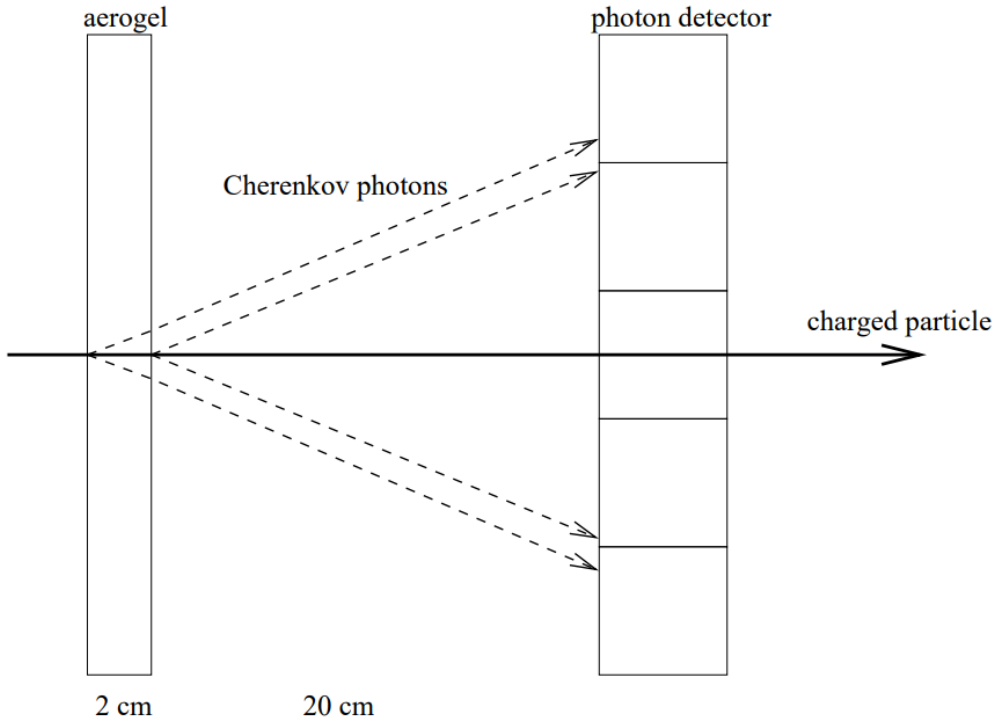


**Figure 2.2:** Proximity focusing ARICH [4]

The ARICH detector consists of several components (Fig. 2.2), including an aerogel radiator that produces Cherenkov photons when charged particles pass through it, an expansion volume that allows the photons to form rings on the photon detector surface, an array of position-sensitive photon detectors capable of detecting single photons with high efficiency and good resolution in two dimensions, and a read-out system to record the detected photons.

## 2.5 Charged particle identification

As mentioned before, the forward end-cap and barrel regions of the detector includes TOP and ARICH systems that provide charged particle information across the full kinematic range. The PID information from these detector systems is combined with specific ionisation or stopping power($dE/dx$) measurements from the SVD and CDC to be the primary sources of information for charged hadron PID. Likewise, the ECL provides the primary information for electron identification, and the KLM provides that for muon identification [6].

To determine the identity of charged particles, likelihood-based selectors are used. Each PID system provides independent information to calculate a likelihood for each charged particle hypothesis, which can then be combined to form a likelihood ratio. Analysis-specific criteria may be applied to construct prior probabilities, which, when combined with likelihoods, can determine the probability of a charged track having a specific identity. Although in principle this approach provides optimal PID performance, it requires analysis-specific optimization and does not allow for pre-determined selection efficiency uncertainty. Likelihood ratios are constructed by summing the PID log likelihoods from each detector to create a combined PID likelihood for each of six long-lived charged particle hypotheses. The difference in log likelihood between two particle hypotheses is then used to form a PID value $L(\alpha : \beta)$ [6].

$$\mathcal{L}(\alpha : \beta) = \frac{\prod_{det} \mathcal{L}(\alpha)}{\prod_{det} \mathcal{L}(\alpha) + \prod_{det} \mathcal{L}(\beta)} = \frac{1}{1 + e^{\left(\ln \mathcal{L}_\beta - \ln \mathcal{L}_\alpha\right)}} \tag{2.3}$$

Where $\mathcal{L}(\alpha : \beta)$ indicates the probability of a charged particle being of type $\alpha$ compared to type $\beta$. The value is greater than 0.5 if the charged particle is more likely to be of type $\alpha$, and less than 0.5 if it's more likely to be of type $\beta$. The performance results presented in this section were obtained from large simulated datasets of $10^6$ $c\bar{c}$ events generated.

### 2.5.1 Ionisation Energy Loss Measurement

To determine the ionization energy loss, $dE/dx$, of a charged particle moving through the Belle II detector, measurements from the VXD and CDC are used. The $dE/dx$ measurement is expected to only depend on the particle velocity, $\beta\gamma = p/m$. However, calibration is needed to avoid systematic effects that can break this dependence. Generally, the $dE/dx$ information is more effective at discriminating between particle momenta below 1 GeV/c.

### 2.5.2 Determination of Likelihoods of Charged Particles

The VXD and CDC detectors measure ionisation energy loss, dE/dx, independently, and both require different calibration procedures. At present, the dE/dx algorithms in both subsystems construct likelihood values using information from individual hits. A lookup table constructed from large MC samples is used to determine a likelihood value for each particle hypothesis, including pion, kaon, proton, muon, electron, and deuteron. To reduce the effect of non-Gaussian tails, the lowest 5% and highest 25% dE/dx measurements of each track are not used in the likelihood determination.

The basf2 could use a truncated mean and resolution parameterisation to determine dE/dx PID discriminators. This method will involve comparing the measured dE/dx truncated mean to a predicted value and resolution and determining a $\chi_h$ value.

$$\chi_h = \frac{I_{\text{mean}} - I_{\text{pred},h}}{\sigma_{\text{pred},h}} \tag{2.4}$$

The predicted values are calculated from a parameterisation of dE/dx as a function of $\beta\gamma$, and the predicted resolutions depend on the dE/dx measurement, the number of hits on the track, and the polar angle of the track. The distributions of this $\chi$ variable are approximately Gaussian and can be converted to a likelihood and combined with the output of other PID systems. This method is expected to have similar performance to the current method but enable a better characterization of the resolution.

## 2.6 PID in Belle II

The PID in Belle II involves using both "Global" and "Binary" likelihood ratios for each particle type which are invoked "Global PID" and "Binary PID" respectively [10]. These ratios are calculated based on the likelihoods from different subsystems within the detector. The analysis framework provides access to relevant variables, and analysts can choose to use predefined values ("tight", "loose") or directly cut on the likelihoods to identify the particles as electrons, muons, pions, kaons, protons, or deuterons. Since the MC (Monte Carlo) modelling used is not perfect, the performance of particle identification needs to be verified using data and corrections derived from MC.

### 2.6.1 GlobalPID

The main idea behind global PID is to use the log-likelihoods of all six charged particle types to identify the particle type. For each event, the detector provides log-likelihood values for each charged particle type, denoted by $\log L_{h,d}$ where $h$ is the particle type hypothesis and $d$ is the detector. These log-likelihood values are combined for each particle type to form an ensemble hypothesis log-likelihood, represented by $\log \mathcal{L}_h$ [10] [2].

$$\log \mathcal{L}_h = \sum_d \log \mathcal{L}_{h,d} \tag{2.5}$$

The sum is performed over all six detectors for six particle types that are used for PID. The log-likelihoods obtained from this sum can be utilized to calculate likelihood ratios, as indicated by the following equation.

$$p(h) = \frac{\mathcal{L}_h}{\sum_{h'} \mathcal{L}_{h'}} = \frac{\exp(\log \mathcal{L}_h)}{\sum_{h'} \exp(\log \mathcal{L}_{h'})} \tag{2.6}$$

where as before, $h$ is the particle type hypothesis of all the detectors and the sum over $h'$ is over all six hypotheses.

### 2.6.2 Binary PID

Binary PID is a particle identification approach that involves the use of only two particle types as hypotheses: the particle type of interest and a background particle type, which is often pions ($\pi$). This approach is particularly useful in situations where pions form a significant background in charged particle identification, and the scope of other particle types is limited in many analyses.

In binary PID, the likelihoods of the two particle types are compared, and the particle is classified as either the particle type of interest or the background particle ($\pi$), based on which hypothesis has a higher likelihood. For instance, in the case of pion separation from kaons, the likelihood ratio is computed using the log-likelihood values for each particle type, and the particle is identified as a kaon if the likelihood ratio for kaon is higher than that of pion [10].

$$P(h_\pi)_{Binary} = P_{(\frac{\pi}{K})} = \frac{\mathcal{L}_\pi}{\mathcal{L}_\pi + \mathcal{L}_K} \tag{2.7}$$

Selecting a particle type based on its global likelihood ratio being the largest also means it

would have been chosen using binary likelihood ratios with a threshold of 0.5 as binary likelihood ratios is often the case that particles are then identified according to whether $P(h) > 0.5$.

This is because choosing a hypothesis based on global likelihood ratios means that the probability of that hypothesis is greater than the probability of all other hypotheses. This in turn means that the log likelihood of that hypothesis is greater than the log likelihood of all other hypotheses, which implies that the binary likelihood ratio for that hypothesis is greater than 0.5. However, this does not work the other way around. In this document, global PID methods are used instead of binary PID methods, and particles are identified using the argument that maximizes the function, which is defined as arg max. When binary PID methods are used, this will be clearly indicated.

## 2.7 Physics Samples

The decay $D^{*+} \rightarrow D^0(\rightarrow K^-\pi^+)\pi^+$ is a standard sample for particle identification studies of charged pions and kaons. In this decay, the charge of $\pi_{\text{tag}}$ (the tagged pion) determines the flavor of the $D^0$ at production time, and the flavor at decay time is determined by the final state particle properties. By selecting charged tracks based on their sign and applying a set of cuts on the reconstruction quality and kinematic properties of the $D^*$ and $D^0$, this decay can be reconstructed cleanly and used to study particle identification performance with minimal bias on the observed quantities.

In the absence of mixing, the initial $D^0$ decays as $D^0 \rightarrow K^-\pi^+$. However, if mixing occurs, the $D^0$ converts to a $\overline{D^0}$ and decays as $\overline{D^0} \rightarrow K^+\pi^-$, with the charge of the $K$ determining the flavor at decay time. Therefore, by observing the charge of the $K$ in the final state, one can determine if mixing occurred.

The $D^*$ method [2] is a well-established "golden" flavour-tagging method used at B-Factories, which identifies the flavour of the $D^0$ meson by the charge of the pion emitted with the $D^0$ in the $D^{*+}$ decay. A new flavour-tagging method, the ROE (the rest of the event) method [2], has been introduced to increase the sample size of tagged $D^0$ candidates.

°The ROE method adds $D^0$ mesons produced in $c\bar{c}$ events that do not come from $D^{*+}$ decays. The principle of the ROE method is to look at the rest of the event with respect to the neutral $D$ meson whose flavour we want to tag, and the method is performed by selecting events with only one $K^{\pm}$ in the ROE and using the charge of the kaon to determine the flavour of the other $D^0$ meson at the time of its production. This method uses a multivariate classification technique to select $K^{\pm}$ candidates in the ROE and applies a pre-selection of tracks based on PID and track fit probability to reject poorly measured candidates.

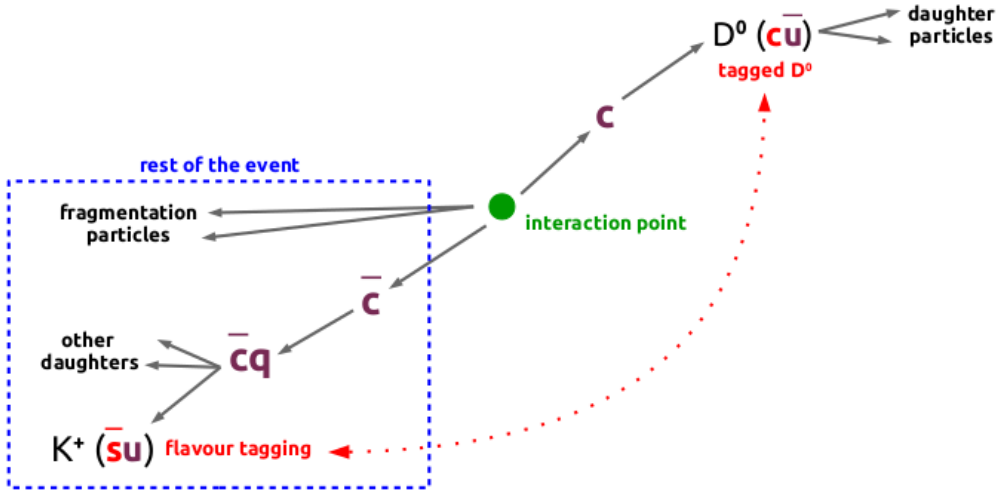


**Figure 2.3:** The ROE flavour-tagging method relies on a certain principle. It involves selecting events that contain only one $K^{\pm}$ in the ROE, and then determining the neutral $D$ meson's flavour based on the charge of that kaon [2].

# 3

# Machine Learning For HEP

## 3.1 Multivariate Analysis (MVA)

Multivariate Analysis (MVA) algorithms utilize multivariate statistics to analyze observed data $\vec{x}$ and derive characteristics of the underlying process that generated the data [1]. These algorithms are commonly used for tasks such as estimating a function $f(\vec{x})$ (known as a regression task), differentiating data points into signal and background (a classification task), and grouping similar data points together (clustering). Machine learning (ML) is an effective way to automatically learn the required statistical model by using a domain-specific dataset, with knowledge extracted from experience. This can be seen as a type of artificial intelligence. The machine learning algorithms typically involve two phases: a fitting-phase where a statistical model is learned from the training dataset, and an inference-phase where the model is used to predict target information for a new test dataset. Three types of learning are identified based on the feedback available during the fitting-phase: supervised learning, unsupervised learning, and reinforcement learning. In high-energy physics (HEP), target information is typically known from Monte Carlo simulated events or can be inferred from data-driven techniques, making supervised learning the most commonly used approach. In supervised learning, the target information $y \sim f(\vec{x}) + \varepsilon$ is predicted by a statistical model $\widehat{f}$ using a feature vector $\vec{x}$. The algorithm adapts the internal weights of the statistical model (represented by $\vec{w}$) to minimize the discrepancy between the true value $y$ and the predicted value $\widehat{y} = \widehat{f}(\vec{x}, \vec{w})$, where the discrepancy is

defined by a loss-function $L(y, \widehat{y}, \vec{w})$ [1].

## 3.2 Tuning Model

The model trains and tests on both real data (comes from detector read-out system) and Monte Carlo simulation data, which are readily available and sufficient to evaluate the effectiveness of particle identification (PID) methods. The simulations involve a pure kaon and pion particle gun that simulates particle tracks and potential decay from the collision point, including some background radiation that creates noise. This is to simulate real-world noise encountered by the models. The simulations fix the particle gun to one point in phase space, shooting all particle tracks at angles of $\theta = [10.0, 170.0]$ at the origin and collision point with the quartz bar, respectively, with a range of momentum of $P = 0.2 \sim 0.5$ GeV [10]. Classification models that perform well on this simulation data can be extended to a larger portion of the phase space and potentially integrated into the real Belle II particle identification process. The dataset initially contains 1466586 events each, but for simplicity, events with only one registered particle track in the module, positively charged particles, and events registered on Module 3 are selected, resulting in a reduced dataset of 1056529 kaons and 830675 pions. The data is randomly split into 60% training data, 20% validation data for tuning hyperparameters during training, and 20% testing data for final model evaluation.

## 3.3 Convolutional neural network (CNN)

A convolutional neural network (CNN) is a type of deep neural network used to recognize visual patterns from pixel images. The PIDML project trains two separate CNNs with 7x7 pixel images of kaons and pions, crystal positions, and transverse momentum of the track in the laboratory frame [11]. Separate CNNs are trained for positive and negative charged tracks due to the geometry of the detector. The CNN architecture has the desirable property of local connectivity, which allows it to summarize spatial and temporal correlation in the data. The same CNN architecture is used for both the quantile and uniformly spaced binning approaches. The testing approach involves a relatively small CNN consisting of two convolutional layers, followed by a max-pooling layer and then three dense layers. The CNN uses the rectified linear

°"The B2BIIConvertMdst module converts the information stored in the Bell PANTHER tables and writes it to the Belle II DataStore. The beam-energy and IP-profile is collected in the Basf2 BeamParameters object and stored in the condition Database of Belle II[1]."

activation function. The image-like representation of the hit grid approaches is a common way to solve this problem, but it has certain shortcomings. One is that it reduces temporal resolution, which is one of the great strengths of the TOP detector. There are also unforeseen impacts on model performance due to hyperparameter choices and data distribution. Approximately 100,000 single kaon and pion candidates are generated, and each track is first reconstructed in the inner tracking detectors and then extrapolated into the Belle II detector with Geant4. Two types of inputs are used for the CNN, the energy depositions in the 7x7 pixel images, and a set of inputs fed after the convolutional layers that include pT, thetaID, and phiID of the extrapolated tracks. A threshold value of 1 MeV on the energy depositions in the pixels is applied, and the pT is already in the range of 0.2 - 1.0 GeV/c, so no scaling is applied. The thetaID and phiID are used as categorical variables, which are implemented as an embedding in the network. The CNN includes two parts, the first of which involves the 7x7 pixel images of kaons and pions as inputs for a convolutional layer. The second part of the training involves a feed-forward neural network (FNN). In this part, the results from the convolutional layer must be flattened and added on top of pT, thetaID, and phiID. The dropout layer is used between the first two layers of FNN [12].

## 3.4 Evaluation the Performance

Basically our initial approach involves using the confusion matrix calculation. A confusion matrix is a table used to evaluate the performance of a classification model. It compares the predicted class labels with the true class labels of a set of data, and shows the number of correct and incorrect predictions for each class. The matrix is often used to calculate several evaluation metrics, including accuracy, precision, recall, and F1-score. In Python, the scikit-learn (sklearn) package provides a simple way to generate a confusion matrix.

This entails identifying the predicted particle type for each event, and then constructing a matrix where each element in row $i$ and column $j$ represents the number of actual particles of type $i$ that were classified as type $j$. If there are no misclassifications, the values in the off-diagonal elements of the matrix will be zero.

Two ways to normalize the matrix are [10] [9]:

- Efficiency matrix, where each row is normalized so that the sum of each row equals one. The value in the $i$th row and $j$th column represents the fraction of particles that are truly

of type $i$ that are identified as type $j$:

$$efficiency(i) = \frac{N(\text{predicted } i \text{ and true } i)}{N(\text{true } i)} \tag{3.1}$$

- Purity matrix, where each column is normalized so that the sum of each column equals one. The value in the $i$th row and $j$th column represents the fraction of particles that are identified as type $j$ that are truly of type $i$:

$$purity(i) = \frac{N(\text{true } i \text{ and predicted } i)}{N(\text{predicted } i)} \tag{3.2}$$

and

$$\text{fake rate} = \frac{N(\text{false } i \text{ and predicted } i)}{N(\text{true } i)} \tag{3.3}$$

To assess how well a particle detector performs based on momentum and track angle (specifically $\theta$), we summarize the information from the confusion matrix into a simpler metric, which involves the signal efficiencies and the overall accuracy.

$$\text{accuracy} = \sum_i \frac{N(\text{true } i \wedge \text{predicted } i)}{N(\text{events})} \tag{3.4}$$

The accuracy represents the proportion of particles that are correctly identified out of all the events, and can be seen as a weighted average of the signal efficiencies, where the weight is the number of events of each particle type. To examine how the performance varies with momentum and track angle, it could be categorized the events into bins based on $p$ and $\theta$, which are the same as those used for correcting the Belle II systematics. Due to the underlying assumptions and symmetries of the problem, the focus is primarily placed on the pion efficiency and the fake rate associated with kaon identification. To elaborate, when studying the decay chain of D mesons, we make specific assumptions and consider certain symmetries of the problem. As a result, we narrow down our attention to evaluating the effectiveness of detecting pions accurately and determining the rate at which kaons are falsely classified.

The pion efficiency refers to the ability of the neural network to correctly identify and classify pions within the decay chain. It measures the proportion of true positive pion identifications out of all the actual pions present. A high pion efficiency indicates that the neural network is effectively detecting and classifying pions.

On the other hand, the kaon fake rate evaluates the rate at which the neural network

misidentifies a kaon instead of a pion. This rate measures the proportion of false positive kaon identifications among all the identified kaons. A lower kaon fake rate implies that the neural network is accurately distinguishing between pions and kaons.

By focusing on these specific aspects, we can assess the performance of the neural network within the constraints of the assumptions and symmetries relevant to the decay chain of D mesons in the Belle II experiment. The confusion matrix provides a comprehensive overview of the network's performance, considering true positive, false positive, true negative, and false negative classifications, which are then used to derive various evaluation metrics such as precision, recall, and F1-score.

For each $(p, \theta)$ bin, we calculate the efficiencies and accuracy for the six particle types. It can be demonstrated these values as a function of $p$ and $\theta$ using two-dimensional visualization techniques that are similar to those used for the confusion matrices.

## 3.5 Area Under Receiver Operating Characteristic

Receiver Operating Characteristic (ROC) is a graphical representation of the performance of a binary classification model. It is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity) for various thresholds of the model's predicted probabilities.

The true positive rate (TPR) is the proportion of actual positives that are correctly identified by the model, while the false positive rate (FPR) is the proportion of actual negatives that are incorrectly classified as positive by the model. The ROC curve shows how the TPR and FPR change as the threshold for classification is varied.

The Area Under the ROC Curve (AUC-ROC) is a scalar value that summarizes the overall performance of the model across all possible thresholds. It ranges from 0 to 1, with 0.5 indicating a random model and 1 indicating a perfect model. A model with a high AUC-ROC indicates that it can distinguish between the positive and negative classes with high accuracy, regardless of the threshold used. A model with an AUC-ROC of 0.5 is no better than a random guess.

## 3.6 Random Forest Regression

To date, various methodologies have been employed in the Belle II experiment to address PID, including Deep Neutral Network, BDT, and feed-forward Neural Network. This motivated us to utilize the random forest approach. The random forest (RF) method is an ensemble learning

algorithm used for classification, regression, and feature selection. It involves creating multiple decision trees, each using a randomly selected subset of features and training data. The decision trees learn to predict based on the features and target values of the training data.

When a new data point is presented, the algorithm uses each decision tree to make a prediction and combines the results to produce a final prediction. This approach is less prone to overfitting than a single decision tree and can handle high-dimensional data with many features.

The RF method can also provide insights into the relative importance of different features in the prediction task. It has been applied to various domains, including image classification, gene expression analysis, and fraud detection.

One advantage of using RF in HEP is that it can be used for both regression and classification tasks, allowing researchers to predict continuous variables or classify events into different categories. Additionally, Random Forest can handle missing or noisy data, making it robust to experimental uncertainties.

The Random Forest Regression is a type of supervised learning algorithm that utilizes an ensemble learning technique for regression and works by following a two-step process. First, it creates a specified number of decision tree regressors (called n-estimators), which are built based on predefined hyperparameters like the minimum number of samples required at leaf nodes and the maximum depth of the trees. Second, the algorithm combines the predictions from each of the decision tree regressors by taking the average of their outputs. This averaging process is what produces the final output of the random forest regression algorithm.

# 4 Methodology

## 4.1 Monte Carlo Simulation

First step in methodology as usual is the simulation. Monte Carlo (MC) simulation is used to generate sample data for various purposes, such as testing detector performance on Particle Identifications, optimizing data analysis methods, and predicting the outcomes of experiments. To generate MC data, a number of steps are typically taken. First, the physics processes of interest are modeled using theoretical calculations and/or experimental measurements. Next, software programs such as Pythia, Geant4 etc are used to simulate the interactions between particles and the detector, taking into account the energy, momentum, and other properties of the particles involved.

### 4.1.1 Generating ParticleGun Simulations as Background

ParticleGun (PG) events are simulated events where particles are generated with specific properties such as momentum, direction, and vertex location. These events are typically used for detector studies, optimization of event selection criteria, and testing of reconstruction algorithms.

To generate the PG samples, by executing a python script, the ParticleGun module is used to generate events with a specified PDG code (six type charged particles), number of tracks (1), and momentum, theta, phi, and vertex generation parameters. The momentum is generated

uniformly between 0.2 and 5 GeV/c, the theta angle is generated uniformly (in cosine), between 10 and 170 degrees, and the phi angle is generated uniformly between -180 and 180 degrees. The vertex location is fixed at $(0, 0, 0)$.

The generated events are then simulated and reconstructed using other modules added to the main path. The final output is in the form of mDST (minimum bias) data, which corresponds to the center of the detector, the nominal point of collision of electron and positron beam, and can be further analyzed using basf2.

### 4.1.2 Generating Signal Samples

To generate signal samples (Charmed Particles) for this study, a Python script is used that imports several modules including generators, simulation, L1trigger, reconstruction, and mDST. The main function creates a path where all the modules will be added. The script generates CCbar events using a decay file, simulates the detector response, performs reconstruction, and produces an mDST output file (see 1.6.3) with the specified file number. The number of events to be generated is set to 500000. The script needs a decay file which defines the decay of a virtual photon (vpho) into a charm-anticharm quark pair using the PYTHIA 91 definition. It also defines the decay of an anti-D meson into an anti-$D^0$ meson and a negative pion ($\pi^-$) using the VSS (Vector, Scalar, Scalar) model, and the decay of the anti-$D^0$ meson (charmed particles) into a positive kaon ($K^+$) and a negative pion using the PHSP (Phase Space) model. In addition to the decay models, the code also sets values for various Pythia parameters such as etaSup, stopMass, and rFactC. These parameters control aspects of the hadronization process in the simulation. The script checks if there is only one command line argument, which should be the file number.

### 4.1.3 Making Ntuple

The Ntuple is typically stored in a binary file, with each event represented by a single row or entry in the file. Ntuples are typically produced by detector simulation and reconstruction software, and can be used for a variety of purposes, including performance studies, physics analysis, and detector calibration.

°The VSS model which is used to describe a decay process and assumes that the decay proceeds via the emission of a virtual vector meson, which then decays into two scalar mesons.

°The PHSP model is an assumption to interpret a decay process. The model takes that the decay products are emitted randomly in phase space, subject only to conservation laws such as energy and momentum conservation.

The script by exploiting PG and CCbar samples, reconstructs a decay chain of D mesons

$$D^{*+} \to D^0 \pi^+_{(\text{slow})} \qquad \textit{and} \qquad D^0 \to K^- \pi^+$$

In $D^*$ decay, pion is called slow pion. Slow pions are produced in strong interaction processes and have lower momenta compared to other pions in the event. The term "slow" indicates that the pion has relatively low energy, which means it has a small momentum and short lifetime. Therefore, slow pions are important as criterion for the $D^{*+}$ decay.

The script first loads in the input data file containing the simulated events, then loads standard particles (charged particles and photons) for use in the analysis. It tightens the selection for pions and kaons by selecting only those with a transverse momentum greater than 0.3 GeV/$c$, and cuts on the distance of closest approach to the interaction point ($d_0$) and longitudinal position ($z_0$).

The script then reconstructs the decay chain using the modularAnalysis package. It first reconstructs the $D^0$ meson from the $K^-$ and $\pi^+$ tracks, and applies a mass cut of $1.794 < M < 1.934$ GeV/$c^2$. The decay is then matched to Monte Carlo truth information. Next, the $D^{*+}$ meson is reconstructed from the $D^0$ and $\pi^+$ tracks, with a mass cut of $1.960 < M < 2.060$ GeV/$c^2$. The decay is also matched to Monte Carlo truth information. The script then adds aliases for several additional variables to be saved in the output file, including variables related to the kinematics in the center-of-mass frame and extra particle identification (PID) variables for each detector type and particle types.

## 4.2 Particle identification calibration

The PID calibration is a crucial step in the data analysis of any high-energy physics experiment, including Belle II. The aim of PID calibration is to determine the response of the detector to different particle types and momenta, and to establish the particle identification criteria used in the analysis.

PID calibration involves several steps:

- TOP Calibration: The process of calibrating the Time of Propagation (TOP) detector in Belle II involves measuring the time it takes for charged particles to travel from the interaction point to the detector. To perform the calibration, the time response of the TOP detector is measured using cosmic rays and charged particles that come from known decay modes. From these measurements, the time offsets and resolutions for each detector module are determined.

- Calibration of ARICH: The ARICH measures the Cherenkov radiation emitted by charged particles traveling through aerogel. The Cherenkov angle, which is the angle between the particle trajectory and the emitted radiation, is used to identify the particle type. The calibration involves measuring the Cherenkov angle using charged particles from known decay modes and then determining the particle identification criteria based on the Cherenkov angle distribution.

- Calibration of CDC: The process of calibrating the CDC involves using cosmic rays and charged particles with known decay modes to measure the drift velocity and magnetic field in the CDC. This allows the CDC to accurately measure the momenta of charged particle tracks. The calibration also includes determining the momentum resolution and scale for each layer of the detector.

- Calibration of ECL: The ECL measures the energy of photons and electrons. The calibration involves measuring the energy response of the ECL using electrons and photons from known decay modes, and then determining the energy resolution and scale for each detector module.

After the calibration process is accomplished, PID criteria can be set for each detector by analyzing the measured responses. The iterative process of PID calibration involves refining the criteria based on the analysis of more data as it is collected.

In this study, we try to find the optimal combination of the detectors likelihood by adjusting the likelihood of the detectors by extracting a $6 \times 6$ Weighted matrix which each element belongs to represent the likelihood of each six detectors and thereafter, evaluating their likelihood ratios.

### 4.2.1 Optimization of Calibration and Extraction of Weighted Matrix

The Weighted Matrix refers to a matrix that the columns of the matrix correspond to the detectors (SVD, CDC, TOP, ARICH, ECL, KML) and the rows correspond to the charged particles (e, $\mu$, $\pi$, K, p, d) and if take each elements as $w_{h,d}$ (as discussed in Chapter 2), then:

$$\log(\tilde{\mathcal{L}}_h) = \sum_{d'} w_{h,d'} \log \mathcal{L}_{h,d'} \rightarrow w_d \log \mathcal{L}_{h,d} \tag{4.1}$$

where a tilde denotes that the quantity is derived from a weighted combination of detector

information and let $w_{h,d} = w_d$ for $\forall\, h$, we would find

$$\tilde{P}(h) = \frac{\exp(w_d \log \mathcal{L}_{h,d})}{\sum_{h'} \exp(w_d \log \mathcal{L}_{h',d})} \rightarrow \frac{\exp(\log \tilde{\mathcal{L}}_h)}{\sum_{h'} \exp(\log \tilde{\mathcal{L}}_{h'})} \tag{4.2}$$

Note that in the case where all weights are unity, e.g. $w_{h,d} = 1$ for all $h, d$, we exactly recover the standard PID quantities.

The weight matrix has been trained on a dataset to learn the optimal set of weights for a particle identification (PID) algorithm. The matrix consists of a set of weights that are applied to various input variables in the PID algorithm to optimize the identification of different particle species. The weights are trained using a machine learning algorithm on a training dataset, and are then applied to a test dataset to evaluate the performance of the PID algorithm. The weighted matrix is a set of weights that are loaded from a file ('models/net_wgt.npy') and applied to the PID variables in the weighted DataFrame to produce a set of weighted PID variables. These variables are then used to evaluate the performance of the PID algorithm on the test dataset. The weighted matrix is also used to create a PIDCalibrationWeight object that can be used to apply the same set of weights to PID variables in other analysis tasks.

## 4.3 Tuning ML algorithms

### 4.3.1 Data Preparation

The initial stage in addressing a machine learning problem is data preparation and management, which is also true for this problem. The script, PrepPIDTrainingSample.py, reads a ROOT file containing Monte Carlo data into a pandas DataFrame, and then employs the *pidDataUtils* module to create slim HDF5 files for each particle type, which are subsequently merged into a single HDF5 file. The merged data is subsequently divided into training, validation, and testing sets for PID optimization weights training. Lastly, the *pidTrainWeights.py* script is invoked to train the weights with the slim_dstar data and a pre-trained neural network model. The script has the following steps:

1. Import required packages/modules: *basf2*, *uproot*, *pidDataUtils*, and *subprocess*.
2. Define the filename of the input ROOT file.
3. Using *uproot*, read the data from the ROOT file into a pandas DataFrame.

4. Employ the *make_h5* function from *pidDataUtils* to generate slim HDF5 files for each particle type ($\pi$, K) and store them in the 'data' directory.

5. Utilize the *merge_h5s* function from *pidDataUtils* to merge the slim HDF5 files into one large file and store it in the 'data' directory.

6. Using the *split_h5* function from *pidDataUtils*, divide the data in the merged HDF5 file into training, validation, and testing sets and save them to the 'data/slim_dstar' directory.

7. Invoke the *pidTrainWeights.py* script using the *subprocess* module to train the PID calibration weights with the slim_dstar data and a pre-trained neural network model. The script's arguments which can be changed are as follows:

    - The directory that contains the slim_dstar data.
    - The path to the pre-trained neural network model.
    - The number of epochs for training.
    - The particle types for which to train the weights.

### 4.3.2 Training Models: Deep Neural Network

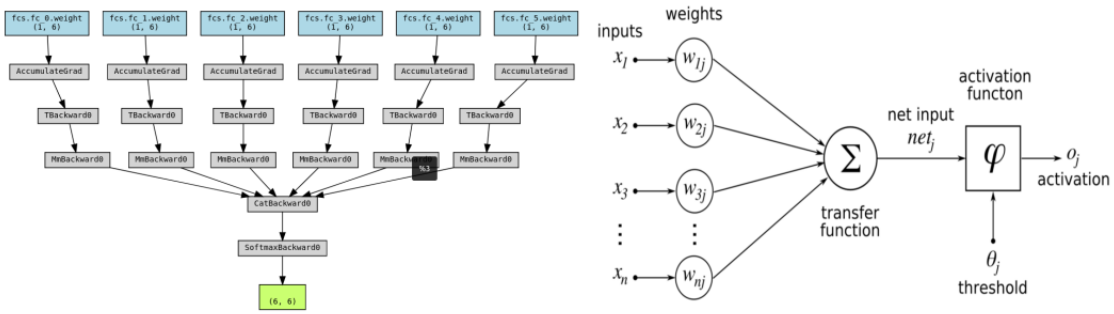


**Figure 4.1:** Visualization of the PyTorch neural network model (WeightNet) with nn.Linears that is trained over the dataset and gets a six by six matrix as an output called weighted matrix (left). Schematic of a typical neural network (right).

The PID calibration weights are trained using a Deep Neural Network with the PyTorch framework. The `pidTrainWeights.py` script begins with importing libraries and defining a private function to define constant lists. Next, a PyTorch neural network module called WeightNet is defined with several methods. The script also defines a function train_net() that

takes in several arguments to loop over the training set for the specified number of epochs and train the network using backpropagation. The function prints out the training loss and accuracy, as well as the test loss and accuracy at each log interval and saves the model after each epoch. Finally, a main function is defined to parse command-line arguments and run the train_net() function with specified arguments such as the directory containing the training and test data, the path to save the trained model, and the number of training epochs.

### 4.3.3 Random Forest

The RF_NET class contains an RF Regressor with fit(), predict(), and score() methods. Additionally, it contains a get_weights() method to return the feature importances as a six by six matrix.

The class ROC_Analysis is also defined to compute and plot the ROC curve for a given model's predictions. The method Compute_ROC_Curve() computes the FPR (False Positive Rate), TPR (True Positive Rate), and AUC score of the ROC curve, and the method Plot_ROC_Curve() is used to plot the ROC curve with the relevant points and threshold values. The method Find_Closest_Parameters() finds the closest FPR, TPR, and threshold values for a given threshold list fpr_th. The method Display_Results() returns a DataFrame containing information about the False Positive Rate, True Positive Rate, and threshold values for a given fpr_th.

Finally, a function RF_Best_Model() is defined to perform a grid search over RF_NET using a given hyperparameter dictionary Hparams and return the best model. The RF Weight Matrix gets at the end.

### 4.3.4 SamplePIDAnalysis

`SamplePIDAnalysis.py` shows how to use trained weights to perform particle identification (PID) in an analysis. The script assumes that the training data has already been prepared using `PrepPIDTrainingSample.py` script, and the weights have already been trained using some machine learning algorithm.

The script first imports necessary modules and reads in a test dataset in `npz` format using `pdu.read_npz()` function from the `pidDataUtils` module. Then, it makes a copy of the DataFrame and loads the trained weights using `np.load()`. The script prepares two DataFrames for analysis: one for standard PID and the other for weighted PID using the calibration weights. The `pdu.produce_analysis_df()` function is used to prepare the DataFrames.

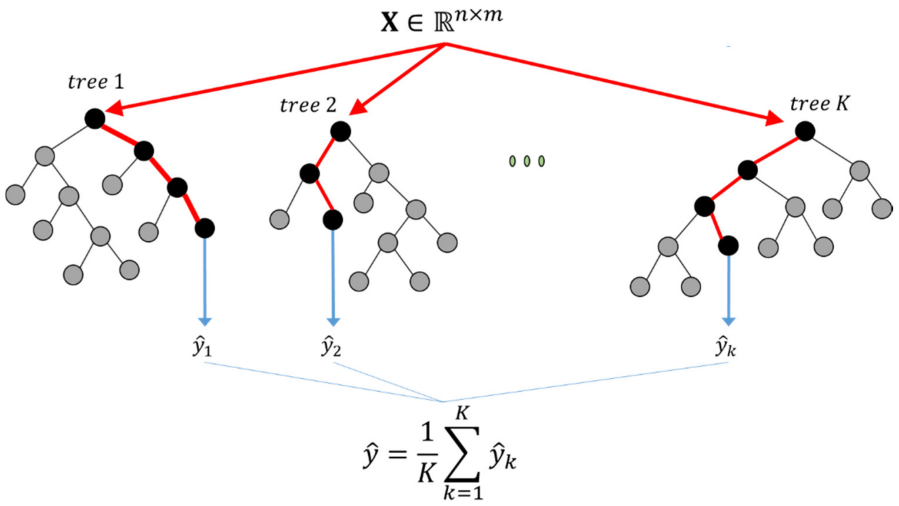


**Figure 4.2:** Schematic Random Forest Regression Model

Next, the script defines a function `compute_accuracy()` to compute the overall accuracy and the pion and kaon efficiencies for both standard and weighted PID. The accuracy is computed as the fraction of events where the predicted particle type matches the true particle type. The pion and kaon efficiencies are computed as the fraction of true pions and true kaons that are correctly identified by the algorithm.

Finally, the script prints out the accuracy and the pion and kaon efficiencies for both standard and weighted PID, and creates a `PIDCalibrationWeight` data object using the PIDCalibrationWeightCreator module. The `PIDCalibrationWeight` data object can be used to apply the calibration weights in other analyses.

### 4.3.5 ApplyWeight

The `ApplyWeight.py` script shows how to apply the PID optimization weights to data and analyze the performance of the weighted PID variables. The script sets up the local database, loads an input ROOT file, fills a particle list, and uses `variablesToNtuple()` to create an ntuple with the PID variables. The resulting ntuple can be used to analyze the performance of the weighted PID variables. Finally, `process()` is called to run the analysis.

# 5 Conclusion

**Results and Outcomes**

This section is dedicated to assessing two machine learning models for pion and kaon PID. The accuracy and efficiency of these models are investigated using MC Data and real data produce at Belle II. The neural network and random forest models were evaluated, and their performances were compared [12].

The results show that the pion identification accuracy significantly improved from 0.711 to 0.897 after implementing the neural network model. This improvement is reflected in the area under the curve (AUC) of the receiver operating characteristic (ROC) curve, as shown in Fig. 5.1.

Additionally, the random forest model increased the pion identification accuracy from 0.711 to 0.919, as demonstrated in Fig. 5.2. As mentioned before, pion identification accuracy is the measure of how well a particle detector can correctly identify pions among other charged particles. Pion identification accuracy is typically expressed in terms of two quantities: the efficiency and the fake rate. The pion efficiency is the fraction of true pions that are correctly identified as pions by the detector, while the fake rate is the fraction of particles identified as pions that are actually kaons (or different particle types).

The weighted matrices of the neural network and the random forest model, along with their visualizations, are reported in Figures 5.3 and 5.4, respectively. Additionally, the normalized matrix of the random forest's weighted matrix is demonstrated in Fig. 5.5. Upon analyzing the provided matrices, we can compare the weighted matrix of the neural network (Fig. 5.3) with

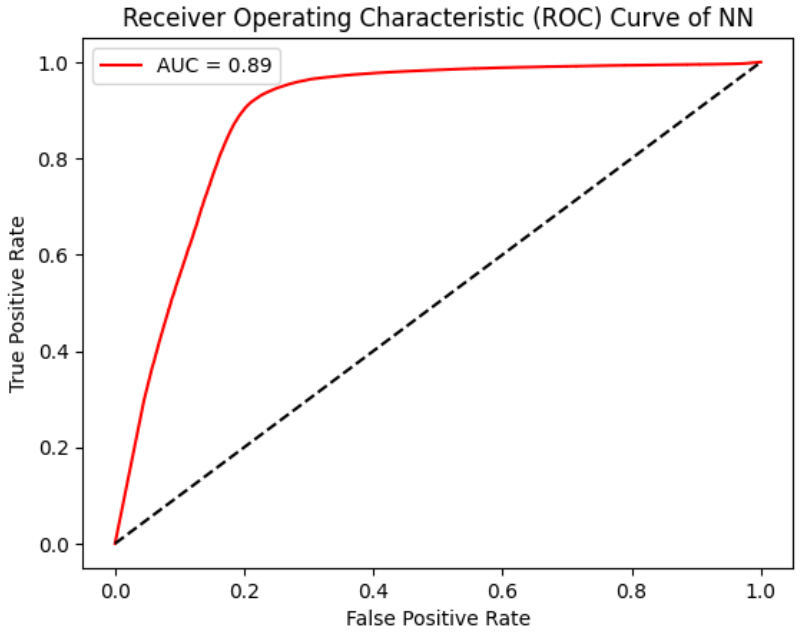


**Figure 5.1:** The Area Under the Receiver Operating Characteristic Curve (AUC ROC) of the neural network. To compute the AUC ROC of a neural network, first it is needed to calculate the true positive rate (TPR) and false positive rate (FPR) for each possible threshold of the predicted probabilities, then plot the TPR against FPR for all possible thresholds to create the ROC curve. The AUC ROC is the area under this curve, which ranges from 0 to 1, with a higher value indicating better performance.

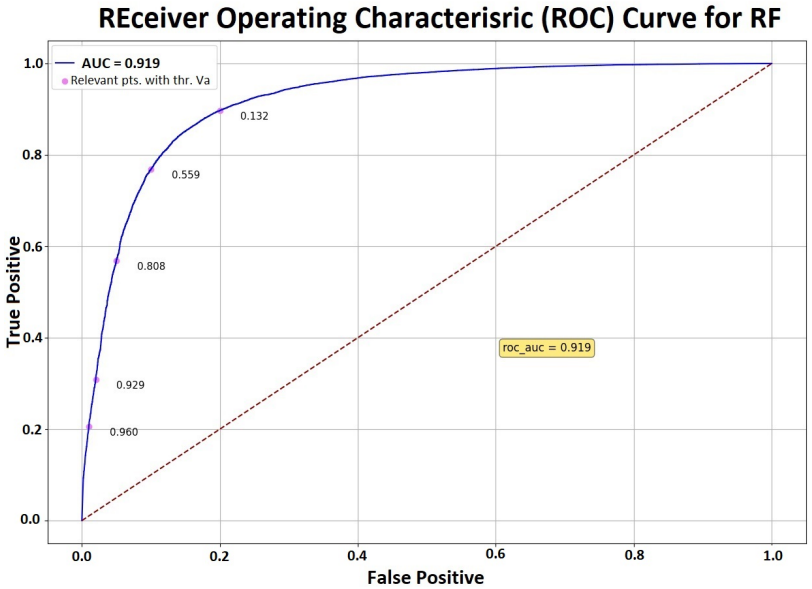


**Figure 5.2:** The Area Under the Receiver Operating Characteristic Curves of performing random forest models by GridSearch (top). The AUC ROC of the best Random Forest model (bottom).

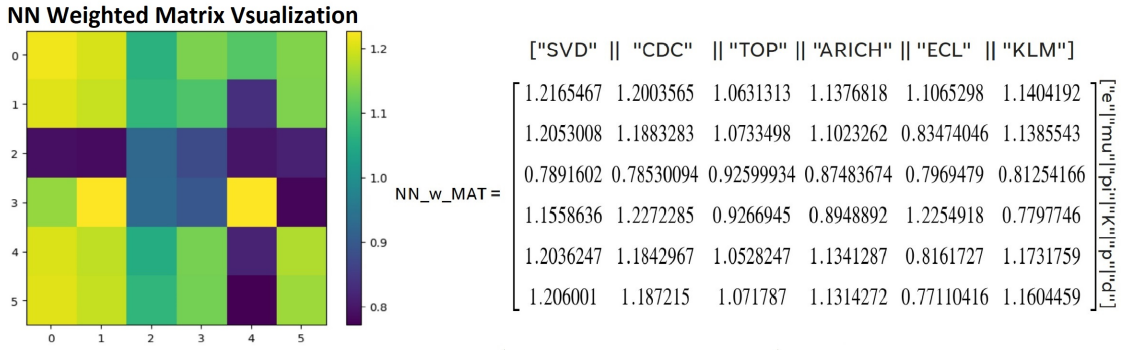


**Figure 5.3:** The weighted matrix contains dimensionless quantities extracted from aneural network model (right) and its visualization by heat map (left)

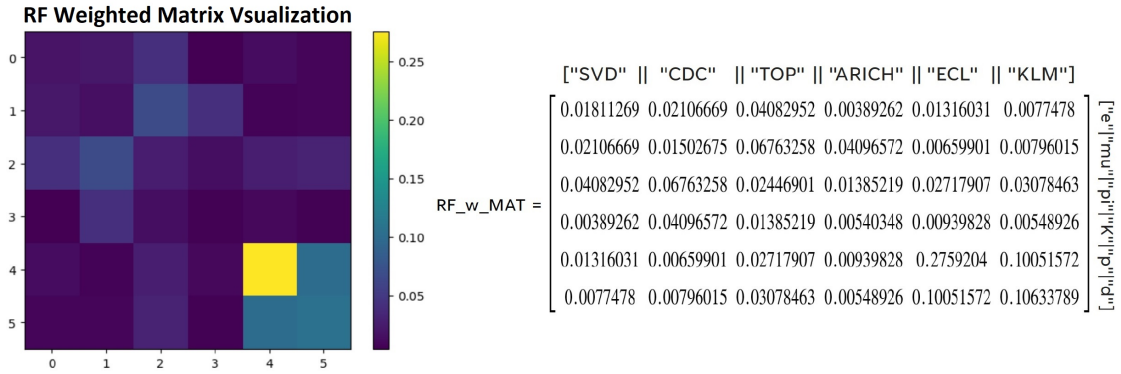


**Figure 5.4:** The weighted matrix contains dimensionless quantities extracted from a Random Forest (right) and its visualization by heat map (left)

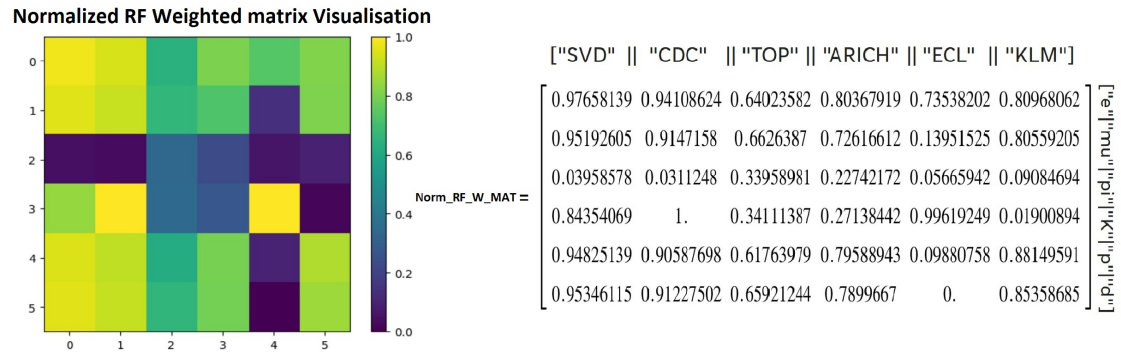


**Figure 5.5:** Normalized Random Forest's weighted matrix (right) and its visualization by heat map (left)

the random forest matrix (Fig. 5.4). The neural network's weighted matrix displays values ranging from approximately 0.77 to 1.23, indicating the relative importance assigned to different sub-detectors and particle types. In contrast, the random forest matrix exhibits considerably smaller values, ranging from approximately 0.0039 to 0.2759, suggesting a significant difference in the assignment of importance by the random forest model across various sub-detectors and particle types.

When comparing the neural network and random forest matrices, we observe distinct distributions. The neural network's weighted matrix exhibits diverse patterns and a broader range of values, implying a more intricate and nuanced allocation of importance. On the other hand, the random forest matrix appears more uniform, indicating a balanced assignment of feature importance across the sub-detectors and particle types.

Furthermore, by comparing the normalized weighted matrix of the random forest in Fig. 5.6 with the neural network's weighted matrix, we find notable similarities. The normalized matrix reflects the relative importance of features within the random forest model. Remarkably, both the neural network and the normalized random forest matrix demonstrate similar trends in the allocation of feature importance. This suggests that, despite differences in range and behavior, the neural network and random forest model exhibit comparable patterns when prioritizing the sub-detectors and particle types.

According to the presented plots in Figures 5.6 to 5.9, it can be observed that the efficiencies of kaon and pion decrease as momentum increases, while their respective false detection rates increase. Furthermore, particle tracks that are in closer alignment with the Belle II detector are detected with greater efficiency for both types of particles, as opposed to tracks that have a more oblique angle.

The evaluations of the models were demonstrated using histograms for MC data. The pion/kaon binary likelihood ratios of each detector before and after updating weights are shown in Fig 5.10.

Comparatively, after updating sub-detectors likelihoods by random forest weight matrix, the likelihoods of any detector before and after correction are compared in Fig. 5.11. In other words, Fig. 5.10 and Fig. 5.11 depict two sets of plots, each containing six sub-plots representing binary likelihood ratios of pion/kaon for different sub-detectors. In Fig 5.10, the likelihood ratios of pion over kaon are shown before and after applying weights derived from a neural network. On the other hand, Fig 5.11 compares the likelihood ratios before and after applying weights obtained from a random forest model. In the set of Fig. 5.10, the first plot represents the SVD. After applying the weights from the neural network, the binary likelihood ratios

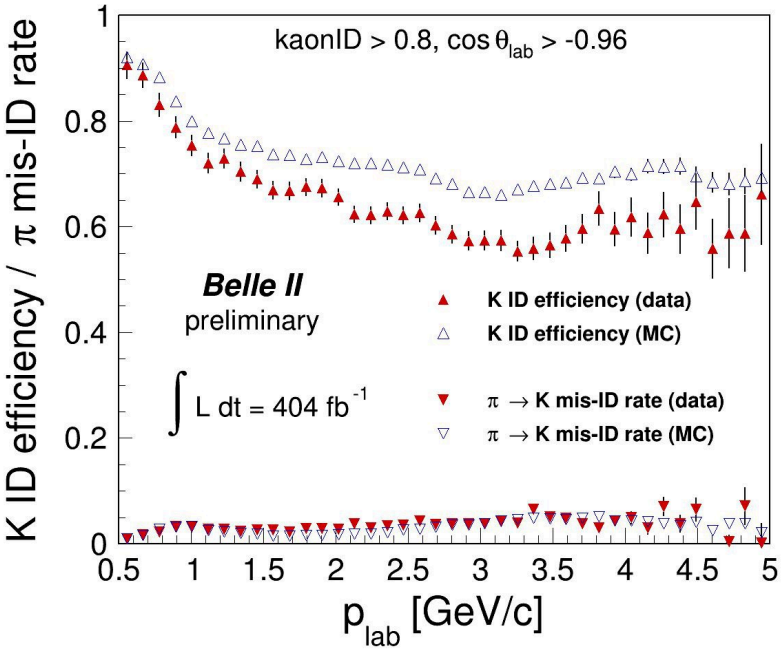


**Figure 5.6:** K signal efficiency and $\pi$ misidentification rates as a function of p on collision $D^*$ decay MC and data.

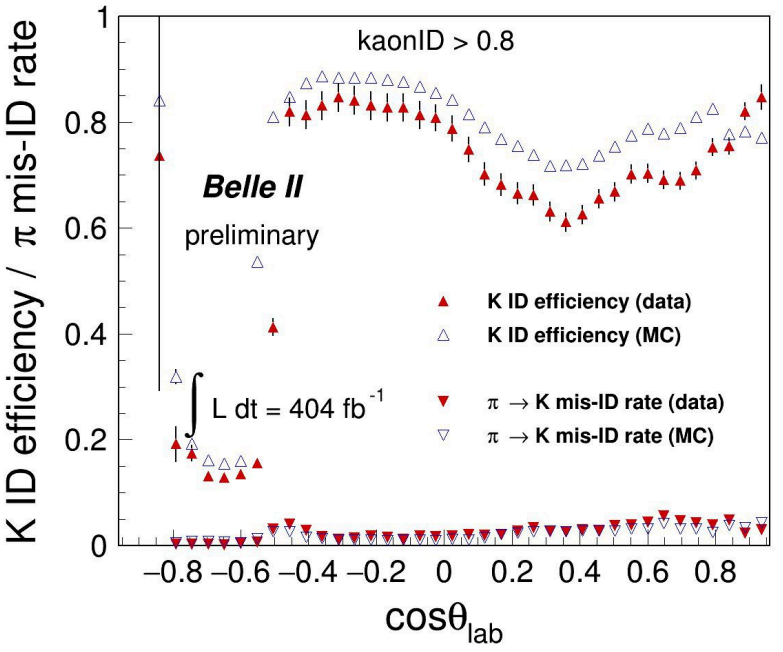


**Figure 5.7:** K signal efficiency and $\pi$ misidentification rates as a function of cos$\theta$ on collision $D^*$ decay MC and data.

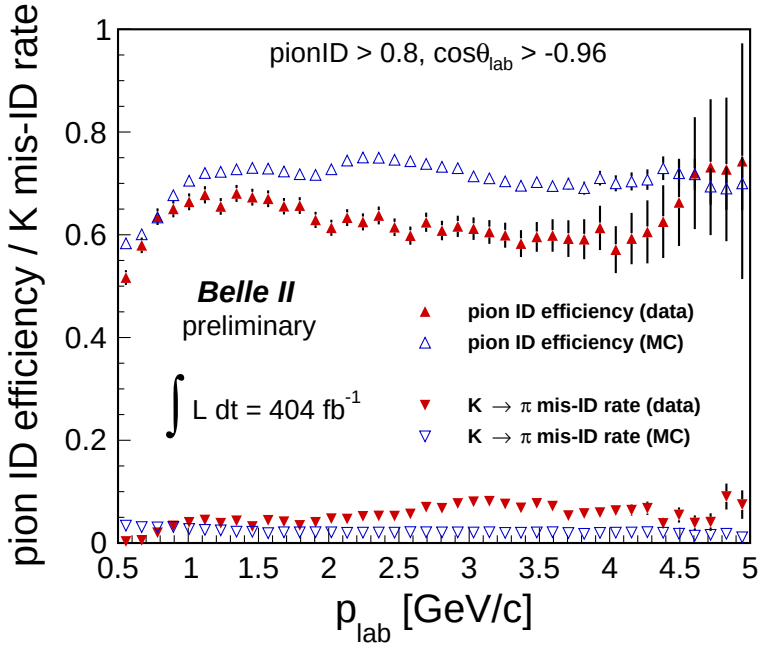


**Figure 5.8:** Pion signal efficiency and K misidentification rates as a function of p on collision $D^*$ decay MC and data.

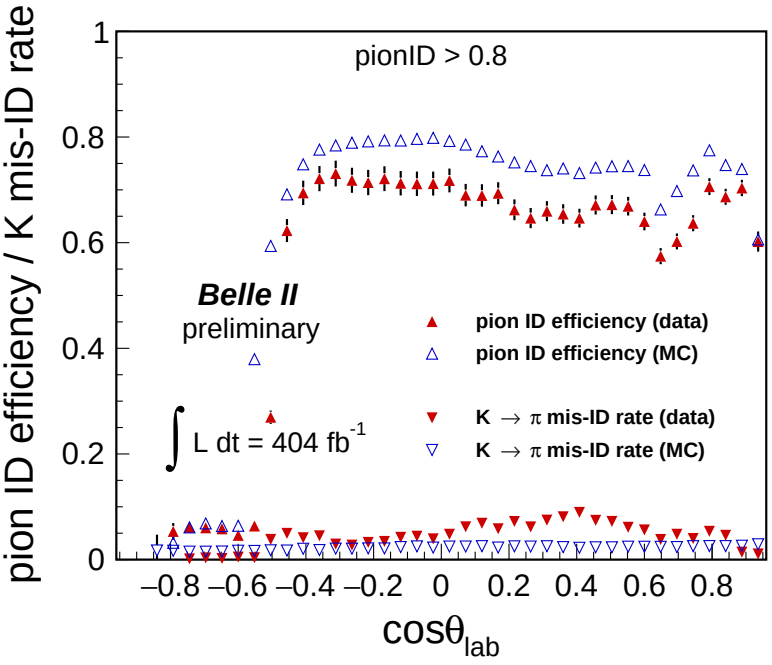


**Figure 5.9:** Pion signal efficiency and K misidentification rates as a function of cos$\theta$ on collision $D^*$ decay MC and data.

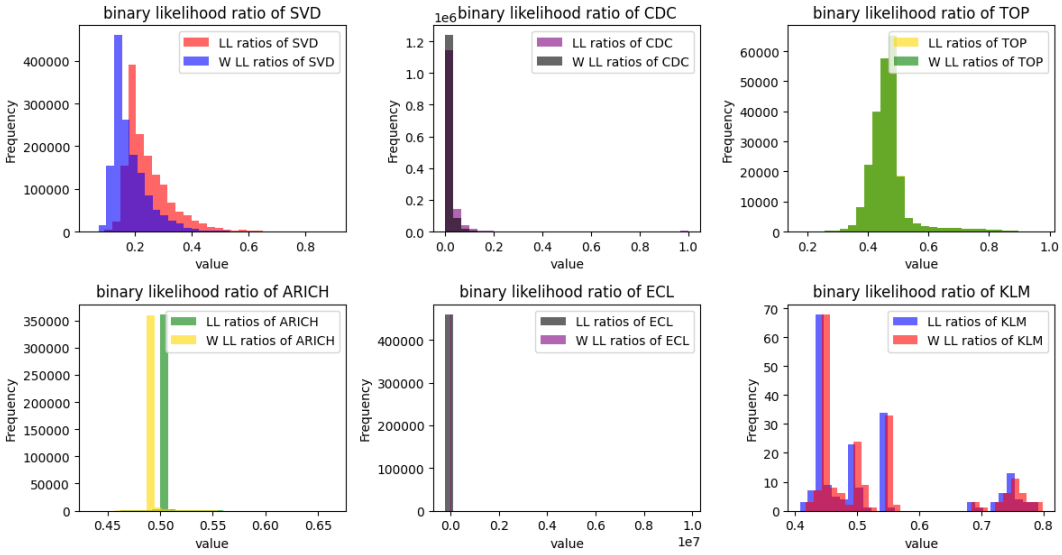


**Figure 5.10:** The $\pi/\mathrm{K}$ binary likelihood ratios of the detectors before and after optimization by Neural Network teq., plotted separately. In each histograms, binary likelihood ratios of pion of different detectors before and after optimization are demonstrated by different colors.

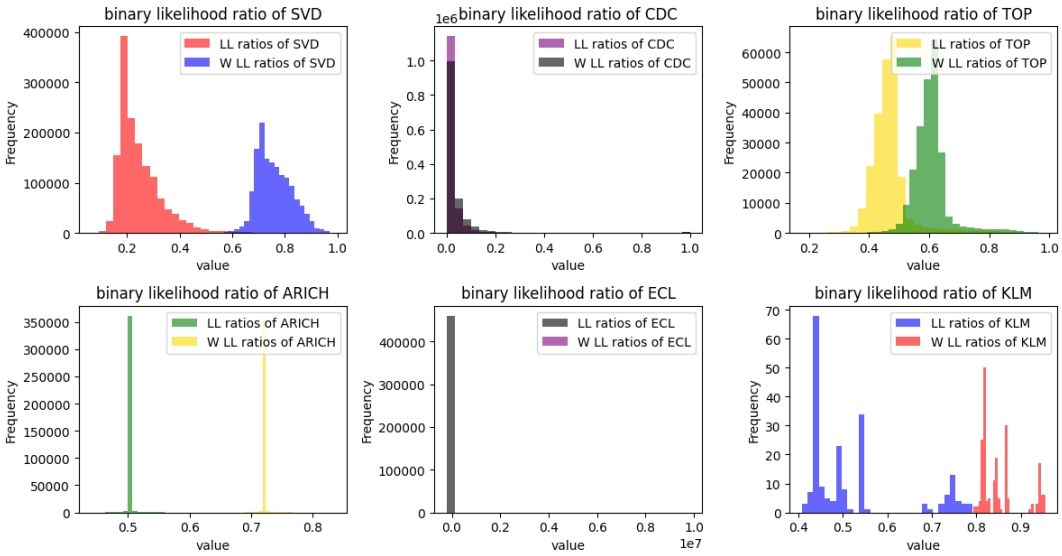


**Figure 5.11:** The $\pi/\mathrm{K}$ binary likelihood ratios of the detectors before and after upgrading weights by Random Forest model, plotted separately. In this plots, like the before one, the binary likelihood ratios of each detectors, before and after optimization are demonstrated by different colors

exhibit a higher peak and are shifted towards zero, indicating an improved identification of kaons. Similarly, in Fig 5.11, the likelihood ratios distribution is shifted towards one and the peak is reduced. For the CDC, both the neural network case in Fig 5.10 and the random forest case in Fig 5.11 show a slight change. In the neural network case, the peak of the likelihood ratios increases after applying the weights, while in the random forest case, it decreases slightly. In the case of the TOP, there is no change in the likelihood ratios before and after applying weights for the neural network case (Fig. 5.10). In fact, by looking at the Neural Network Weighted Matrix in Fig. 5.3 and the weights of TOP for pion and kaon hypotheses which are comparatively equal, it becomes clear why the histograms show no change before and after the application of weights. However, for the random forest case, the entire likelihood distribution is shifted to the right (one) without altering its shape. Regarding the ARICH, in Fig 5.10, the main lag of the likelihood ratios histogram is slightly shifted from approximately 0.51 to 0.48. In Fig 5.11, for the random forest case, the histogram shifts from around 0.51 to 0.72. The binary likelihood ratios for the ECL sub-detector remain unchanged in both Fig. 5.10 and Fig. 5.11, as their values are equal to zero. The most interesting binary likelihood ratios distribution is observed for the KLM. In Fig. 5.10, after applying the weights from the neural network, the KLM likelihood ratios histogram is slightly shifted towards zero, and the two histograms overlap. However, in Fig. 5.11 of the random forest case, the entire KLM likelihood ratios histogram is shifted to the right (one), and there are no shared points between the two histograms. The log likelihood differences of K and $\pi$ are visualised by histograms in Fig. 5.12 and Fig. 5.13.

In set of plots in Fig. 5.14, it is visible that the distribution of likelihood ratios after applying optimization ratios exhibits a lower peak compared to the original distribution, it indicates that the optimization has led to a decrease in the likelihood of the observed events. This could imply that the optimization has introduced some bias or that the chosen optimization ratios are not suitable for the dataset.
In the set of plots in Fig. 5.15, in contrast, the distribution of likelihood ratios after multiplication of weights come out from a random forest model reaches the same peak but shifts, which it means that the shape of the distribution remains similar, but the position or location of the peak changes. This shift in the peak indicates that the weights have had an impact on the distribution by redistributing the events. The shift can be attributed to the influence of the weights or optimization factors on the underlying data or MC, altering the relative contribution of different events or samples. Thesefore, the shift in the peak suggests a change in the relative importance or distribution of the events due to the application of weights.

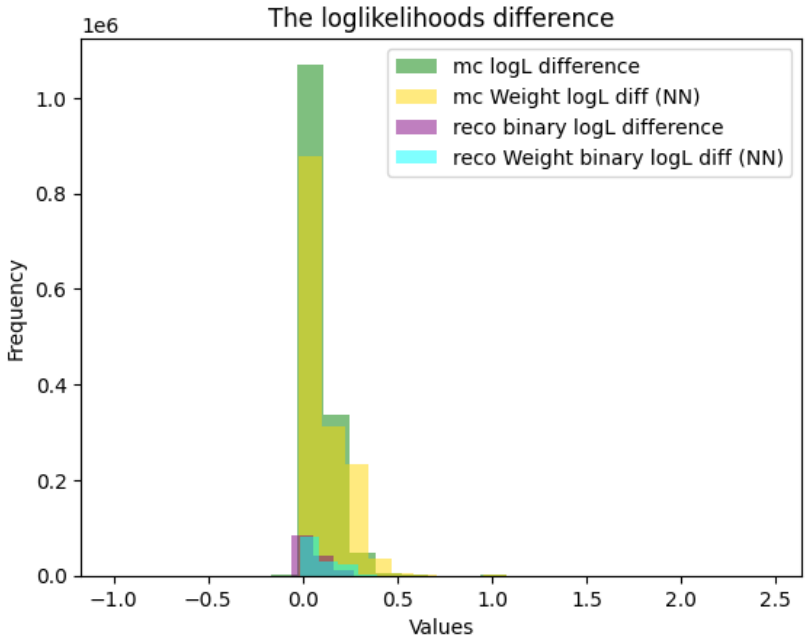


**Figure 5.12:** The log likelihood difference of K and $\pi$ for MC (green and gold) and data (purple and cyan) with and without updating weights by NN

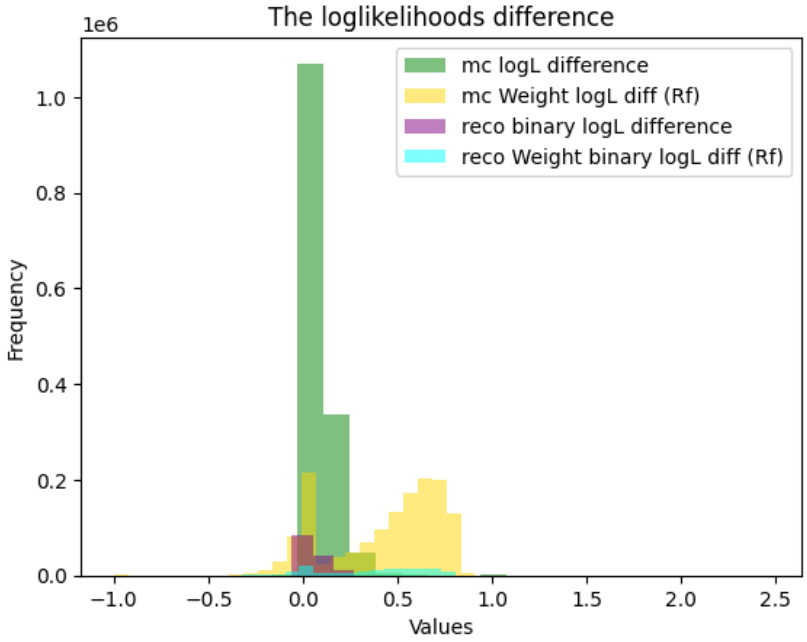


**Figure 5.13:** The log likelihood difference of K and $\pi$ for MC (green and gold) and data (purple and cyan) with and without updating weights by RF

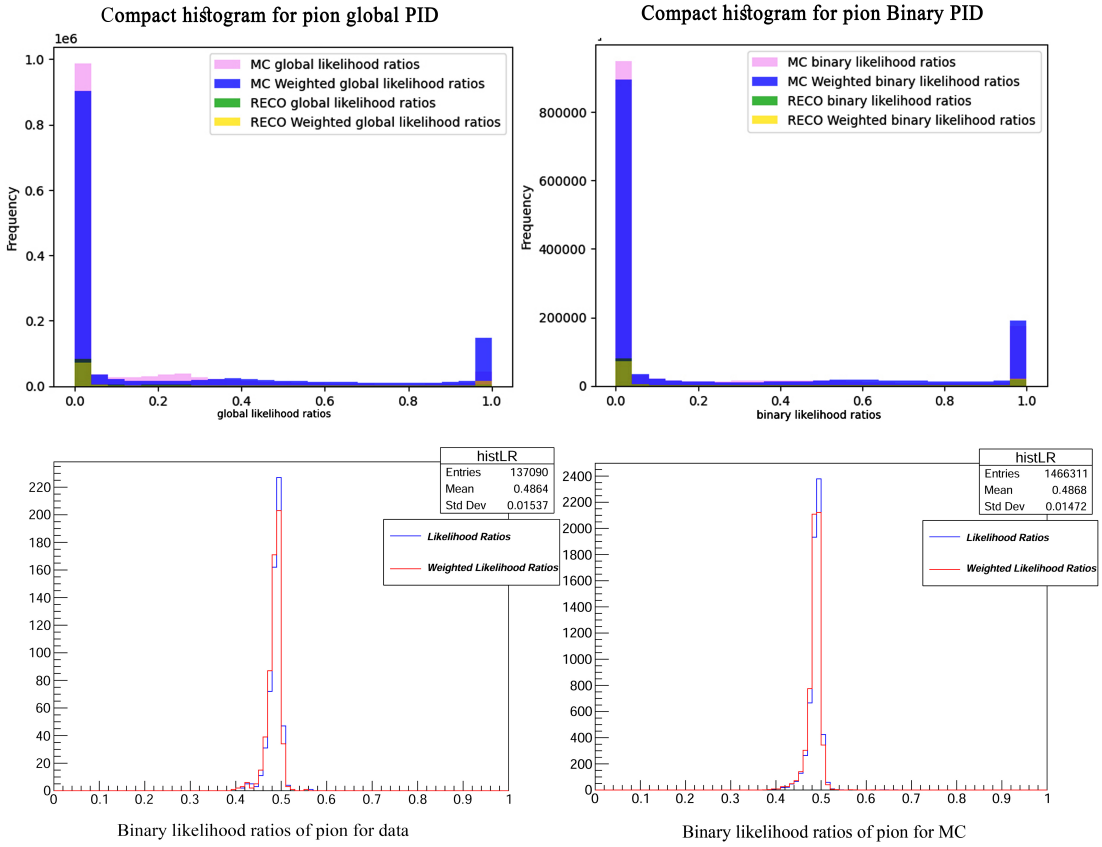


**Figure 5.14:** The comparission of likelihood ratios of pion, before and after applying weights by neural network model. The global likelihood ratios of $\pi$; demonstration of MC (pink), weighted MC (blue), data (green) and weighted data(yellow) [top left], binary likelihood ratios of $\pi$; demonstration for MC (pink), weighted MC performed (blue), data (green) and weighted data(yellow) [top right], the binary likelihood ratios of $\pi$ before (blue line) and after (red line) applying weights[below left], the binary likelihood ratios of $\pi$ before (blue line) and after (red line) applying weights [below right].

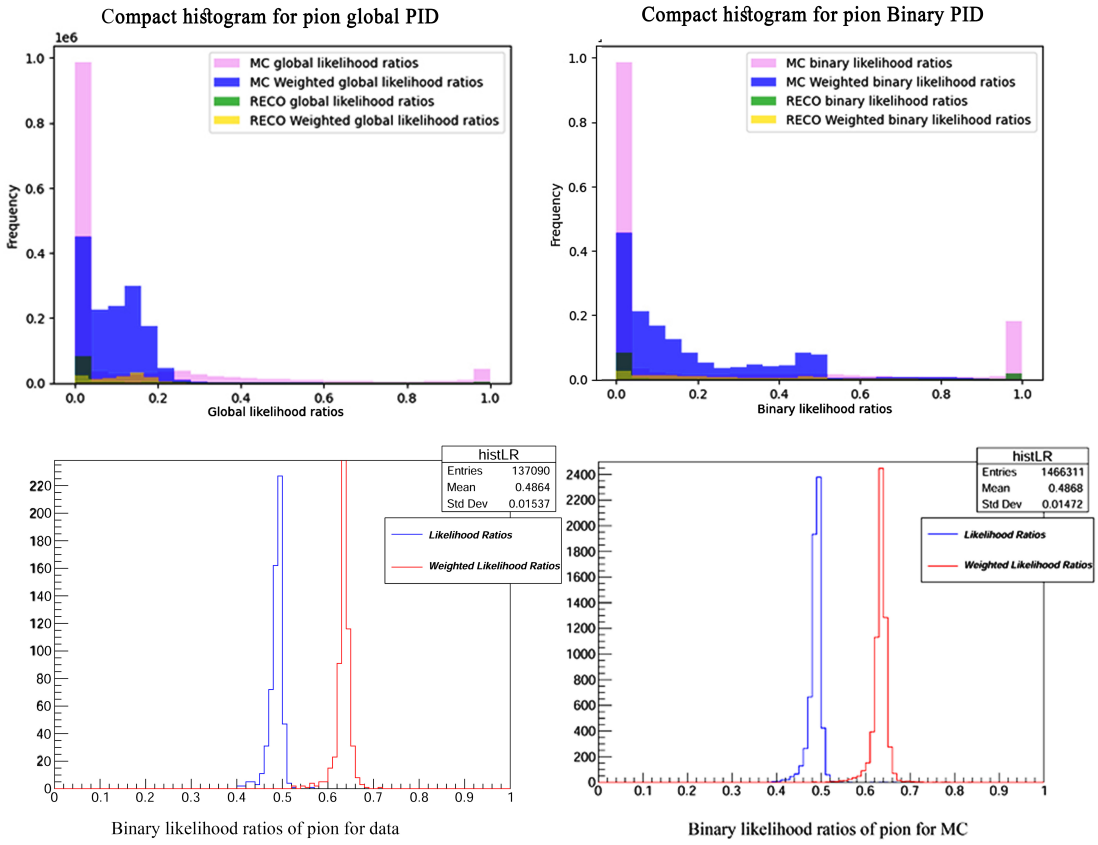


**Figure 5.15:** The comparission of likelihood ratios of pion, before and after applying weights by random forest model. The global likelihood ratios of $\pi$; demonstration of MC (pink), weighted MC (blue), data (green) and weighted data(yellow) [top left], binary likelihood ratios of $\pi$; demonstration for MC (pink), weighted MC performed (blue), data (green) and weighted data(yellow) [top right], the binary likelihood ratios of $\pi$ before (blue line) and after (red line) applying weights[below left], the binary likelihood ratios of $\pi$ before (blue line) and after (red line) applying weights [below right].

In conclusion, the neural network and random forest models both show promising results for pion and kaon PID. The random forest model is slightly more accurate in machine learning aspects, while the neural network model is still significant in its improvement of (charged) particle identification accuracy in PID study. These findings provide useful insights for the development of machine learning models in future particle physics research.

# References


[1] T. Keck, *Machine Learning Algorithms for the Belle II Experiment and Their Validation on Belle Data*. ETP-KA, 2017, no. ETP-KA/2017-31.

[2] B. I. Collaborator, "The belle ii physics book," *Progress of Theoretical and Experimental Physics*, vol. 2019, p. 123C01, 2019.

[3] Y. Sato, "Introduction to the analysis package," in *Belle II Starter Kit Workshop*. KEK, January 2020.

[4] J. Kemmer and G. Lutz, "New detector concepts," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 253, no. 3, pp. 365–377, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0168900287905183

[5] Z. Dolezal and S. Uno, "Belle ii technical design report," KEK, Tech. Rep. arXiv:1011.0352 [hep-ex], 2010.

[6] M. Milesi *et al.*, "Muon and electron identification performance with 189 fb1*ofbelleiidata*," *BELLE2-NOTE-TE-2021-011*, 2022, *internalDraftversion*6.1, *October*19.

[7] Belle II Collaboration. (2021) Belle ii software. [Online]. Available: https://software.belle2.org/development/sphinx/

[8] B. I. S. Group. (n.d.) Simulation. [Online]. Available: https://software.belle2.org/development/sphinx/online_book/fundamentals/03-simulation.html

[9] S. Sandilya, "Kaon and Pion Identification Performances in Phase III data," Belle II Collaboration, Belle II Public Notes 022. [Online]. Available: https://belle2.cc/downloads/publications/publicnotes/2019/BELLE2-NOTE-PL-2019-022.pdf

[10] C. Hainje, A. Albert, C. Hadjivasiliou, and J. Strube, "A comprehensive study of belle ii particle identification performance, efficiencies, and detector effects," *BELLE2-NOTE-TE-2021-022*, 2022, dRAFT Version 5.

[11] Y. Glaser, "Physics-informed neural networks for belle ii kaon versus pion particle identification," Master of Science thesis, University of Hawai'i at Manoa, August 2020.

[12] C. H. et al., "Machine learning methods to evaluate and optimize the charged particle identification in belle ii," Belle II Collaboration, Belle II Note TE-2021-027, January 2022.

**Appendix:**

## Random Forest Algorithm python script

```
\#!/usr/bin/env python3

##############################################################################
##############################################################################
# This script tune RF Moldel for Charged Particles and extract Weight Matrix #
# to Correct the loglikelihoods of Belle II Detectors                        #
#                                                                            #
#  Writtien by Alì Bavarchee                                                 #
#         March 2023                                                         #
##############################################################################

import basf2
import numpy as np
import pandas as pd
from argparse import ArgumentParser
import h5py
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from os import makedirs
from os.path import join, dirname
from tqdm.auto import tqdm
import itertools
from mpl_toolkits.mplot3d import Axes3D
from scipy import linalg as la
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import log_loss
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ParameterGrid
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
import seaborn as sns
import glob
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from os import makedirs
from os.path import join, dirname
import time
import pickle
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.datasets import make_regression

######
data_folder = 'data/slim_dstar'
######

# To perforem a GridSearch over RF_Model, defining Hyper Params
Hparams = { 'bootstrap': [True],
           'max_features': ['auto'],
           'n_estimators': [2, 6, 20],
           'max_depth' : [2, 4, 10, 50],'min_samples_leaf': [2, 6, 10],
           'min_samples_split': [2, 6, 10]}
######.
```

```
# _make_const_lists gives some cte values ; Particles and their corrispond PDG_code
# and detectors
def _make_const_lists():
    """Moving this code into a function to avoid a top-level ROOT import."""
    import ROOT.Belle2

    PARTICLES, PDG_CODES = [], []
    for i in range(len(ROOT.Belle2.Const.chargedStableSet)):
        particle = ROOT.Belle2.Const.chargedStableSet.at(i)
        name = (particle.repr()[7:-1]
                .replace("-", "")
                .replace("+", "")
                .replace("euteron", ""))
        PARTICLES.append(name)
        PDG_CODES.append(particle.getPDGCode())
    # PARTICLES = ["e", "mu", "pi", "K", "p", "d"]
    # PDG_CODES = [11, 13, 211, 321, 2212, 1000010020]

    DETECTORS = []
    for det in ROOT.Belle2.Const.PIDDetectors.set():
        DETECTORS.append(ROOT.Belle2.Const.parseDetectors(det))
    # DETECTORS = ["SVD", "CDC", "TOP", "ARICH", "ECL", "KLM"]

    return PARTICLES, PDG_CODES, DETECTORS

#This is a common pytorch data loader which loads data and splits them to
#train and test(val) :

def load_training_data(directory, p_lims=None, theta_lims=None, device=None):
    """Loads training and validation data within the given momentum and theta
    limits (if given).

    Args:
```

```
        directory (str): Directory containing the train and validation sets.
        p_lims (tuple(float), optional): Minimum and maximum momentum. Defaults
            to None.
        theta_lims (tuple(float), optional): Minimum and maximum theta in
            degrees. Defaults to None.
        device (torch.device, optional): Device to move the data onto. Defaults
            to None.

    Returns:
        torch.Tensor: Training log-likelihood data.
        torch.Tensor: Training labels.
        torch.Tensor: Validation log-likelihood data.
        torch.Tensor: Validation labels.
    """
    p_lo, p_hi = p_lims if p_lims is not None else (-np.inf, +np.inf)
    t_lo, t_hi = theta_lims if theta_lims is not None else (-np.inf, +np.inf)
    t_lo, t_hi = np.radians(t_lo), np.radians(t_hi)

    def _load(filename):
        data = np.load(filename)
        X, y, p, t = data["X"], data["y"], data["p"], data["theta"]
        mask = np.logical_and.reduce([p >= p_lo, p <= p_hi, t >= t_lo, t <= t_hi])
        X = torch.tensor(X[mask]).to(device=device, dtype=torch.float)
        y = torch.tensor(y[mask]).to(device=device, dtype=torch.long)
        return X, y

    X_tr, y_tr = _load(join(directory, "train.npz"))
    X_va, y_va = _load(join(directory, "val.npz"))
    return X_tr, y_tr, X_va, y_va


#data_folder = 'data1/slim_dstar'
#df = load_training_data(data_folder)
```

```
X_tr, y_tr, X_va, y_va = load_training_data(data_folder)

#mask pdg code of pion and kaon (target sets) to 1 and -1

Y_tr = torch.where(y_tr==2,torch.tensor(1),y_tr)
Y_tr = torch.where(Y_tr==3,torch.tensor(-1),Y_tr)

Y_va = torch.where(y_va==2,torch.tensor(1),y_va)
Y_va = torch.where(Y_va==3,torch.tensor(-1),Y_va).

class RF_NET:
    def __init__(self, **params):
        self.rf = RandomForestRegressor(**params)

    def train(self, X_train, y_train):
        self.rf.fit(X_train, y_train)

    def predict(self, X_test):
        return self.rf.predict(X_test)

    def score(self, X_test, y_test):
        y_pred = self.predict(X_test)
        mse = mean_squared_error(y_test, y_pred)
        return mse

    def get_weights(self, to_numpy=True):
        """Returns the feature importances as a six-by-six array or tensor.

        Args:
            to_numpy (bool, optional): Whether to return the weights as a numpy
                array (True) or torch tensor (False). Defaults to True.

        Returns:
            np.array or torch.Tensor: The six-by-six matrix containing the
```

```
                feature importances.
        """
        feature_importances = self.rf.feature_importances_
        weights = np.zeros((6, 6))
        idx = 0
        for i in range(6):
            for j in range(i, 6):
                weights[i][j] = feature_importances[idx]
                weights[j][i] = feature_importances[idx]
                idx += 1
        if to_numpy:
            return weights
        else:
            return torch.tensor(weights)

#Hparams = {'bootstrap': [True],
#           'max_features': ['auto'],
#           'n_estimators': [2, 10],
#           'max_depth': [2, 4, 10, 50],
#           'min_samples_leaf': [2, 6, 10],
#           'min_samples_split': [2, 6, 10]}

class ROC_Analysis():

    def __init__(self, name, true_labels, predictions,
                 fpr_th=[0.01, 0.02, 0.05, 0.1, 0.2], model=None):

        self.name = name
        self.true_labels = true_labels
        self.predictions = predictions
        self.fpr_th = fpr_th
        self.fpr = None
        self.tpr = None
        self.thresholds = None
```

```
        self.AUC_score = None
        self.model = model.
\end{lstlisting}
\begin{lstlisting}[language=Python]
def Compute_ROC_Curve(self):

    self.fpr, self.tpr, self.thresholds = roc_curve(self.true_labels,
              self.predictions)
    self.AUC_score = roc_auc_score(self.true_labels, self.predictions)


def Find_Closest_Parameters(self):

    s = len(self.fpr_th)
    closest_fpr, closest_tpr, indexes = np.zeros(s),
    np.zeros(s), np.zeros(s, dtype=int)

    for ii in range(s):
        indexes[ii] = np.searchsorted(self.fpr, self.fpr_th[ii])

    closest_fpr = self.fpr[indexes]
    closest_tpr = self.tpr[indexes]
    closest_th  = self.thresholds[indexes]

    return (closest_fpr, closest_tpr, closest_th)


def Plot_ROC_Curve(self, axes=None):

    if self.AUC_score == None: self.Compute_ROC_Curve()

    if axes == None: fig, axes = plt.subplots(figsize=(7,6))
    axes.plot(self.fpr, self.tpr, color="blue", label=self.name+"_ROC")
    axes.plot([0, 1], [0, 1], color="darkred", linestyle='--', label="Random Guess")
```

```
    cl_fpr, cl_tpr, cl_th = self.Find_Closest_Parameters()
    axes.scatter(cl_fpr, cl_tpr, color="violet", label="Relevant pts,
    with thr. values")
    for ii in range(len(self.fpr_th)):
        plt.annotate('%.3f' % cl_th[ii], (cl_fpr[ii], cl_tpr[ii]),
                     xytext=(cl_fpr[ii]+0.03,cl_tpr[ii]-0.02),fontsize=11)

    txt_roc_auc = "roc_auc = %.3f" % self.AUC_score
    props = dict(boxstyle="round", facecolor="gold", alpha=0.5)
    axes.text(0.6, 0.4, txt_roc_auc, transform=axes.transAxes,
    fontsize=12, verticalalignment='top', bbox=props)

    axes.set_xlabel("False Positive")
    axes.set_ylabel("True Positive")
    axes.set_title("ROC Curve of the " + self.name)
    axes.grid()
    axes.legend()
    plt.savefig('ROC_Curve.png')


def Display_Results(self):

    if self.AUC_score == None: self.Compute_ROC_Curve()
    cl_fpr, cl_tpr, cl_th = self.Find_Closest_Parameters()

    inf_df = pd.DataFrame(np.vstack((np.array(self.fpr_th),
    cl_fpr, cl_tpr, cl_th)).round(4))
    inf_df.index = ['values', 'False Positive','True Positive','threshold']
    inf_df.columns = ['']*5

    return inf_df

# Define The best RF model function to select the best one
```

```
def RF_Best_Model(analysis_objects, verbose=False):

    fig, ax = plt.subplots(2, 1, figsize=(15,22 ))

    # Setup comparison plot
    ax[0].plot([0, 1], [0, 1], color="green", linestyle='-.', label="Random Guess")
    ax[0].set_xlabel("False Positive")
    ax[0].set_ylabel("True Positive")
    ax[0].set_title("ROC Curves")

    best_model = None
    max_auc = 0
    auc_values = []

    for key in analysis_objects:
        analysis_objects[key].Compute_ROC_Curve()
        ax[0].plot(analysis_objects[key].fpr, analysis_objects[key].tpr)
        roc_auc = analysis_objects[key].AUC_score
        auc_values.append(roc_auc)
        if roc_auc > max_auc:
            best_model = analysis_objects[key]
            max_auc = roc_auc

        if verbose:
            print(key, "AUC ROC = ", roc_auc)
    \begin{lstlisting}[language=Python]
# Setup best model plot
best_model.Plot_ROC_Curve(axes=ax[1])

hist_ax = fig.add_axes([0.22, 0.23, 0.1, 0.13])
hist_ax.hist(auc_values, bins=len(analysis_objects))
hist_ax.set_yticks([])

return best_model
```

```
# Define a GS Function and then initialize search
def GS_4_BRF():
    bulding_forest_start=time.time()
    best_RF = {}
    #best_RF_ROC = {}
    counter = 1
    for params in tqdm(list(ParameterGrid(Hparams))):
        rff = RF_NET(**params)
        rff.train(X_tr, Y_tr)
        predii = rff.predict(X_va)
        scoreee = rff.score(X_va, Y_va)
        mse = mean_squared_error(Y_va, predii)
        #poisson = poisson(Y_test, predii)
        #evaluate_model(rff, df, pdg_codes=PDG_CODES)
        #print("|||N|Z|O||||||")
        #print('hparams:=>', params)
        #print("==================================>>>>>")
        #print("~~~~~~~~~~")
        #print('accuracy:=>', scoreee)
        print("~~~~~~~~~~")
        #print('mse===>', mse)
        #print("<<<<<==================================")
        print('----------------------------------------')
        #best_RF['RF_'+str(counter)] = evaluate_model(model = rff, df = df,
        #pdg_codes=PDG_CODES)
        best_RF['RF_'+str(counter)] = ROC_Analysis(name='RF_'+str(counter),
        true_labels=Y_va, predictions=predii, model=rff)
        print('----------------FIN----------------')
        counter+=1
    bulding_forest_end=time.time()

    return print('wall time=>', bulding_forest_end - bulding_forest_start)
```

```
#GS_4_BRF()

bulding_forest_start=time.time()
#best_RF = {}
best_RF_ROC = {}
counter = 1
for params in tqdm(list(ParameterGrid(Hparams))):
    rff = RF_NET(**params)
    rff.train(X_tr, Y_tr)
    predii = rff.predict(X_va)
    SQORE = rff.score(X_tr, Y_tr)
    mse = mean_squared_error(Y_va, predii)
    #poisson = poisson(Y_test, predii)
    #evaluate_model(rff, df, pdg_codes=PDG_CODES)
    print("|||N|Z|O||||||")
    #print('hparams:=>', params)
    #print("==================================>>>>>")
    #print("~~~~~~~~~~")
    print('SCORE:=>', SQORE)
    print("~~~~~~~~~~")
    print('mse===>', mse)
    #print("<<<<<==================================")
    print('----------------------------------------')
    #best_RF['RF_'+str(counter)] = evaluate_model(model = rff, df = df,
    #pdg_codes=PDG_CODES)
    best_RF_ROC['RF_'+str(counter)] = ROC_Analysis(name='RF_'+str(counter),
                        true_labels=Y_va, predictions=predii, model=rff)
    print('----------------FIN----------------')
    counter+=1
bulding_forest_end=time.time()
print('wall time=>', bulding_forest_end - bulding_forest_start)

# Visualise the GS
best_randomforest = RF_Best_Model(best_RF_ROC, verbose=False)
```

```
#Finding the best RF model and RF_WMat
BestRF = best_randomforest.model
BestRF.train(X_tr, Y_tr)
prediii = BestRF.predict(X_va)
scoreee = BestRF.score(X_tr, Y_tr)
mseee = mean_squared_error(Y_va, prediii)
print('------SCORE:====>', scoreee)
print('mean squared error', mseee)

RF_WMat = BestRF.get_weights()
print('RF Weight Matirix:', RF_WMat)
np.savetxt('test.out', RF_WMat, delimiter=',').
```

# Acknowledgments

I would like to express my heartfelt gratitude and appreciation to the following individuals who have played a significant role in the completion of this thesis:

First and foremost, I am deeply indebted to Professor Alessandro Gaz, my esteemed supervisor, for his unwavering guidance, invaluable insights, and continuous support throughout this research endeavor. His expertise, patience, and dedication have been instrumental in shaping my ideas and refining my work. I am truly grateful for his mentorship and the knowledge I have gained under his guidance.

I would also like to extend my sincere thanks to Professor Marco Zanetti, the coordinator of this study program, for his valuable input, constructive feedback, and meticulous oversight. His commitment to academic excellence and his willingness to share his knowledge have greatly enriched my research journey.

Inoltre, sono immensamente grato ai miei due fratelli, Signori Giuseppe e Rosario Traina Tito, per il loro costante sostegno, incoraggiamento e fiducia nelle mie capacità. La loro motivazione costante e la loro presenza ispiratrice sono stati i pilastri di forza che mi hanno sostenuto durante i momenti difficili. I loro saggi consigli e l'assistenza instancabile sono stati di inestimabile valore per il completamento di questa tesi.

Furthermore, I extend my deepest gratitude to Hana Bavarchi, my beloved daughter, whose love, understanding, and unwavering patience have been a constant source of inspiration and motivation for me. Despite the demands of my studies, she has been a pillar of support, always offering encouragement and understanding. Her presence in my life has been a driving force behind my achievements, and I am truly blessed to have her by my side.

Finally, I would like to express my heartfelt appreciation to all the individuals who have contributed to this thesis in various ways, including my friends, colleagues, and the countless individuals who participated in my research. Your willingness to share your time, knowledge, and experiences has been crucial to the successful completion of this work.

To all those mentioned above, and to anyone else who has contributed to this journey in ways big or small, please accept my sincere gratitude for your unwavering support and encouragement. This thesis would not have been possible without your guidance, assistance, and belief in my abilities. Thank you from the bottom of my heart.