[27.05.2021]

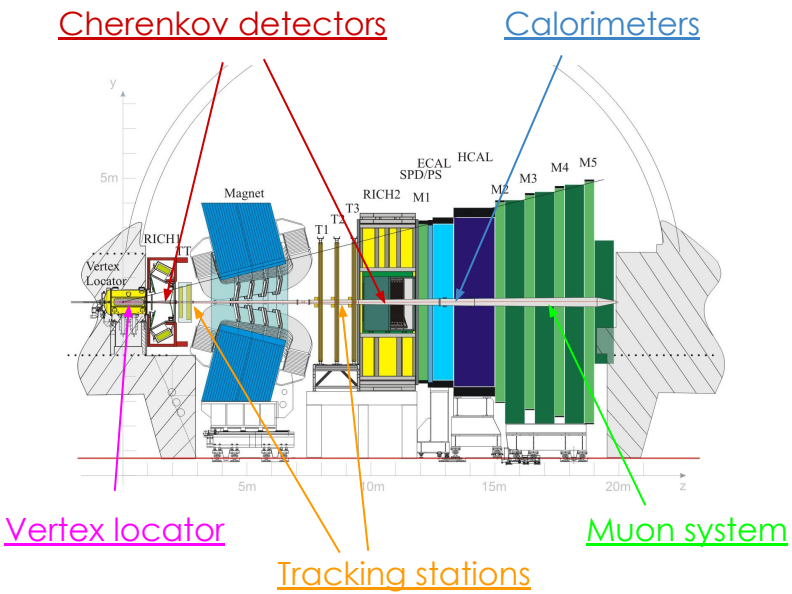
Simulating the LHCb detector with Generative Adversarial Networks

Matteo Barbetti
on behalf of the LHCb Collaboration

University of Firenze
INFN - Firenze

Workshop CCR
[24-28 May 2021]

LHCb detector and its upgrades



Detector paper [JINST 3 (2008) S08005]

Upgrade design [LHCb-TDR-12]

The **Upgrade I** of the LHCb experiment is currently in commissioning. It will go back to work for the LHC Run 3, after a replacement of about 90% of detector channels. The new LHCb will also rely on a new **fully software** trigger system.

The upgraded experiment will operate with an increased instantaneous luminosity (x5) and a more performant selection algorithm (x2), enough to collect datasets at least **one order of magnitude larger**.

In order to match the increase of collected data, **larger** simulated samples and a strategy to **speed-up** their production will be an **unavoidable** evolution of the LHCb Computing Model.

fully software trigger system

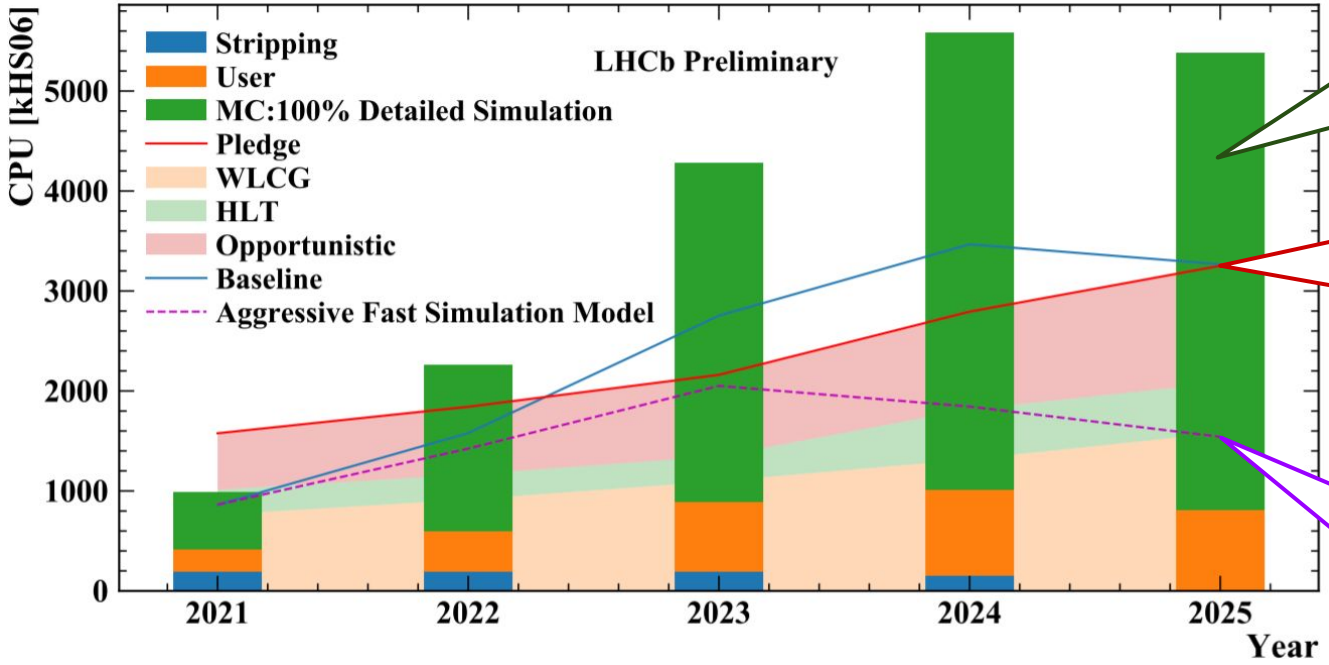
x 5 instantaneous luminosity

x 2 selection efficiency

x 10 data sample size

Computing requirements for Run 3

[LHCb-FIGURE-2019-018]



CPU resources needed for a simulation scheme only including Geant4-based ***Detailed Simulation***

Available resources according to the planned **funding scheme**

Computing resources required by an excellent ***Fast*** and ***Ultra-Fast Simulation***

Fast and Ultra-Fast Simulation

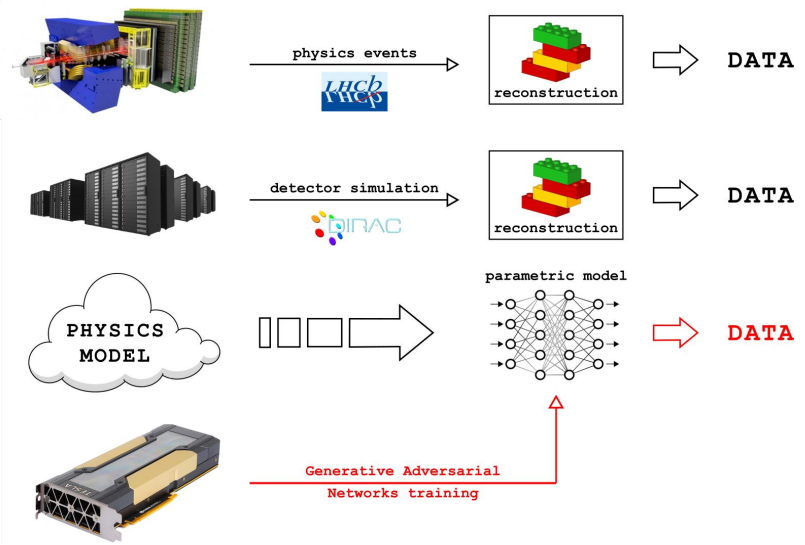
Geant4 consumes the majority of the computing resources (~90%) intended for simulation. Then, to match the demand for simulation with the available resources, the CPU-greed ***Full Simulation*** needs to be replaced with faster solutions.

When the experimental setup is reproduced only partially, we talk about ***Fast Simulation***:

- disabling specific processes (e.g. Cherenkov effect);
- simplifying the geometry (e.g. removing sub-detectors);
- re-using the underlying event (e.g. soft QCD processes).

When the simulation of the detector and the reconstruction phase is replaced with **statistical models** embedding the experimental errors introduced in the detection and reconstruction process, we talk about *Ultra-Fast* solutions:

- if the statistical models are described by parameters, this approach is named ***Parametric Simulation***;
- if non-parametric techniques are also used (such as Neural Nets), the name ***Ultra-Fast Simulation*** is preferred.

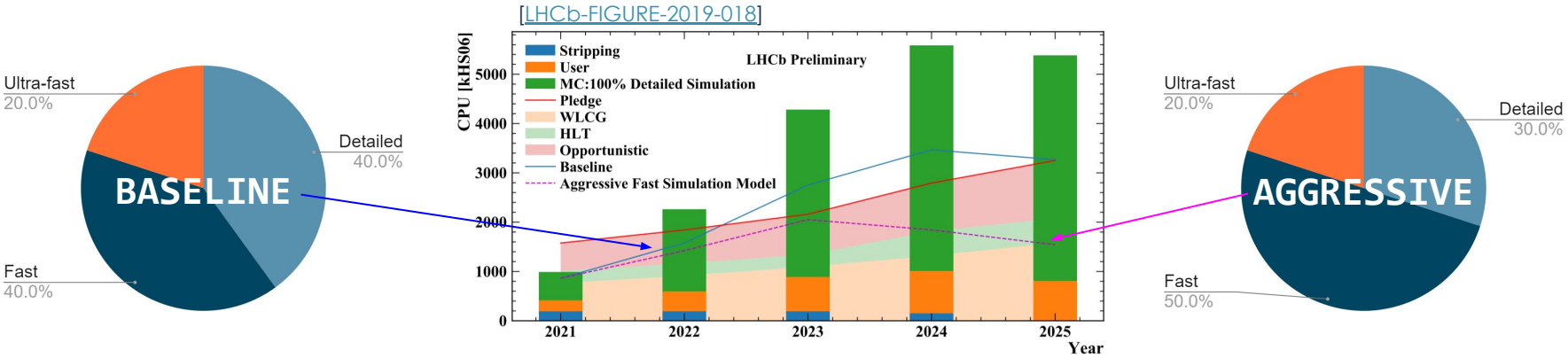


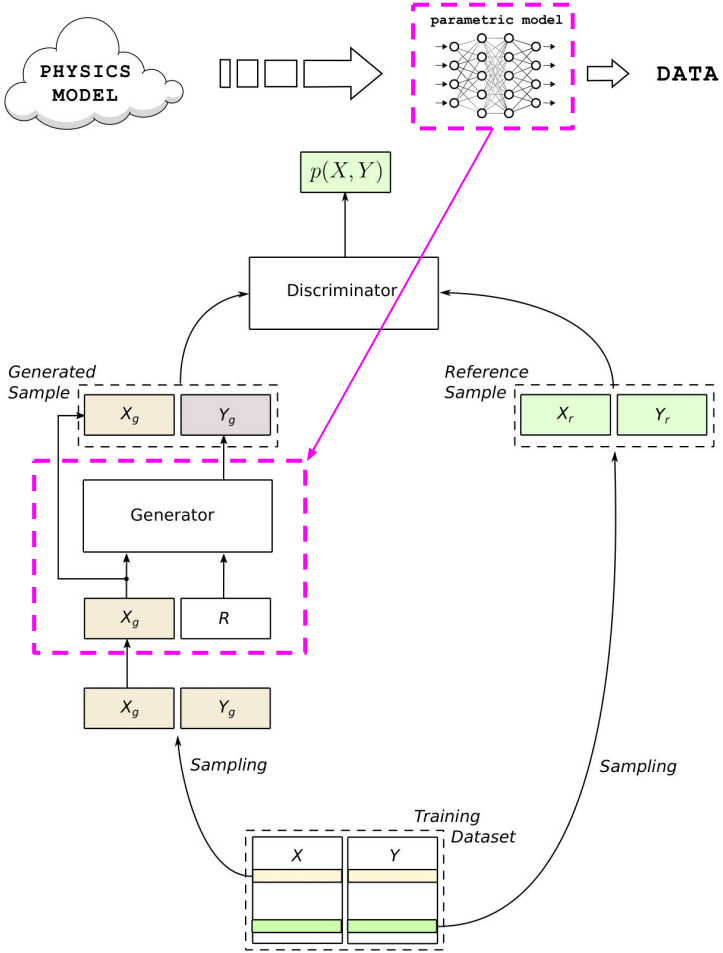
A new Computing Model for LHCb

- Projections have been obtained assuming that at least 20% of the simulated samples requested by the LHCb Collaboration can be obtained through ***Fast Simulation***
- Only 20÷30% will need ***Full Simulation*** (whole detector simulation)

> ***Ultra-Fast Simulation*** is important to succeed and
> it should cover a fraction of the use-cases as large as possible.
> But it's **not about abandoning Geant4** for all analyses,
> that will remain *necessary* to build models for the detector response.

[LHCb-FIGURE-2019-018]





Generative Adversarial Networks

Advanced algorithms such as ***Generative Adversarial Networks*** (GANs) allow to create *statistical models* able to describe effectively the LHCb detectors [CaloGAN, RichGAN].

GANs rely on the simultaneous training of two neural nets named **generator** and **discriminator**:

- the discriminator is trained to distinguish the generator output from the *reference dataset*;
- while the generator is trained to reproduce the *reference dataset* trying to fake the discriminator.

At the end of the training, the generator can be used as *statistical model* embedding directly the **high-level response** of the detector.

Boosted Decision Tree for models assessment

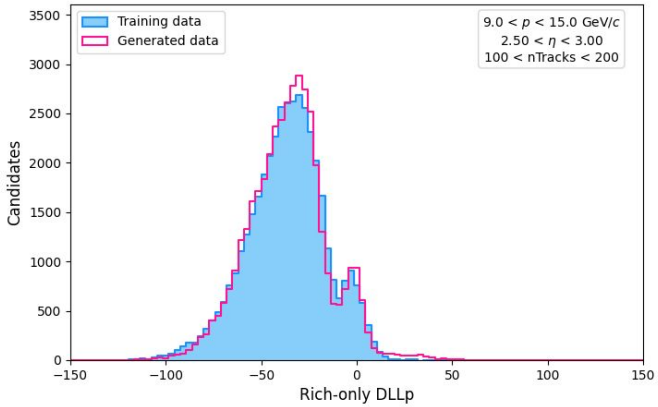
In most cases, we are interested in *statistical models* able to reproduce **accurately** the detector response, even on tails or for different ranges of *conditional variables* (such as kinematic parameters for PID).

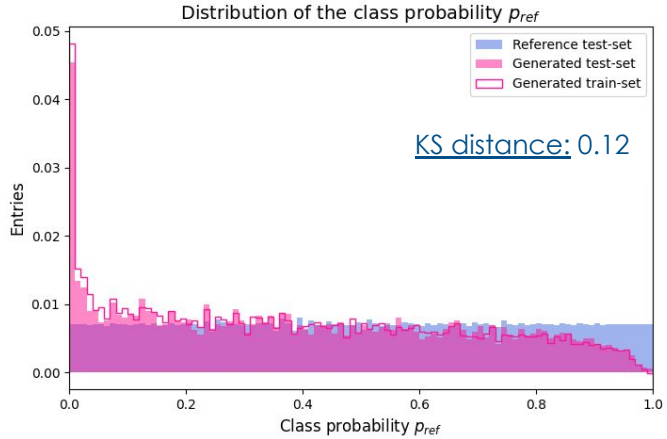
The performance of trained GANs are assessed evaluating the ability of **stacks of 4÷10 GBDTs** to distinguish *simulated* and *real data*. GBDTs define a transformation from the multidimensional space describing the detector to a 1D-space encoding the origin of input variables (*simulated* or *real*). In this space, using the **Kolmogorov-Smirnov test**, it's possible to obtain a *score* for the trained model.

It was decided to actively avoid neural networks for the validation to keep it **completely independent** from training.

For additional info on the method see arXiv:1612.07186

Pion Tracks



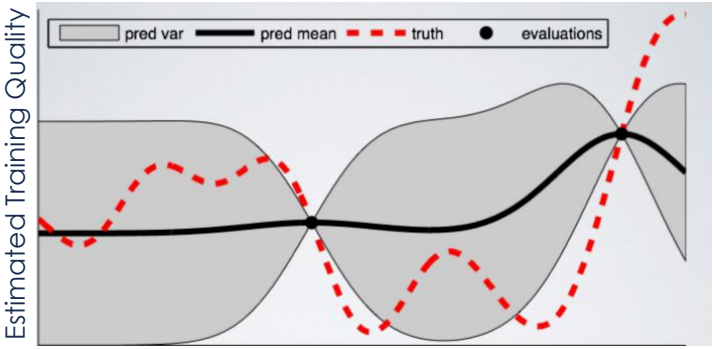


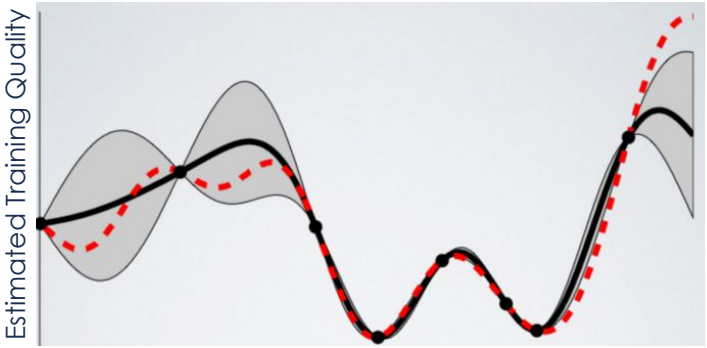
Automatic hyperparameters optimization

Neural Networks are non-parametric models since the *weights* are automatically set by a training procedure. Hence, to obtain the best possible models, one needs to **optimize the *hyperparameters***, namely the set of parameters defining the training process (such as the *learning rate* or the *batch size*).

GANs output is particularly sensitive to hyperparameters settings, requiring many subsequent training procedures to optimize the choice. **Automatic hyperparameter optimization** frameworks, such as Optuna, offer ***Bayesian search*** strategies to converge towards good models faster than what can be done with grid or random search.

Example of Bayesian optimization stolen from Will Koehrser





Distributed automatic hyperparameters optimization

The need for intensive optimization studies pointed to **OptunAPI**, a simple API designed to **distribute** an automatic hyperparameters optimization through *HTTP requests*. Each set of hyperparameters can be studied independently since the best-model research does't allows for any gradient computation.

OptunAPI is based on two modern, MIT-licensed and highly performant frameworks:

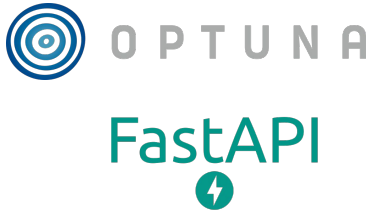
Optuna implementing bayesian search

- Organization of training "*trials*" in a simple **SQL database**
- Proposal of hyperparameter sets based on the *Tree Parzen Estimator* algorithm
- Designed to be totally independent of the training procedure

FastAPI simplifying RESTful API design.

- Highly **optimized** framework
- Based on (and fully compatible with) the **open standards** for APIs
- Design to operate with **minimal server** application implementing ASGI specifications, such as Uvicorn

OptunAPI: *key features*



OptunAPI inherits most of the modern functionalities of Optuna and FastAPI:

- **Lightweight and versatile**
  - OptunAPI is entirely written in Python and has few dependencies.
- **Easy to configure**
  - For hyperparameters sampling, OptunAPI relies on configuration files easy to set up.
- **Easy to integrate**
  - The hyperparameters values can be easily recovered from HTTP response content from the server.
- **Easy parallelization**
  - Different machines can run the hyperparameters study in parallel, centrally coordinated by the server.
- **Efficient optimization algorithms**
  - The optimization task is headed by Optuna and its *state-of-the-art* algorithms.
- **Quick visualization for study analysis**
  - [*todo*] OptunAPI provides a set of reports to monitor the status of the hyperparameters study.

OptunAPI for Cloud Computing services integration

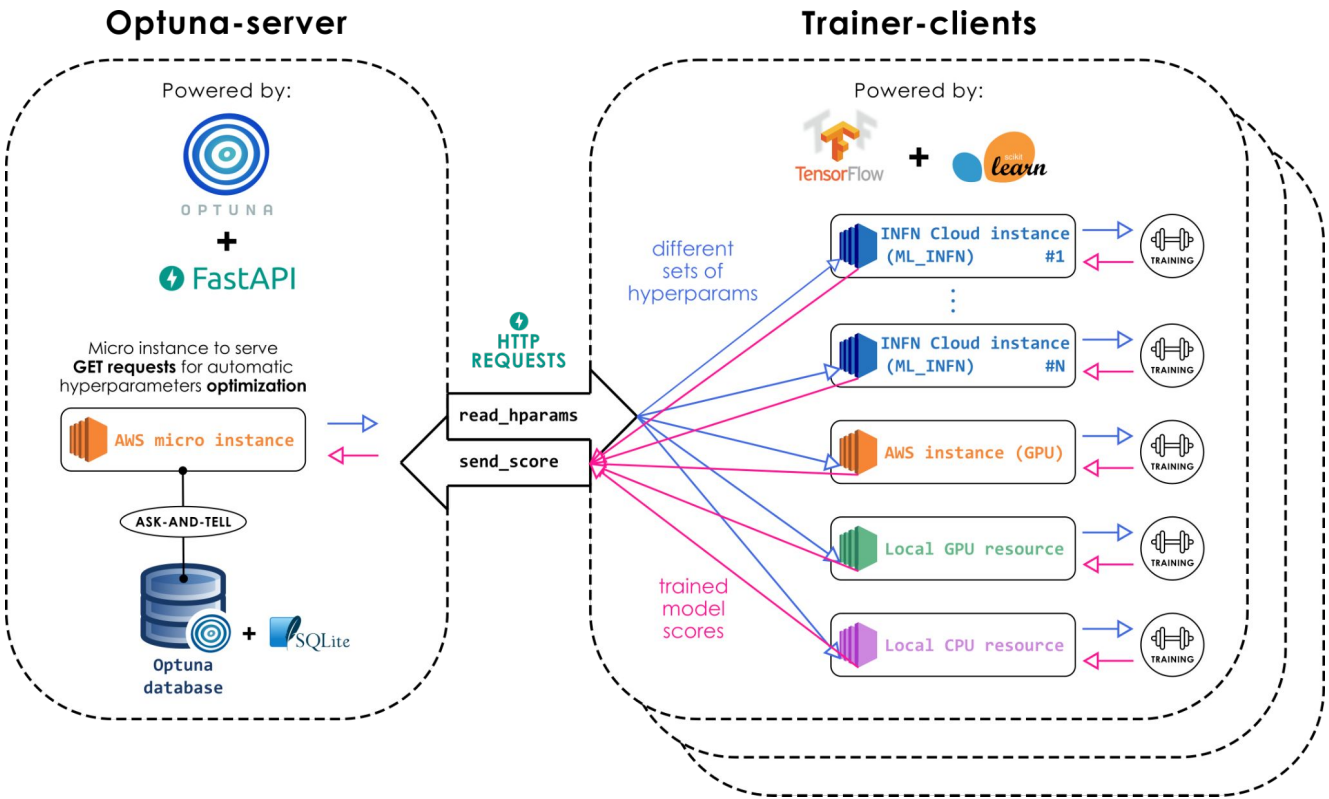
The training of large neural networks clearly benefits from **Cloud Computing**, requiring a large amount of resources for a limited amount of time (*burst payloads*).

In particular, the exploration of the hyperparameter space of a training procedure can be effectively performed by training the algorithm **in parallel** on different machines, with *different architectures* and/or *hardware accelerators*, with *limited connectivity*, among the working nodes.

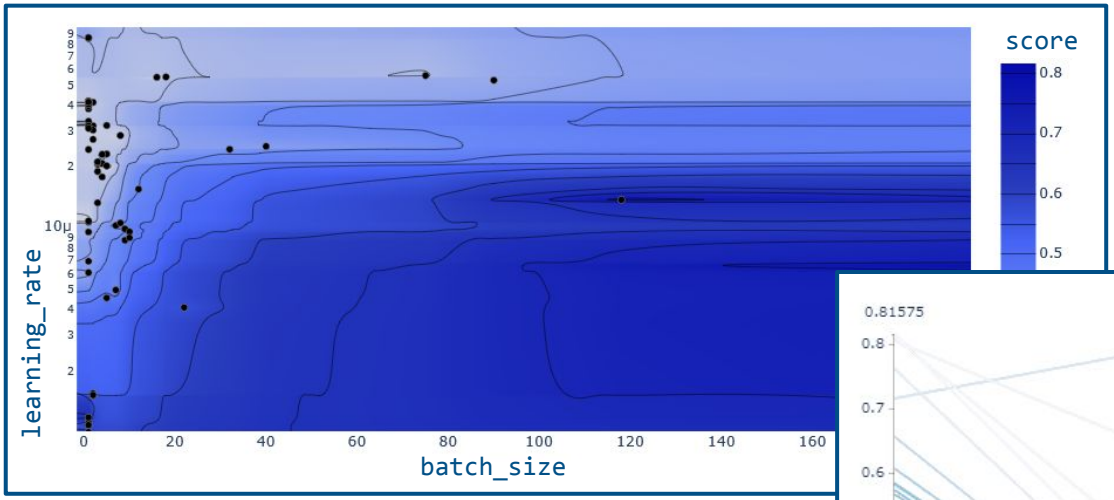
As a real use-case, the LHCb GANs for Fast Simulation were trained using resources made available by **INFN Cloud** through ML_INFN, Amazon Web Services (AWS) through the **CloudBank EU** initiative, plus several private nodes at CERN and at INFN-Firenze. All contributing to construction of a single *trials* DB via plain *HTTP requests*.

When scaling to world-wide networks, cyber-security becomes a concern. We protect the OptunAPI server behind an SSH Bastion accessed by the **authorized training clients** via SSH tunnels.

Sketch of the OptunAPI client-server system

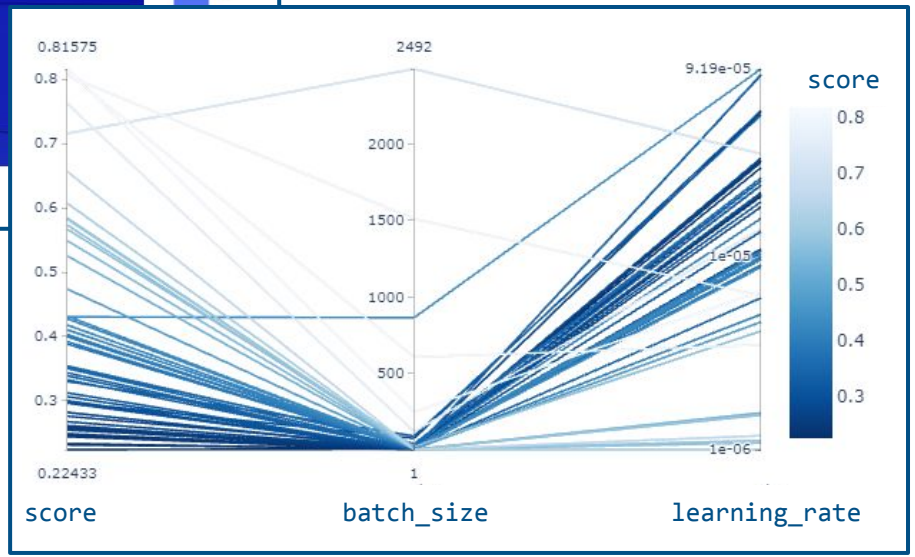


Results powered by OptunAPI [1/2]

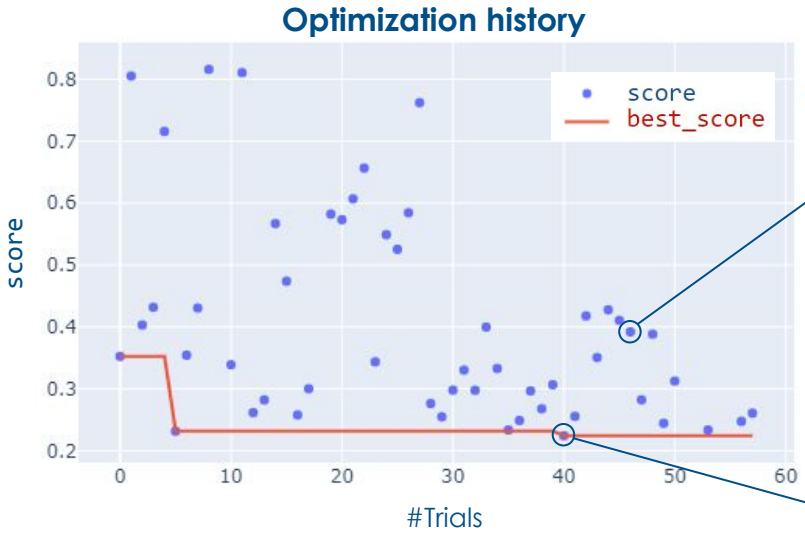


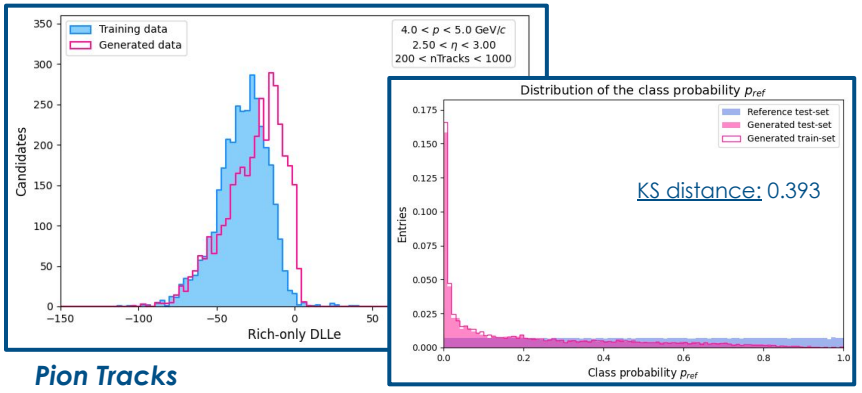
Map of the hyperparameters space

Trials values

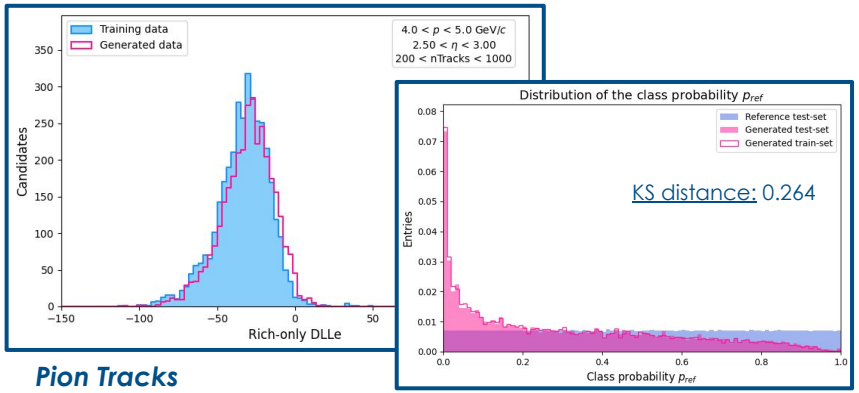


Results powered by OptunAPI [2/2]





Pion Tracks



Pion Tracks

Summary and outlook

- **Ultra-Fast Simulation** plays a *key-role* in matching the future simulation demands of the LHCb experiment
- **GANs** are effective in producing statistical models describing the detector response
- Several local and cloud resources have been used (opportunistically) to explore the large hyperparameter space contributing to the huge effort to push the **performance** of the trained generators
- We propose **OptunAPI**, combining *state-of-the-art* frameworks for Bayesian optimization and RESTful APIs

We will keep expanding the API to provide access to more features of the Optuna package!

Backup

OptunAPI: *key components*

OptunAPI relies on three main components: (1) `Study` and `Trial` objects from Optuna, (2) Optuna's *Ask-and-Tell* interface and (3) *HTTP requests* to map the hyperparameters space.

Study and Trial

A ***study*** corresponds to an optimization task, i.e., a set of *trials*. Started a *study*, different machines can access it asking for a new set of hyperparameters or sending back the training score obtained by the previous one.

A ***trial*** allows to prepare a particular set of hyperparameters and evaluate its capability of optimizing a defined score function. OptunAPI provides a very easy interface to define hyperparameters, scanning regions and sampling mode.

Ask-and-Tell Interface

The Optuna's ***Ask-and-Tell*** interface provides a more flexible interface for hyperparameter optimization: the ***ask* instance** creates a trial that can sample hyperparameters, while the ***tell* instance** finishes the trial by passing `trial` and a score value.

In OptunAPI, machines asking for sets of hyperparameter receive trials resulting from *ask* instances. Then, evaluated the scores, the same machines send a new request allowing to close the running trials with the corresponding *tell* instances.

HTTP Requests

OptunAPI provides a simple Python module to run a server able to centrally manage the optimization studies. It is equipped with a set of ***path operation functions*** relying on the FastAPI ecosystem:

- `[GET] read_hparams`
  - calls an *ask* instance and provides a set of hyperparameters
- `[GET] send_score`
  - calls a *tell* instance to finish the trial with the evaluated score

If the server is opened to the public Internet, OptunAPI provides a simplified interface to secure *HTTP requests* through **SSH authentication** based on sshtunnel.