To Emulate or to Differentiate? A Delphes3 optimization study

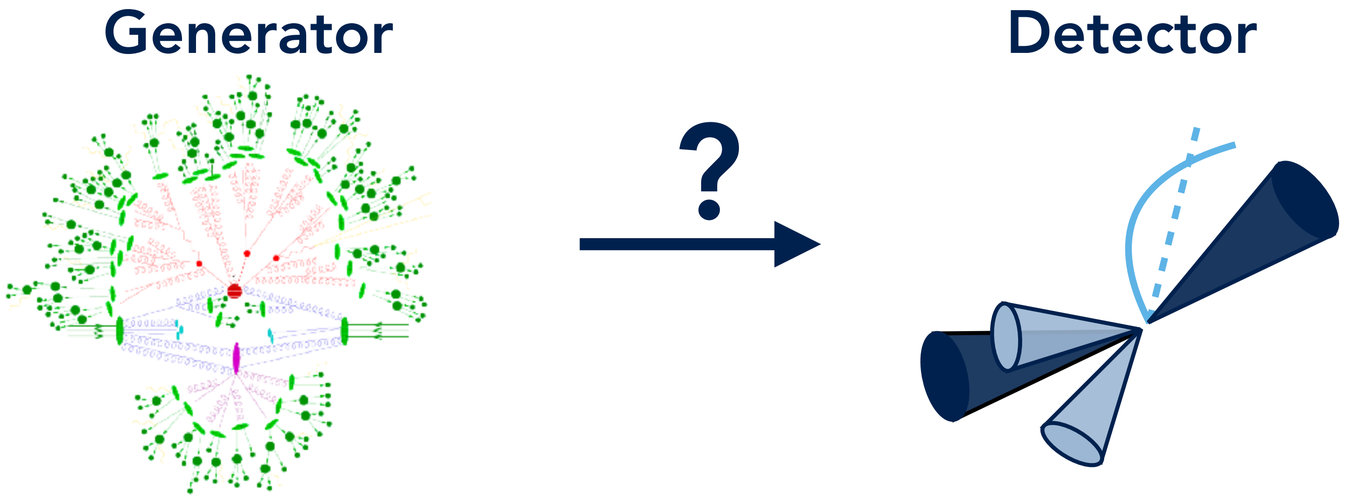
Luigi Favaro, Andrea Giammanco

15.09.2026

ML4Jets26

Vienna

End-to-end simulation



Motivations:

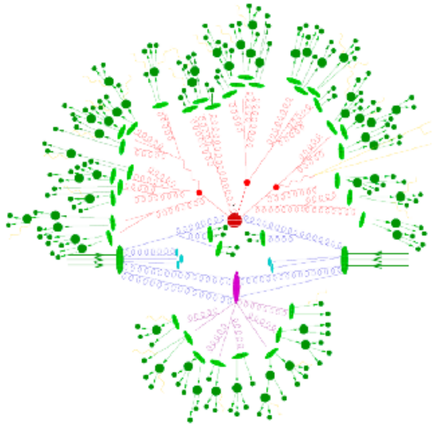
Compact and accurate representation of detectors

Publicly available to the community

Simulation and storage cost of full simulations

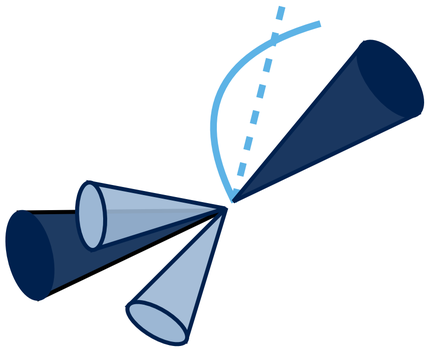
End-to-end simulation

Generator



?

Detector



Motivations:

Compact and accurate representation of detectors

Publicly available to the community

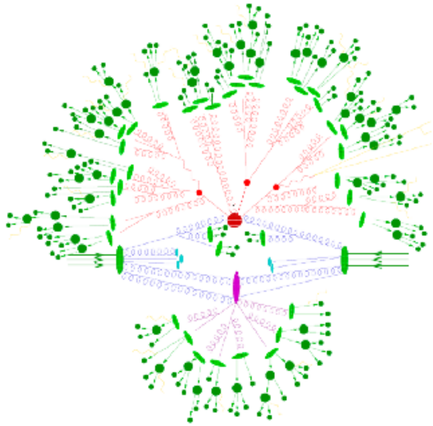
Simulation and storage cost of full simulations

How to directly map from gen. to reco. objects?

(electrons, muons, photons, jets)

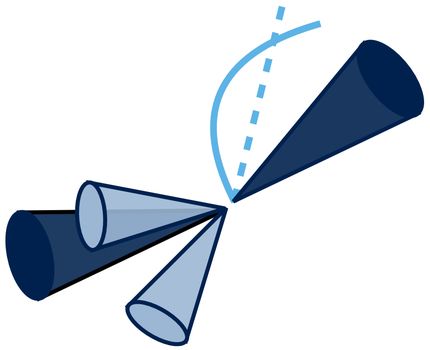
End-to-end simulation

Generator



?

Detector



Motivations:

Compact and accurate representation of detectors

Publicly available to the community

Simulation and storage cost of full simulations

How to directly map from gen. to reco. objects?

(electrons, muons, photons, jets)

Delphes3 is a great example of modularised and simple detector modelling

To emulate or to differentiate?

Setting:

Access to paired data, CMS open data full sim Drell-Yan + 3 jets

Distance-based matching between gen. and reco. objects

To emulate or to differentiate?

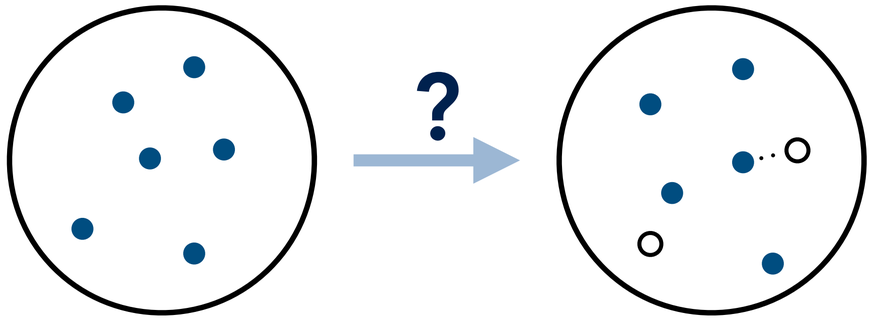
Setting:

Access to paired data, CMS open data full sim Drell-Yan + 3 jets

Distance-based matching between gen. and reco. objects

Matching:

- distance-based matching: $\Delta R < R_{\text{max}}$
- p_T matching: $p'_T/p_T < r$
- exclusive selection



Data allows to use any other matching scheme

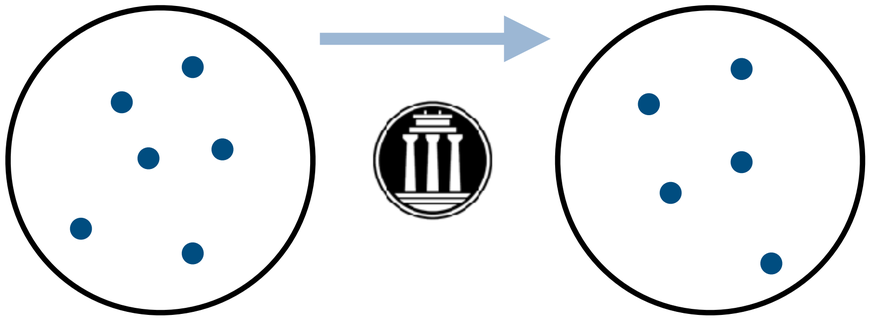
To emulate or to differentiate?

Setting:

Access to paired data, CMS open data full sim Drell-Yan + 3 jets

Distance-based matching between gen. and reco. objects

Differentiable simulator:



Physics model
with access to gradients $\partial\mathcal{L}/\partial$

Turn Delphes into a differentiable simulator*:

- no hand tuning of detector cards
- directly differentiate through the simulator

*similar work done with Parnassus in 2608.22873, A. Elabd et al.

To emulate or to differentiate?

Setting:

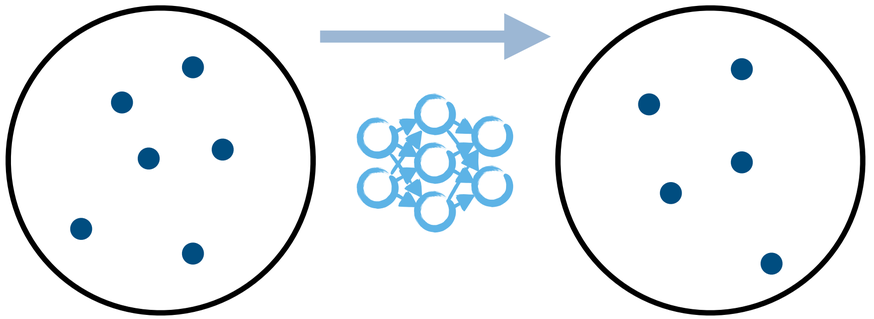
Access to paired data, CMS open data full sim Drell-Yan + 3 jets

Distance-based matching between gen. and reco. objects

Generative network trained on paired data:

- removes limitations from simple detector cards

Generative emulator:



Neural network encodes $p(x_{\text{reco}} \mid x_{\text{gen}})$

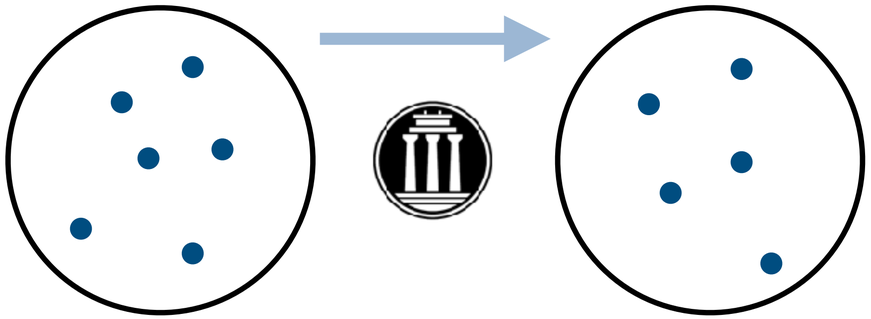
To emulate or to differentiate?

Setting:

Access to paired data, CMS open data full sim Drell-Yan + 3 jets

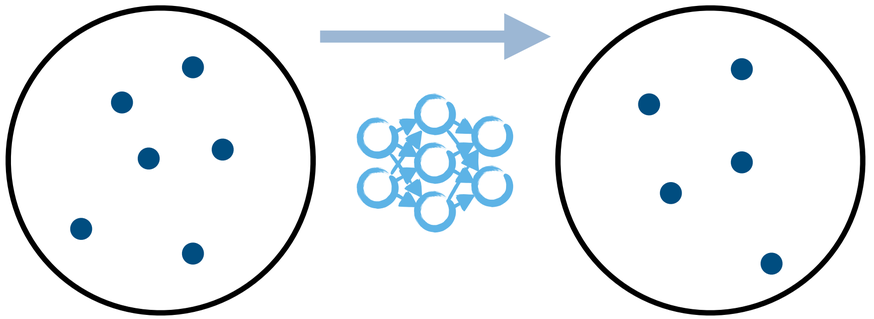
Distance-based matching between gen. and reco. objects

Differentiable simulator:



Physics model
with access to gradients $\partial\mathcal{L}/\partial$

Generative emulator:



Neural network
encodes $p(x_{\text{reco}} \mid x_{\text{gen}})$

Differentiable Delphes

HEPMC data

Step 1: reimplement Delphes in PyTorch

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Tracker:
apply smearing and efficiency to

- charged hadrons
- electrons
- muons

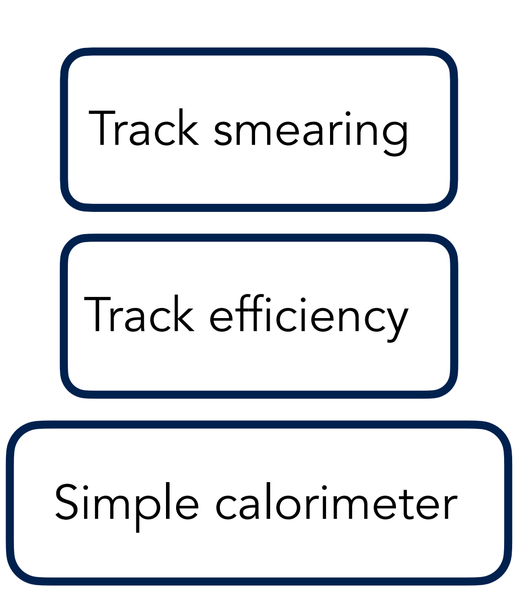
Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Tracker:
apply smearing and efficiency to

- charged hadrons
- electrons
- muons

Calorimeter:
define and apply smearing to energy towers in ECal and HCal



Differentiable Delphes

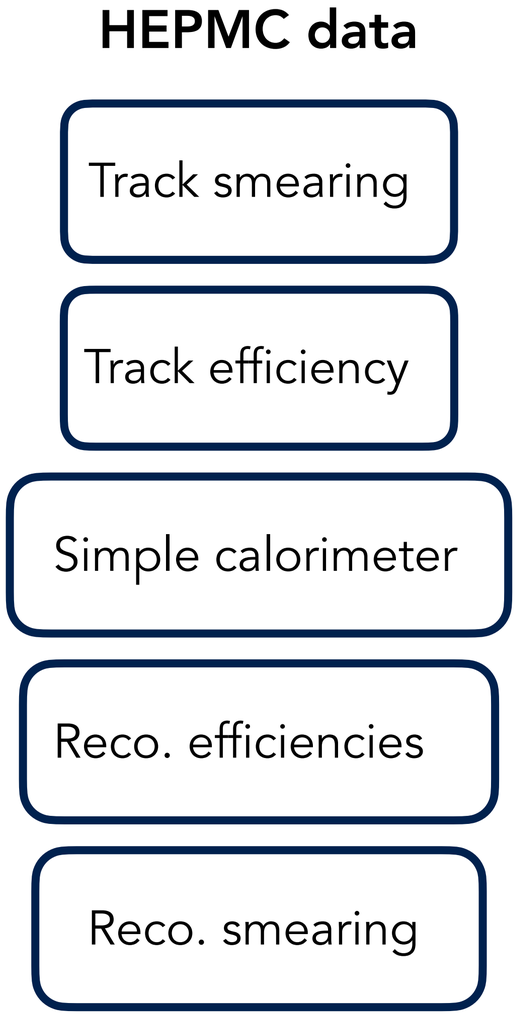
Step 1: reimplement Delphes in PyTorch

Tracker:
apply smearing and efficiency to

- charged hadrons
- electrons
- muons

Calorimeter:
define and apply smearing to energy towers in ECal and HCal

Efficiency and isolation for electrons, photons, and muons



Differentiable Delphes

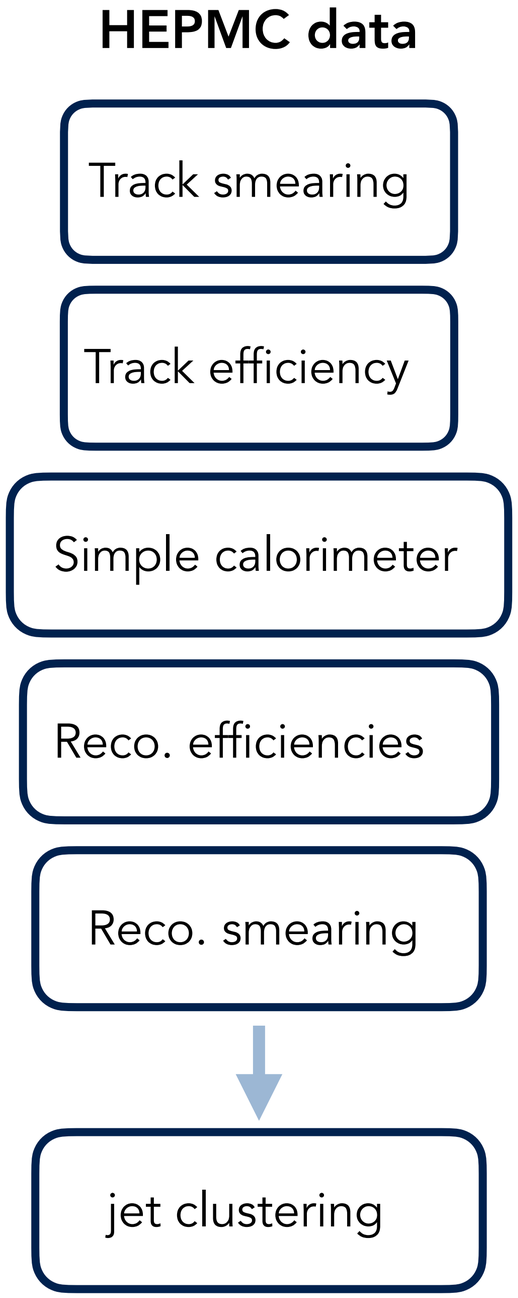
Step 1: reimplement Delphes in PyTorch

Tracker:
apply smearing and efficiency to

- charged hadrons
- electrons
- muons

Calorimeter:
define and apply smearing to energy towers in ECal and HCal

Efficiency and isolation for electrons, photons, and muons



Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Smearing functional form: Log-Normal distribution

$$\sigma_{\text{res}}(p_T, \eta) = \sqrt{\theta_0(\eta)^2 + \theta_1(\eta)^2 p_T^2}$$

(total of 18 smearing parameters)

Delphes default (pre-fit)

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Smearing functional form: Log-Normal distribution

$$\sigma_{\text{res}}(p_T, \eta) = \sqrt{\theta_0(\eta)^2 + \theta_1(\eta)^2 p_T^2}$$

(total of 18 smearing parameters)

Delphes default (pre-fit)

Example momentum smearing

```
# resolution formula for muons
set ResolutionFormula {                    (abs(eta) <= 0.5) * (pt > 0.1) * sqrt(0.01^2 + pt^2*1.0e-4^2) +
                       (abs(eta) > 0.5 && abs(eta) <= 1.5) * (pt > 0.1) * sqrt(0.015^2 + pt^2*1.5e-4^2) +
                       (abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 0.1) * sqrt(0.025^2 + pt^2*3.5e-4^2)}
}
```

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Smearing functional form: Log-Normal distribution

$$\sigma_{\text{res}}(p_T, \eta) = \sqrt{\theta_0(\eta)^2 + \theta_1(\eta)^2 p_T^2}$$

(total of 18 smearing parameters)

Delphes post-fit

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Smearing functional form: Log-Normal distribution

$$\sigma_{\text{res}}(p_T, \eta) = \sqrt{\theta_0(\eta)^2 + \theta_1(\eta)^2 p_T^2}$$

(total of 18 smearing parameters)

Delphes post-fit

Example momentum smearing

```
# resolution formula for muons
set ResolutionFormula {
              (abs(eta) <= 0.5) * (pt > 0.1) * sqrt(0.0088232^2 + pt^2*0.00012419^2) +
              (abs(eta) > 0.5 && abs(eta) <= 1.5) * (pt > 0.1) * sqrt(0.014642^2 + pt^2*0.00014692^2) +
              (abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 0.1) * sqrt(0.022765^2 + pt^2*0.00036676^2)}
}
```

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Efficiency functional form: surviving probability

$$q(x;\theta) = \begin{cases} x & \text{if} \quad a < \theta \\ 0 & \text{if} \quad a > \theta \end{cases} \qquad \text{where} \quad a \sim \mathcal{U}(0,1), \theta \in [0,1)$$

Delphes default (pre-fit)

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Efficiency functional form: surviving probability

$$q(x;\theta) = \begin{cases} x & \text{if} \quad a < \theta \\ 0 & \text{if} \quad a > \theta \end{cases} \qquad \text{where} \quad a \sim \mathcal{U}(0,1), \theta \in [0,1)$$

Delphes default (pre-fit)

Example efficiencies

```
# efficiency formula for electrons
set EfficiencyFormula {                                          (pt <= 10.0) * (0.00) +
                                        (abs(eta) <= 1.5) * (pt > 10.0)  * (0.95) +
                       (abs(eta) > 1.5 && abs(eta) <= 2.5) * (pt > 10.0)  * (0.85) +
                       (abs(eta) > 2.5)                                   * (0.00)}
}
```

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Efficiency functional form: surviving probability

$$q(x;\theta,p_T) = \theta_0(1+\theta_1\log(p_T))$$

2 learnable parameters (post-fit)

Free to use more complex cards

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Efficiency functional form: surviving probability

$$q(x;\theta,p_T) = \theta_0(1+\theta_1\log(p_T))$$

2 learnable parameters (post-fit)

Free to use more complex cards

Example efficiencies

```
set EfficiencyFormula {
  (abs(eta) <= 0.5) * (0.98154*(1.0 + (+0.008235)*log(pt/50.0))) + \
  (abs(eta) > 0.5 && abs(eta) <= 1.0) * (0.98485*(1.0 + (+0.008649)*log(pt/50.0))) + \
  (abs(eta) > 1.0 && abs(eta) <= 1.5) * (0.97318*(1.0 + (+0.046652)*log(pt/50.0))) + \
  (abs(eta) > 1.5 && abs(eta) <= 2.0) * (0.97574*(1.0 + (+0.034300)*log(pt/50.0))) + \
  (abs(eta) > 2.0 && abs(eta) <= 2.5) * (0.96812*(1.0 + (+0.069869)*log(pt/50.0))) + \
  (abs(eta) > 2.5 && abs(eta) <= 5) * ((pt > 0.5) * (0.92506 * (1.0 + (-0.042945) * log(pt/50))
  (abs(eta) > 5) * (0.00)
}
```

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Step 3: optimization

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Step 3: optimization

Efficiency: Bernoulli likelihood

$$\mathscr{L}_{\text{eff}}(\theta) = \left\langle x \log q(x;\theta) + (1-x)\log(1-q(x;\theta)) \right\rangle_{x\sim p_{\text{data}}}$$

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Step 3: optimization

Efficiency: Bernoulli likelihood

$$\mathscr{L}_{\text{eff}}(\theta) = \left\langle x \log q(x;\theta) + (1-x)\log(1-q(x;\theta)) \right\rangle_{x\sim p_{\text{data}}}$$

Smearing: Log-Normal likelihood

$$\mathscr{L}_{\text{smear}}(\theta) = \left\langle \frac{1}{2}\log(2\pi b^2) + \frac{(\log(x'(\theta)/x) + b^2/2)^2}{2b^2} \right\rangle_{x\sim p_{\text{data}}} \qquad b^2 = \log(1+\sigma^2_{\text{res}}(\theta))$$

Differentiable Delphes

Step 1: reimplement Delphes in PyTorch

Step 2: block parametrizations

Step 3: optimization

Efficiency: Bernoulli likelihood

$$\mathscr{L}_{\text{eff}}(\theta) = \left\langle x \log q(x;\theta) + (1-x)\log(1-q(x;\theta)) \right\rangle_{x\sim p_{\text{data}}}$$

Smearing: Log-Normal likelihood

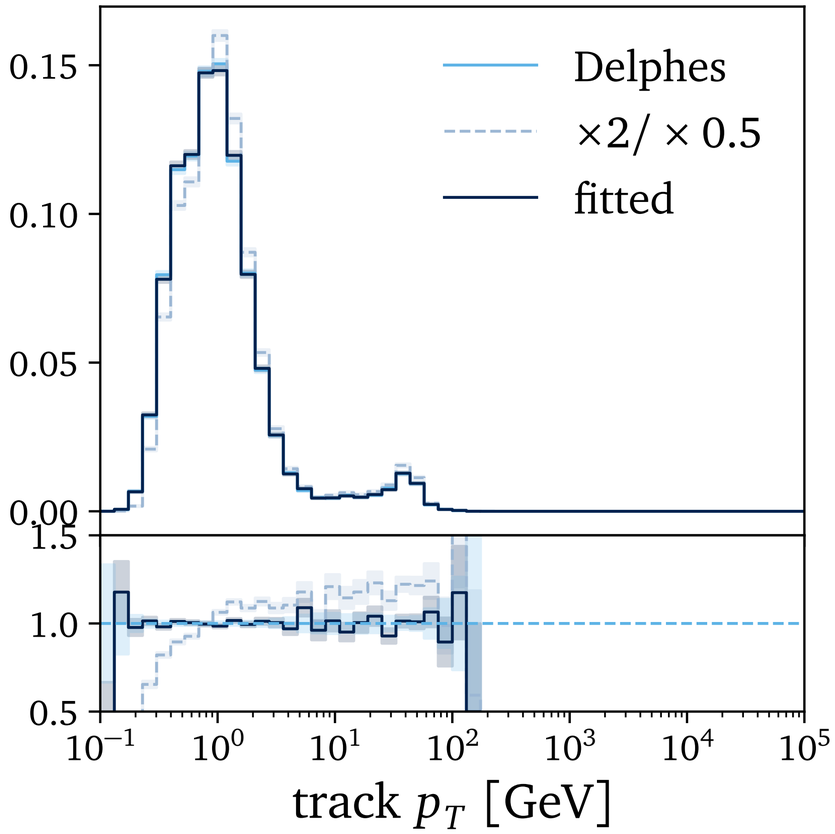
$$\mathscr{L}_{\text{smear}}(\theta) = \left\langle \frac{1}{2}\log(2\pi b^2) + \frac{(\log(x'(\theta)/x) + b^2/2)^2}{2b^2} \right\rangle_{x\sim p_{\text{data}}} \qquad b^2 = \log(1+\sigma^2_{\text{res}}(\theta))$$

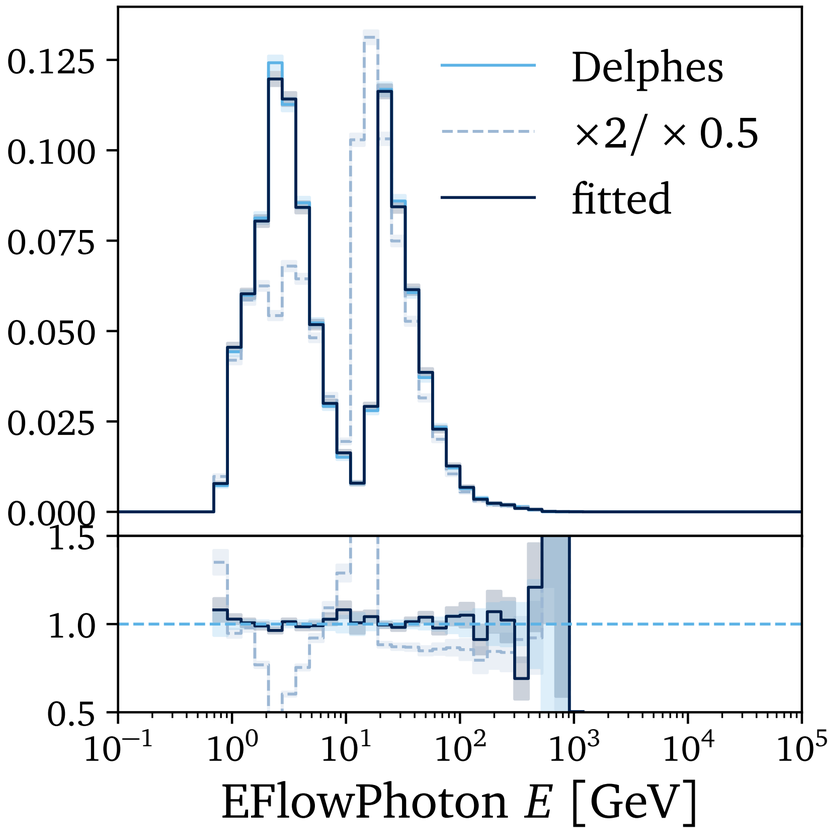
The optimization is a maximum likelihood estimate (MLE): negative log-likelihood loss*

$$\mathscr{L}_{\text{joint}} = \sum_{\text{species}} \mathscr{L}_{\text{eff}} + \mathscr{L}_{\text{smear}}$$

*when intermediate states are available

Delphes closure tests



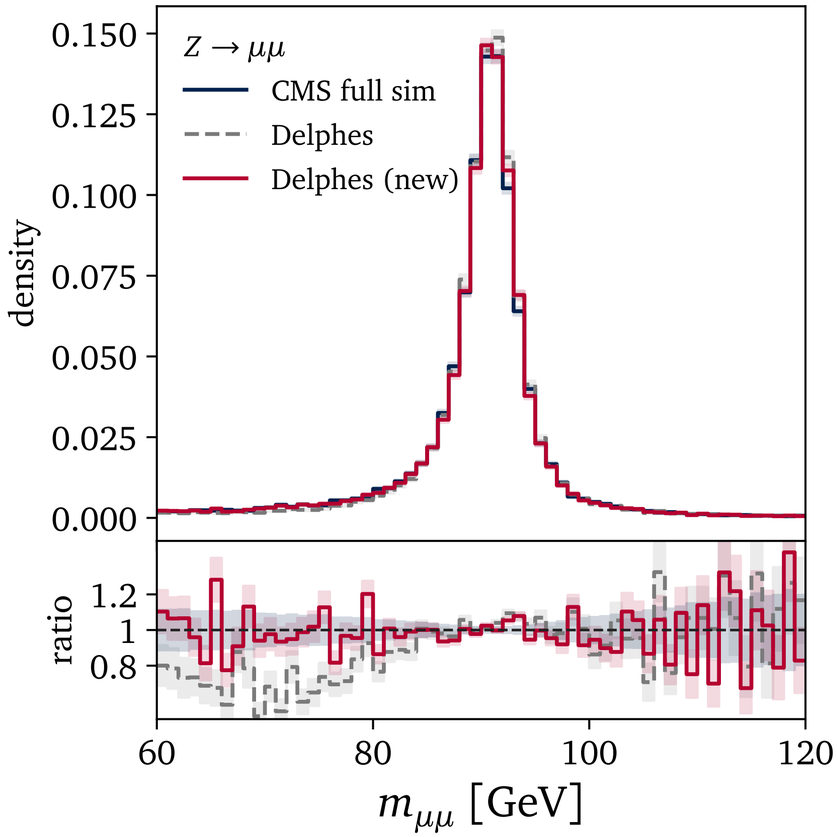


Applied to DY events generated with MG5+Pythia+Delphes:

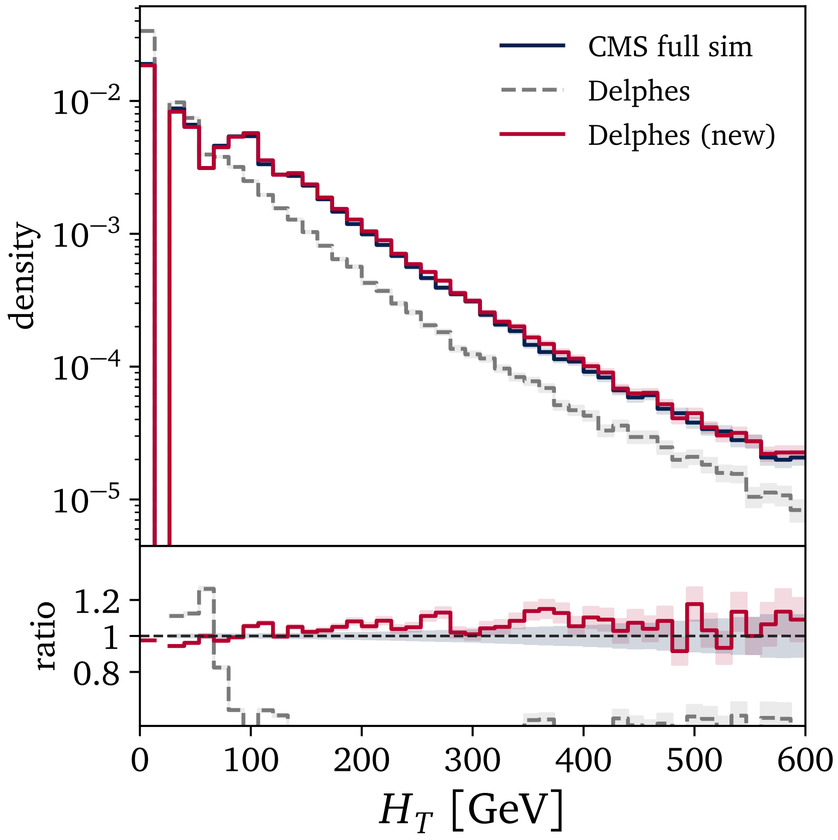
we correctly recover the original values of the Delphes card

CMS full sim DY

Downstream observables



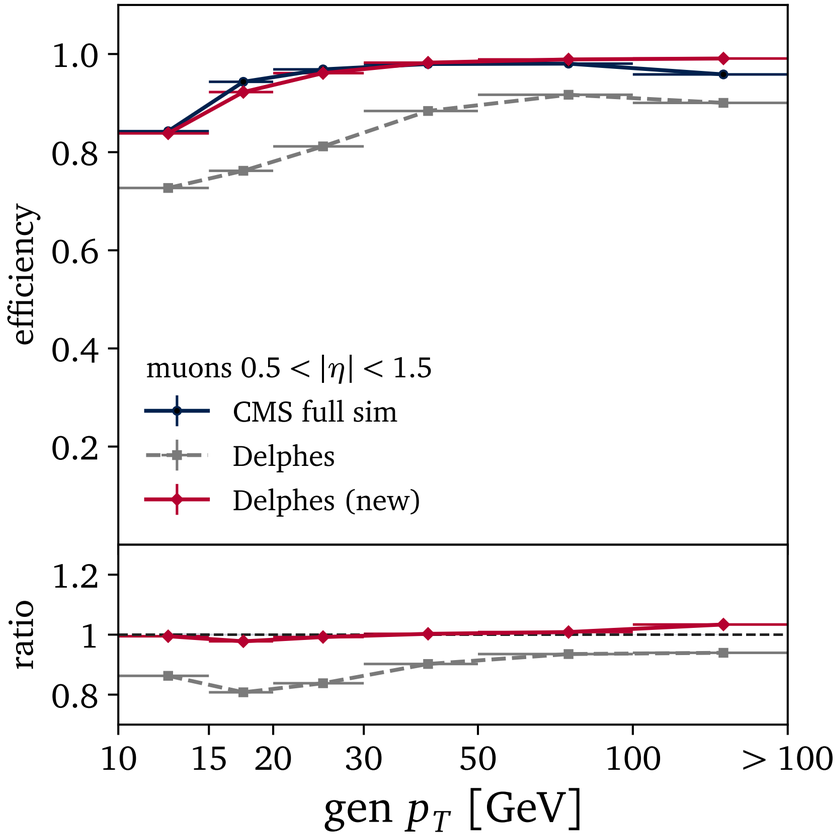
dimuon mass



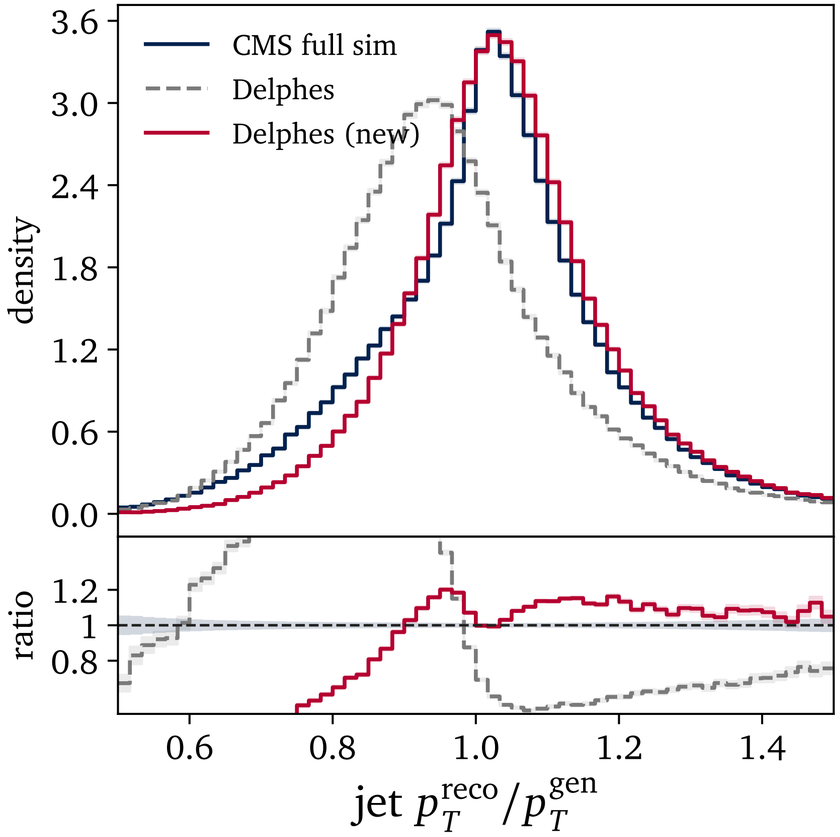
jets transverse momentum

CMS full sim DY

Downstream observables



muon efficiency



jet smearing

Limitations due to the simple card functional form

Generative Delphes

Emulate the detector response through a generative network

Generator-level Inputs:

- for each generated object: $\{\log p_T, \eta, \sin\phi, \cos\phi, m\}$
- object type: electron, muon, photon, jet
- event-level input: number of objects, MET, H_T

Generative Delphes

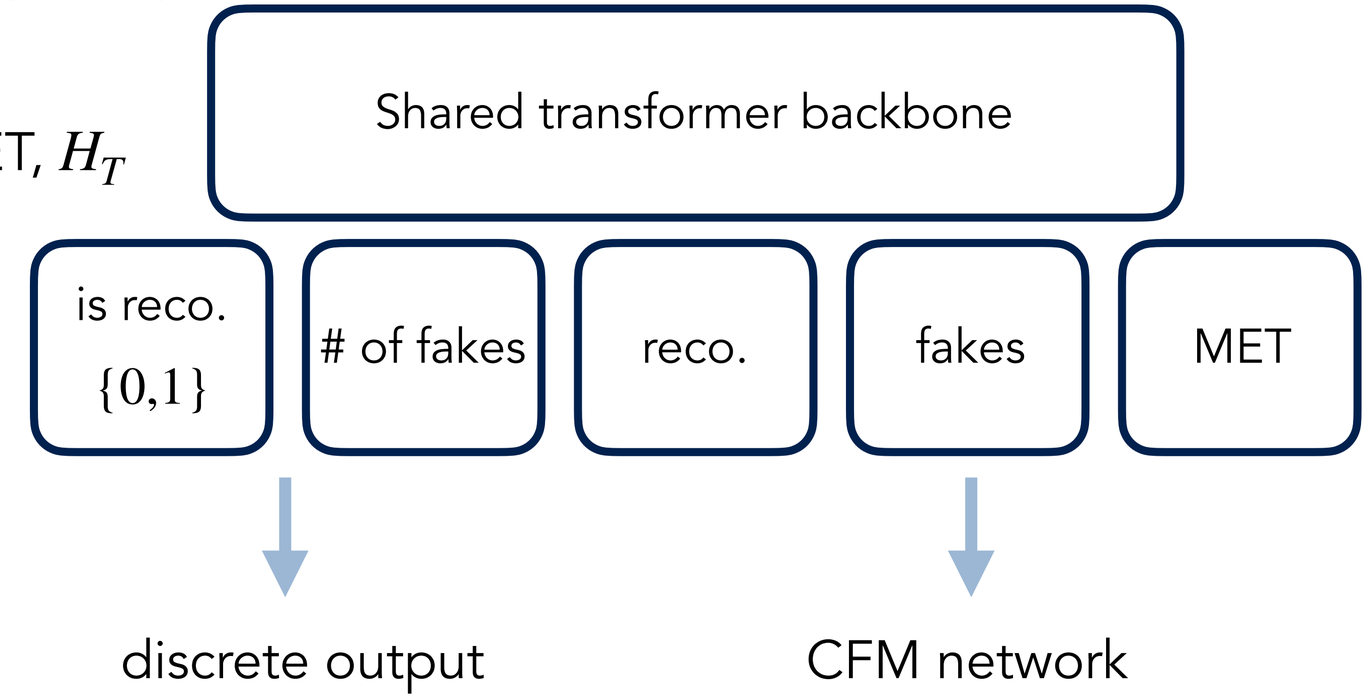
Emulate the detector response through a generative network

Generator-level Inputs:

- for each generated object: $\{\log p_T, \eta, \sin\phi, \cos\phi, m\}$
- object type: electron, muon, photon, jet
- event-level input: number of objects, MET, H_T

Generated quantities:

- # of unmatched and reco. objects
- kinematics: $\{\log p_T, \eta, \sin\phi, \cos\phi, m\}$ of reco., fakes, and MET



Generative Delphes

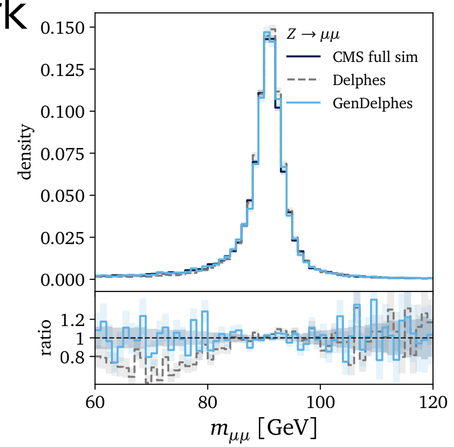
Emulate the detector response through a generative network

Generator-level Inputs:

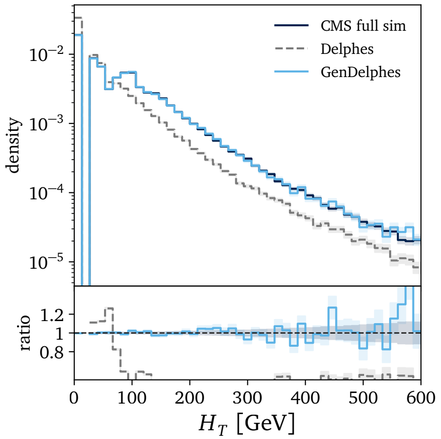
- for each generated object: $\{\log p_T, \eta, \sin\phi, \cos\phi, m\}$
- object type: electron, muon, photon, jet
- event-level input: number of objects, MET, H_T

Generated quantities:

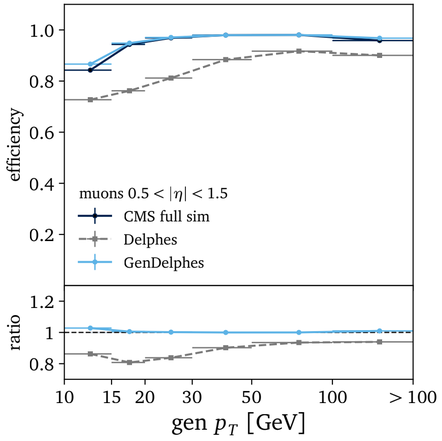
- # of unmatched and reco. objects
- kinematics: $\{\log p_T, \eta, \sin\phi, \cos\phi, m\}$ of reco., fakes, and MET



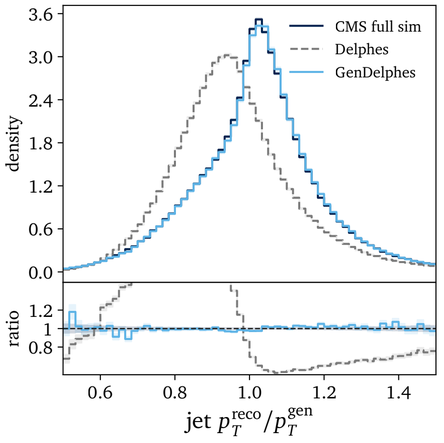
dimuon mass



jets transverse momentum



muon efficiency



jet smearing

Generative Delphes

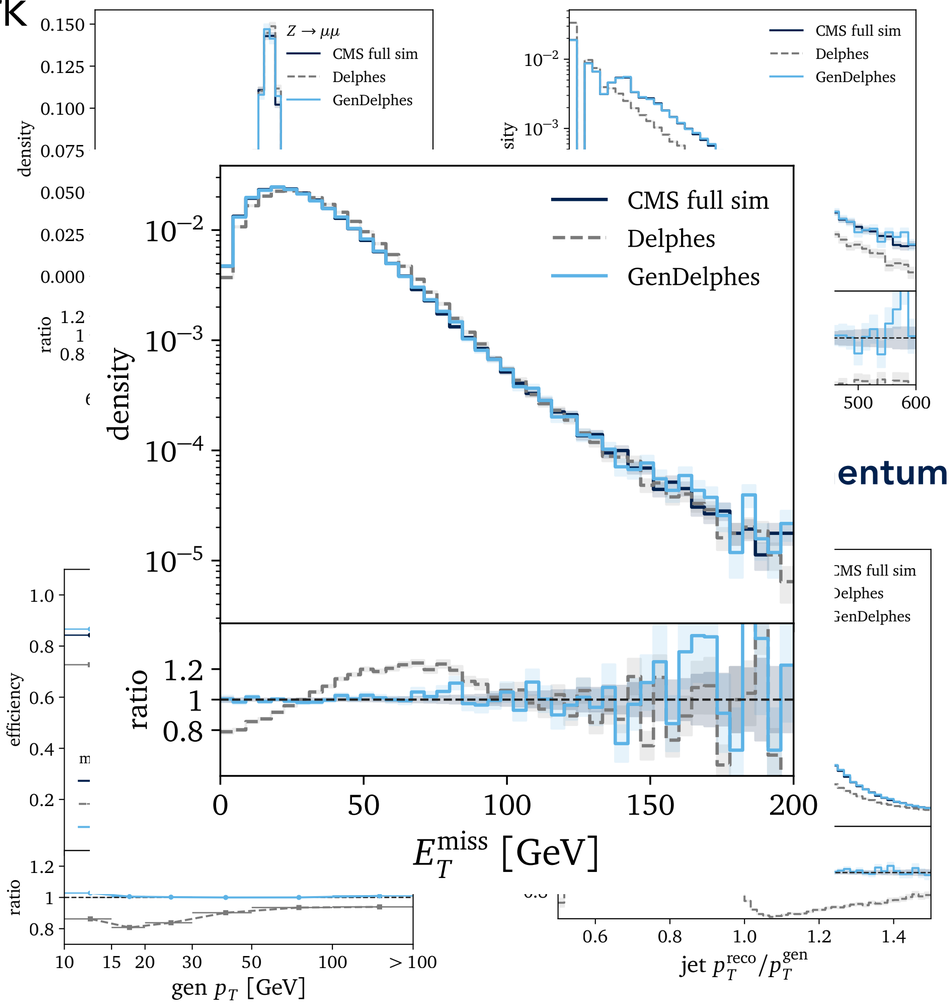
Emulate the detector response through a generative network

Generator-level Inputs:

- for each generated object: $\{\log p_T, \eta, \sin\phi, \cos\phi, m\}$
- object type: electron, muon, photon, jet
- event-level input: number of objects, MET, H_T

Generated quantities:

- # of unmatched and reco. objects
- kinematics: $\{\log p_T, \eta, \sin\phi, \cos\phi, m\}$
  of reco., fakes, and MET



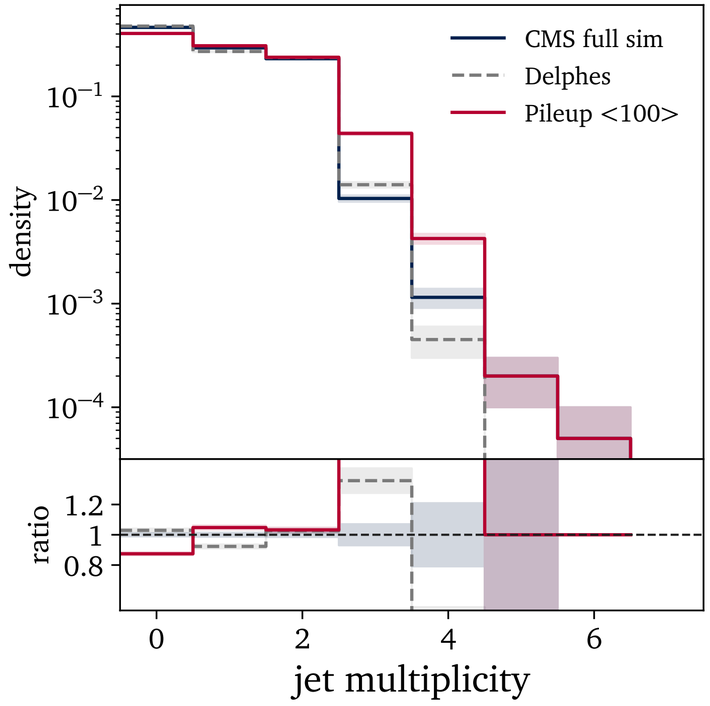
entum

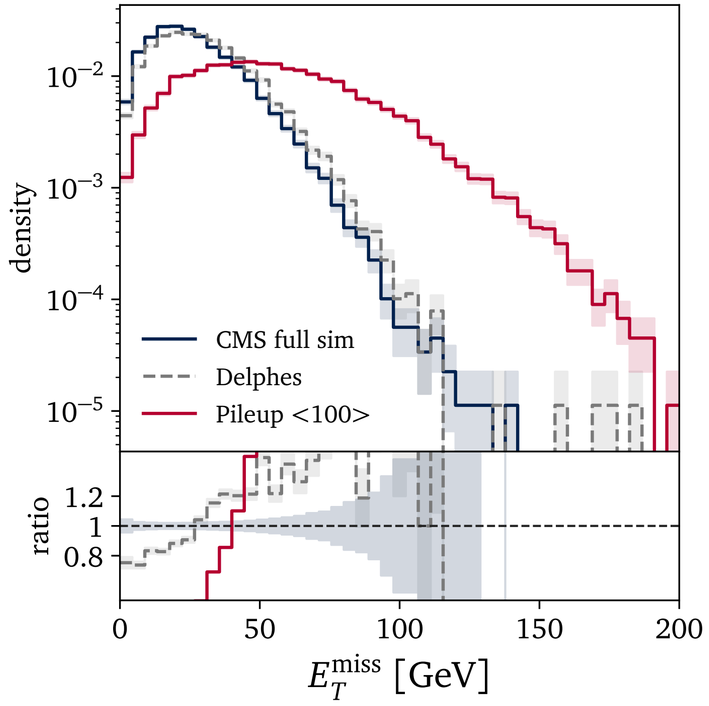
muon efficiency

jet smearing

Pileup

Delphes has a pileup model ⟶ it can be varied (set by $\langle\mu\rangle$)





A generative network uses the pileup seen in the training data ($< 25 >$)

Conclusions/Outlook

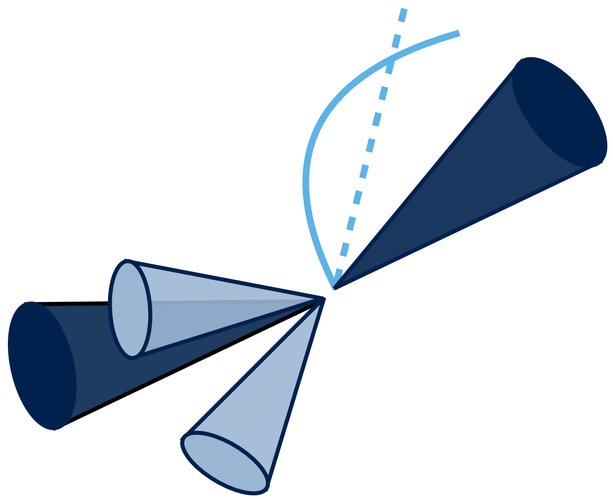
Emulation and differentiation are two approaches to solve the same problem

Differentiate:

- make use of already efficient simulators
- simple and portable: new detector cards can be effortlessly deployed
- interpretable: explicit smearing formulas and efficiencies
- fastest

Emulate:

- best agreement with CMS full sim
- still fast (on GPU)



Conclusions/Outlook

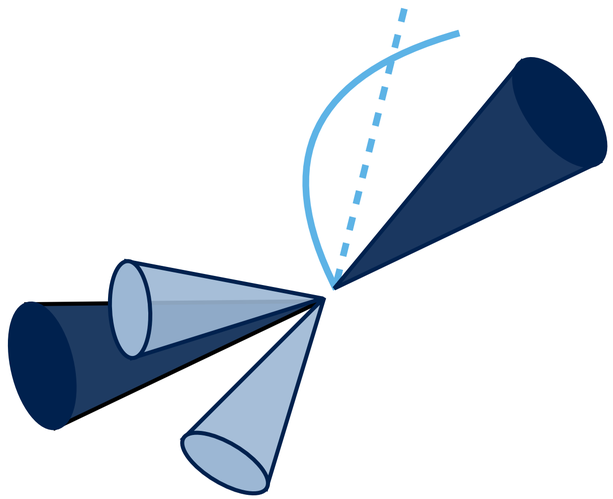
Emulation and differentiation are two approaches to solve the same problem

Differentiate:

- make use of already efficient simulators
- simple and portable: new detector cards can be effortlessly deployed
- interpretable: explicit smearing formulas and efficiencies
- fastest

Emulate:

- best agreement with CMS full sim
- still fast (on GPU)



Good reconstruction of downstream DY observables in both case

Pileup is treated differently

Conclusions/Outlook

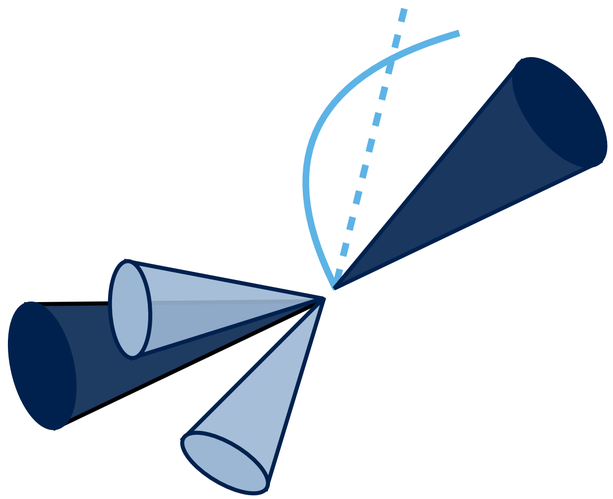
Emulation and differentiation are two approaches to solve the same problem

Differentiate:

- make use of already efficient simulators
- simple and portable: new detector cards can be effortlessly deployed
- interpretable: explicit smearing formulas and efficiencies
- fastest

Emulate:

- best agreement with CMS full sim
- still fast (on GPU)



Good reconstruction of downstream DY observables in both case

Pileup is treated differently

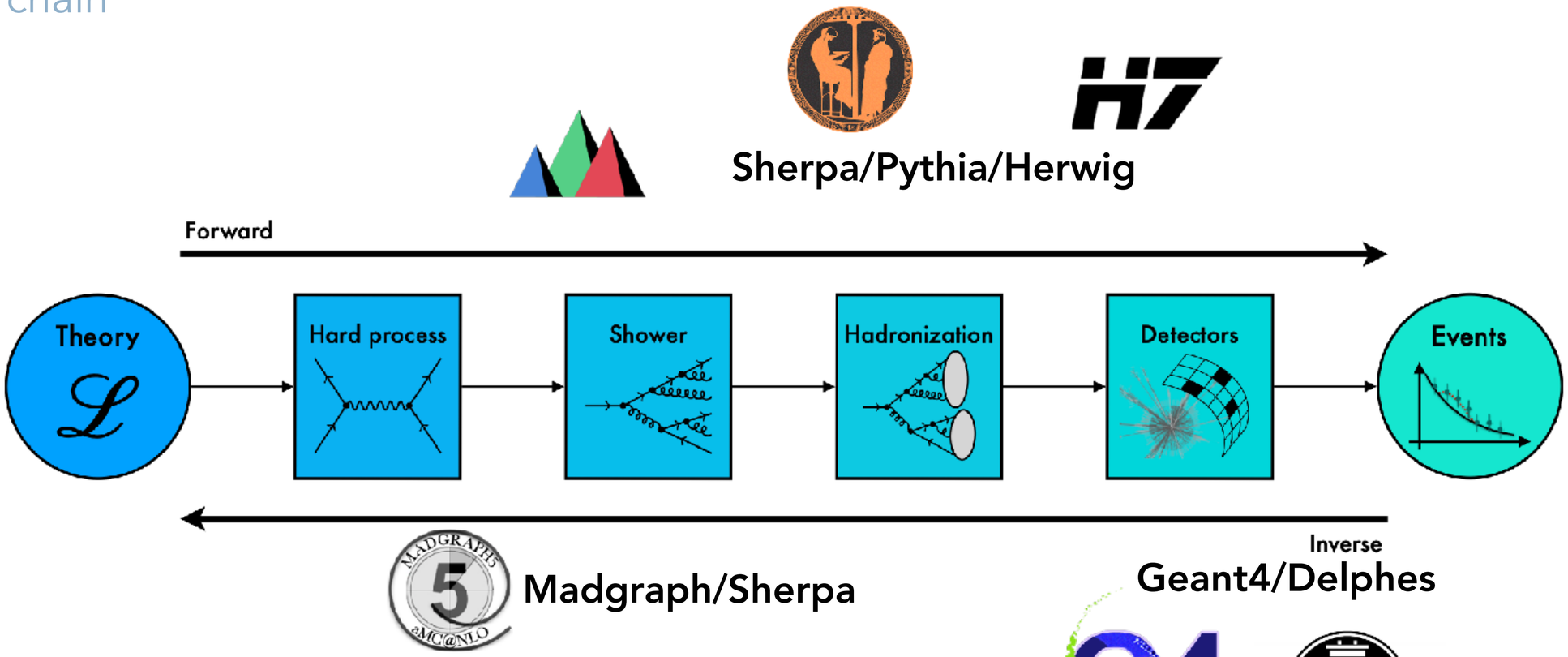
A lot of new ideas and discussions with Parnassus, stay tuned!

Thanks for your attention!

Backup

Context

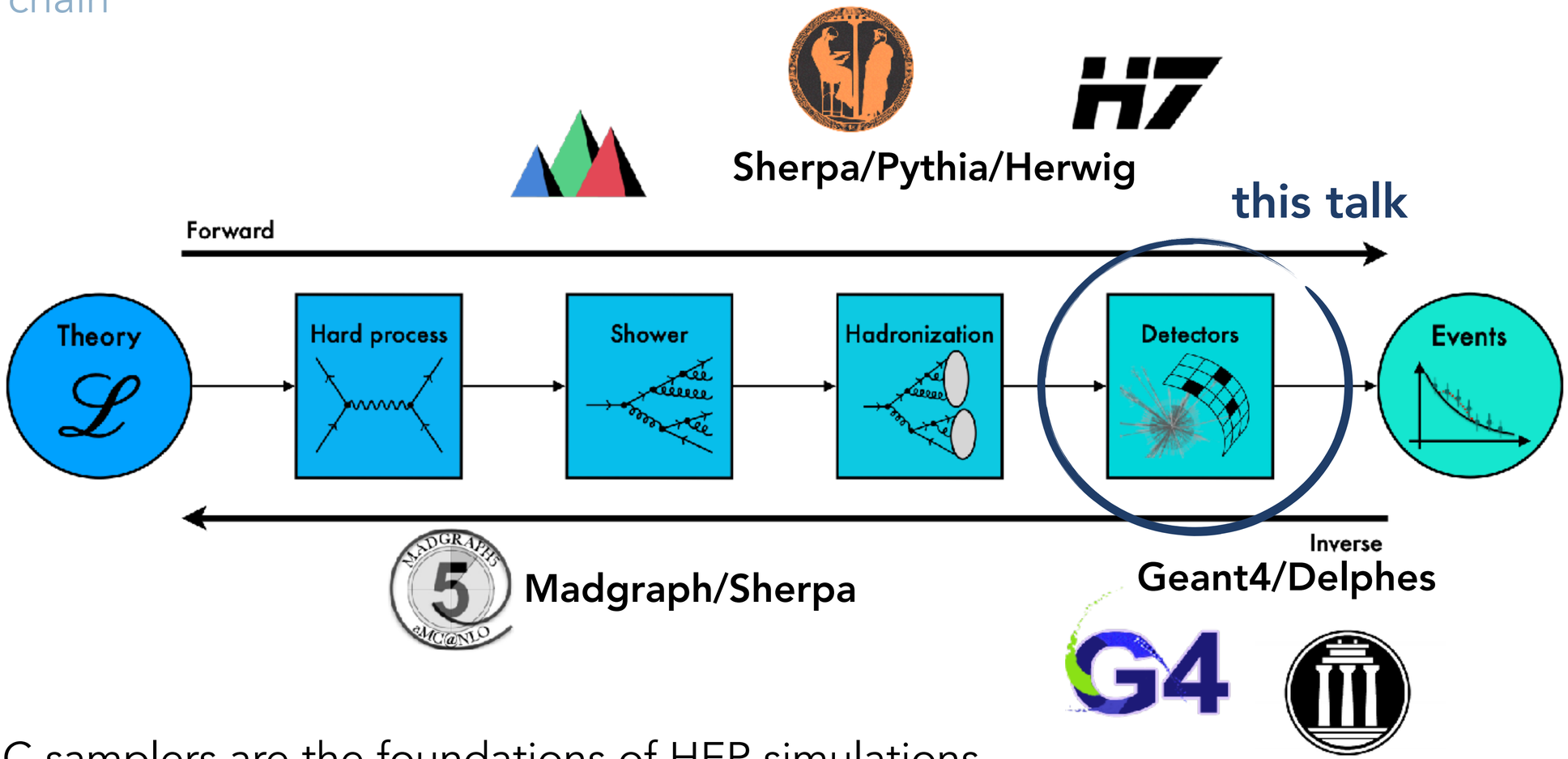
Simulation chain



- MCMC samplers are the foundations of HEP simulations.

Context

Simulation chain

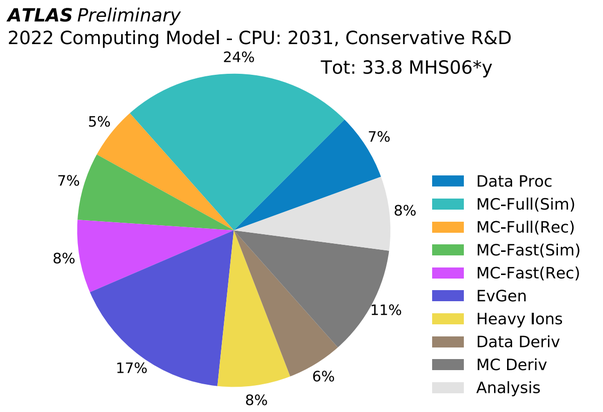


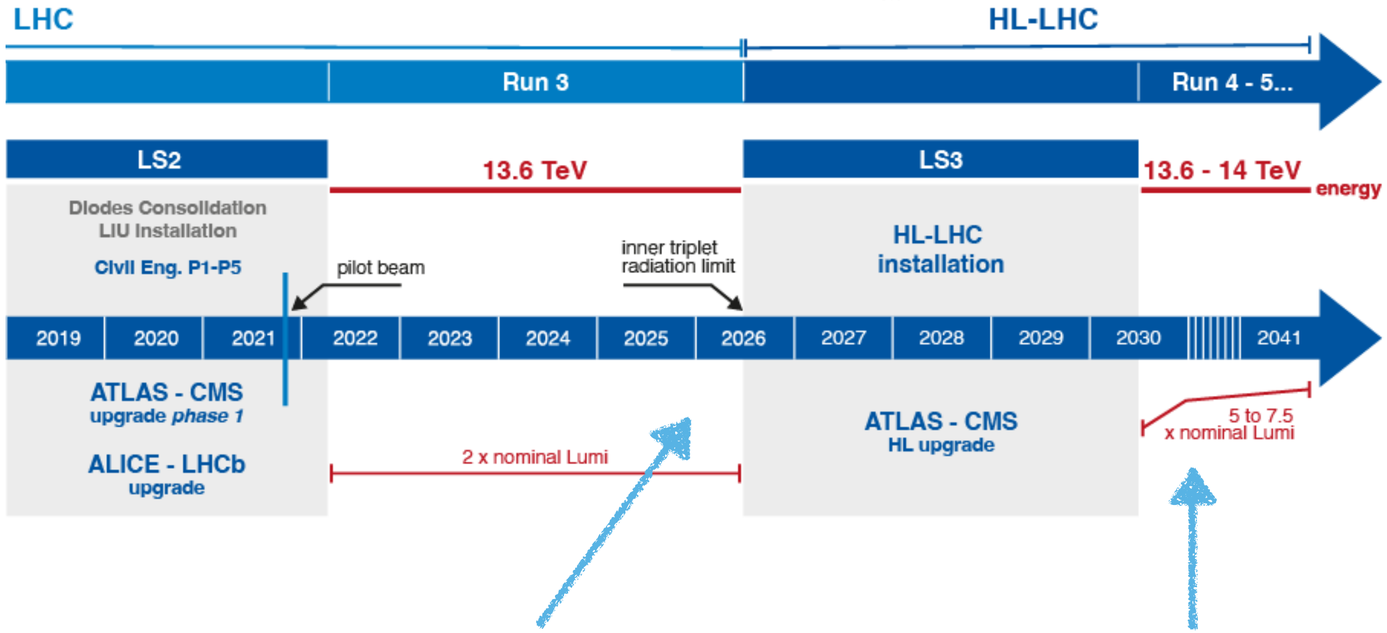
- MCMC samplers are the foundations of HEP simulations.

Context

Challenges

- High-Luminosity LHC will produce unmatched amount of data;
- simulations need to match the collected statistics.



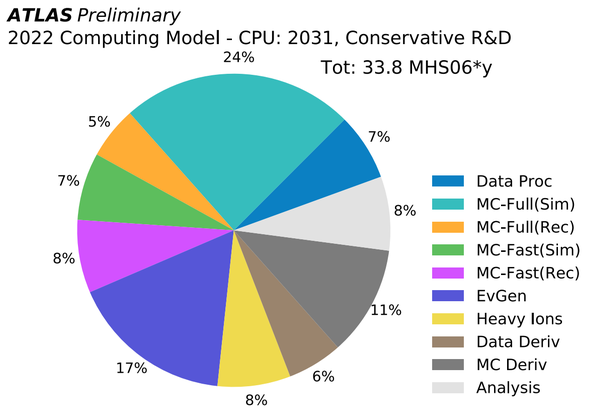


2026: end of Run3 **~3x increase in luminosity**

Context

Challenges

- High-Luminosity LHC will produce unmatched amount of data;
- simulations need to match the collected statistics.

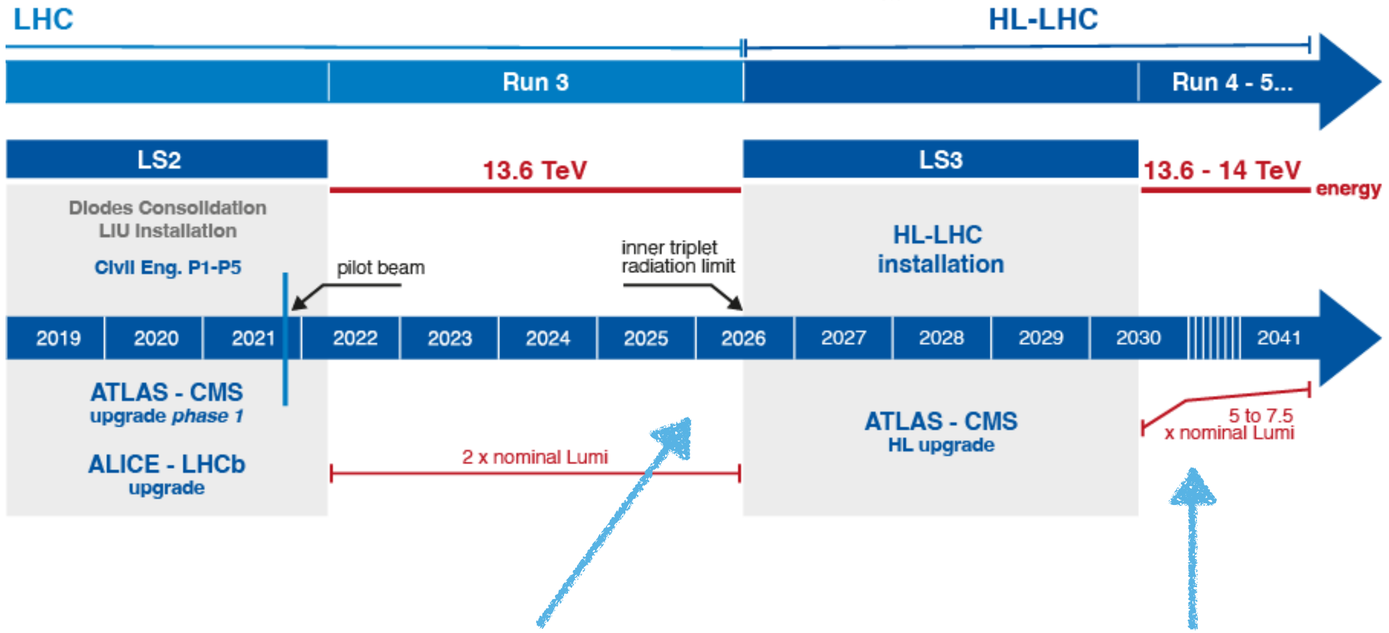


Overarching goal:

Maximise LHC potential

- find BSM physics;
- understand LHC data w/ SM.

Requires better and faster simulators



2026: end of Run3 **~3x increase in luminosity**

CMS full sim evaluation

More validation plots