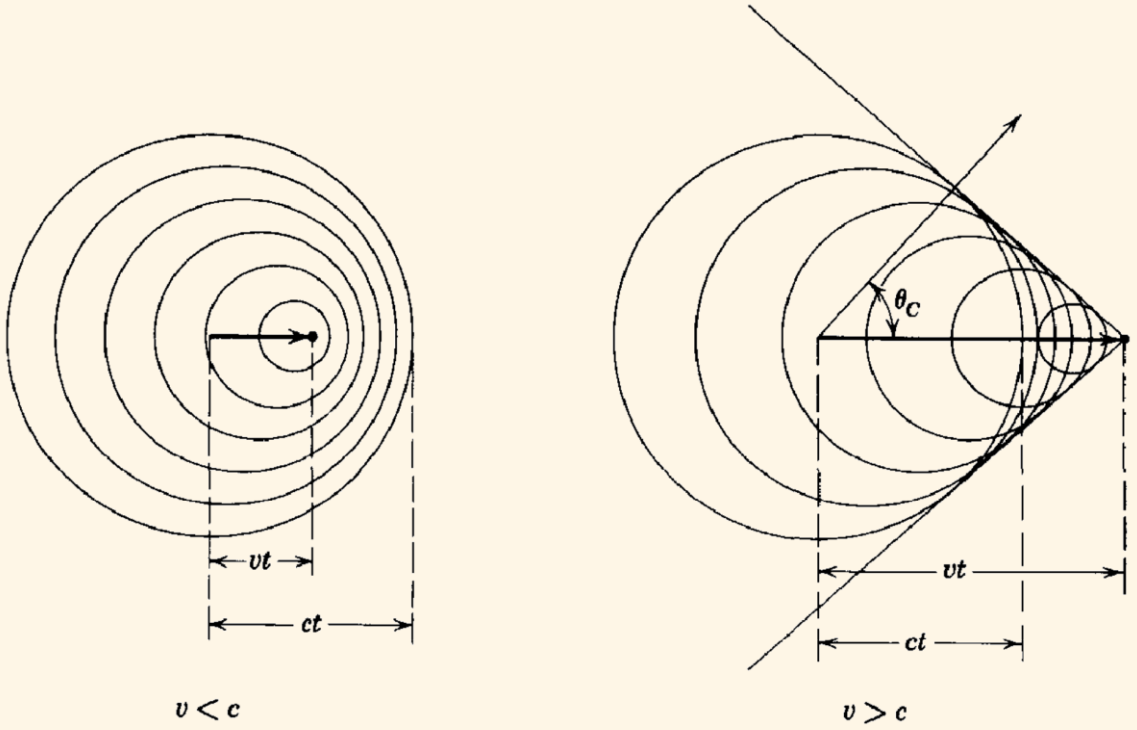
RICH Particle Identification with YOLO-based Pipeline

Martino Borsato **Maurizio Martinelli** **Giovanni Laganà**

University of Milano Bicocca and INFN

Cherenkov Radiation

Cherenkov radiation is produced when a charged particle passes through a medium with speed greater than the phase speed of light in such medium.



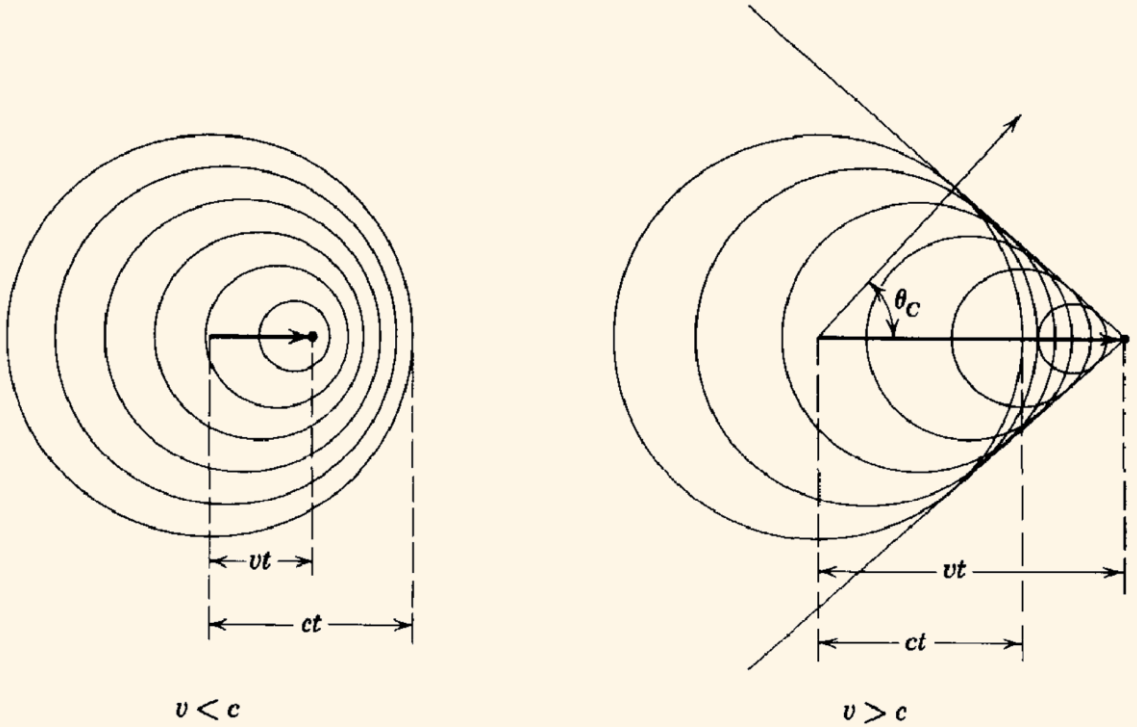
$$\cos\theta_c = \frac{1}{n\beta}$$

Cherenkov Radiation

Cherenkov radiation is produced when a charged particle passes through a medium with speed greater than the phase speed of light in such medium.

The Cherenkov angle depends on the particle velocity and the refractive index of the medium.

This phenomenon is used in particle detectors to identify charged particles by combining Cherenkov angle measurement with momentum measured by the trackers.



$v < c$ $v > c$

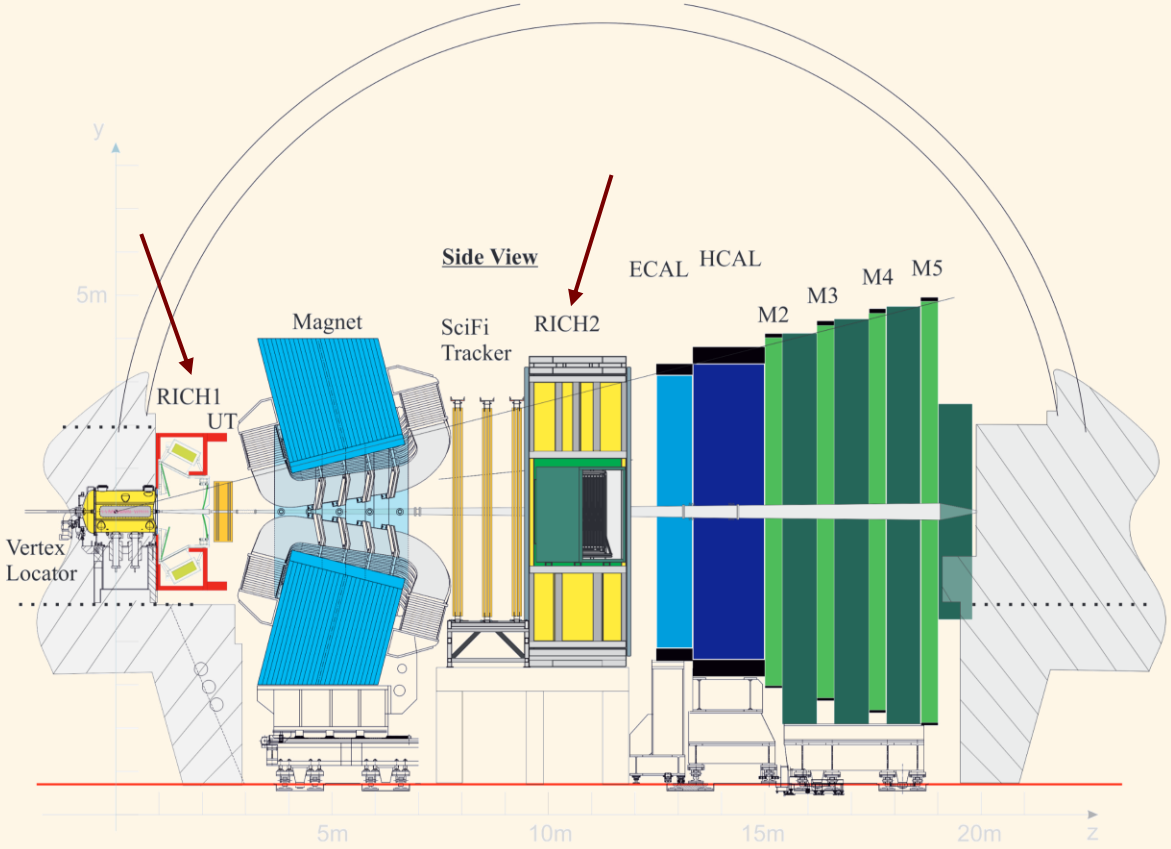
$$\cos\theta_c = \frac{1}{n\beta}$$

The RICH detectors at LHCb at CERN

Two RICH (Ring Imaging Cherenkov) detectors:

- RICH1, upstream of the magnet
- RICH2, downstream of the magnet

In this work, we focused on RICH1.



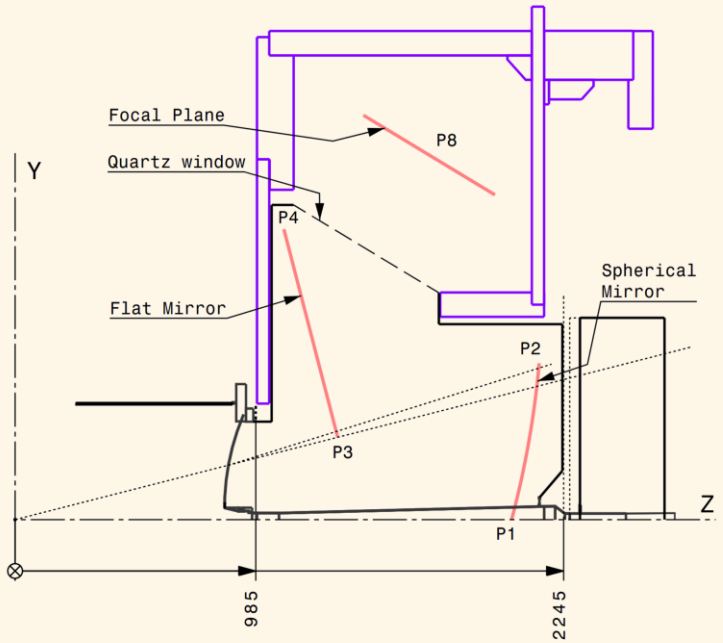
JINST 19 (2024) P05065

The RICH detectors at LHCb at CERN

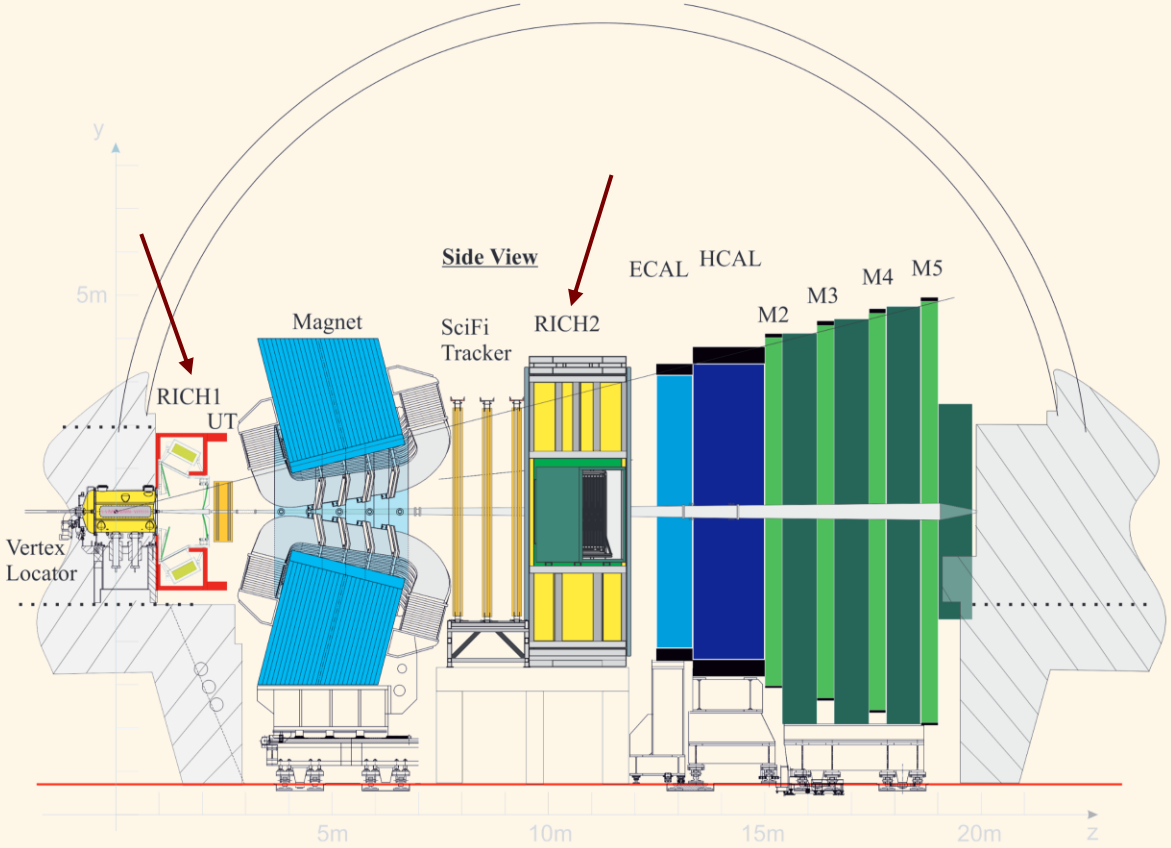
Two RICH (Ring Imaging Cherenkov) detectors:

- RICH1, upstream of the magnet
- RICH2, downstream of the magnet

In this work, we focused on RICH1.



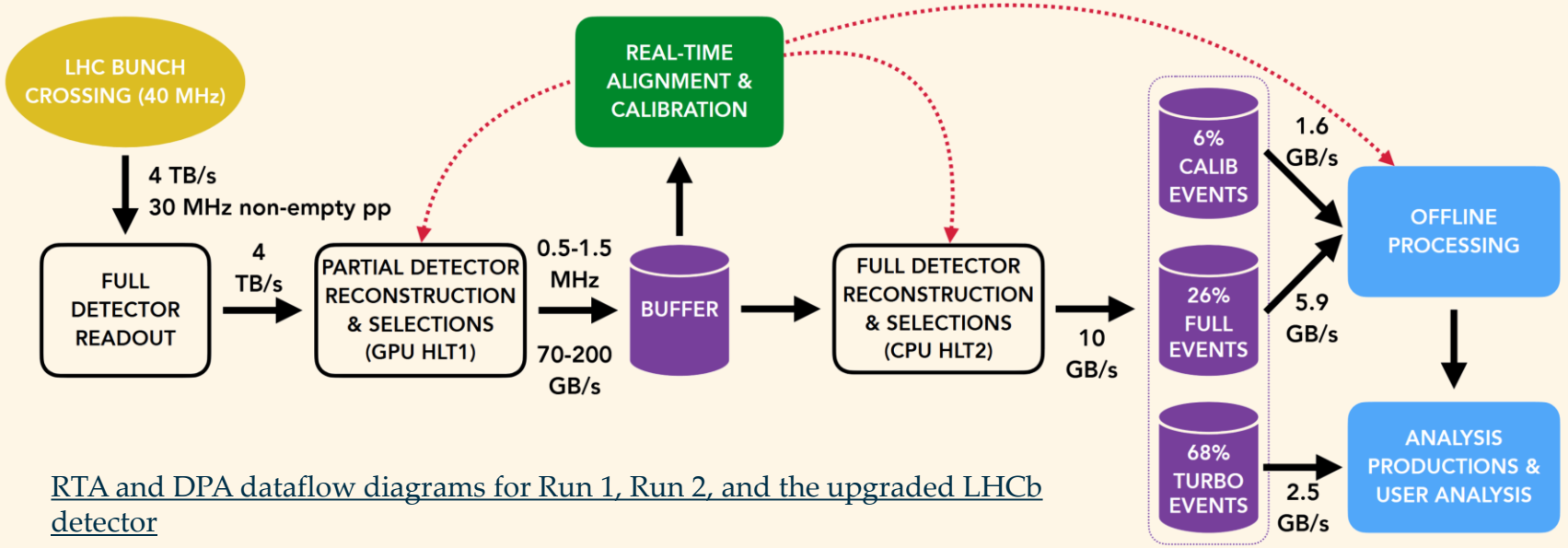
LHCb Collaboration. LHCb PID Upgrade Technical Design Report. Tech. rep. 2013.



JINST 19 (2024) P05065

The Trigger System

The trigger system is crucial for **reducing the data volume** from 4 TB/s, generated at the nominal instantaneous luminosity, to approximately 10 GB/s, which can be stored for offline analysis.

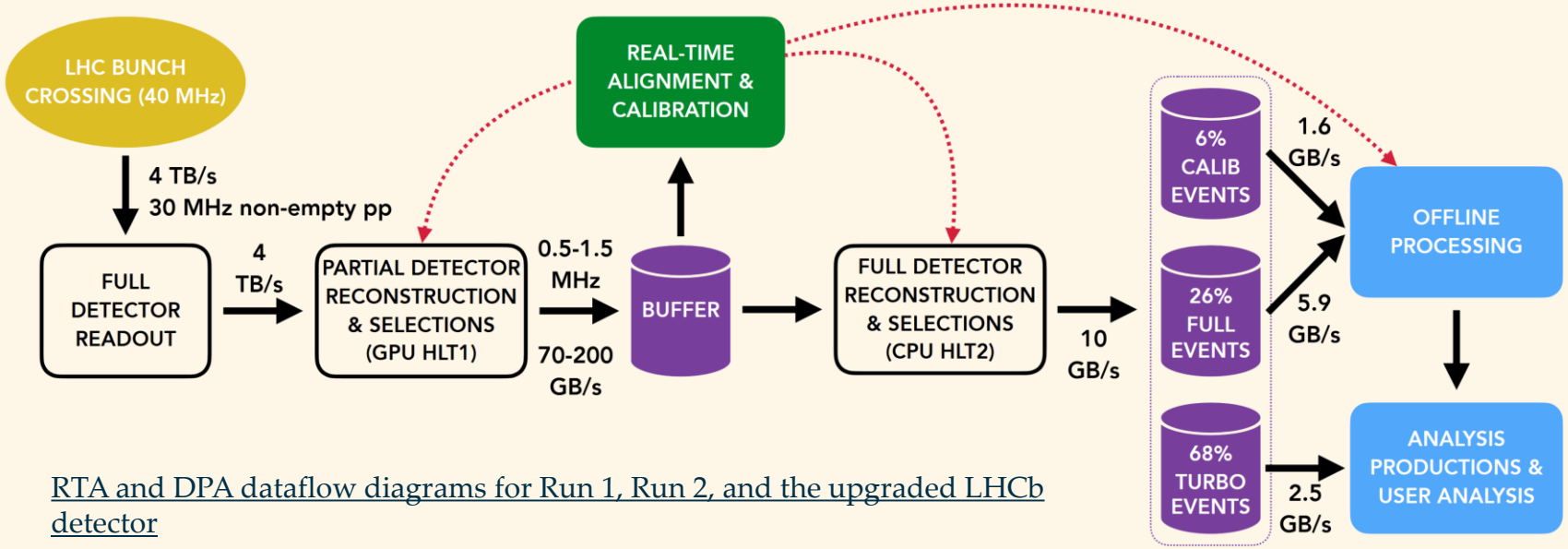


RTA and DPA dataflow diagrams for Run 1, Run 2, and the upgraded LHCb detector

The Trigger System

The trigger system is crucial for **reducing the data volume** from 4 TB/s, generated at the nominal instantaneous luminosity, to approximately 10 GB/s, which can be stored for offline analysis.

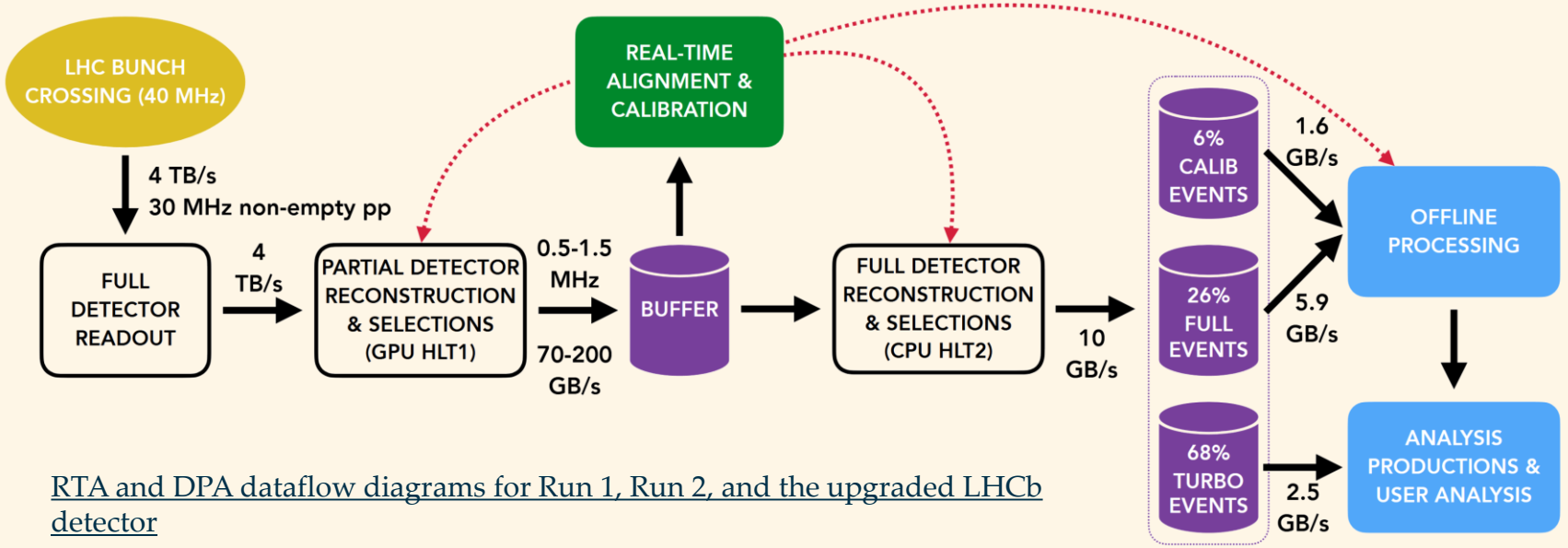
RICH PID is done in HLT2, online, at a reduced rate of 0.5-1.5 MHz (from the 30 MHz generated), using a likelihood-based approach.



RTA and DPA dataflow diagrams for Run 1, Run 2, and the upgraded LHCb detector

Why ML for RICH PID?

RICH PID is crucial for **reducing combinatorial backgrounds and improving event selection** but consumes **20%-40%** of the CPU budget, posing a significant computational cost.

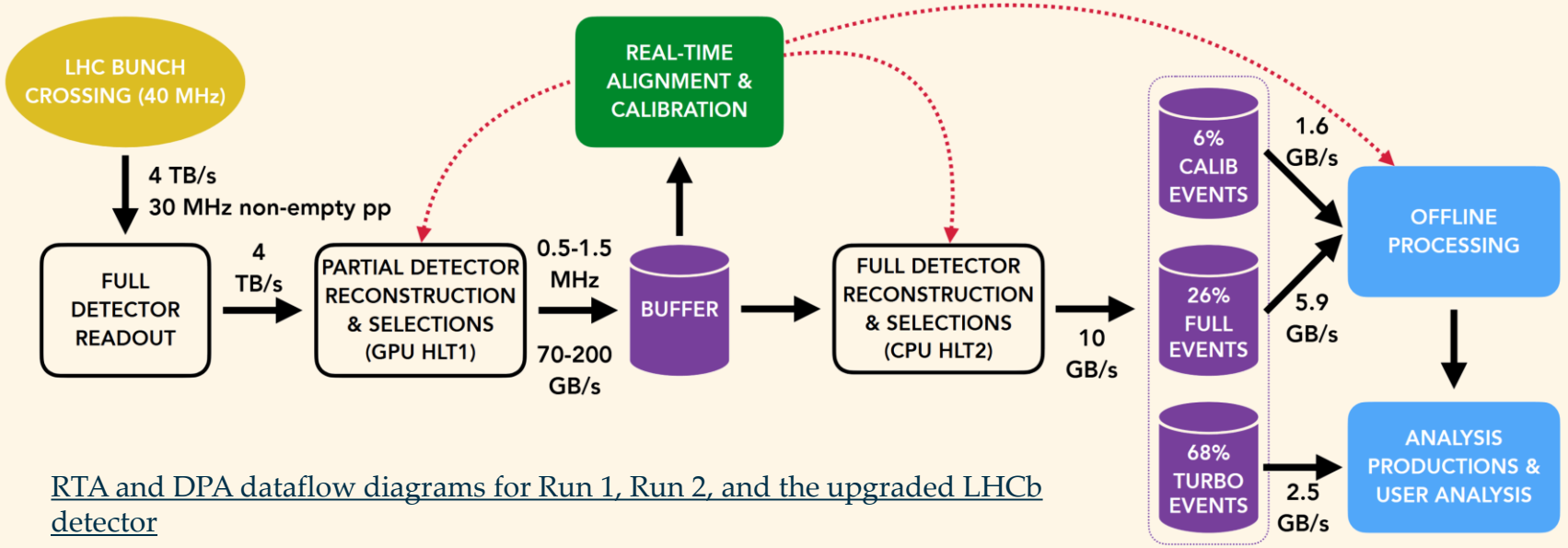


RTA and DPA dataflow diagrams for Run 1, Run 2, and the upgraded LHCb detector

Why ML for RICH PID?

RICH PID is crucial for **reducing combinatorial backgrounds and improving event selection** but consumes **20%-40%** of the CPU budget, posing a significant computational cost.

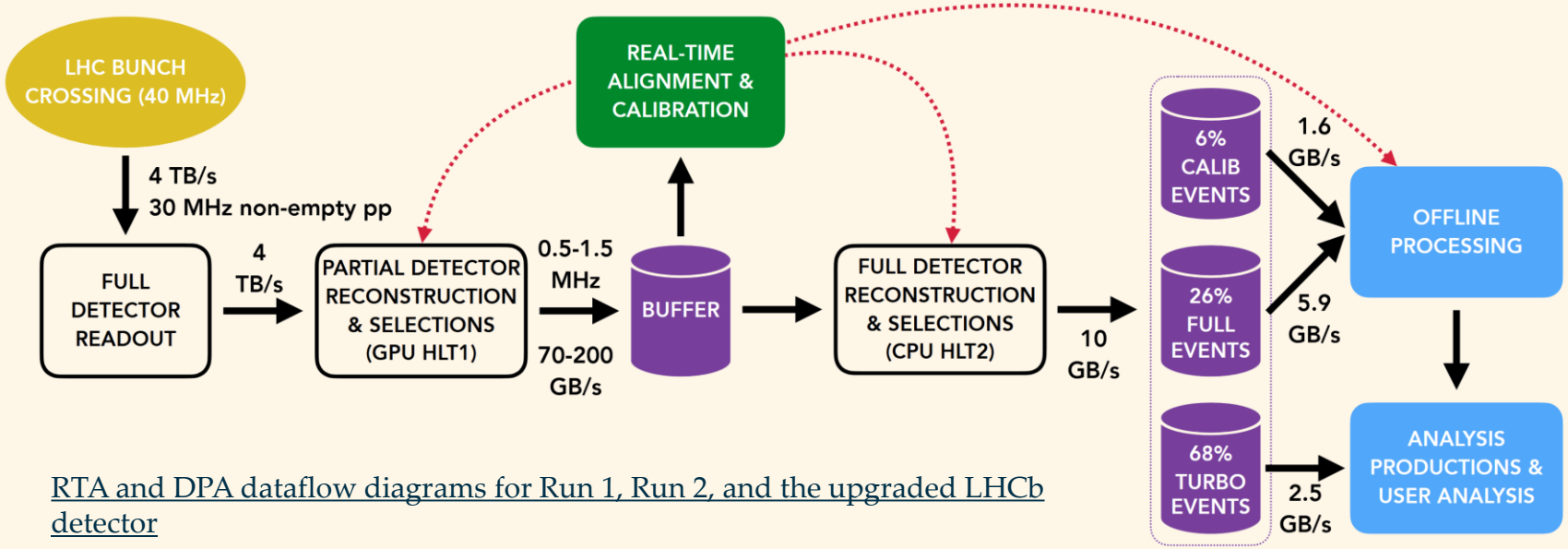
Machine learning for RICH PID can ease HLT2's CPU load through parallelization and potentially improve PID performance.



RTA and DPA dataflow diagrams for Run 1, Run 2, and the upgraded LHCb detector

Why ML for RICH PID?

CNN-based PID has already been tried, showing promising accuracy:
Blago, M. P., *Convolutional Neural Networks and Photonic Crystals for Particle ID*, Univ. of Cambridge, 2021.

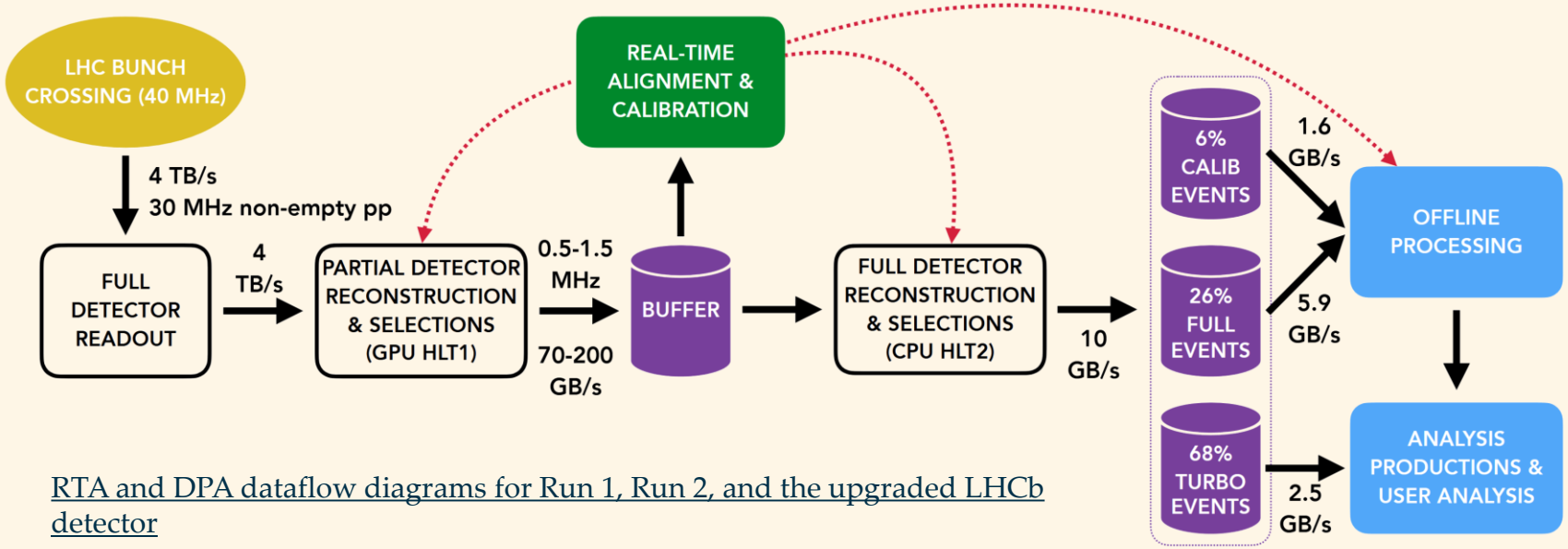


RTA and DPA dataflow diagrams for Run 1, Run 2, and the upgraded LHCb detector

Why ML for RICH PID?

CNN-based PID has already been tried, showing promising accuracy: Blago, M. P., *Convolutional Neural Networks and Photonic Crystals for Particle ID*, Univ. of Cambridge, 2021.

We wanted to try a more modern architecture to possibly gain accuracy and efficiency.

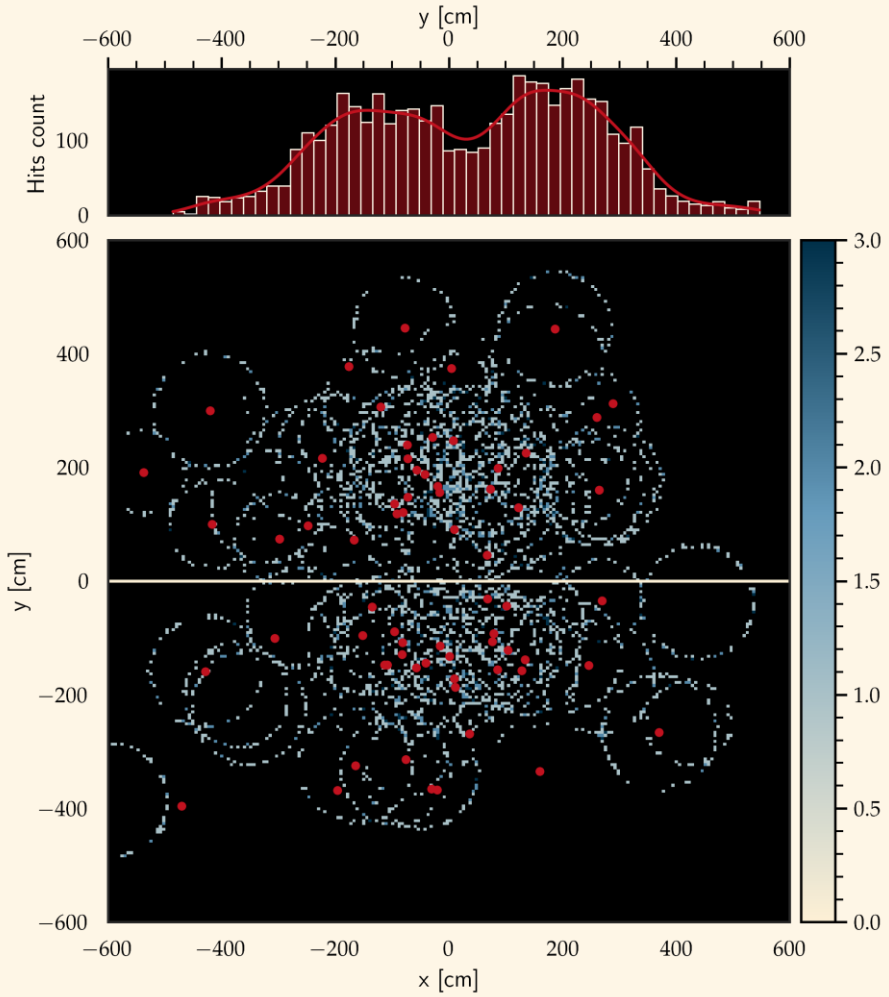


RTA and DPA dataflow diagrams for Run 1, Run 2, and the upgraded LHCb detector

The Dataset

Synthetic Cherenkov data Generator

Synthetic Cherenkov data Generator

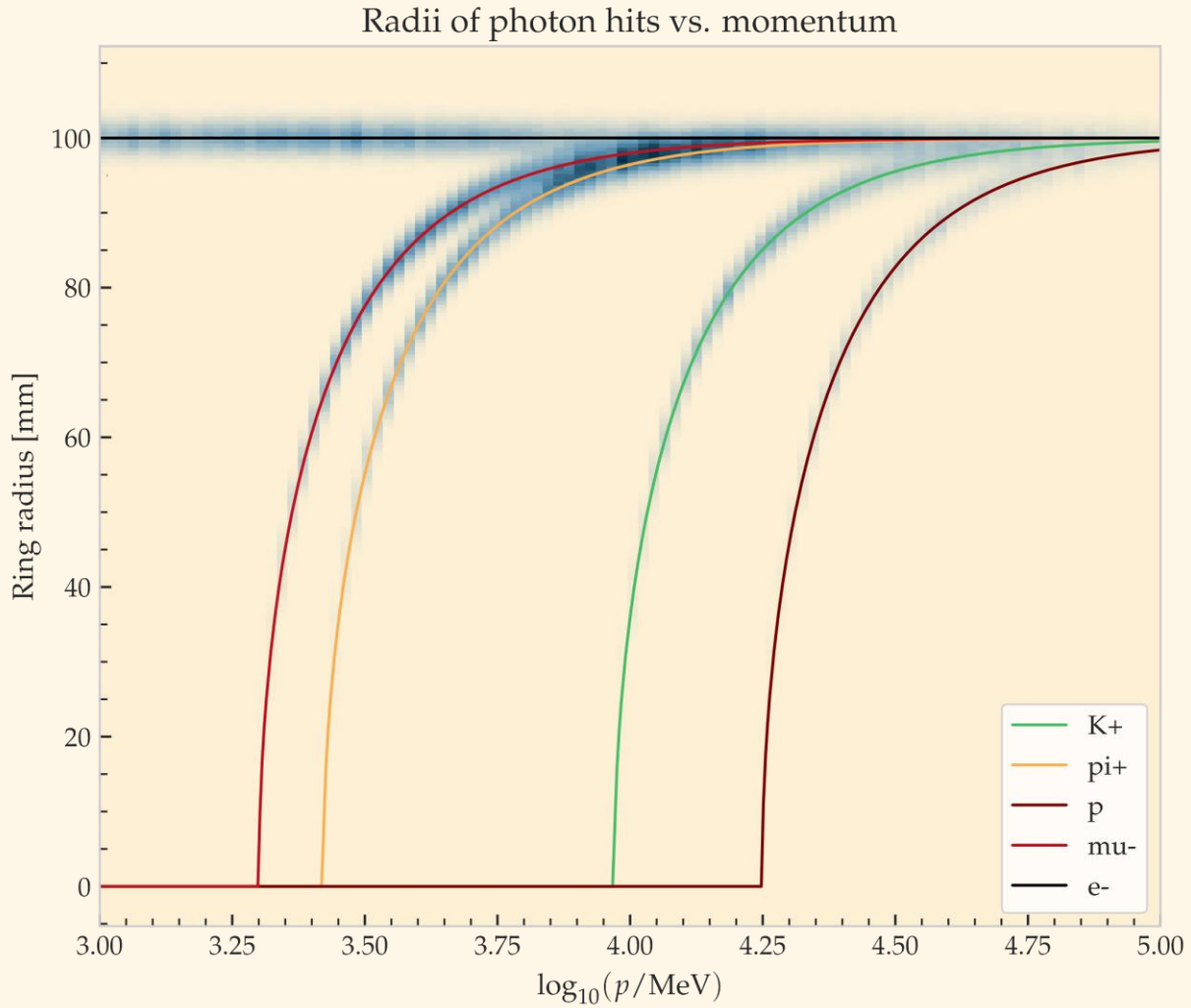


We developed a data generation framework that mimics RICH detector response.

Mean number of rings in RICH1 plane: 170
Mean number of tracked rings: 54

Giovanni Laganà, Maurizio Martinelli, Martino Borsato University of Milano Bicocca and INFN

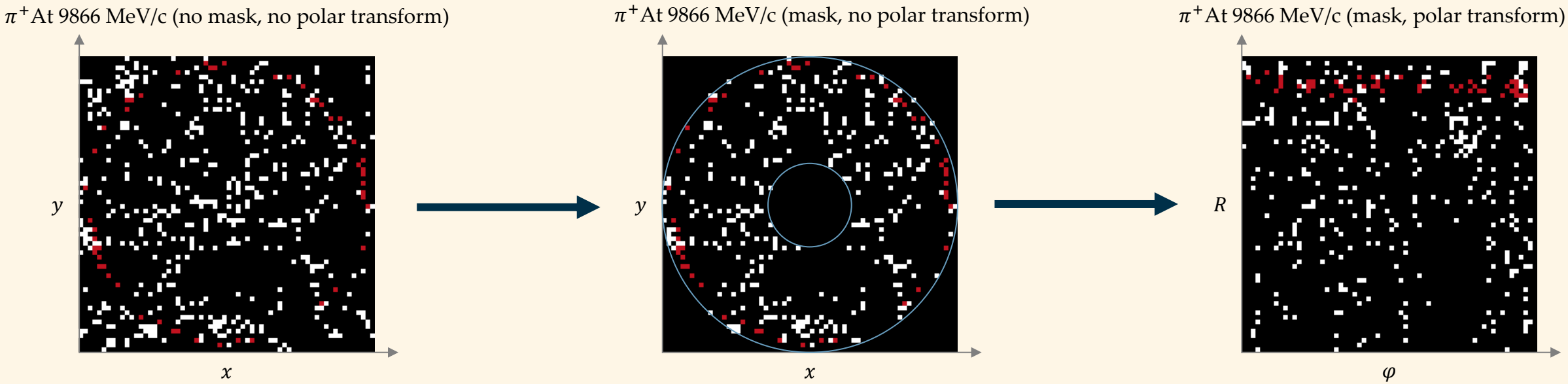
Synthetic Cherenkov data Generator



We developed a data generation framework that mimics RICH detector response.

The produced data has the most important features that the real data would have:

- Cherenkov angle dependence on momentum
- The number of photon hits is proportional to $\sin(\theta_c)^2$.
- Realistic Cherenkov angle resolution of $0.8\,mrad$
- Track distributions
- Relative abundance of particle types



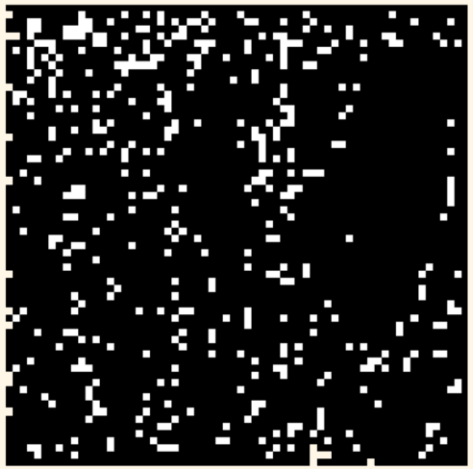
Signal is in red

YOLO images generation

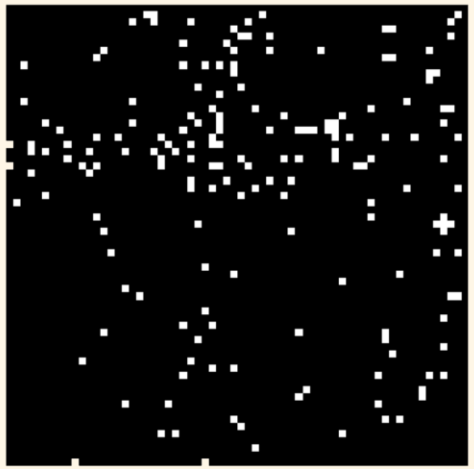
To train the YOLO models, images of the rings are required. We start from the generated event. Then we add rings in a specific momenta range, and for each of them we **cut out a squared image**.

Before rasterization, we apply a ring-shaped mask to limit background and transform the data points into polar coordinates.

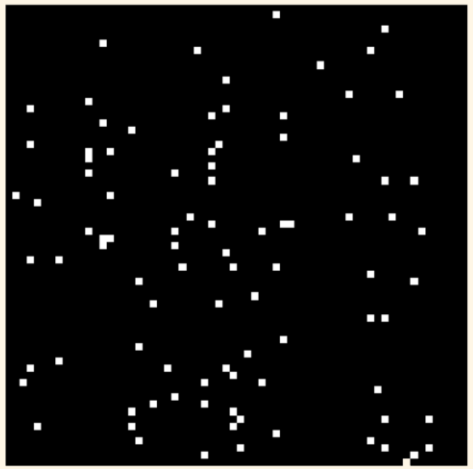
e^- at 10179 MeV



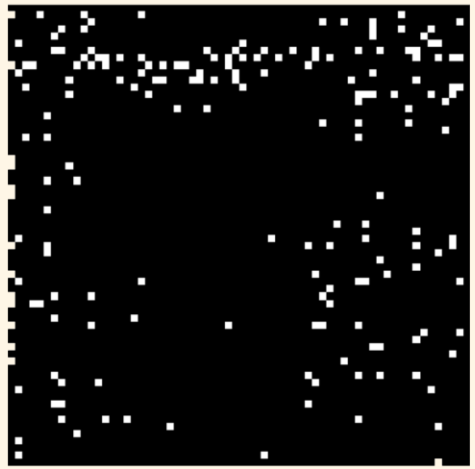
K^+ at 27274 MeV



p at 6317 MeV



π^+ at 11992 MeV



Some more ring images

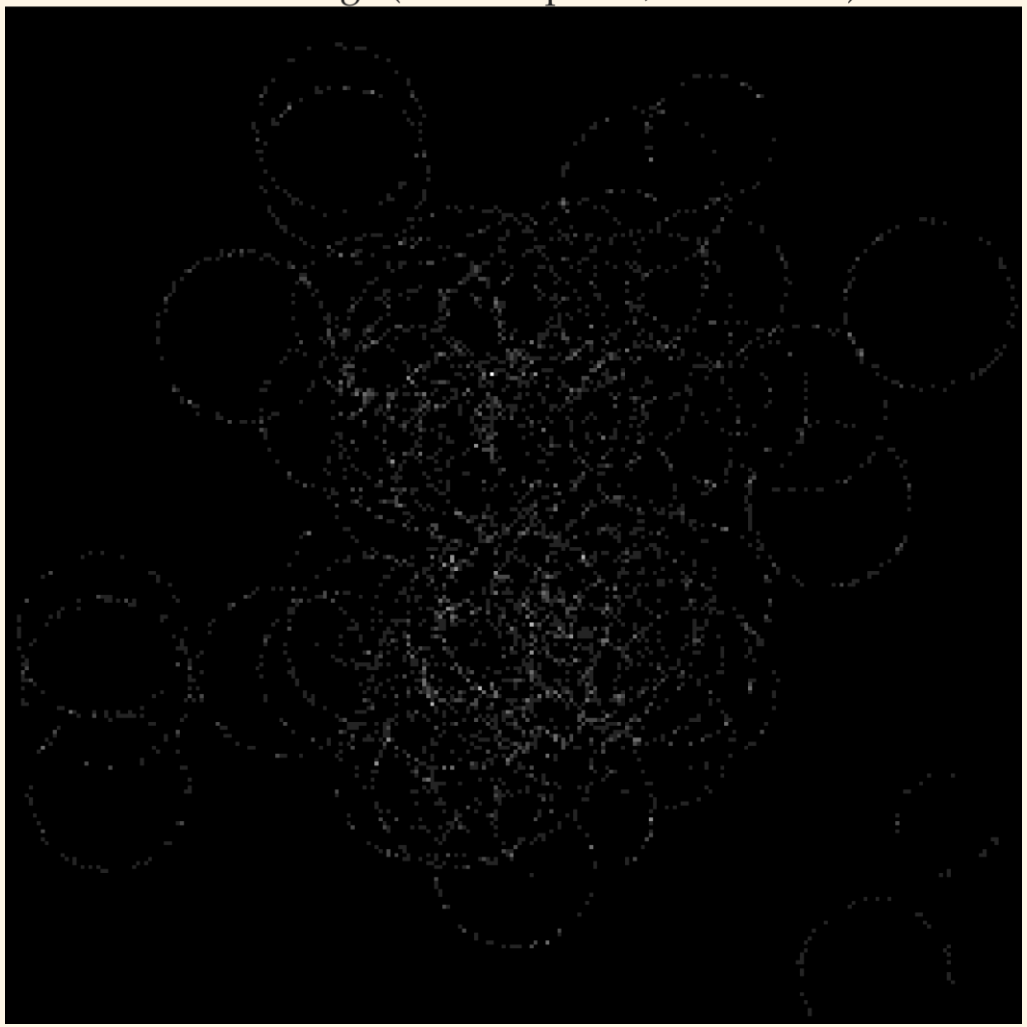
Full event Image

We can also rasterize an image of the full event. This is used for the classical maximum-likelihood fitter.

The image resolution can be specified, but it should be chosen such that a pixel is smaller than the smearing induced by the Cherenkov angle resolution.

Note: the image here is normalized (0-255) but the real "images" produced by the generator for the full event contain the real number of photon hits in each pixel. Also, the image shown is of smaller resolution for visualization.

Event Image (256x256 pixels, full RICH1)



Giovanni Laganà, Maurizio Martinelli, Martino Borsato University of Milano Bicocca and INFN

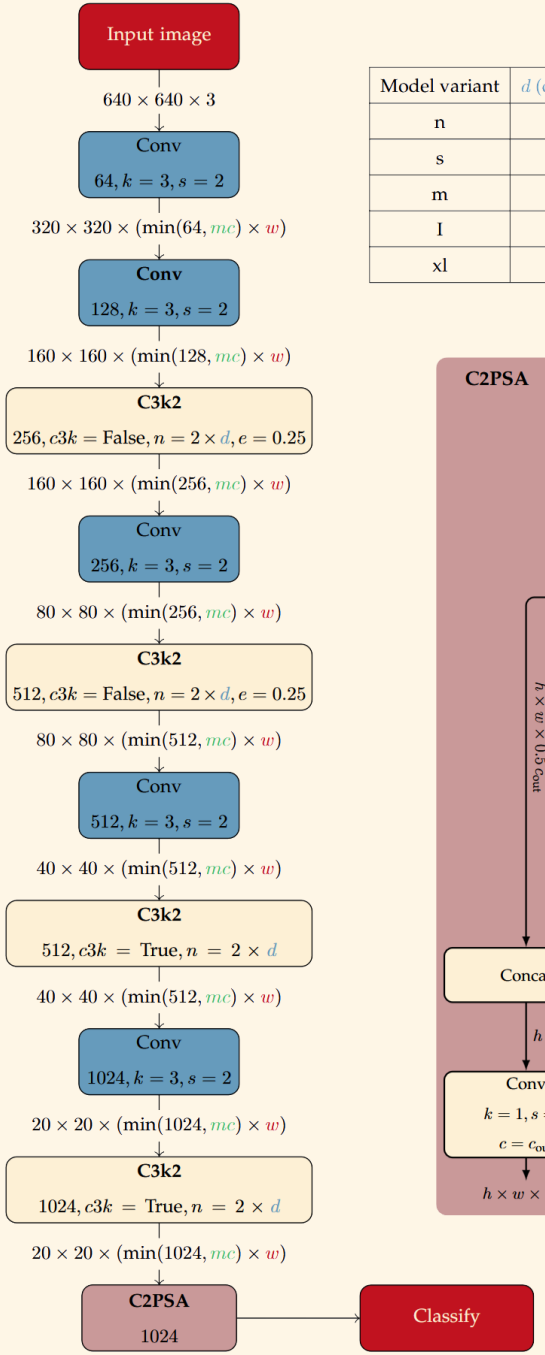
YOLO-Based Pipeline

You Only Look Once: real-time image classification

Image Classification

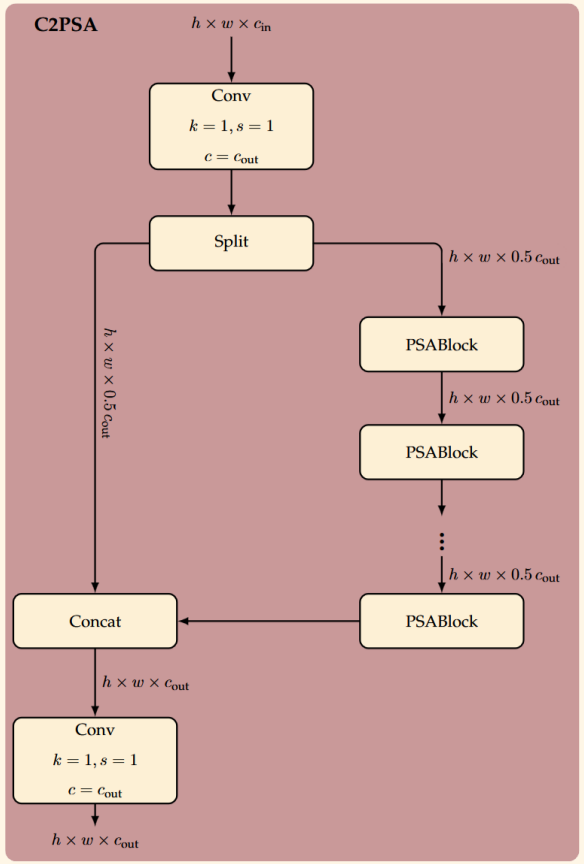
Cherenkov photons form ring patterns in RICH detector images, making **image-based methods** ideal for particle identification.

Given the need for fast classification, **YOLO** models are a natural choice since they are optimized for speed and they are even used in real time applications.



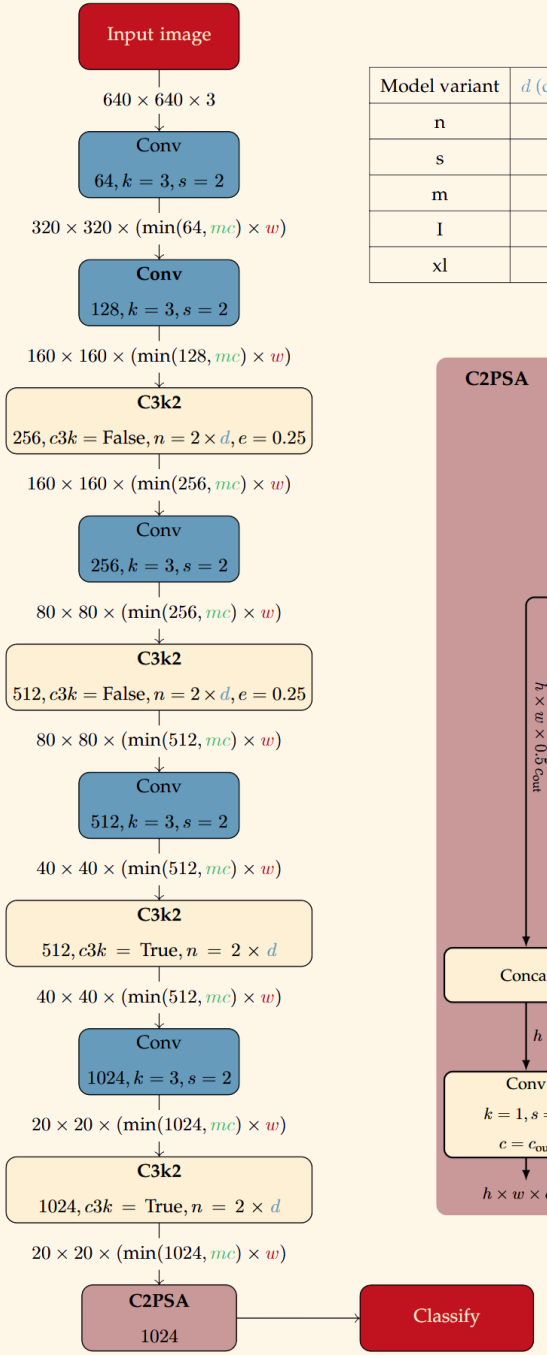
Model variant dependent parameters

Model variant	d (depth multiple)	w (width multiple)	mc (max channels)
n	0.50	0.25	1024
s	0.50	0.50	1024
m	0.50	1.00	512
l	1.00	1.00	512
xl	1.00	1.50	512



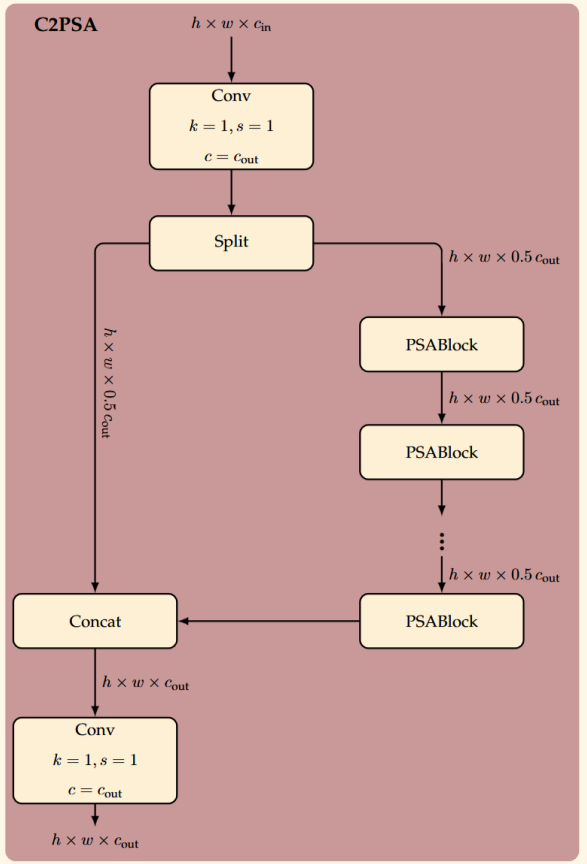
YOLO Classification

There are different variants of the 11th YOLO version. We use the classification one, which classifies a whole image.



Model variant dependent parameters

Model variant	d (depth multiple)	w (width multiple)	mc (max channels)
n	0.50	0.25	1024
s	0.50	0.50	1024
m	0.50	1.00	512
l	1.00	1.00	512
xl	1.00	1.50	512

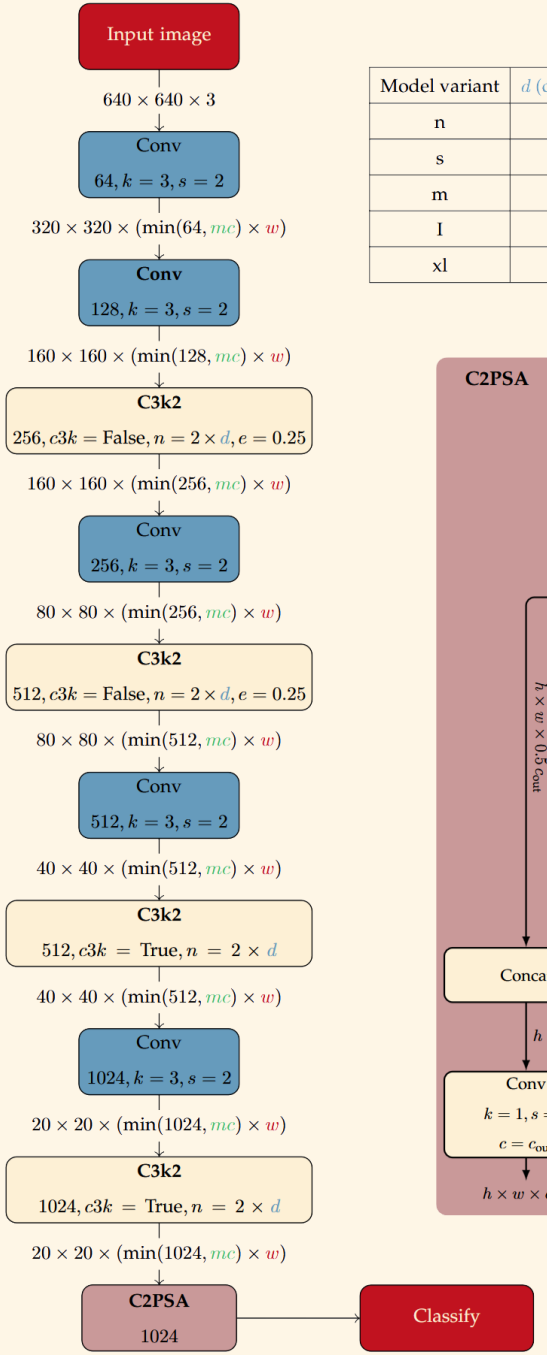


Glenn Jocher and Jing Qiu. Ultralytics YOLO11. Version 11.0.0. 2024.

YOLO Classification

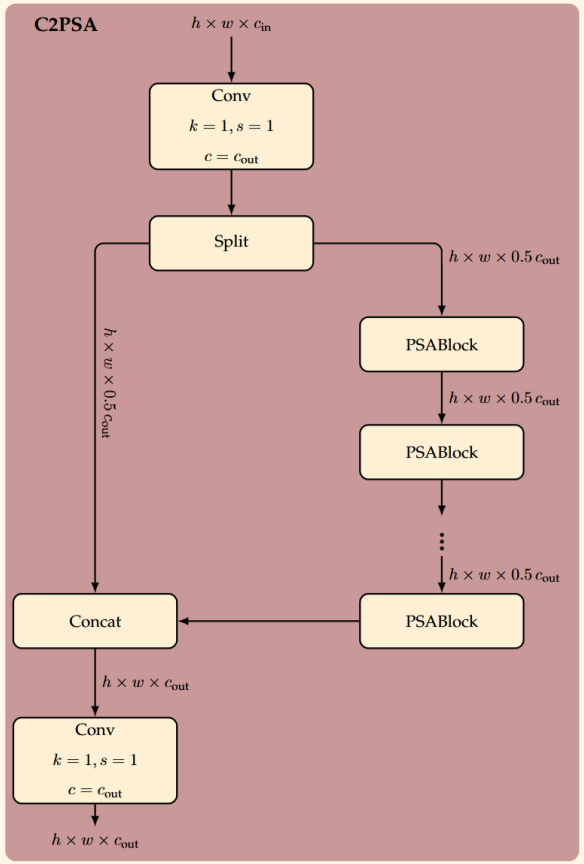
There are different variants of the 11th YOLO version. We use the classification one, which classifies a whole image.

It is comprised of a CSP-based **backbone**, a **neck**, the C2PSA block, and a **head** (the classify block).

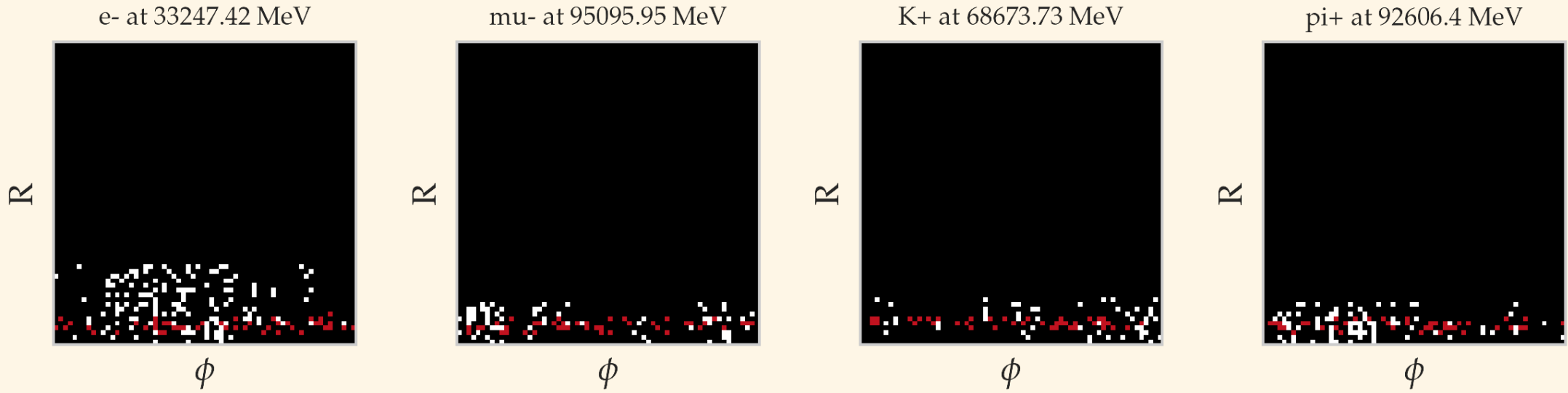


Model variant dependent parameters

Model variant	d (depth multiple)	w (width multiple)	mc (max channels)
n	0.50	0.25	1024
s	0.50	0.50	1024
m	0.50	1.00	512
l	1.00	1.00	512
xl	1.00	1.50	512



Glenn Jocher and Jing Qiu. Ultralytics YOLO11. Version 11.0.0. 2024.



As we can see, different particle types can produce rings of the same size (at different momenta).

YOLO however can **only take images as input**. What we can do is to use different models for bins of momenta.

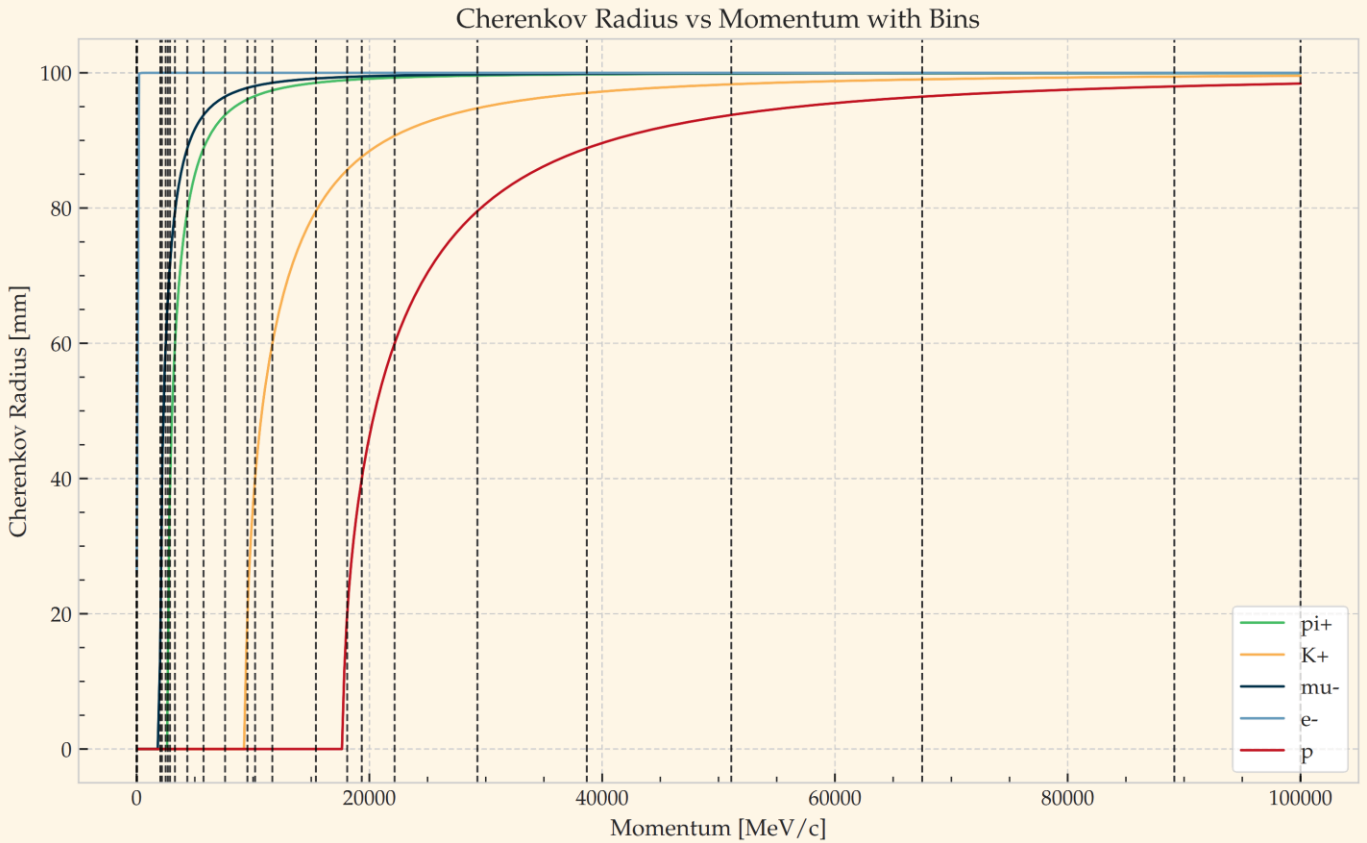
This is not optimal, but it is good to ensure the approach with YOLO can work.

The Pipeline

We **divide the momenta range into bins**; in each we train a YOLO model.

An algorithm for finding the best bin configuration was devised, aiming to minimize the number of bins, while maintaining good separation in ring radii for different particle types.

We found an optimal number of **25 bins**.



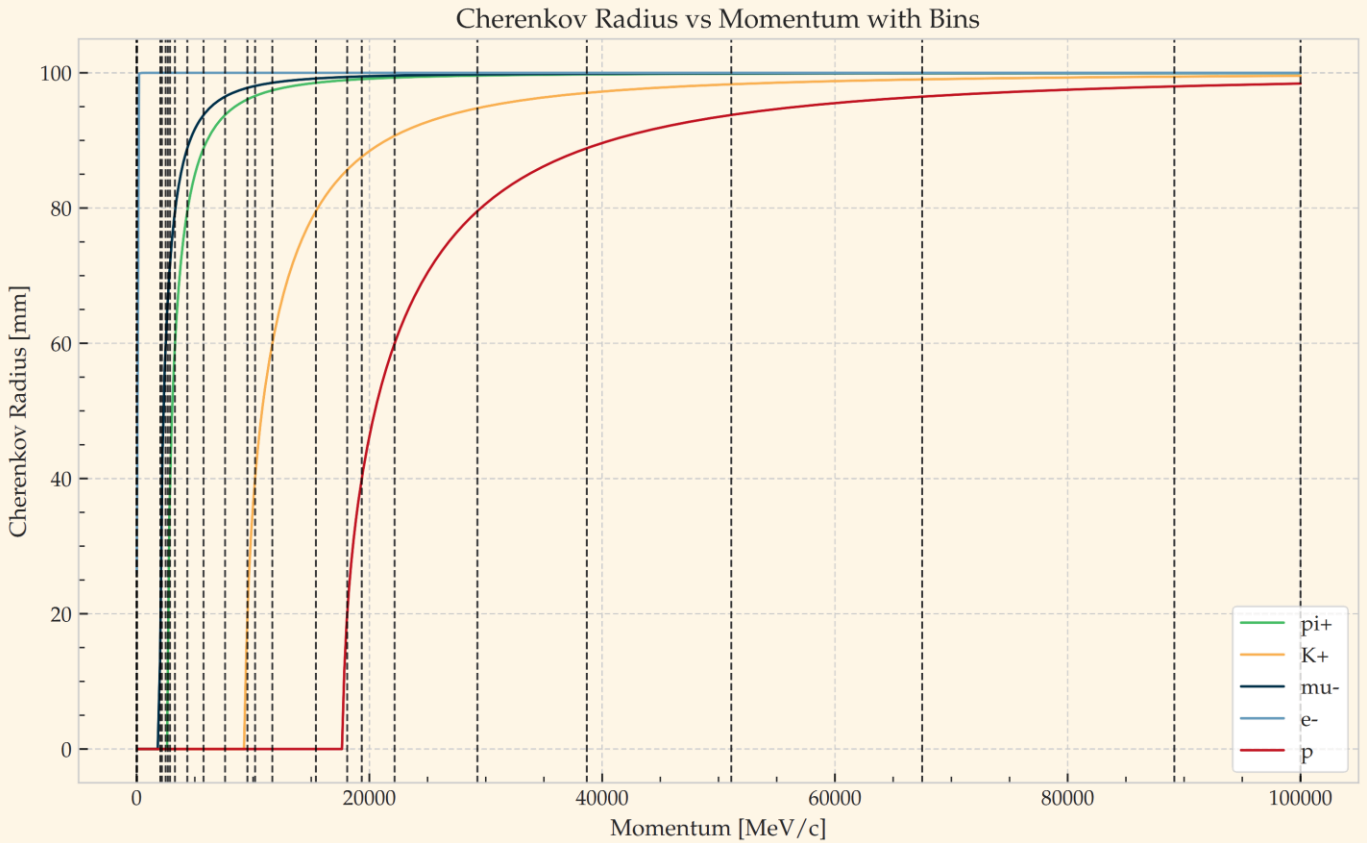
The Pipeline

We **divide the momenta range into bins**; in each we train a YOLO model.

An algorithm for finding the best bin configuration was devised, aiming to minimize the number of bins, while maintaining good separation in ring radii for different particle types.

We found an optimal number of **25 bins**.

In high momenta bins, Cherenkov radii of different particle types are too close to be physically distinguishable.



Training, Inference & Evaluation

For each bin we train a YOLOv11-cls model (starting from pre-trained weights).

A dataset with uniform distribution of particle type and momenta is used. It consists of 25k images for each bin (80% training, 20% validation)

Training, Inference & Evaluation

For each bin we train a YOLOv11-cls model (starting from pre-trained weights).

A dataset with uniform distribution of particle type and momenta is used. It consists of 25k images for each bin (80% training, 20% validation)

All models for different momentum bins are loaded on the device and images to be classified are distributed to the correct model.

The test set consists of a realistically distributed set of ring images, about 850k (for all the momenta range)

Training, Inference & Evaluation

For each bin we train a YOLOv11-cls model (starting from pre-trained weights).

A dataset with uniform distribution of particle type and momenta is used. It consists of 25k images for each bin (80% training, 20% validation)

All models for different momentum bins are loaded on the device and images to be classified are distributed to the correct model.

The test set consists of a realistically distributed set of ring images, about 850k (for all the momenta range)

The main evaluation metrics is the **pion misidentification rate** vs the **kaon identification efficiency**.

Similar metrics have also been used for other particle pairs.

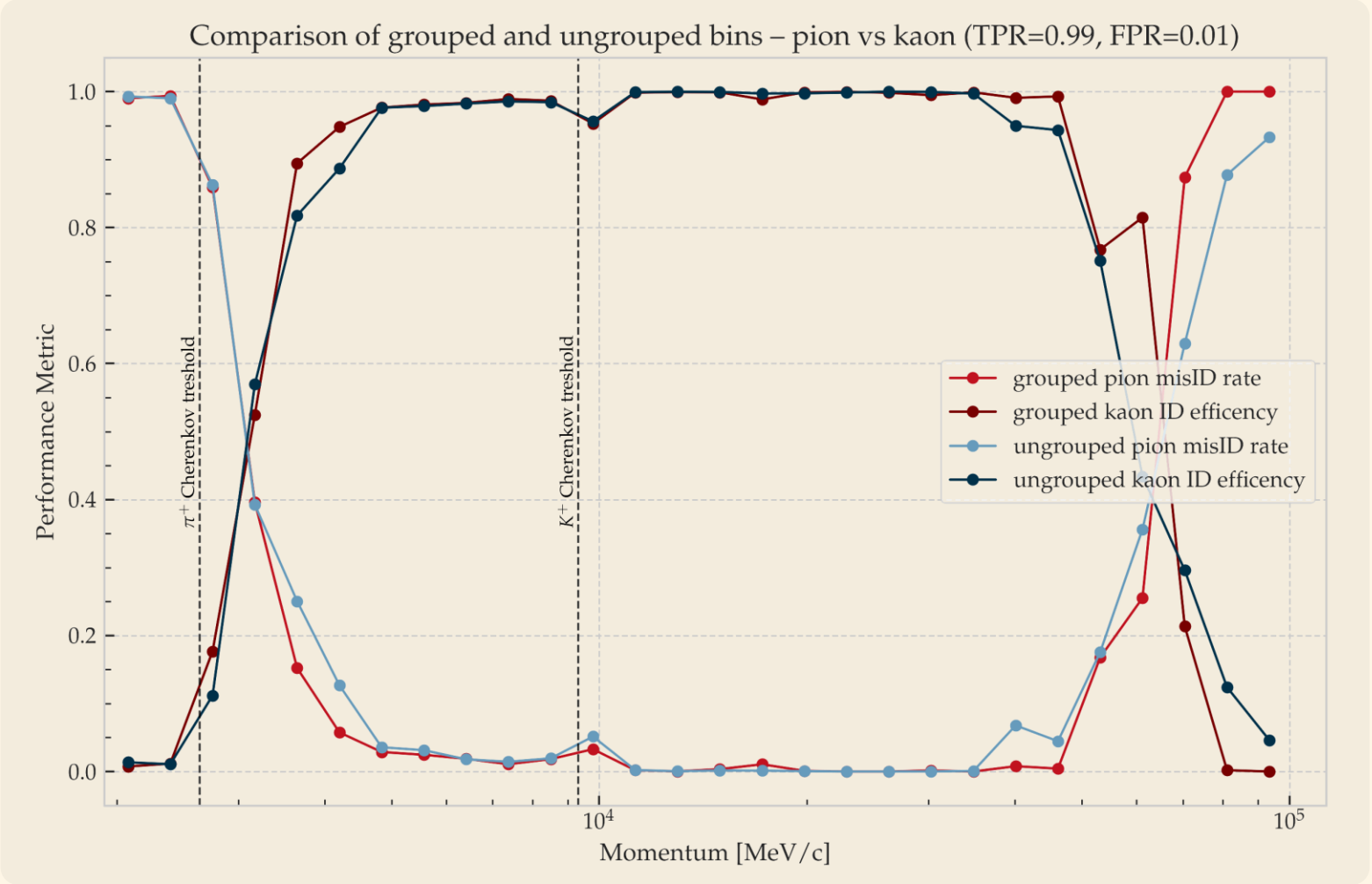
Grouping vs Non-Grouping

Initially an experiment was conducted to check if the grouping of particle types in a same class was advantageous.

It turns out that using grouping gives higher performances in higher momenta range.

Grouping i.e.

At higher momenta muons and pions are grouped in the same class and at inference time, the probability for such class is split equally among pion and moun hypothesis.

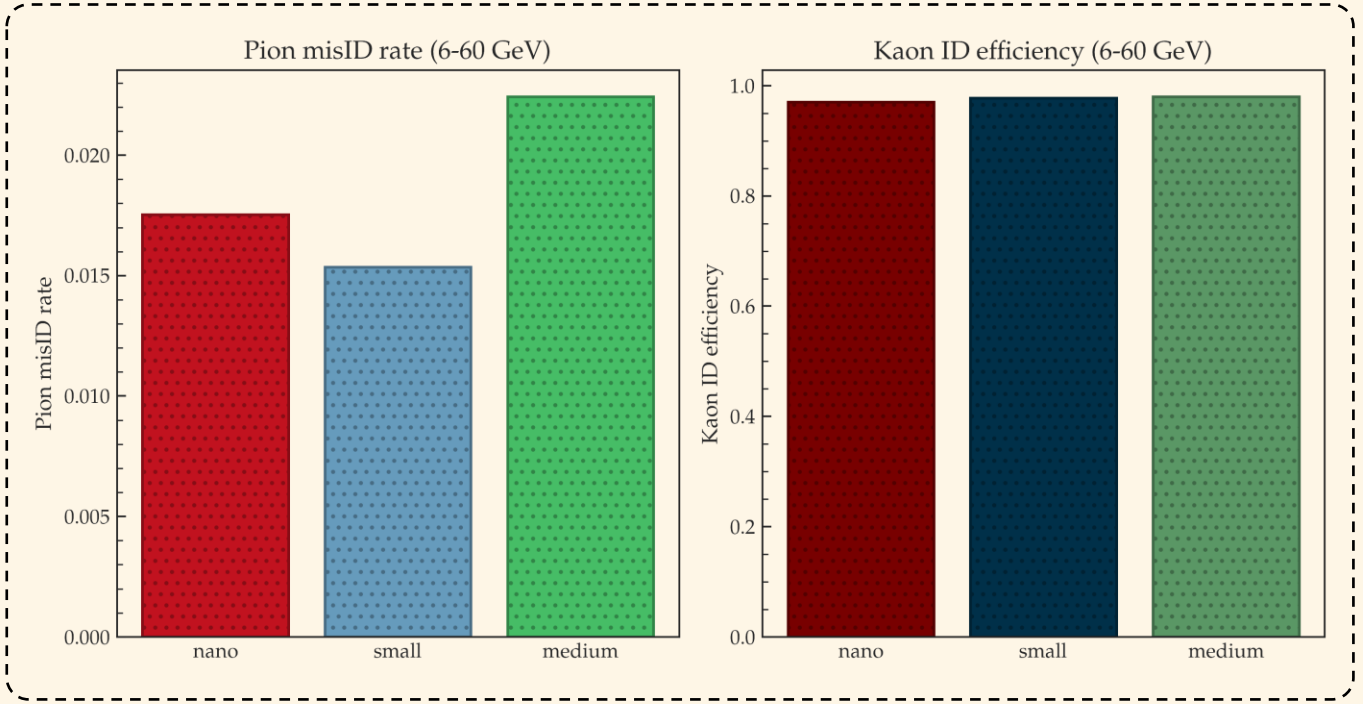


Model Size Experiment

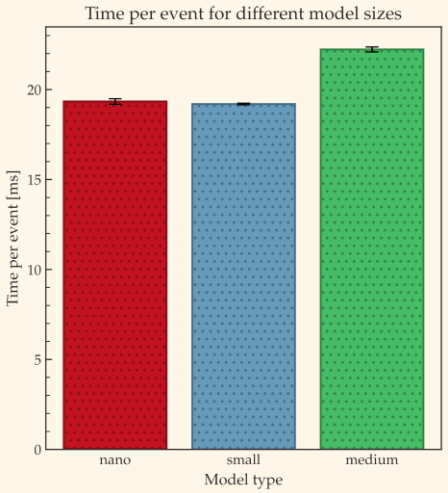
YOLOv11 comes in different size variants: *nano*, *small*, *medium*, *large* & *extra-large.*

We compered three of this variants.

Pion misID rate & Kaon ID efficiency in 6 to 60 GeV range.



Note: an event consists of 118 rings



Evaluation was done on an NVIDIA A6000 GPU.

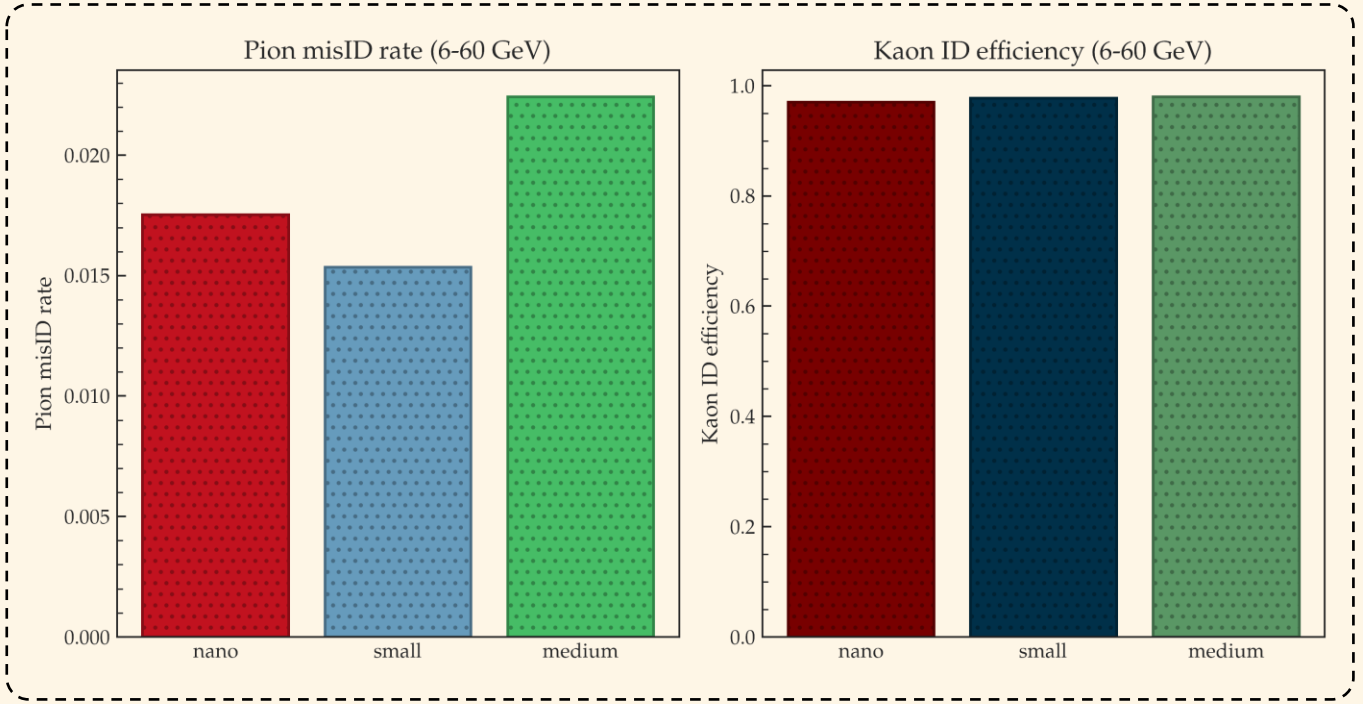
Model Size Experiment

YOLOv11 comes in different size variants: *nano*, *small*, *medium*, *large* & *extra-large.*

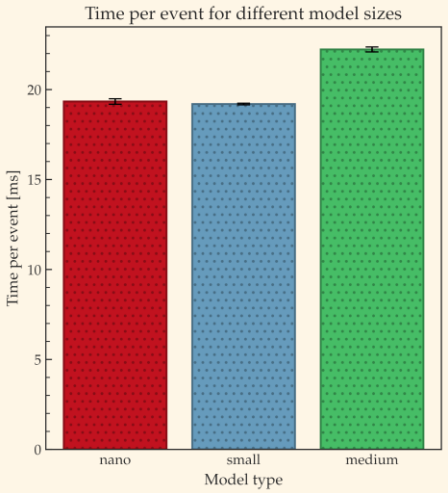
We compered three of this variants.

The accuracy of the small and medium are not too dissimilar, while the inference time of the small is smaller, nearly equal to the nano.

Pion misID rate & Kaon ID efficiency in 6 to 60 GeV range.



Note: an event consists of 118 rings

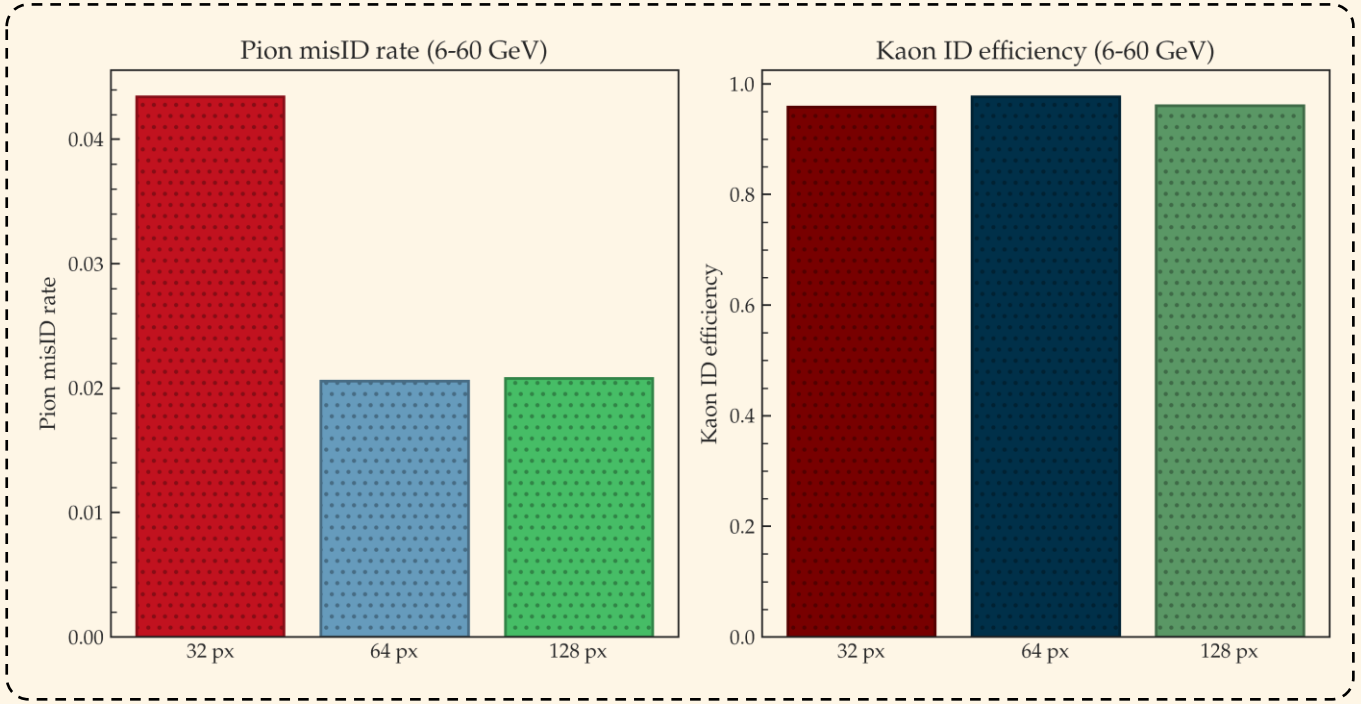


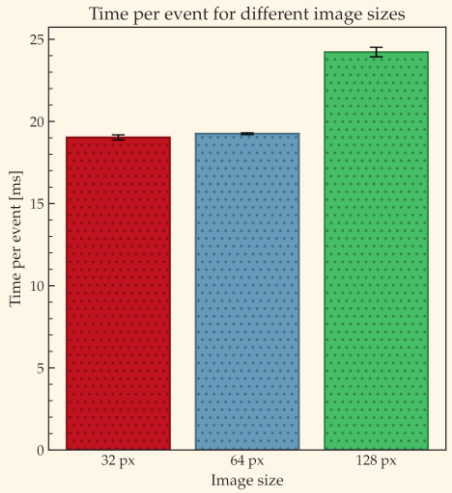
Evaluation was done on an NVIDIA A6000 GPU.

Image Size Experiment

We tried using different image size for the Cherenkov rings; we tried: *32x32px*, *64x64px* & *128x128px*

Pion misID rate & Kaon ID efficiency in different momenta ranges.





Evaluation was done on an NVIDIA A6000 GPU.

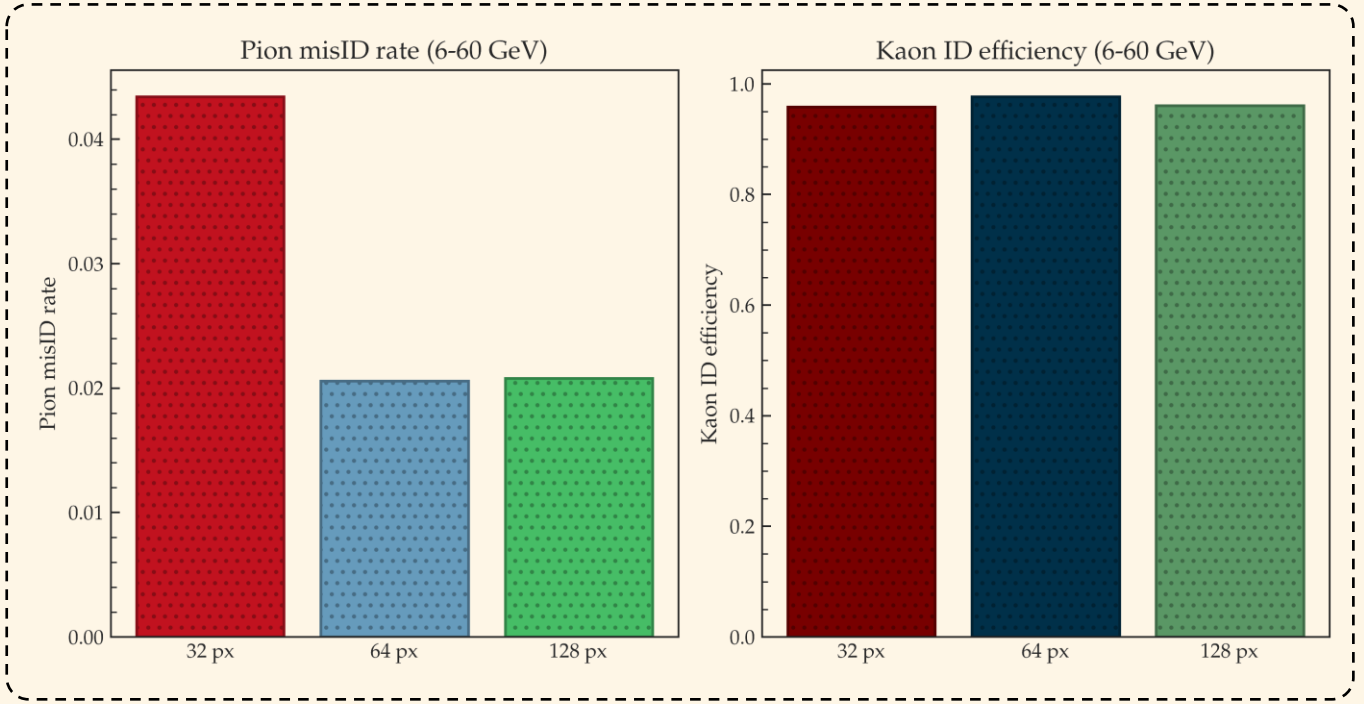
Image Size Experiment

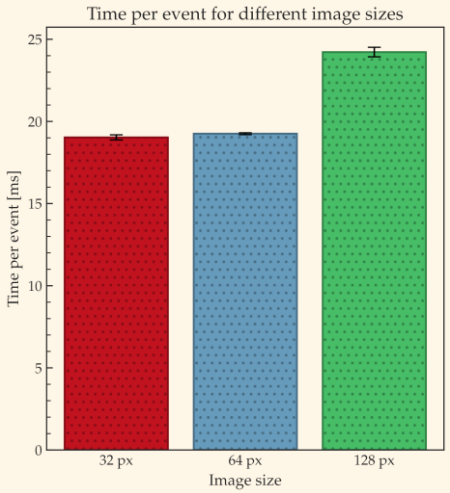
We tried using different image size for the Cherenkov rings; we tried: *32x32px*, *64x64px* & *128x128px*

The 64x64px images are the best ones, striking a balance between accuracy and inference speed.

N.B: The 128^2 image has a resolution close to the Cherenkov resolution we used to generate the samples.

Pion misID rate & Kaon ID efficiency in different momenta ranges.





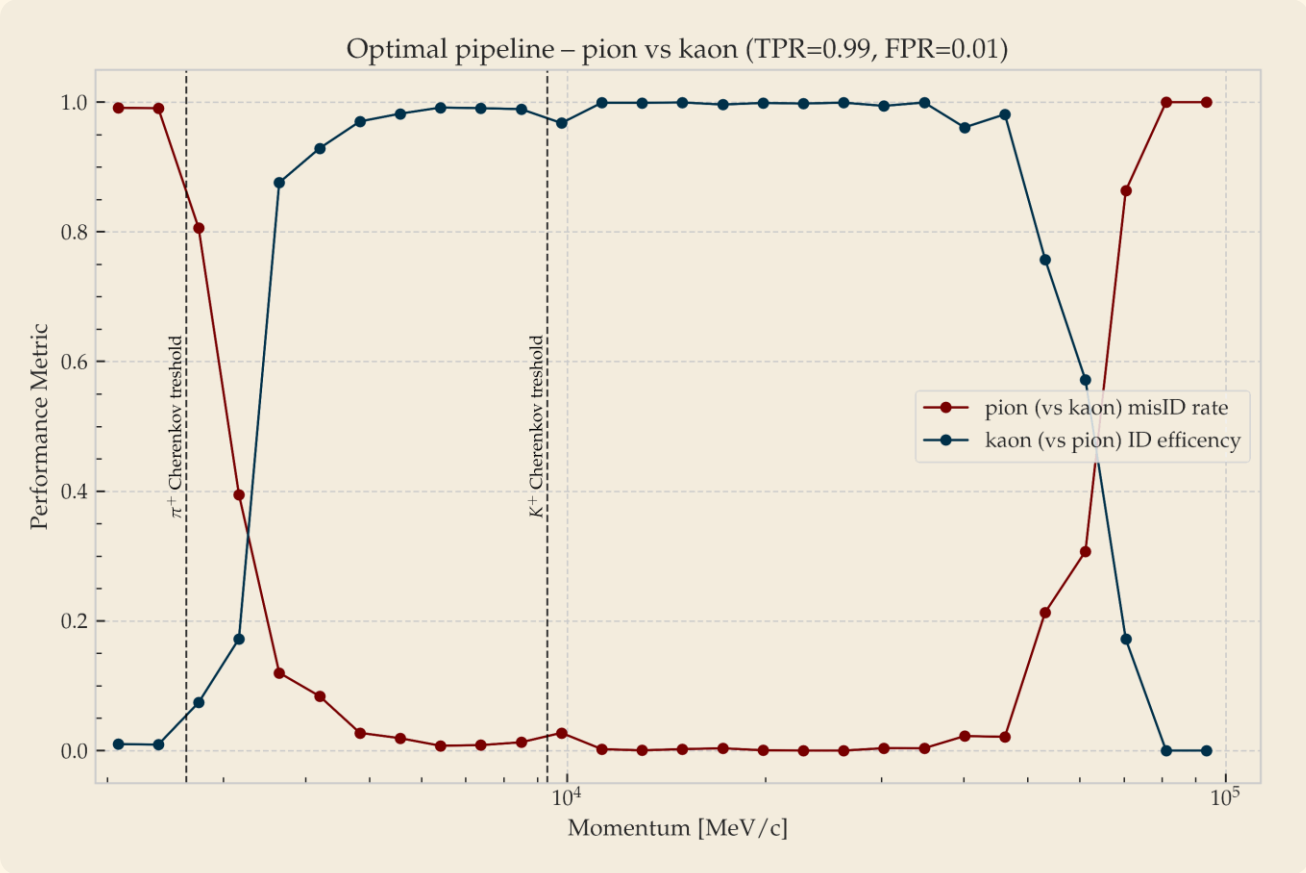
Evaluation was done on an NVIDIA A6000 GPU.

The Optimal Pipeline

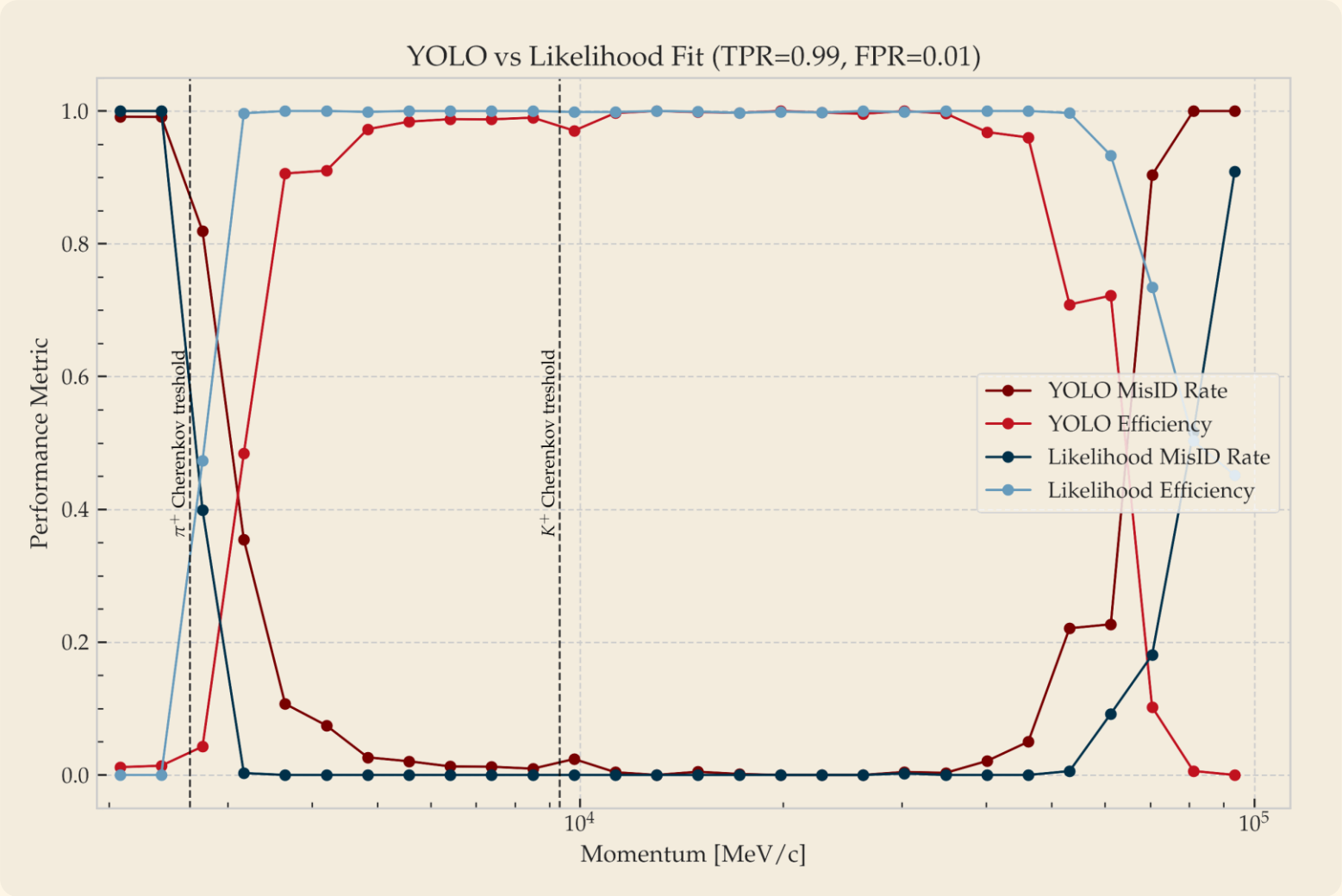
After our experiments, we concluded that the best pipeline was the one using the small variant of YOLO and with 64x64px images.

It achieves a **kaon ID efficiency above 95%** for a **pion misID rate of 1%** in a broad momenta range. This results are comparable with current standards.

It takes about **20ms** to infer a full event (RICH1+RICH2: 118 rings*) on an NVIDIA A6000 GPU.



YOLO vs Maximum-Likelihood Fit

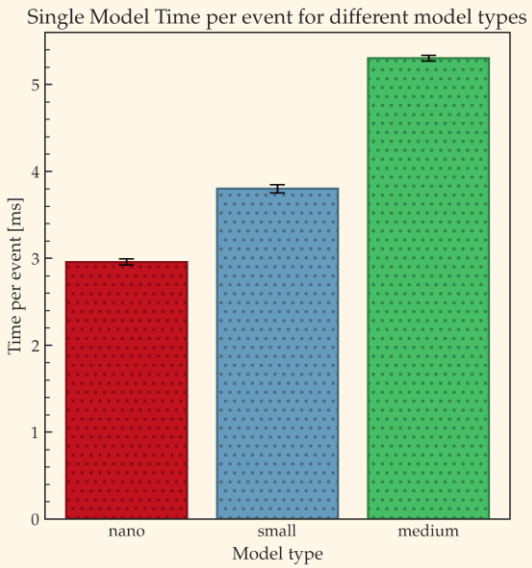


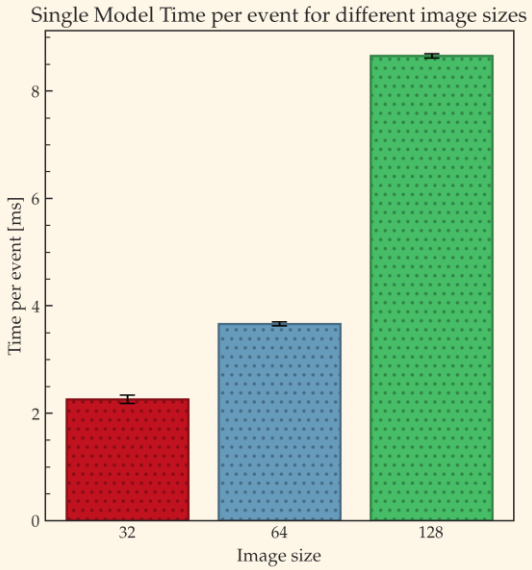
Distribution Bottleneck

Looking at the previous performance result it would seem there is a bottleneck.

We hypothesized it was due to the distribution of images to the models of the pipeline.

We ran a test using only a single model to exclude the distribution process. We consider the same 118 rings per event to estimate performances.





Note: In the single-model setup, inference was performed using only one instance of the model.

Outlook

An accurate and fast RICH PID

Achievements

Our YOLO-based pipeline delivered very promising results.

We achieved quite good accuracy in PID. In the range 5-40 GeV we get pion misID rates and kaon ID efficiencies very close to the global maximum-likelihood fit.

The pipeline has a quite fast inference time of 20ms per event on an NVIDIA A6000 graphics card.

Current limits

Routing images to the correct model creates a performance bottleneck.

The classification is based on local data, contrary to the maximum-likelihood fit. The models do not know where the ring is on the detector plane.

Real data exhibit significant position-dependent ring distortions.

Next Steps

1. Collaborate with LHCb to use data from Monte Carlo simulation or real data. Include RICH2 data.
2. Use just one YOLO model. In this way we reduce the overhead of data distribution and remove the need for binning. We are currently working on modifying the YOLOv11-cls model to incorporate information about momentum and center position on the detector.

Thanks for the $softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

-

Backup Slides

-

SCGen Radii and number of photon hits

Radii of Cherenkov rings were determined from the Cherenkov angle as:

$$R_c = \frac{\tan(\theta_c) \cdot R_{max}}{\tan(\theta_{c_{max}})}$$

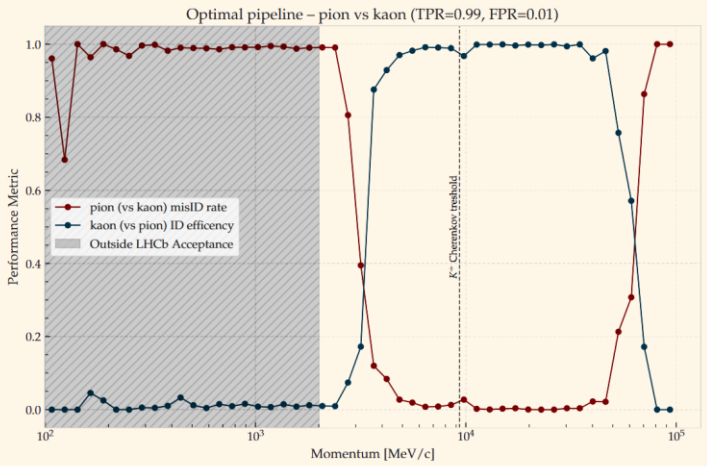
Con $R_{max} = 100\ mm$ e $\theta_{c_{max}} = 1/n$

The number of Cherenkov photon hits was determined, starting from a user defined N value, then updating it as:

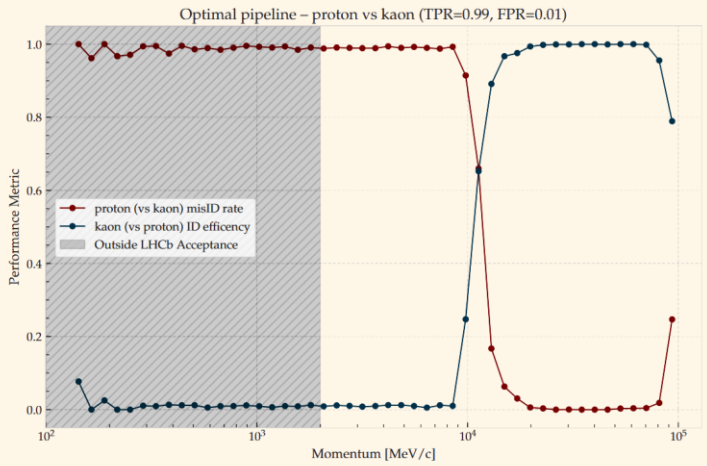
$$\widetilde{N} = N \frac{\sin(\theta_c)^2}{\sin(\theta_{c_{max}})^2}$$

And finally sampling the photon hits number from a Poisson distribution with mean $\widetilde{N}$.

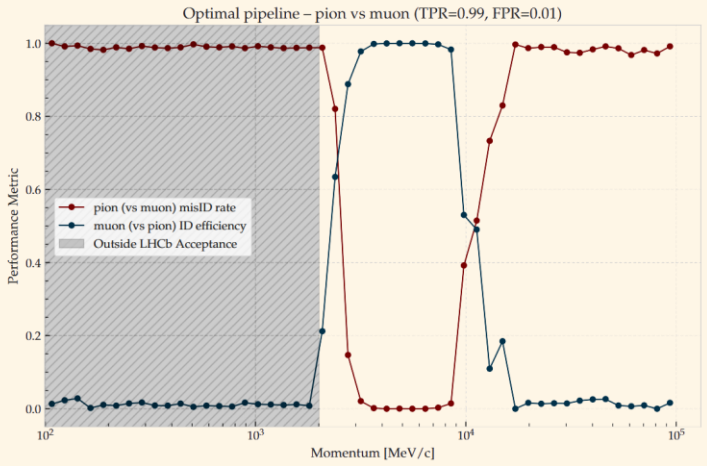
All the optimal pipeline results



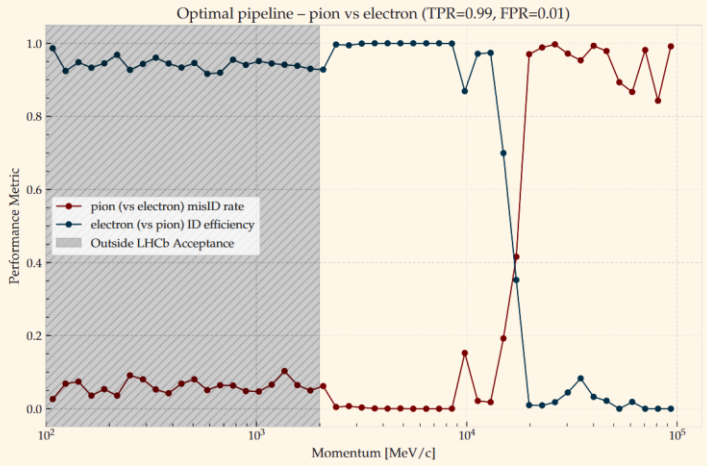
(a) Pion vs Kaon



(b) Proton vs Kaon

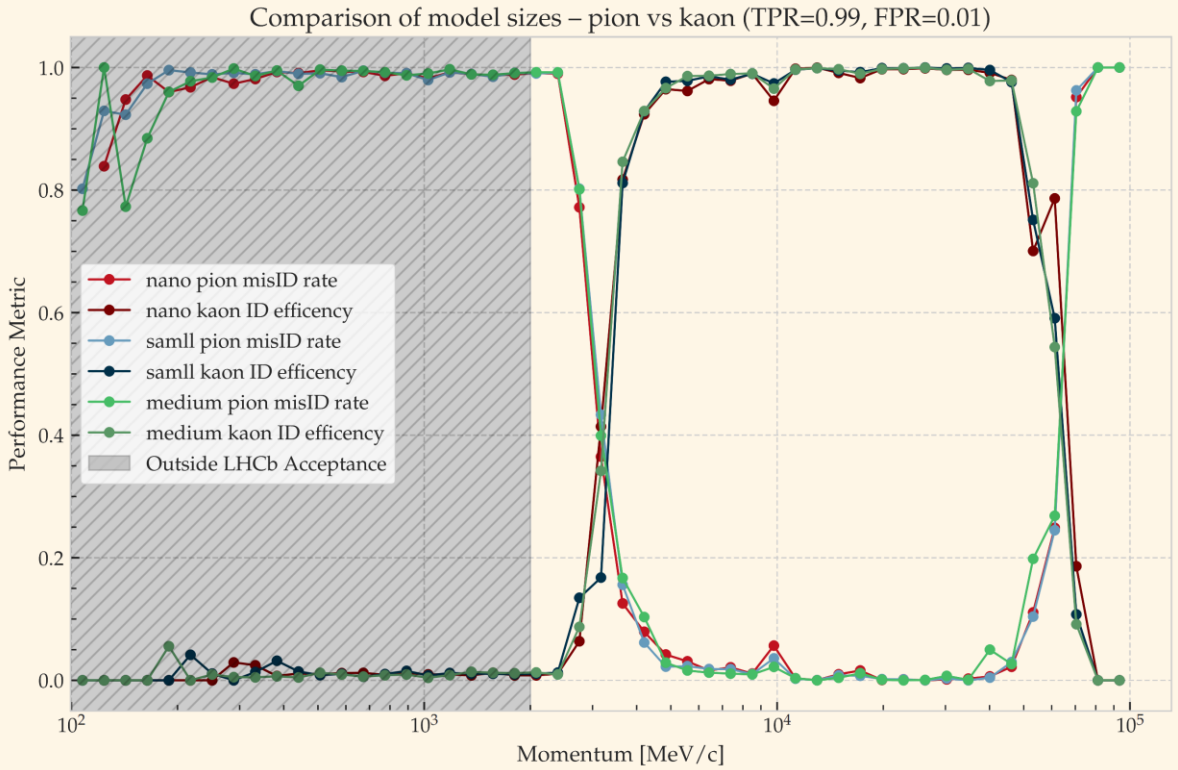


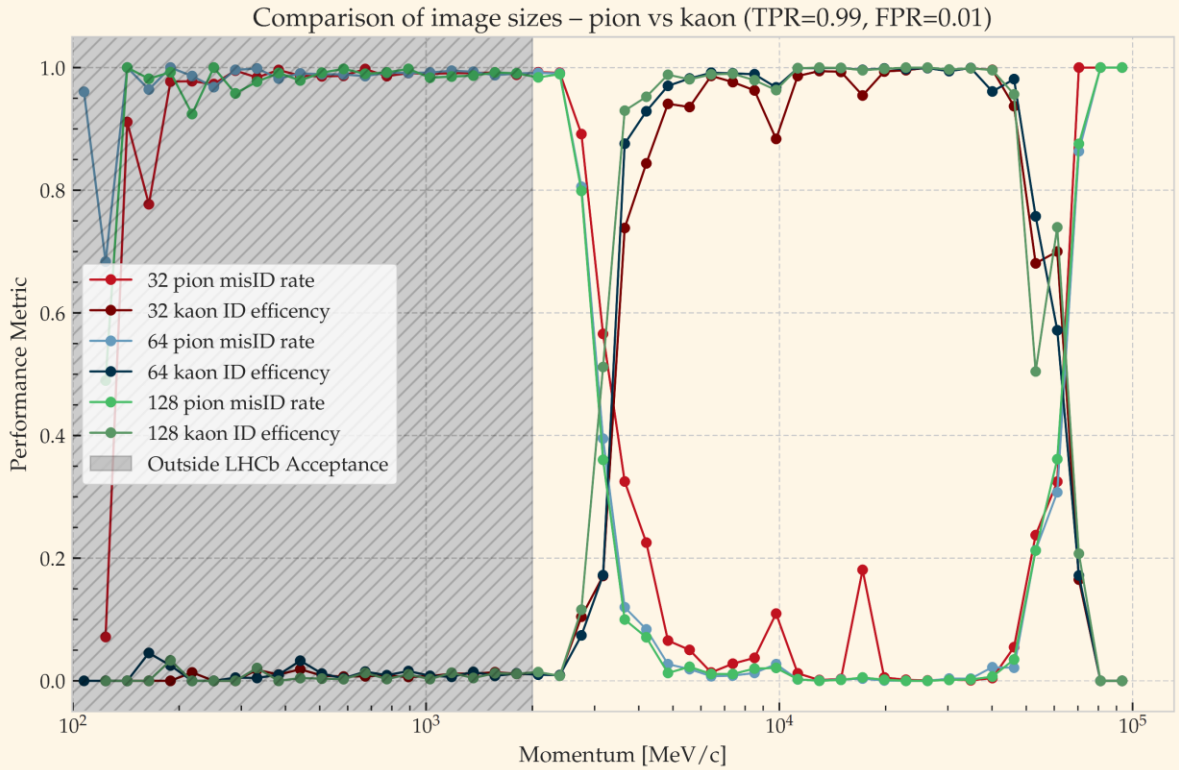
(c) Pion vs Muon



(d) Pion vs Electron

Experiments pion misID vs kaon ID eff.



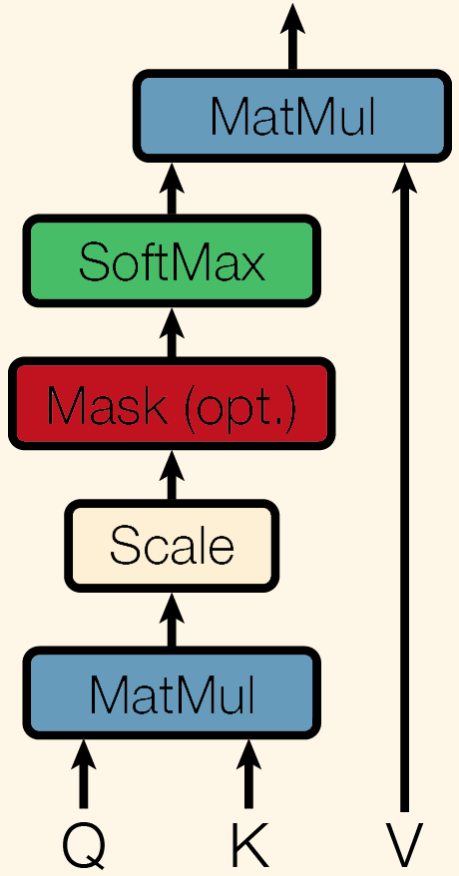


The attention mechanism

- backup slides -

What is Attention?

It is a strategy that allows a neural network to dynamically weigh input elements, **focusing on the most relevant features** when building a contextual representation.

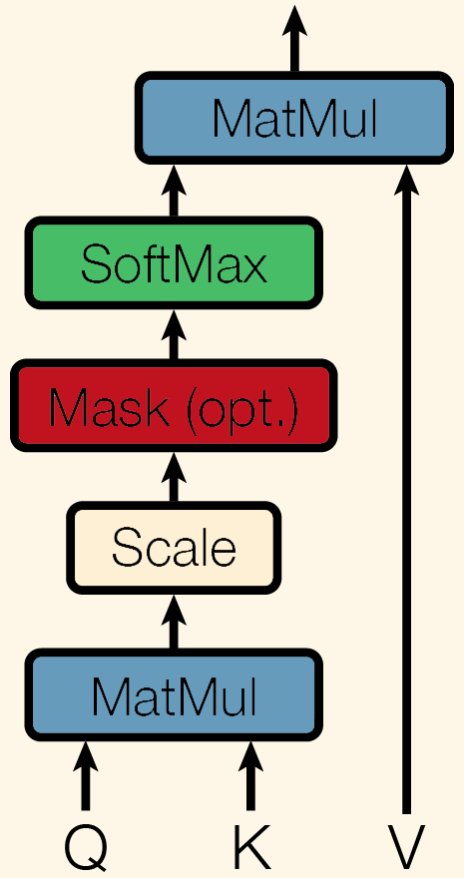


What is Attention?

It is a strategy that allows a neural network to dynamically weigh input elements, **focusing on the most relevant features** when building a contextual representation.

For a given query, the mechanism computes a compatibility score with each key.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$



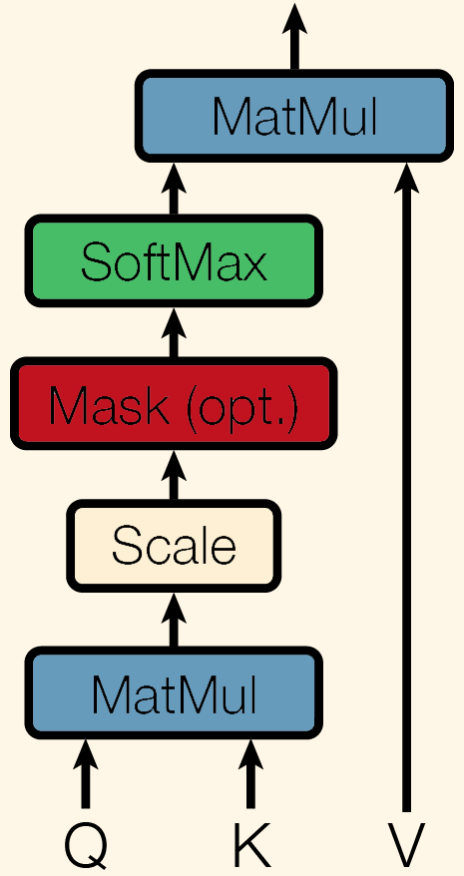
What is Attention?

It is a strategy that allows a neural network to dynamically weigh input elements, **focusing on the most relevant features** when building a contextual representation.

For a given query, the mechanism computes a compatibility score with each key.

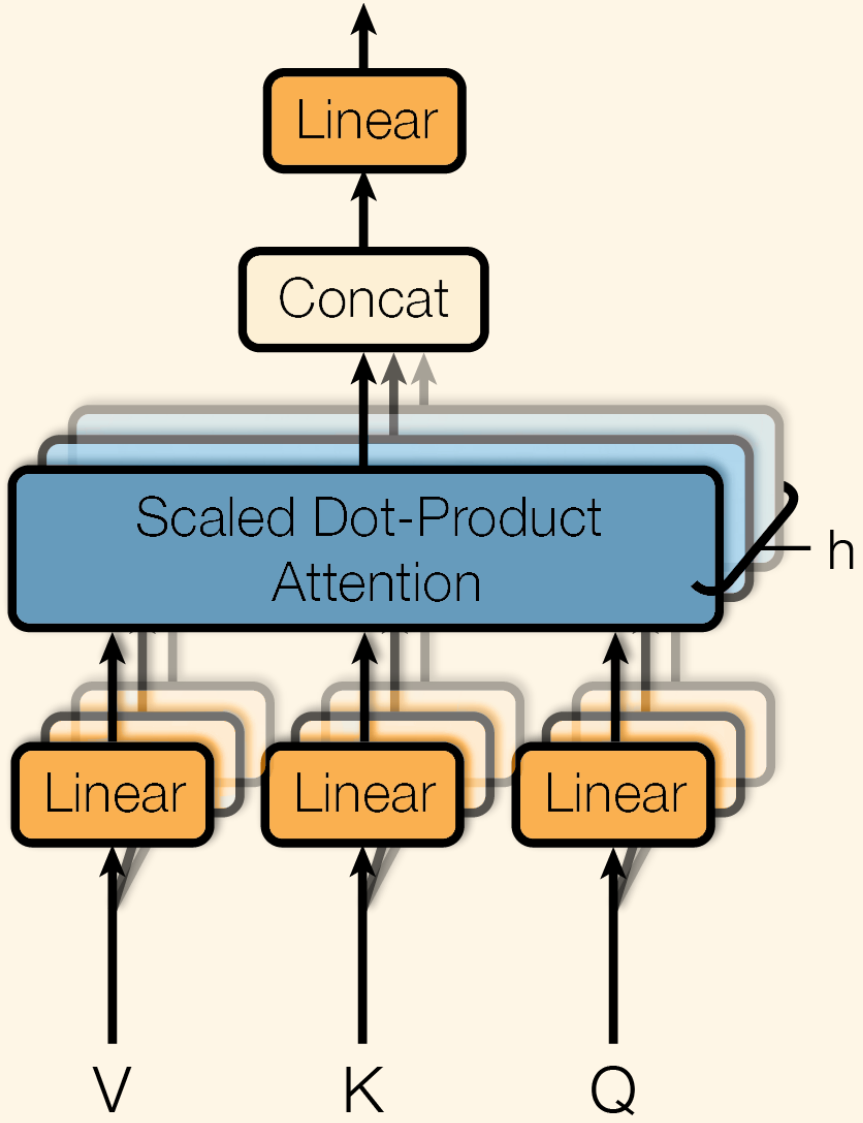
$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

In **self-attention**, the input itself is used to generate the queries, keys, and values, enabling each element to attend to all others in the sequence.



What is Attention?

In multi-head attention, each head focuses on different aspects or relationships within the data.
The outputs of these heads are then concatenated or averaged.



Maximum Likelihood Fit

- backup slides -

Max-Likelihood Fitter

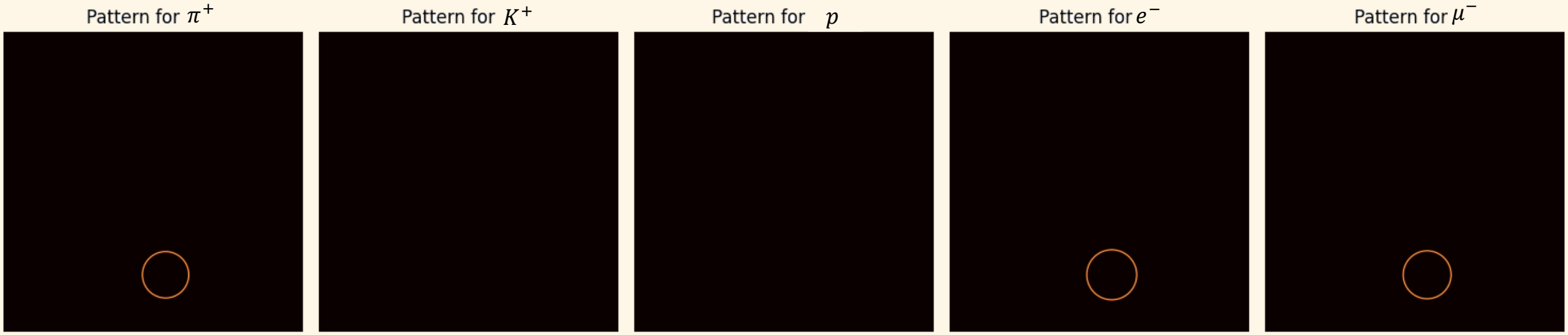
GPU implementation of the original Forty R. & Schneider O. algorithm

https://cds.cern.ch/record/684714

We now want to compare the YOLO-based Pipeline results with the classical approach used at LHCb, which consists in a max-likelihood fit.

However, since we used custom generated data, we will need to adapt the fitter accordingly. We therefore developed an implementation of the original fitter used for RICH data.

To speed testing times, and to enable fairer comparisons between our approach and the classical method in the future, we implemented the algorithm on GPU (note however, that it is still not fully optimized and inference time comparisons have not been done yet)

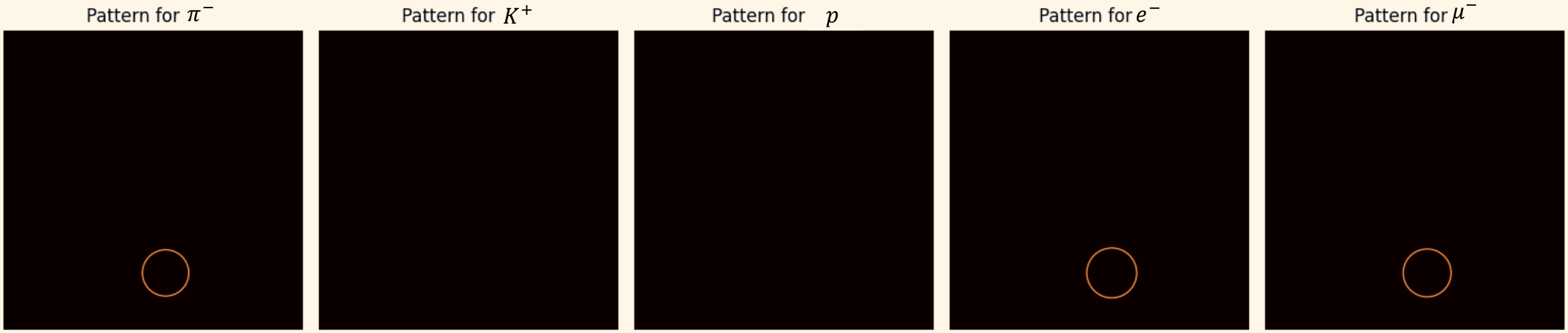


Momentum = 7066.87 GeV

The idea is to **generate hypothesis hit patterns for every track, mass hypothesis combination**, in the event.

Then we can **create global patterns combining** such **single-track patterns** and use those to evaluate the likelihood.

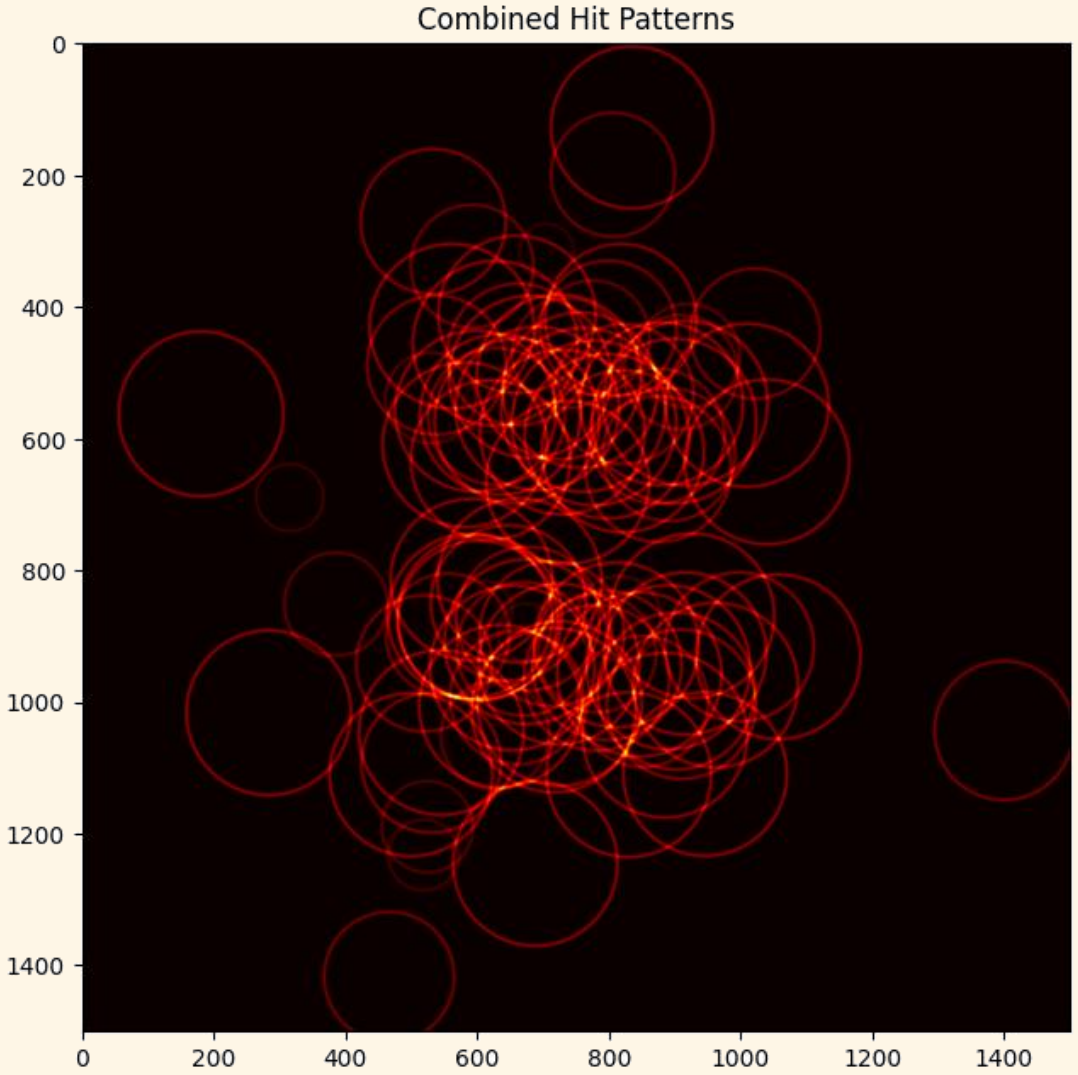
The patterns are generated by computing for each pixel the expected photon yield, following the same distribution used in the generator.



Momentum = 7066.87 GeV

Actually, we only store in memory the **total pixel count of each pattern** and the **pixels in correspondence with lit pixels on the event image**. This can be done due to how the likelihood is calculated.

Global Pattern example



Note: here we show the patterns corresponding to the true type of each track.

Likelihood calculation

$$\ln \mathcal{L}(\vec{h}) = \sum_{i=1}^{M^{tot}} \left[-\nu_i(\vec{h}) + n_i \ln \nu_i(\vec{h}) - \ln n_i! \right]$$

Global pattern at pixel i given hypothesis $\vec{h} = (h_1, h_2, \ldots, h_N)$ with N number of tracks.

Event image at pixel i.

M^{tot} it the total number pixels in the event, while M is the number of **lit** pixels.

Likelihood calculation

$$\ln \mathcal{L}(\vec{h}) = \sum_{i=1}^{M^{tot}} \left[-\nu_i(\vec{h}) + n_i ln\, \nu_i(\vec{h}) - \ln n_i! \right]$$

$$= -\sum_{i=1}^{M^{tot}} \nu_i(\vec{h}) + \sum_{i=1}^{M} n_i ln\, \nu_i(\vec{h}) - C$$

M^{tot} it the total number pixels in the event, while M is the number of **lit** pixels.

Likelihood calculation

$$\ln \mathcal{L}(\vec{h}) = \sum_{i=1}^{M^{tot}} \left[-\nu_i(\vec{h}) + n_i ln\, \nu_i(\vec{h}) - \ln n_i! \right]$$

$$= -\sum_{i=1}^{M^{tot}} \nu_i(\vec{h}) + \sum_{i=1}^{M} n_i ln\, \nu_i(\vec{h}) - C$$

$$= -\sum_{j=1}^{N} \mu_j(h_j) + \sum_{i=1}^{M} n_i ln \sum_{j=1}^{N} a_{ij}(h_j) - C$$

Pattern of hyp j at pixel i.

$$\mu_j(h_j) = \sum_{i=1}^{M^{tot}} a_{ij}(h_j)$$

The optimization procedure

To find the optimal combination of hypothesis for each track the Forty & Schneider procedure is illustrated in the pseudocode.

Algorithm 1 Fit event

Initialize: Assume all tracks are pions (hypothesis h_0)
`selected_patterns` $\leftarrow [\vec{a}_0(h_0), \vec{a}_1(h_0), \cdots, \vec{a}_N(h_0)]$
`LL` $\leftarrow -\infty$
for each iteration **do**
    `LLs` $\leftarrow [\,]$
    for $j = 0$ to N **do**
        for $m = 0$ to $M^{\max}$ **do**
            Replace j-th pattern: `selected_patterns`$[j] \leftarrow \vec{a}_j(h_m)$
            `global_pattern` $\leftarrow \sum$ `selected_patterns`
            `LLs.append(compute_LL(global_pattern))`
        end for
    end for
    Update `selected_patterns` with the best ones from `LLs`
    `LL` $\leftarrow \max($`LLs`$)$
end for
Compute: `DLLs` $\leftarrow$ differences in log-likelihoods between π^+ and K^+

The optimization procedure

To find the optimal combination of hypothesis for each track the Forty & Schneider procedure is illustrated in the pseudocode.

However, our implementation runs on GPU (we used pytorch) and massively parallelize the hypothesis testing for each iteration.
In fact, we **compute all the hypothesis** (for each track and for each particle type) **in parallel** with few CUDA kernel launches.

Algorithm 1 Fit event

Initialize: Assume all tracks are pions (hypothesis h_0)
`selected_patterns` $\leftarrow [\vec{a}_0(h_0), \vec{a}_1(h_0), \cdots, \vec{a}_N(h_0)]$
`LL` $\leftarrow -\infty$
for each iteration **do**
    `LLs` $\leftarrow [\,]$
    for $j = 0$ to N **do**
        for $m = 0$ to $M^{\max}$ **do**
            Replace j-th pattern: `selected_patterns`$[j] \leftarrow \vec{a}_j(h_m)$
            `global_pattern` $\leftarrow \sum$ `selected_patterns`
            `LLs.append(compute_LL(global_pattern))`
        end for
    end for
    Update `selected_patterns` with the best ones from `LLs`
    `LL` $\leftarrow \max($`LLs`$)$
end for
Compute: `DLLs` $\leftarrow$ differences in log-likelihoods between π^+ and K^+