FlashSim:
End-to-end simulation with Machine Learning

Andrea Rizzi, University and INFN Pisa
on behalf of The CMS Collaboration

CHEP 2024 - Krakow, Poland
21/10/2024

Ref: CMS DP-2024/080

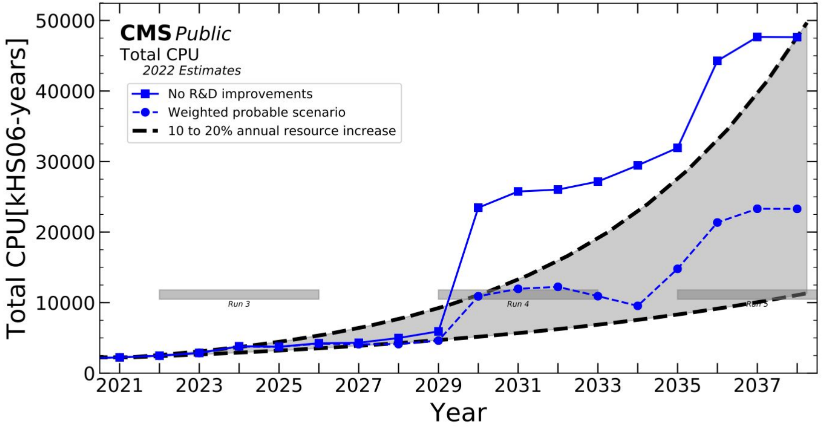
Outline

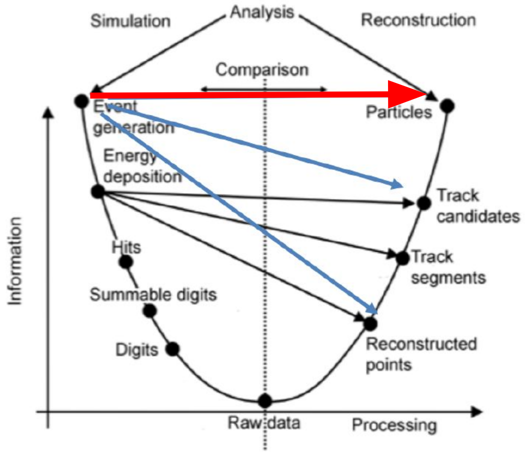
- Why faster simulation?
- What we mean with end-to-end?
- Generative AI
- Normalizing flows and flow matching
- CMS Flashsim structure
- Accuracy of simulated variables
- Speed, bottlenecks and oversampling
- Conclusions

Simulation at LHC

- Simulation is a large fraction of LHC experiment computing costs
- Tens/hundreds of **billions of events** needed in analysis for proper modelling of backgrounds and signals
- The increase in number of events and complexity of single events for HL-LHC further increases the simulation needs
- Various R&D approaches in CMS to speed-up simulation, often using ML (see Phat's Talk)
  - Speed-up of slowest parts of fullsim (Kevin's talk)
  - FastSim accuracy improvements (Dorukhan's poster)
  - Usage of Delphes for current HL-LHC studies
  - **End-to-End ML for analysis**

this talk!





"la mia parabola" Figure by Federico Carminati, independent parallel inventions by Vincenzo Innocente & Kyle Cramer

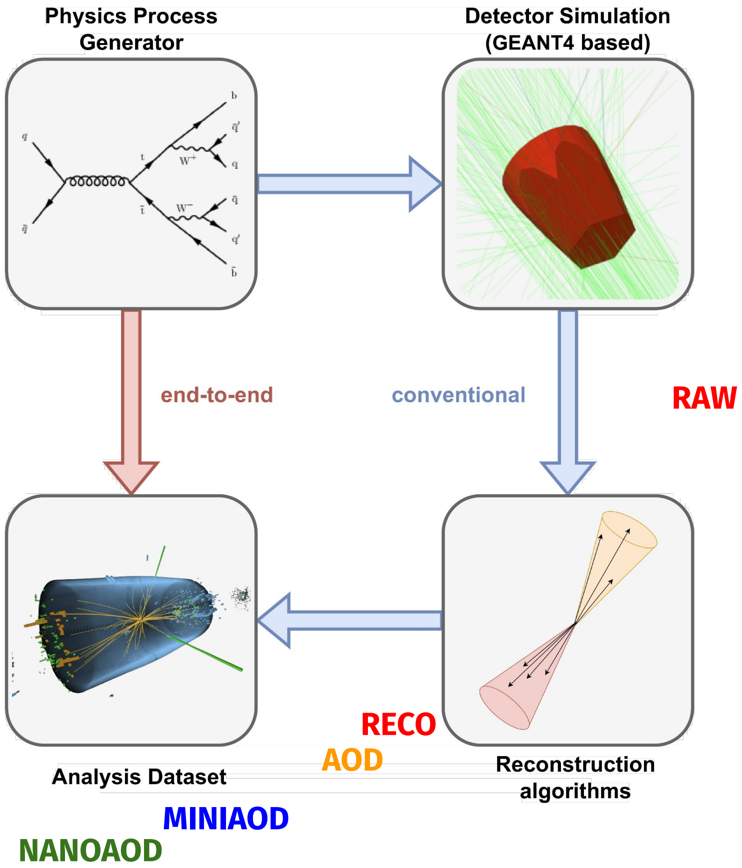
CMS Data Tiers / end-to-end

CMS data tiers

- **RAW & RECO** => lowest level: detector hits, reconstructed objects including all intermediate steps
- **AOD** => subset of RECO with higher level objects
- **MINIAOD** => compact version of AOD
- **NANOAOD**=> ntuple like format usable by most analysis, only ~1-2Kb/event of information

NANOAOD is one of the enabling factors for a general purpose end-to-end simulation:

- Reasonably “simple” target
- Still usable for analysis

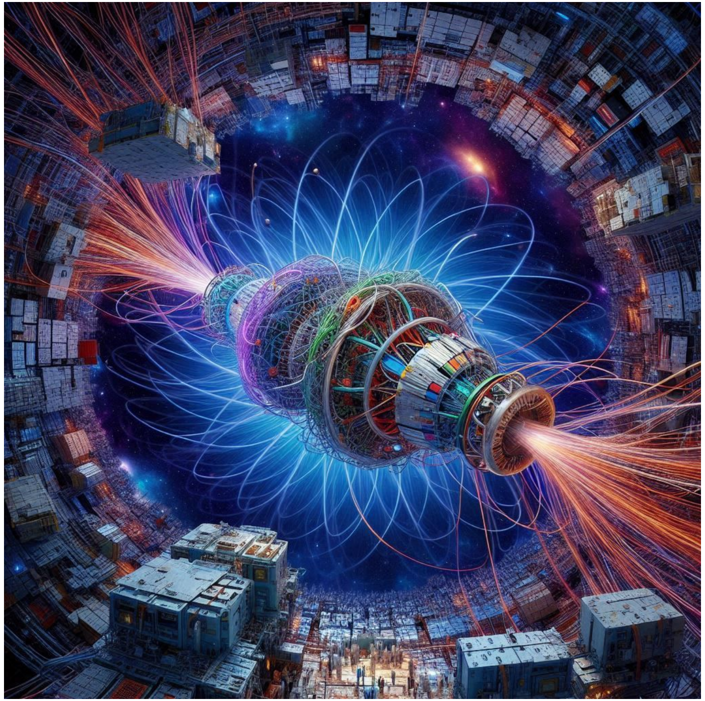


Generating LHC events with AI

“conditioning”
as in P(x|cond)

Generate an LHC event

an LHC event of CMS experiment with a Higgs boson decay to a pair of muons

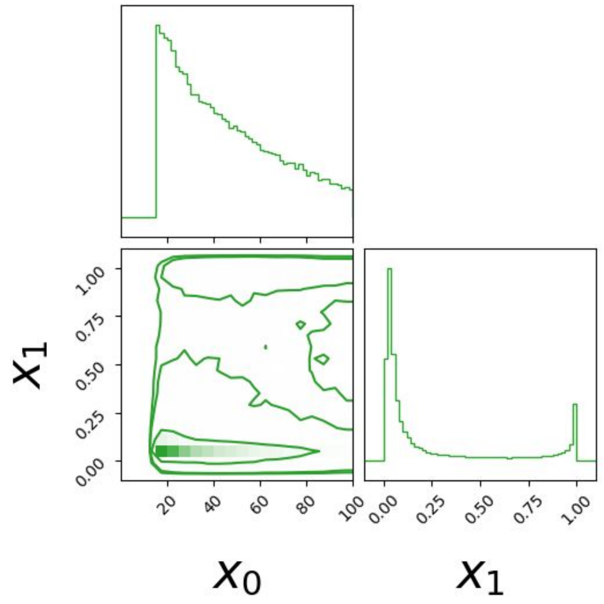


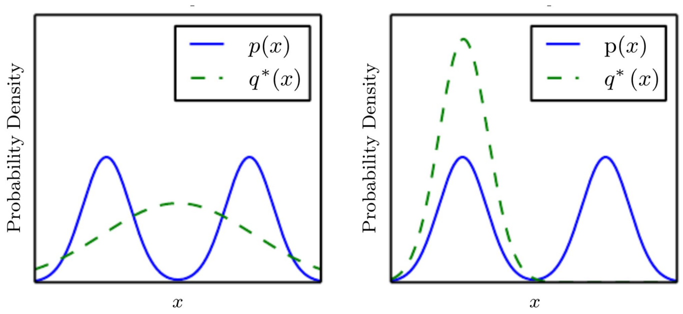
Accuracy in image generation

Qualitative definitions, no requirement of statistical properties of generated samples

Accuracy for Physics Analysis

- A simulation **usable for analysis** should properly **reproduce PDFs** both for individual variables and for their correlations
- Many AI generative tools (e.g. GAN, VAE) work reasonably well *qualitatively* but are severely limited when looking at distribution details
  - mode collapse for multimodal distributions
  - bridging between peaks
  - accurate in mean and variance but limited handling of long tails
- We tested various alternative models including W-GAN and other modified losses
  - limited success on the low dimensionality problem we face

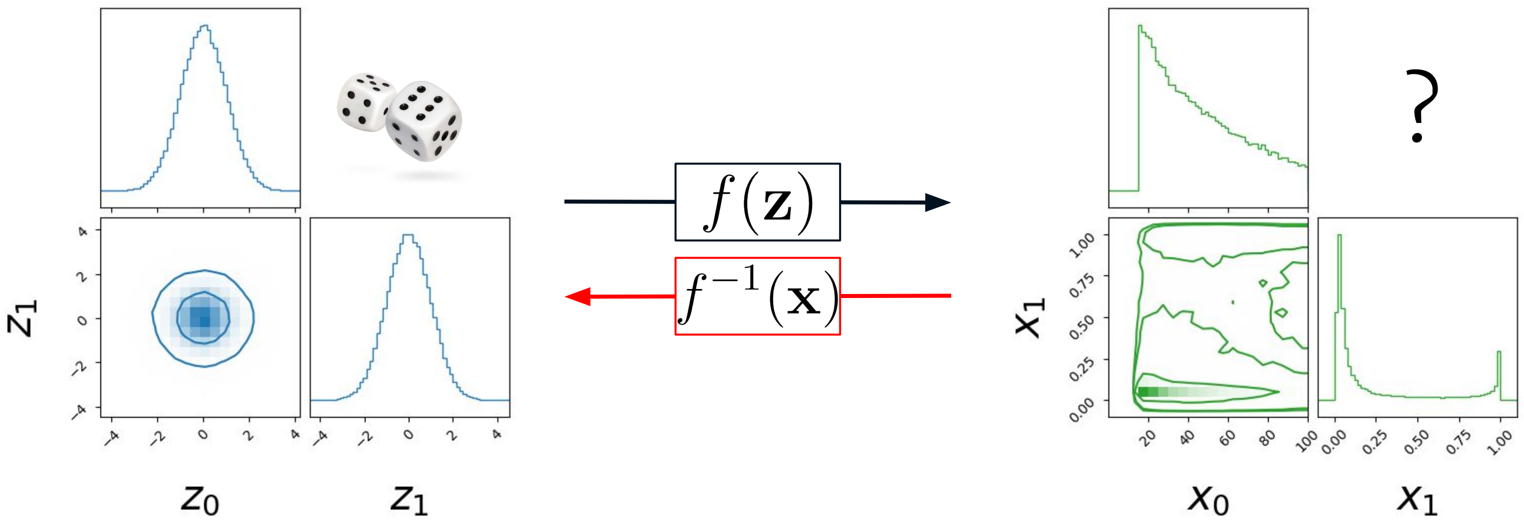




Normalizing Flows: generative model for *pdfs*!

What we know

What we need



$\mathbf{z}$

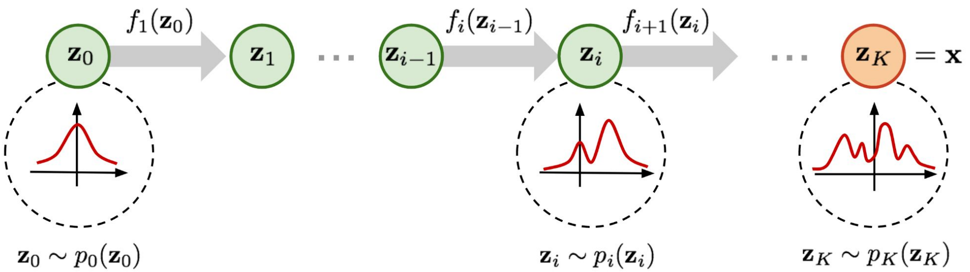
multi-dimensional gaussian

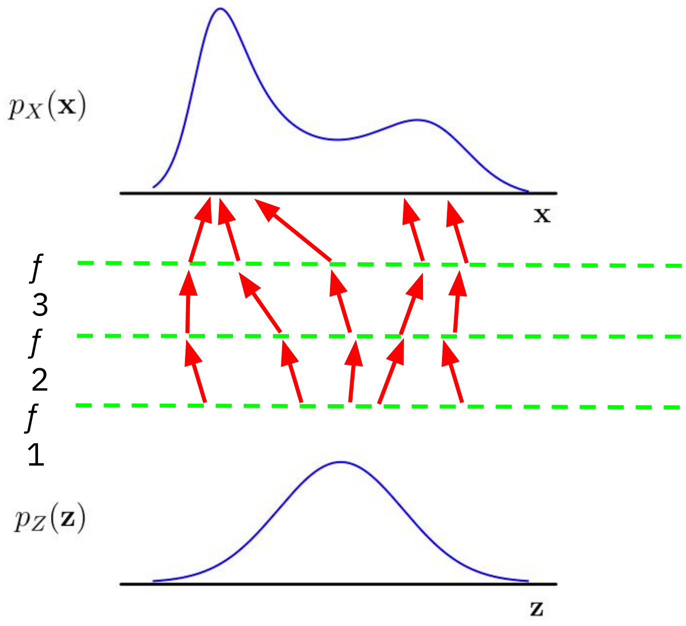
$\mathbf{x}$

FullSim data, pdf unknown!

$f(x)$ as a discrete flow

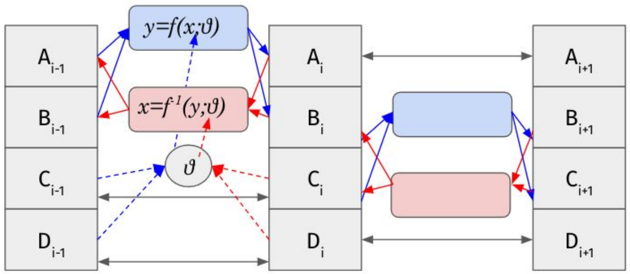
- In order to increase the expressivity of $f(x)$ we can use a **chain** of simple invertible transformations
- The parameters of each transformation are determined by a DNN that takes as input the previous state and the external conditioning information





- In order to catch **correlations** you want one variable to depend on others f(x;θ), but to keep it **invertible** you cannot transform the variables θ as they are need to compute $f^{-1}(y;\theta)$
  - Autoregressive: 1st variable depends on nothing, 2nd variable depends on 1st, 3rd on (1st, 2nd)... etc..
  - Coupling: at each step only transform some variables, and explicitly depend on the others

coupling architecture



Continuous Flows

Possible solution: continuous flow

$$f(0; z) = z = Gaussian$$

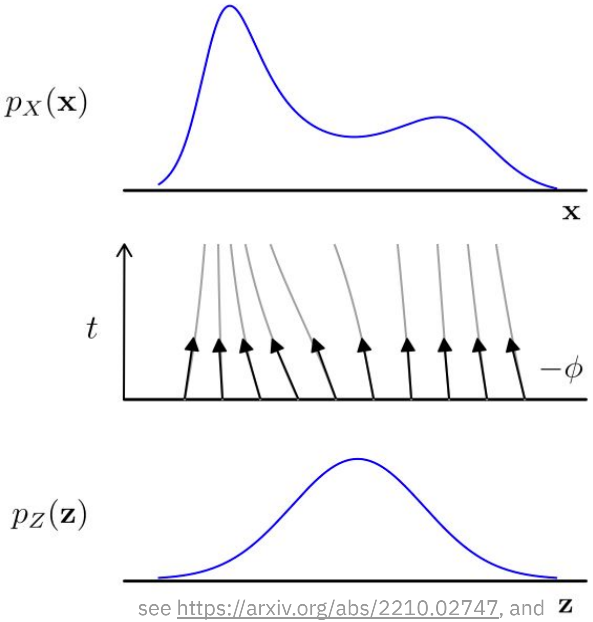
$$f(1; z) = \text{target p.d.f.}$$

$$f(t + dt) = f(t) + v(t) \cdot dt$$

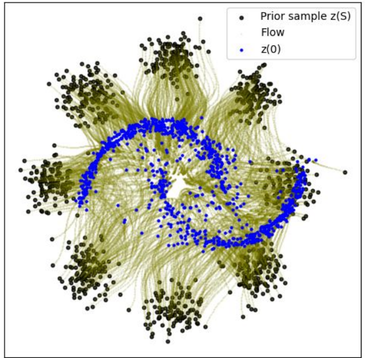
$$f(t + dt) = f(t) + DNN(f(t)) \cdot dt$$

A technique called Flow Matching allows to train continuous flows learning the vector field v_t

- Solves the conditioning problem: each step is infinitesimal hence *f(t) ~ f(t+dt)*
- No need to chose a function for the transformation as we simply learn its gradient in every point of space
- At inference time we will need to integrate the path from t=0 to t=1
- Use a single DNN to predict the vector field in any point
  - while discrete flow have one DNN for each step



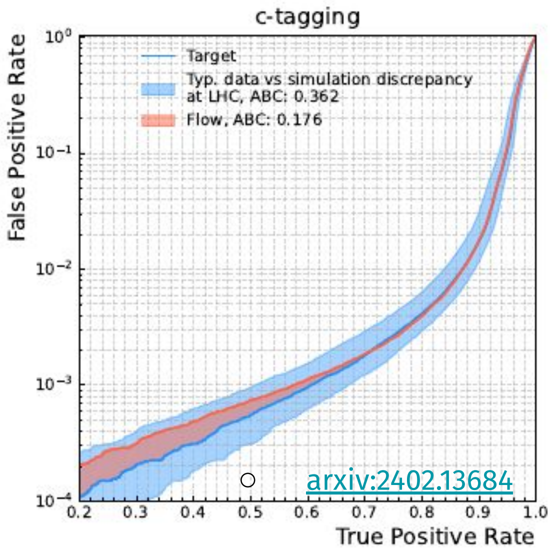
see https://arxiv.org/abs/2210.02747, and https://arxiv.org/abs/2302.00482, figure from https://ehoogeboom.github.io/post/en_flows/



CMS FlashSim

Goals and ideas of CMS “FlashSim”

- Provide an **analysis agnostic simulation** exploiting the common NANAOD format as a baseline target of the simulation
- Be **sample independent**, learning the **“detector response”** to different type of generated particles and in different running conditions
- Reach a speed that is **orders of magnitude faster** than existing simulations
- Maintain an accuracy that is good enough for analysis, with a “delta” to full-sim that is the same order of magnitude of the delta between full-sim and real data.



- arxiv:2402.13684

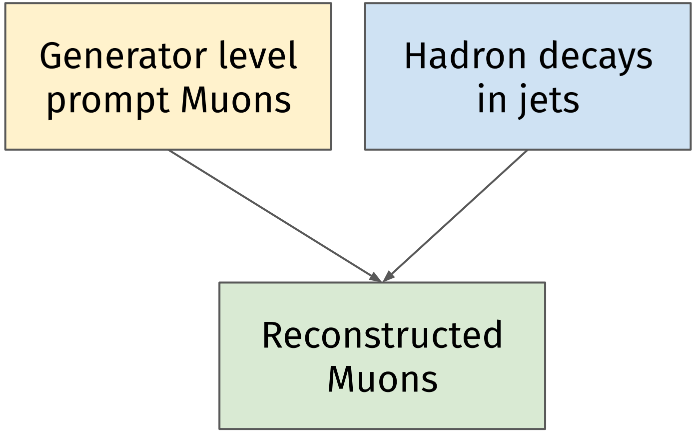
FlashSim structure

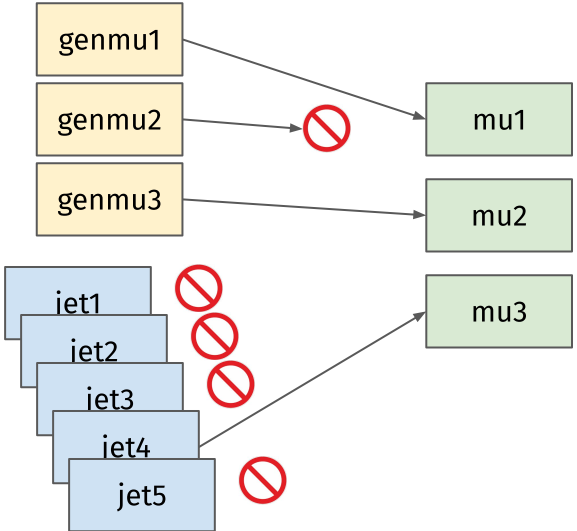
A **reconstructed object** may originate from **multiple sources**

- genuine signal
- particles with similar signature
- detector interactions and decays
- fakes, duplicates, pileup

Each object is handled by FlashSim with various models

- An **efficiency model** for each source
- A **properties model** for each source



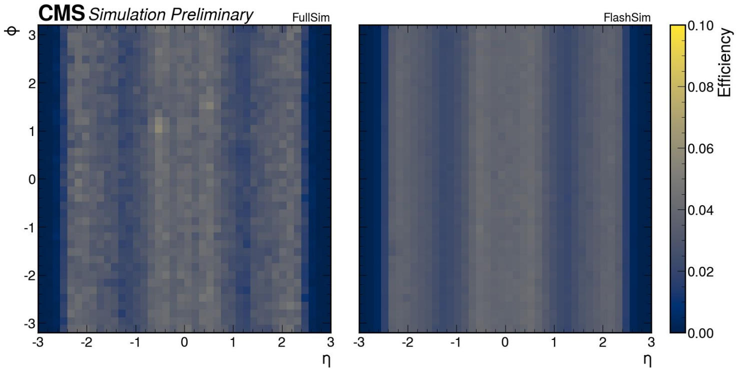


Efficiency models

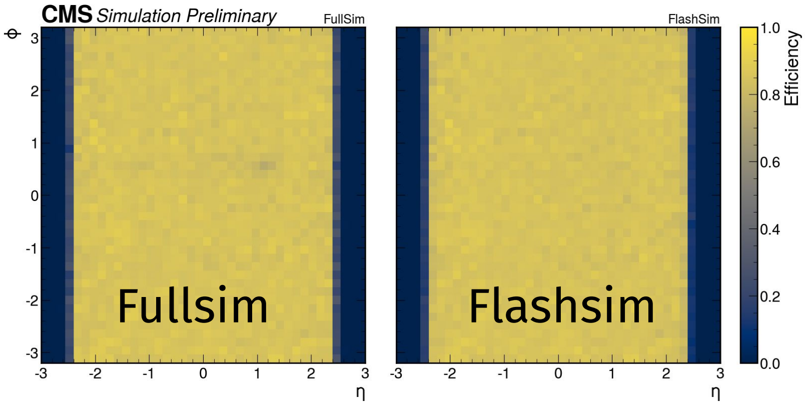
Given a soure object to we get a reconstructed one?

- Efficiency models are **trained as simple classifiers** with binary cross-entropy loss
  - output can be interpreted as a probability!
- At inference time we just **toss in [0,1] and compare with model probability**

Probability of a jet producing a mu

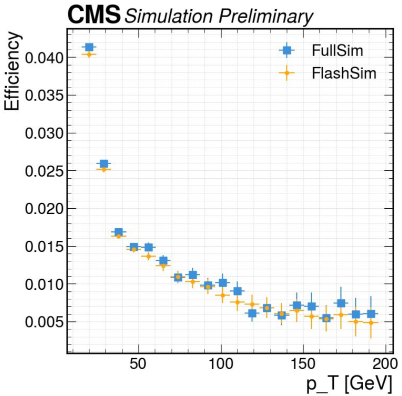


Prompt muon efficiency



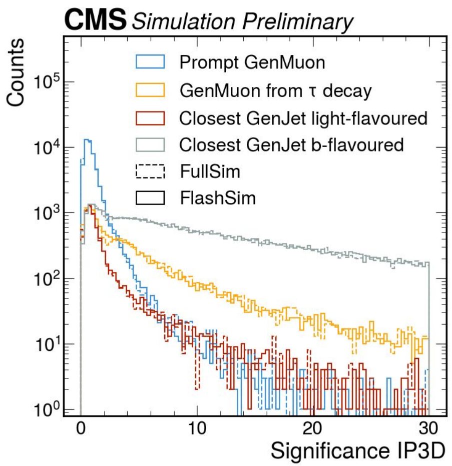
Prompt muon duplicate probability

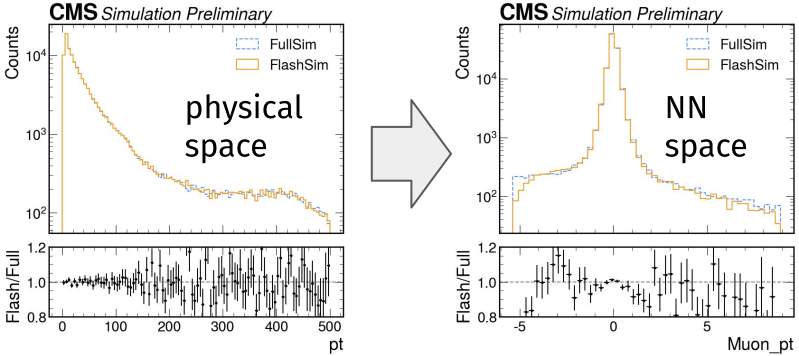
Duplicates can be handled by training a second classifier to predict when a second copy is produced



Properties models

- For each object we need to **simulate all its properties**
  - e.g. momentum, eta, phi, tagging variables, ID, isolation, etc..
- Some properties have obvious correlations with generator level information
  - generated vs reconstructed four-momentum
  - MC flavour with tagging variables
- Two crucial points to reproduce correlations
  - Conditioning:
    - e.g. is it b-quark jet?
  - Transformations:
    - standard scaling
    - better learn P_T^{reco} or P_T^{reco}/P_T^{gen} ?
    - tails matter for physics (apply logs when needed)





Flashsim status

- Current FlashSim prototypes simulates all object properties for most of the NANOAOD format collections
- Major sources of signal and background are considered, more to be added in the future
- Currently missing: trigger information (some part is trivial, some is less)

Physics objects	Sources (one NN model for each source)			Number of simulated attributes per object
Jets	Generator Jet	Fake from PU		39
Muons	Generator Muons	Fake from Jets/PU	Duplicates	53
Electrons	Generator Electrons	Generator Photons (prompt)	Fake from Jets/PU	48
Photons	Generator Photons (prompt)	Generator Electrons	Fake from Jets/PU	22
MET	GenMET and HT			25
FatJets	Generator AK8 Jets			53
SubJets	Generator AK8 SubJets			13
Tau	Reconstructed Jets with a Tau	RecoJets without a Tau		27
Secondary Vertices	Jets with Heavy Flavour	Light Jets	Taus	16
Non MET scalars (e.g. PV)	Various event level inputs			16
FSRPhotons	GenMuon/RecoMuon			6

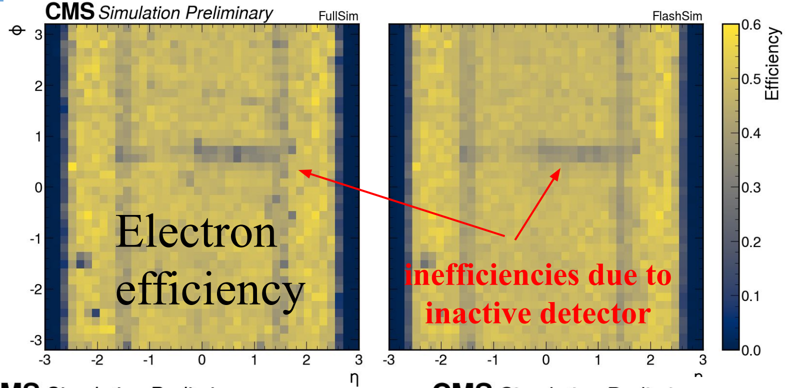
Results on individual models

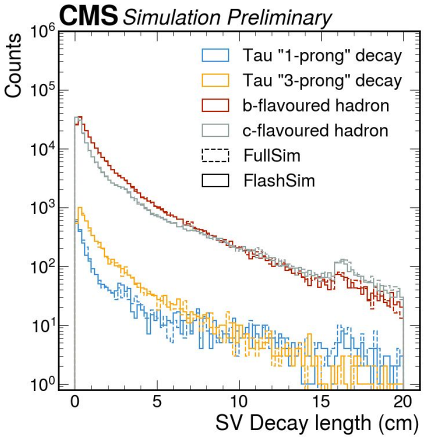
Results on 4M events training

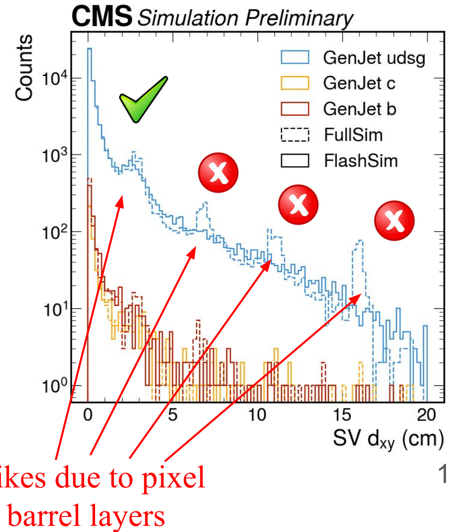
- Trained on a cocktail of a few samples with different signatures covering different corners of the phase space
  - likely suboptimal choice, dedicated samples (e.g. flat QCD or particle guns) could also be considered

Sample	Events
$t\bar{t}$	800k
DY HT [100, 200], 2J MLL [200-1400]	930k
HH → bb bb	840k
X(3000) → Y(500) H(125) → (bb) (WW → 2q 2lν)	147k
X → HH → qq qq (M_X 900, 1200, 1800; M_H 365, 400, 18)	90k
SMS TchiZH mNLSP200-1500	300k
X(1200) → Y(300) H(125) → bb $\gamma\gamma$	400k
VBF H → $\tau\tau$	270k
bbA → ZH → ll $\tau\tau$ (M = 900)	33k

- FlashSim learned some of the detector features present in the simulation (and missed some other)

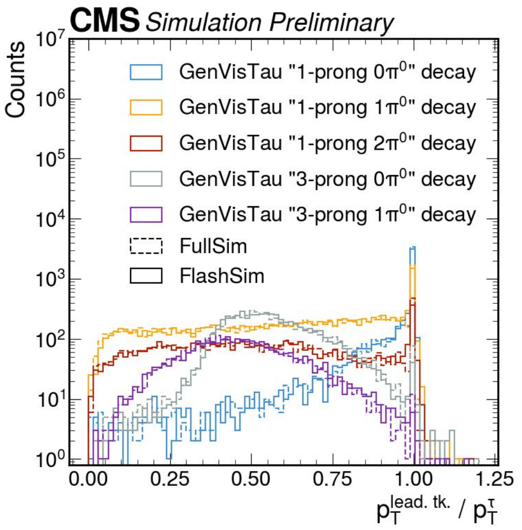


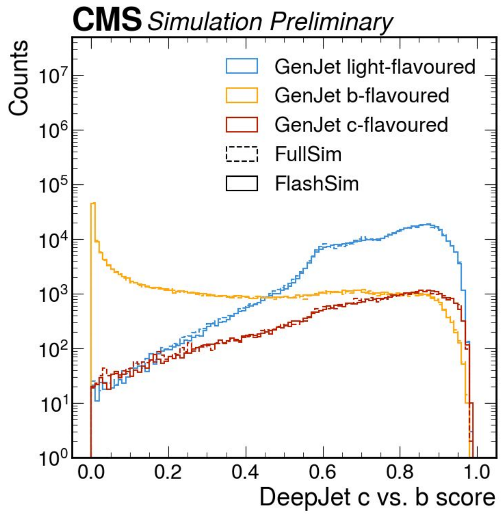


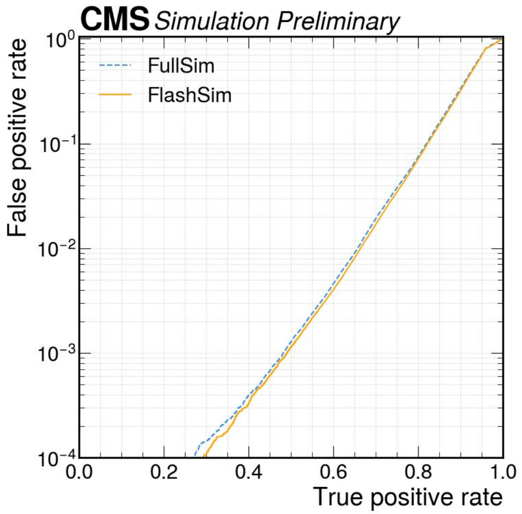


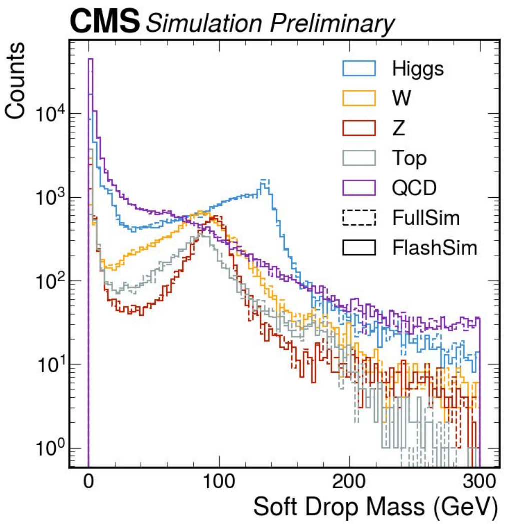
Accurate conditioning

- A single model should learn to produce different distributions for different conditioning values (momentum of a particle, flavour of the quark producing a jet, decay mode of a particle, etc...)
- flowmatching is incredibly accurate at catching the multidimensional correlations between conditioning variables and output ones

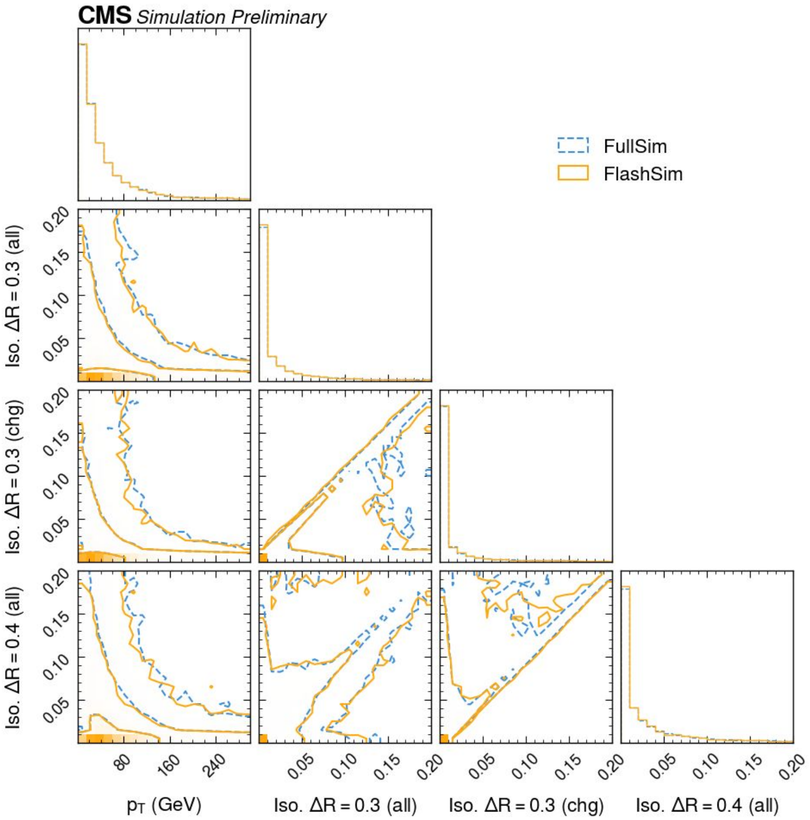


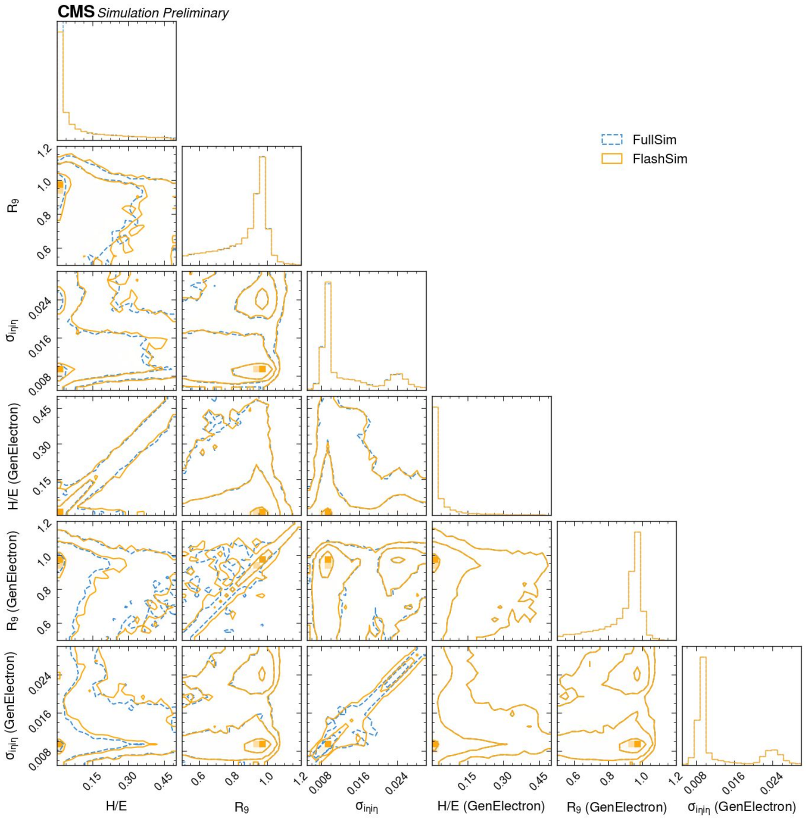






Example of correlations





Full Event simulation
and
toy analyses

Event simulation chain

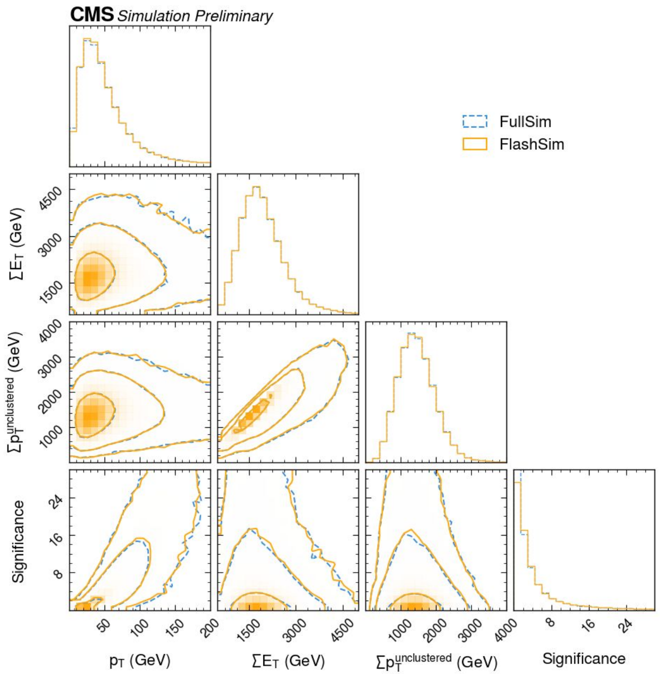
Simulating a full NANOAOD event implies several steps, to be repeated for each object, for each source

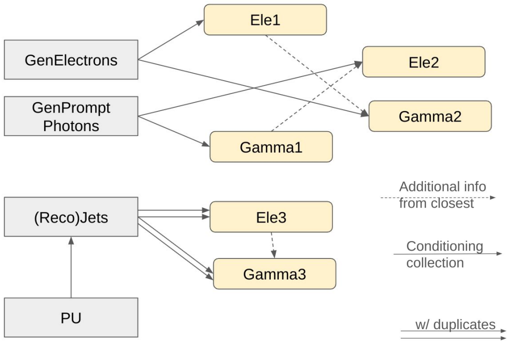
- extract the conditioning information
- run the efficiency model
- run the properties model
- (merge output from different sources)

Some models are conditioned not only on *generator* information but on *reconstructed* information from previous steps (e.g. MET is conditioned on the various reconstructed objects ; electron and photon reconstruction are cross-conditioned)

FlashSim event simulation is **extracting data with RDataFrame** and processing batches of events in parallel with **PyTorch**

Simulated ~100M events from various processes, some of them never seen during training.



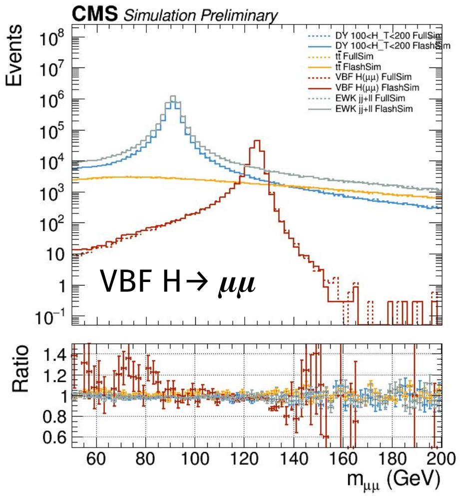


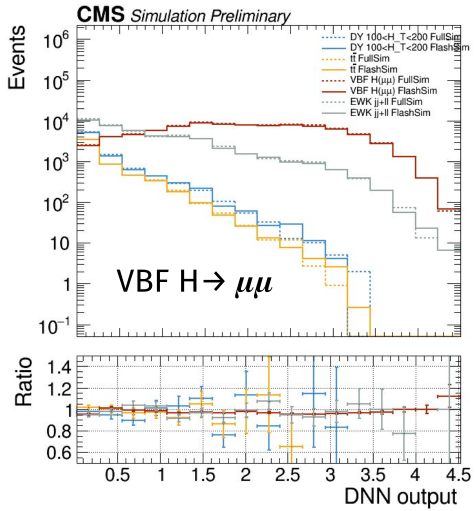
Sample	Events
$t\bar{t}$	100M
DY HT [100, 200]	25M
$H \to \mu\mu$	1M
ZH	300k
jj + ll (ewk)	8M

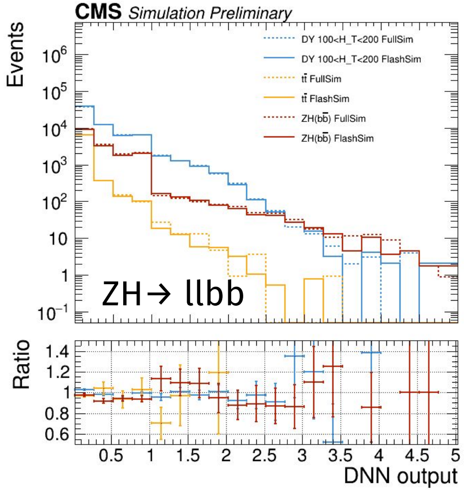
Derived quantities

Once full NANOAOD event are available we can compare derived quantities and implement some analyses

- Two toy analysis corresponding to VBF Higgs to muons search and ZH→ llbb have been tested comparing flashsim with fullsim
- Analyses tested all the way down to the final DNN output, comparing different samples, some never seen during training







VBF H→ $\mu\mu$	Selection
Muons	$p_T > 20$ GeV, $\|\eta\| < 2.4$, Iso < 0.25, MediumID
Jets	$p_T > 25$ GeV, $\|\eta\| < 4.7$, puId > 0, jetId > 0
Signal Region	$115 < m(ll) < 135$, $p_T^{j1} > 35$, $p_T^{j2} > 25$, $m(jj) > 150$, $\|\Delta\eta(jj)\| > 2$

ZH→ llbb	Selection
Muons	$p_T > 20$ GeV, $\|\eta\| < 2.4$, Iso < 0.25, MediumID
Jets	$p_T > 20$ GeV, $\|\eta\| < 2.5$, puId > 0, jetId > 0
Medium b-tag	DeepFlavour btag > 0.27
Signal Region	$75 \leq m(Z) < 105$, $90 < m(jj) < 150$, Medium b-tag (lead. jet)

Speed and bottlenecks

How fast is FlashSim?

- The current prototype with ~20 properties model and ~20 efficiency models, starting from existing generated samples runs between 10Hz and 1KHz
  - Accuracy of integration
  - Availability of GPU vs Single CPU
- How fast do we need FlashSim to be
  - If you already have generated samples, as fast as possible
  - If the generator is very slow, we are easily in the shadow of the generator
- What if we can avoid being generator-speed limited by **reusing** generated events?
  - Overampling!

Processor	ODE accuracy (timesteps)	Event simulation rate
GPU 3060	100	325 Hz
GPU 3060	20	690 Hz
CPU 1-core	100	15 Hz
CPU 1-core	20	60 Hz
CPU 4-core	20	120 Hz

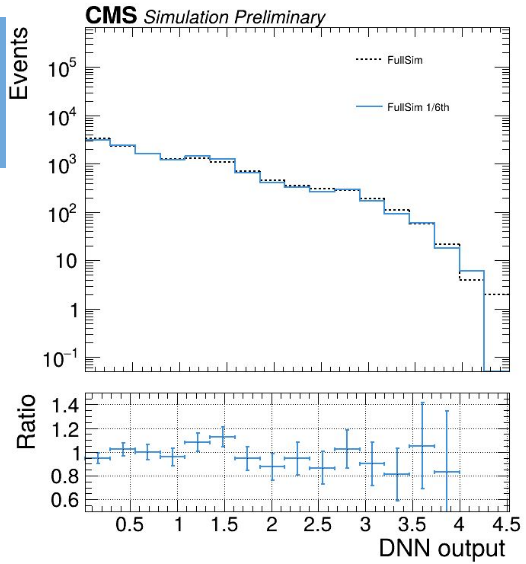
		Event generation speed				Ratio to Geant4-based		
Generator speed (Hz)	Oversample factor	0.1Hz Geant4 based sim	10Hz Flashsim	100Hz Flashsim	1KHz Flashsim	*10Hz Flashsim*	*100Hz Flashsim*	*1KHz Flashsim*
available	1x	0.10 Hz	10.00 Hz	100.00 Hz	1000.00 Hz	*100.0x*	*1000.0x*	*10000.0x*
50.00 Hz	1x	0.10 Hz	8.33 Hz	33.33 Hz	47.62 Hz	*83.5x*	*334.0x*	*477.1x*
50.00 Hz	10x	0.10 Hz	9.80 Hz	83.33 Hz	333.33 Hz	*98.1x*	*833.5x*	*3334.0x*
1.00 Hz	1x	0.09 Hz	0.91 Hz	0.99 Hz	1.00 Hz	*10.0x*	*10.9x*	*11.0x*
1.00 Hz	10x	0.10 Hz	5.00 Hz	9.09 Hz	9.90 Hz	*50.5x*	*91.8x*	*100.0x*
0.05 Hz	1x	0.03 Hz	0.05 Hz	0.05 Hz	0.05 Hz	*1.5x*	*1.5x*	*1.5x*
0.05 Hz	10x	0.08 Hz	0.48 Hz	0.50 Hz	0.50 Hz	*5.7x*	*6.0x*	*6.0x*

Oversampling

- Typical LHC MC samples are randomly sampled "twice"
  - in the generator
  - in simulating the detector response
- In many cases a large part of the uncertainty originates from the detector response
  - generator information can be reused

We call "**oversampling**" the repeated usage of the same generator event for multiple simulations

- Proper statistical treatment is needed for events originating from "same gen"
  - count events that end up in the same bin of a histogram as correlated
  - consider events in different bins as uncorrelated



Is oversampling introducing biases?

Let's test it against full sim

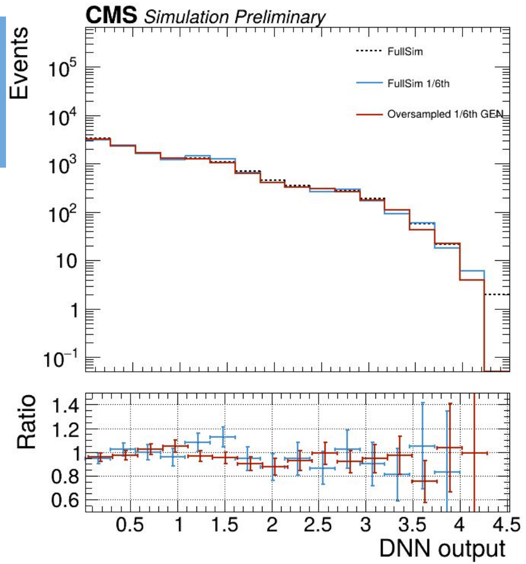
- We start from a sample for which we have 8M full sim events
- We take a fraction (1/6th, 1.3M events) of the full sim events and we can check how oversampling (6x or 10x) it would compare to the full sim sample

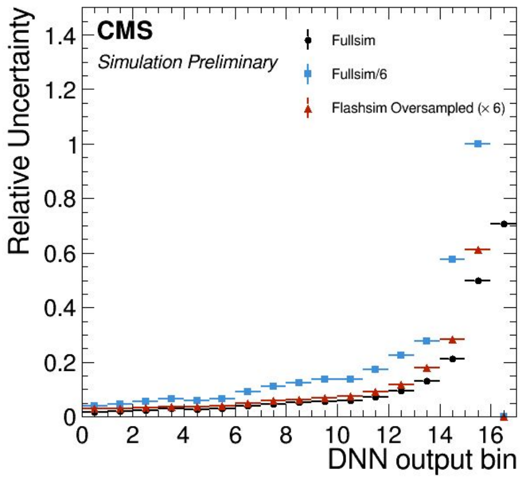
Oversampling

- Typical LHC MC samples are randomly sampled “twice”
  - in the generator
  - in simulating the detector response
- In many cases a large part of the uncertainty originates from the detector response
  - generator information can be reused

We call “**oversampling**” the repeated usage of the same generator event for multiple simulations

- Proper statistical treatment is needed for events originating from “same gen”
  - count events that end up in the same bin of a histogram as correlated
  - consider events in different bins as uncorrelated



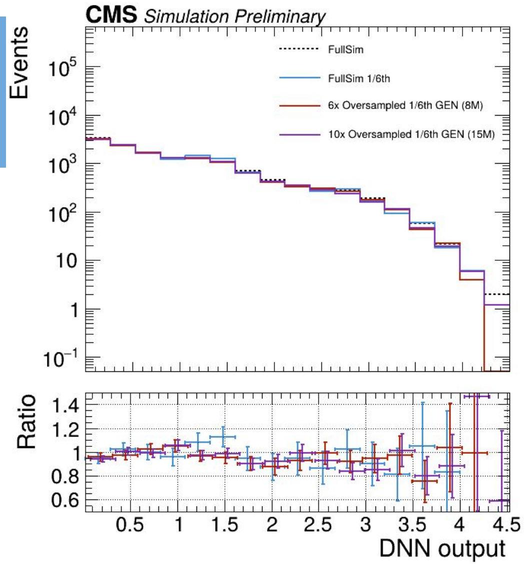


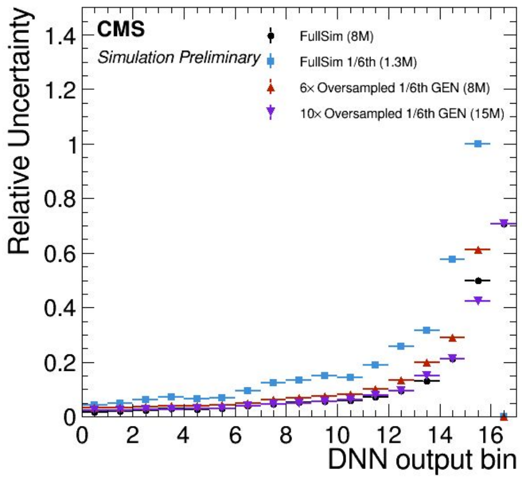
Oversampling

- Typical LHC MC samples are randomly sampled “twice”
  - in the generator
  - in simulating the detector response
- In many cases a large part of the uncertainty originates from the detector response
  - generator information can be reused

We call “**oversampling**” the repeated usage of the same generator event for multiple simulations

- Proper statistical treatment is needed for events originating from “same gen”
  - count events that end up in the same bin of a histogram as correlated
  - consider events in different bins as uncorrelated





Conclusions

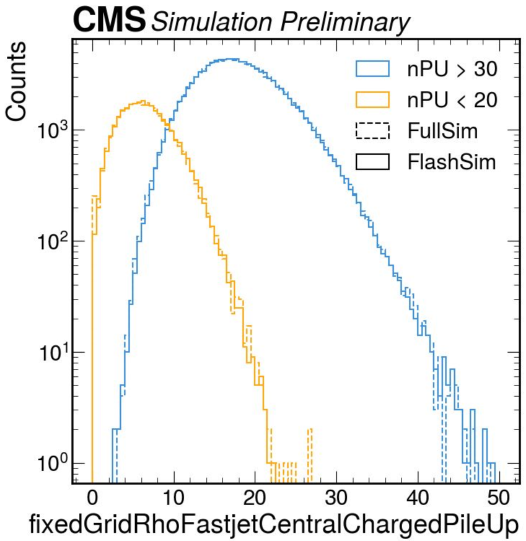
- We implemented the first complete working prototype of an end-to-end simulation, using ML, for CMS NANOAOD format
- A good tradeoff between speed and accuracy has been found, but we can further tune it as needed
- Tests on toy analyses show a good accuracy also for derived quantities, next tests could be on real analysis
- We introduce the oversampling technique to maximize the exploitation of generator level MC event

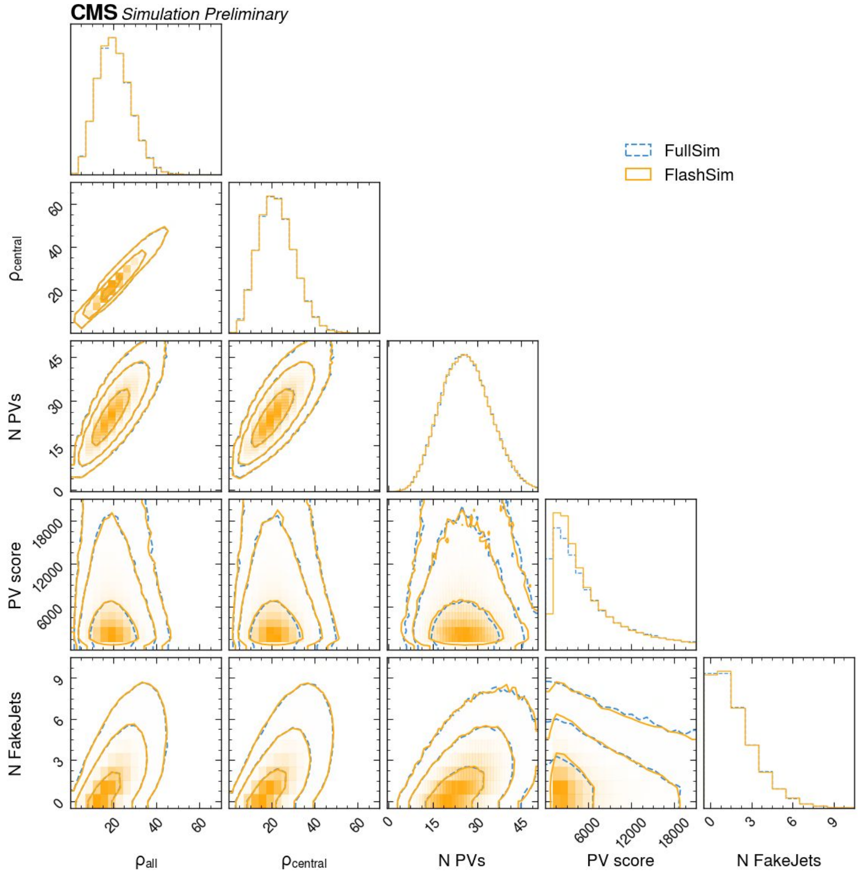
References:

- DPS Note with more plot and details:
  - CMS DP-2024/080
- CMS Note with earlier prototype
  - CMS-NOTE-2023-003
- Paper on toy dataset (see Filippo's talk on Tuesday 17:27 track 5)
  - arxiv:2402.13684

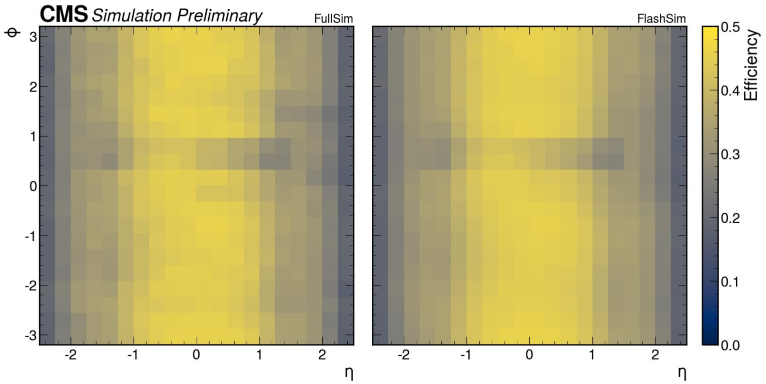
Backup

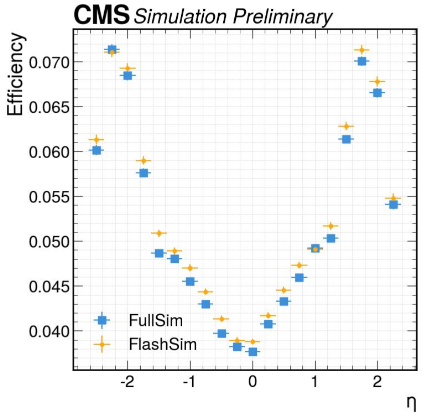
Vertex and Pileup

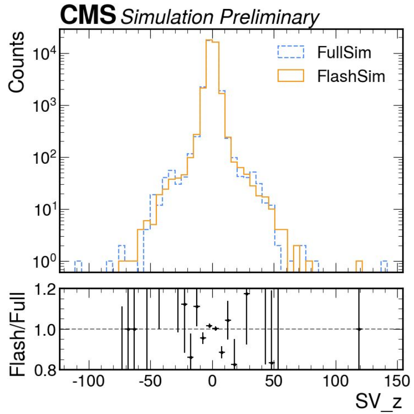




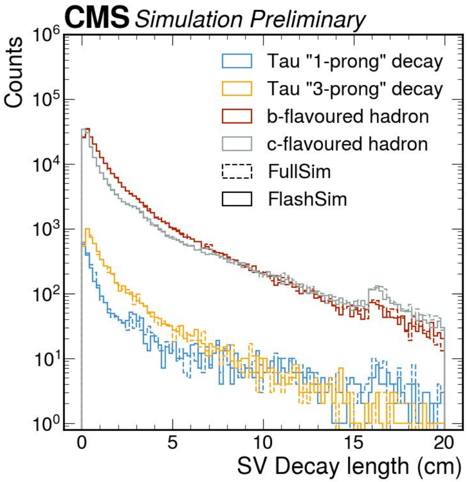
Secondary Vertices

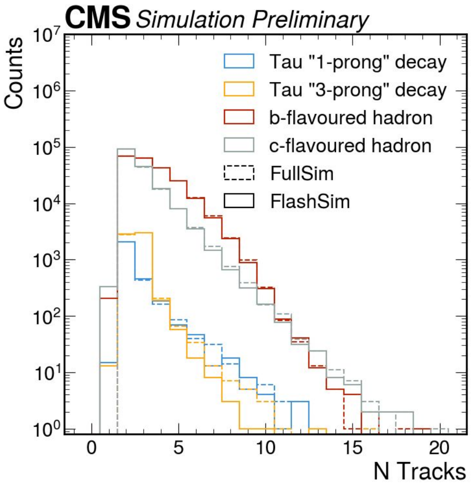


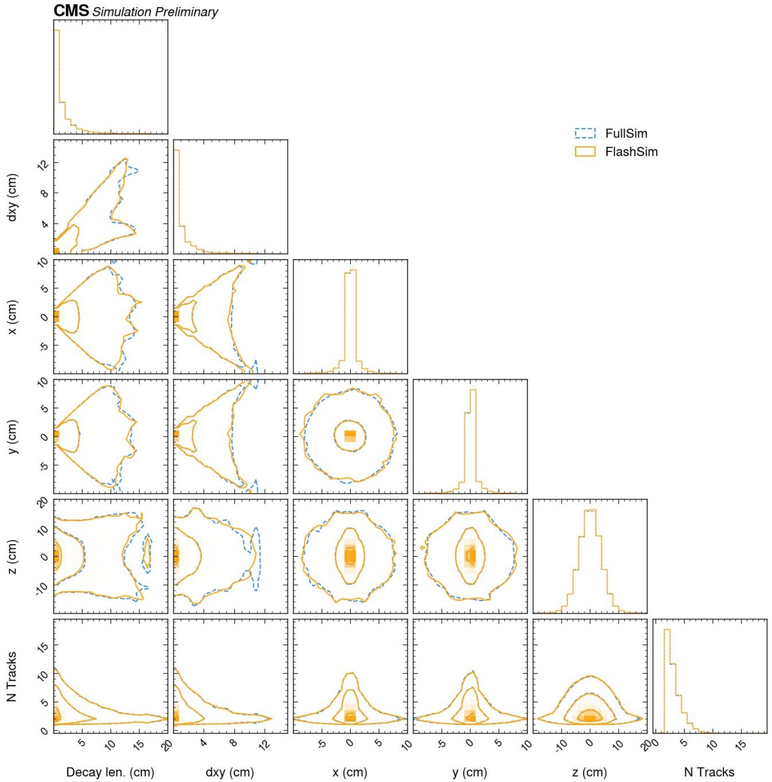




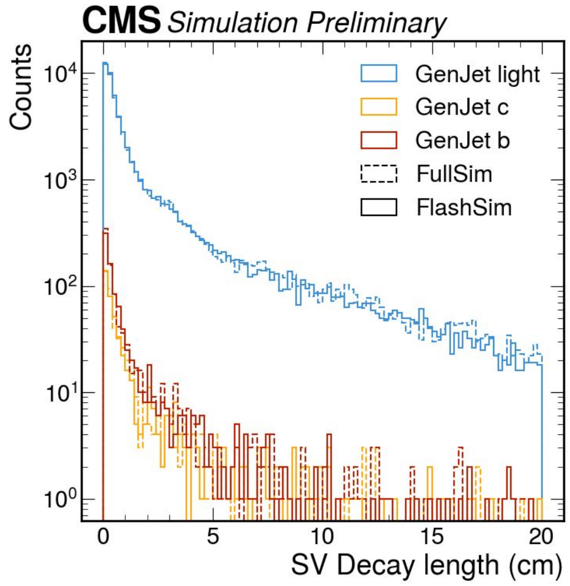
Secondary Vertex from Taus and Heavy Flavour

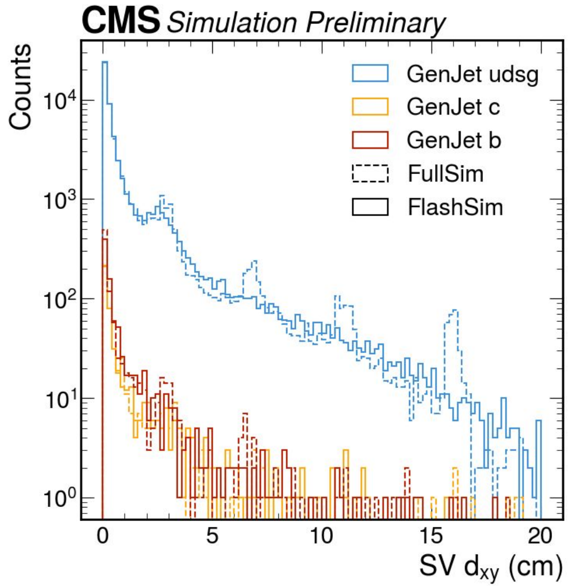


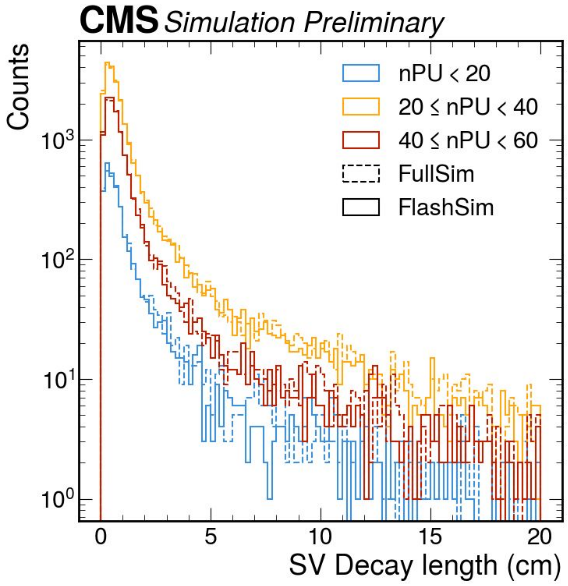




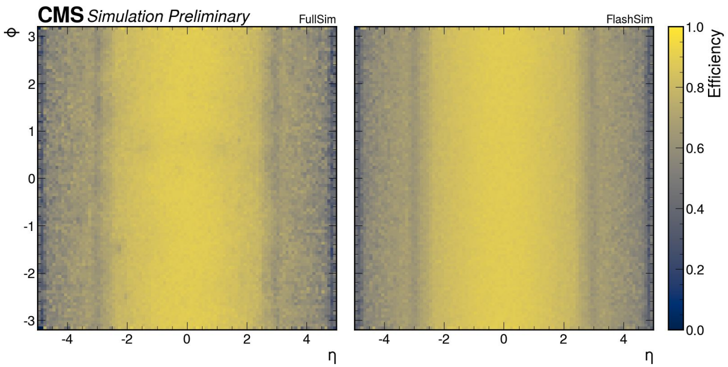
SV from GenJets

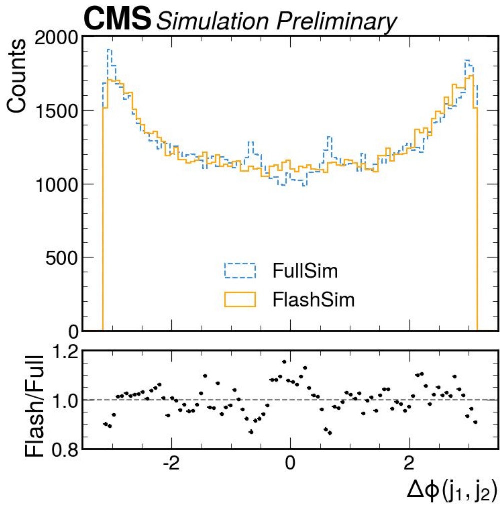


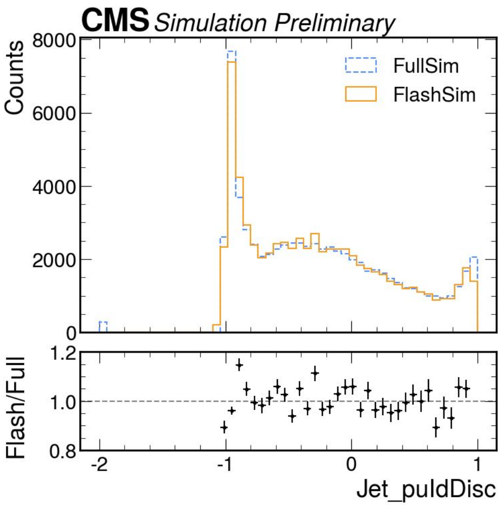


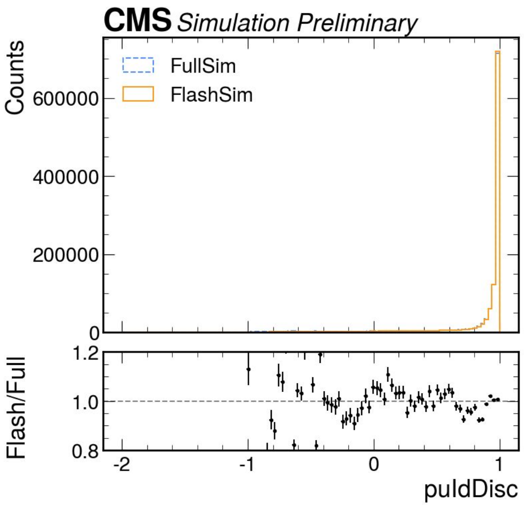


Jets and Fake Jets

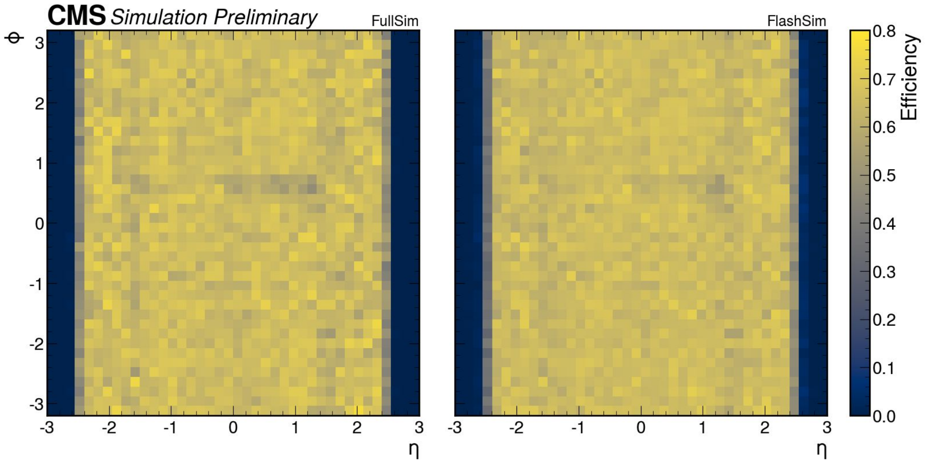




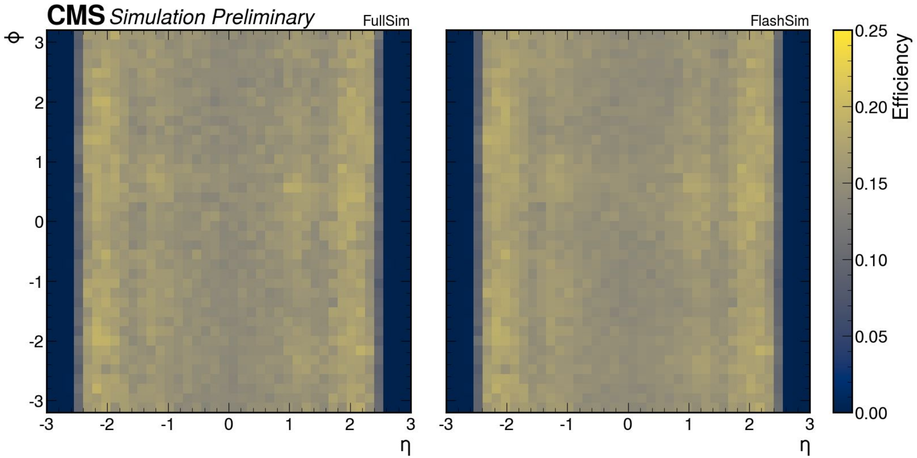




Tau

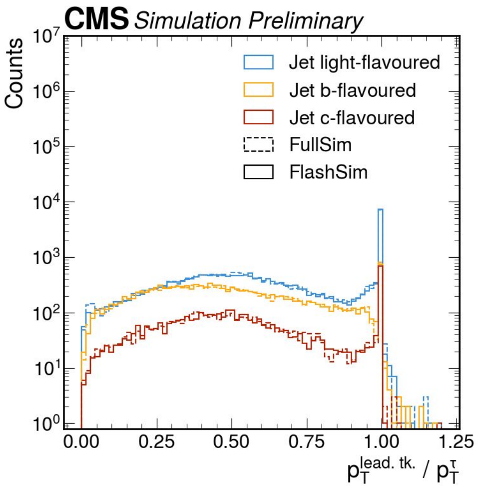


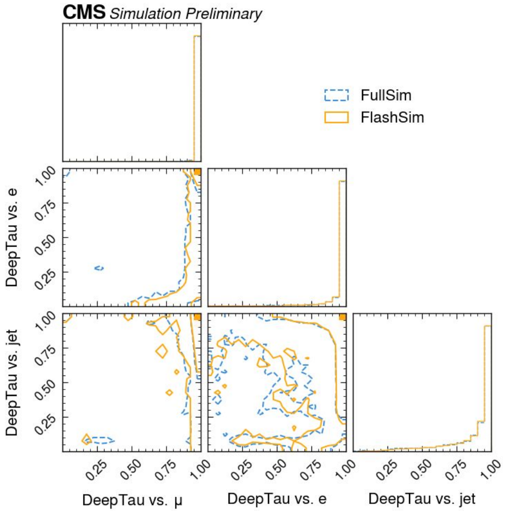
Signal

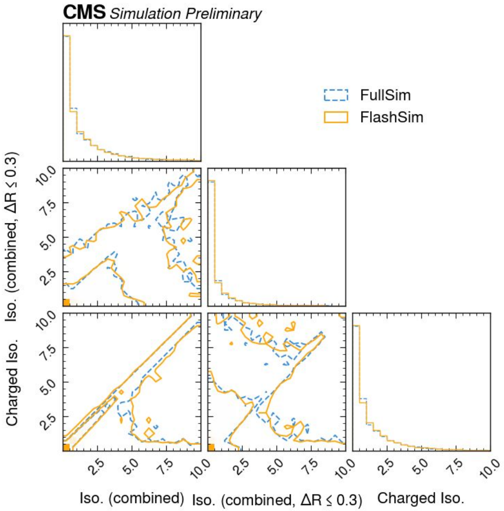


Background

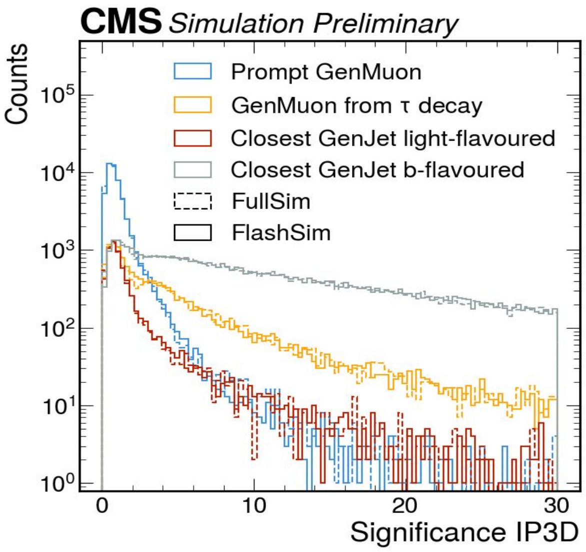
Tau properties

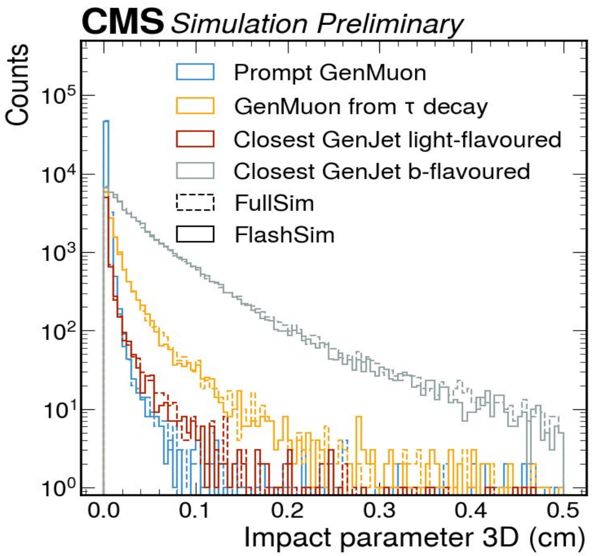


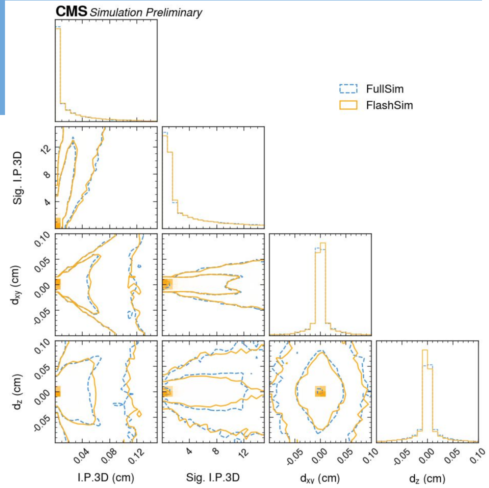


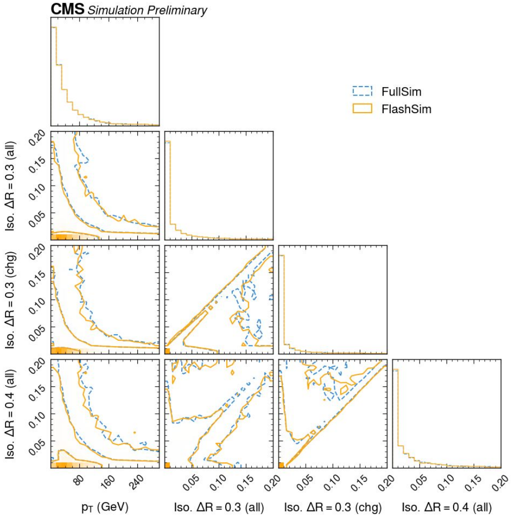


Muon features

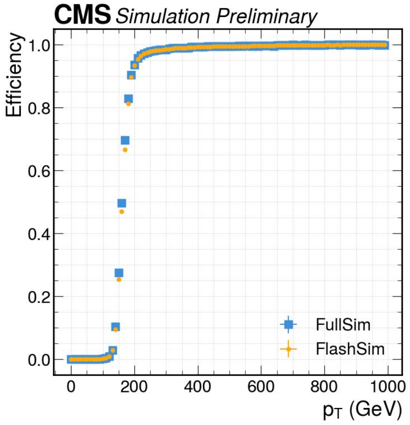


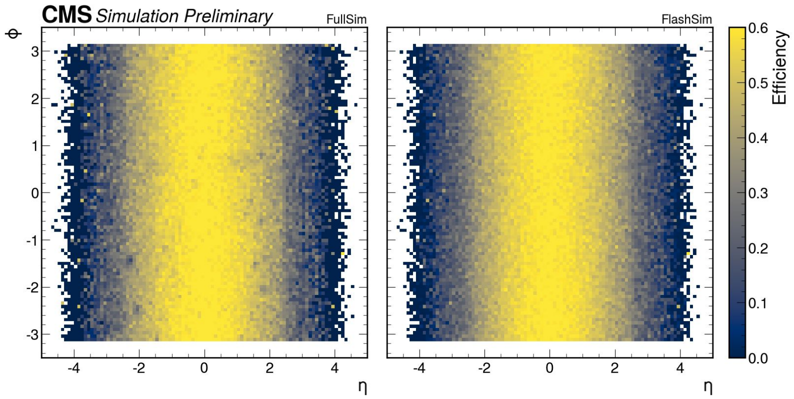


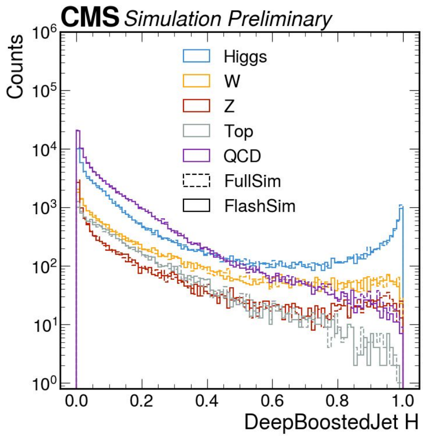




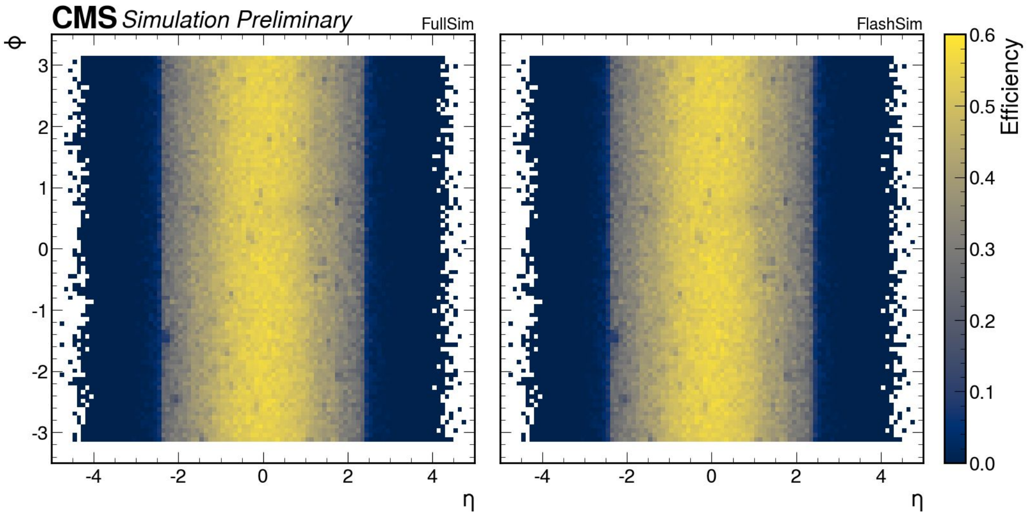
FatJets

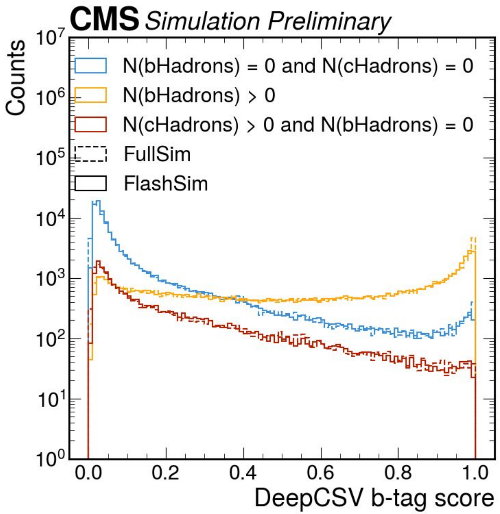

CMS Simulation Preliminary
Efficiency
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
p_T (GeV)
FullSim
FlashSim


CMS Simulation Preliminary
FullSim
FlashSim
ϕ
η
Efficiency
0.6
0.5
0.4
0.3
0.2
0.1
0.0

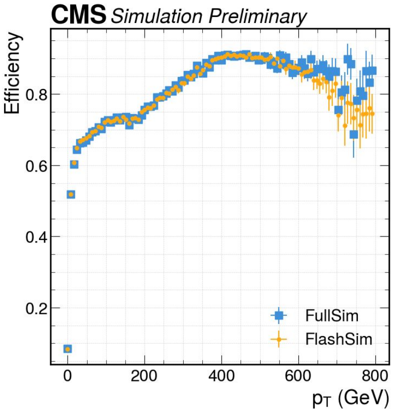

CMS Simulation Preliminary
Counts
Higgs
W
Z
Top
QCD
FullSim
FlashSim
DeepBoostedJet H

SubJets

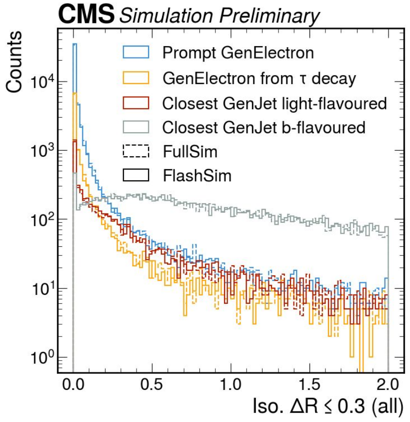

CMS Simulation Preliminary
FullSim
FlashSim
ϕ
η
Efficiency


CMS Simulation Preliminary
Counts
N(bHadrons) = 0 and N(cHadrons) = 0
N(bHadrons) > 0
N(cHadrons) > 0 and N(bHadrons) = 0
FullSim
FlashSim
DeepCSV b-tag score

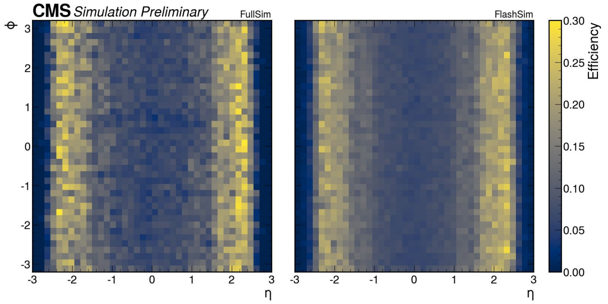
Electrons



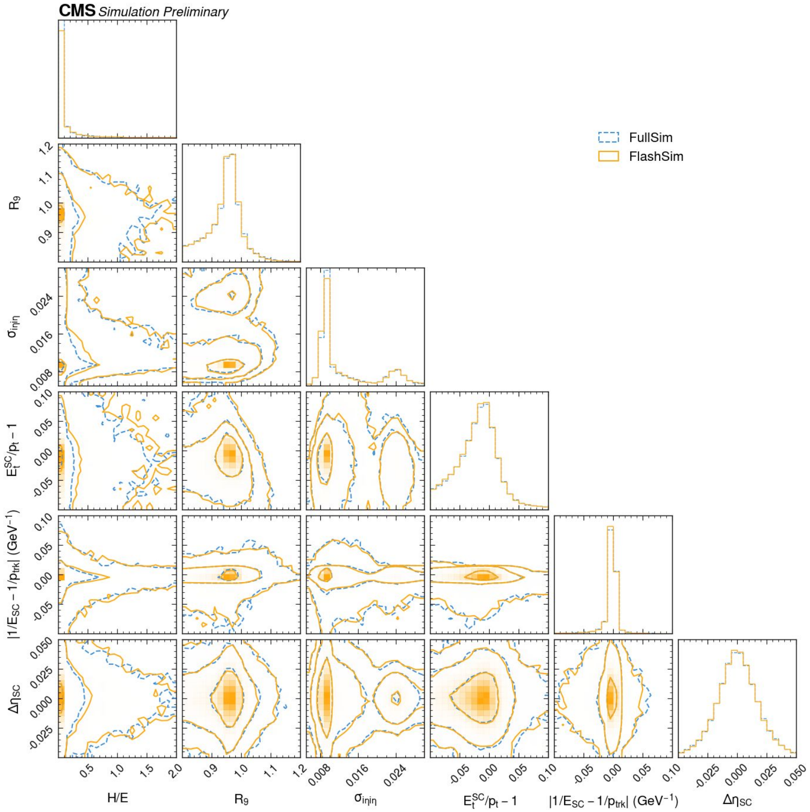
CMS Simulation Preliminary
Efficiency
pT (GeV)
FullSim
FlashSim



CMS Simulation Preliminary
Counts
Prompt GenElectron
GenElectron from τ decay
Closest GenJet light-flavoured
Closest GenJet b-flavoured
FullSim
FlashSim
Iso. ΔR ≤ 0.3 (all)

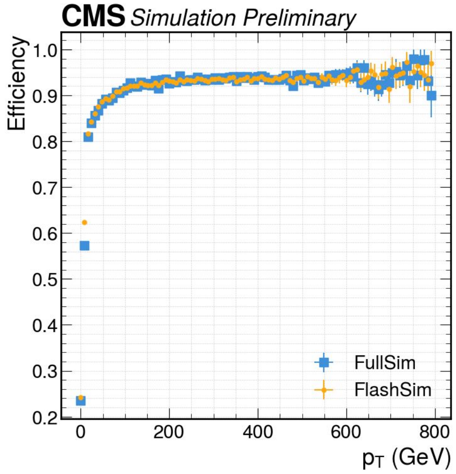


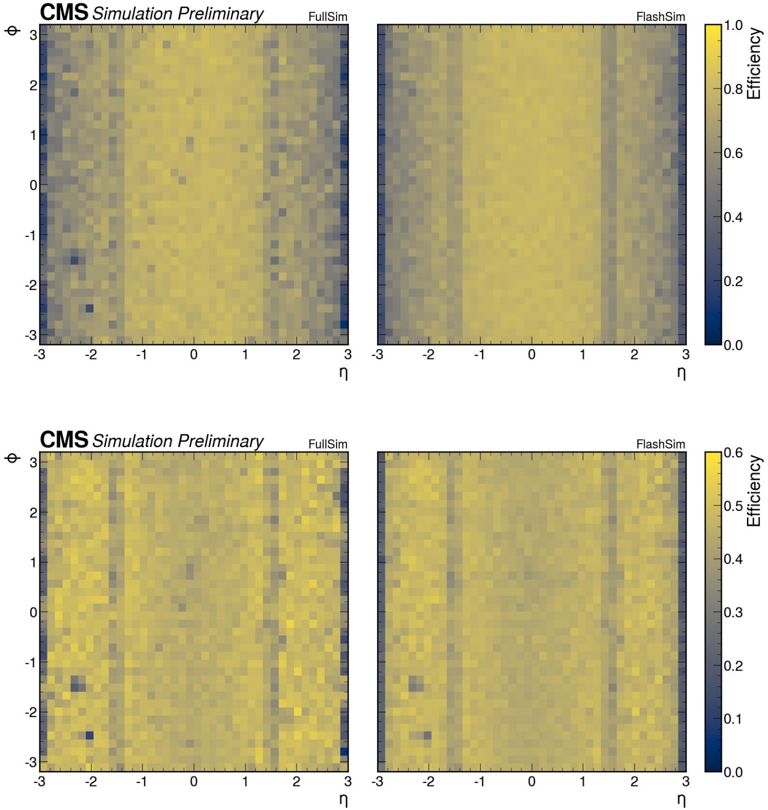
CMS Simulation Preliminary
FullSim
FlashSim
Efficiency
φ
η

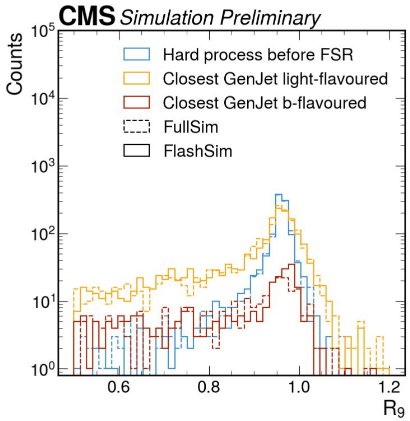


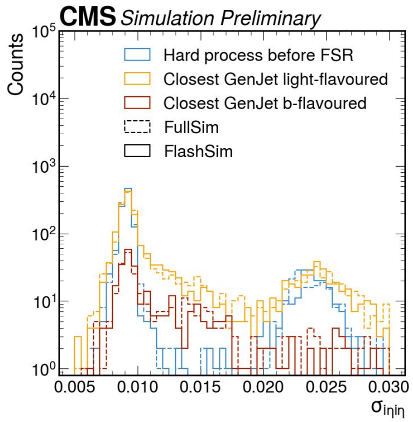
CMS Simulation Preliminary
FullSim
FlashSim
H/E
R9
σiηiη
E_t^SC/p_t − 1
|1/E_SC − 1/p_trk| (GeV⁻¹)
Δη_SC

Photon from generator level photons

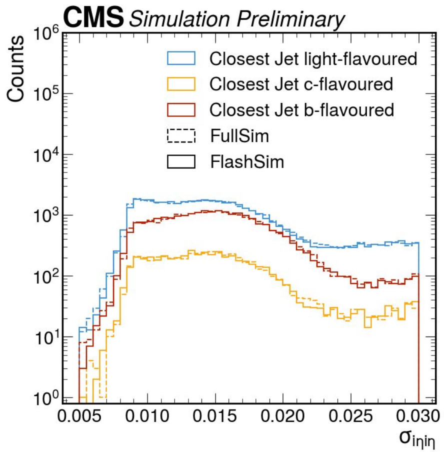


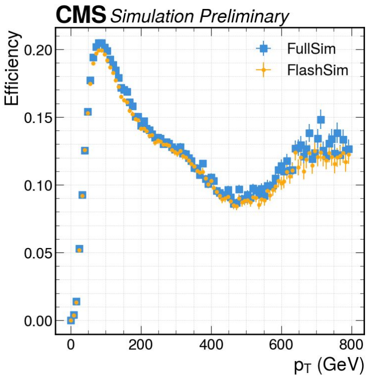


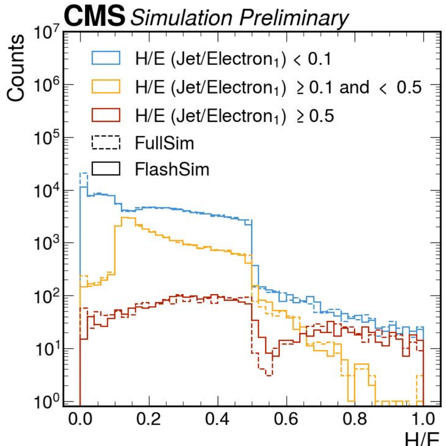


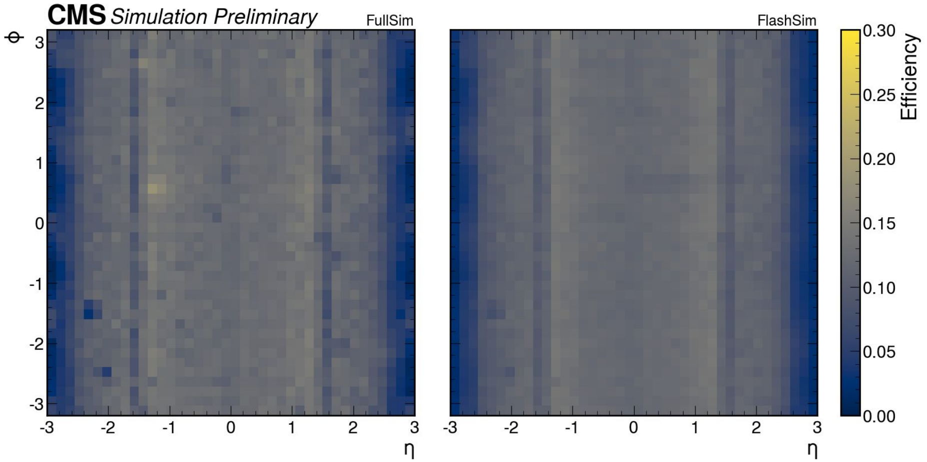


Photon from Jets

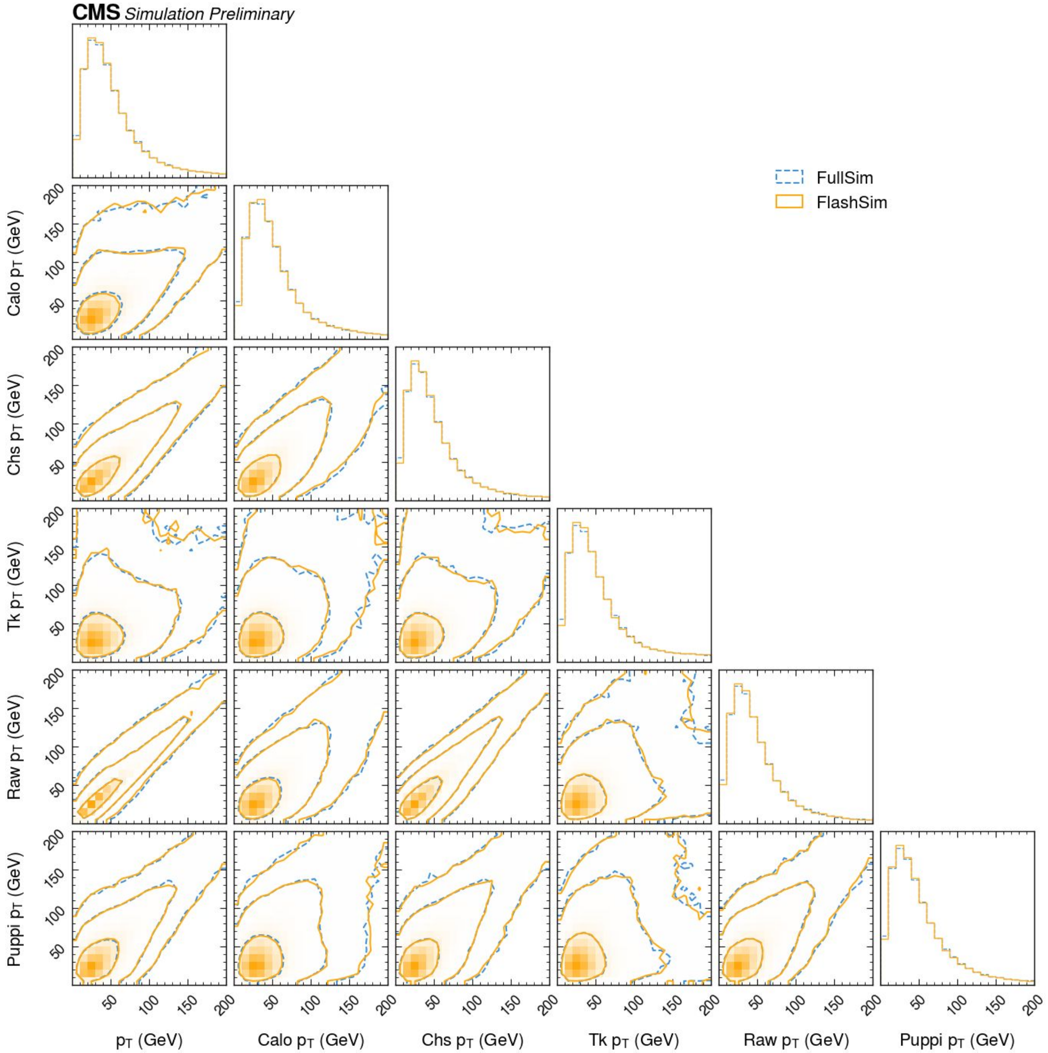




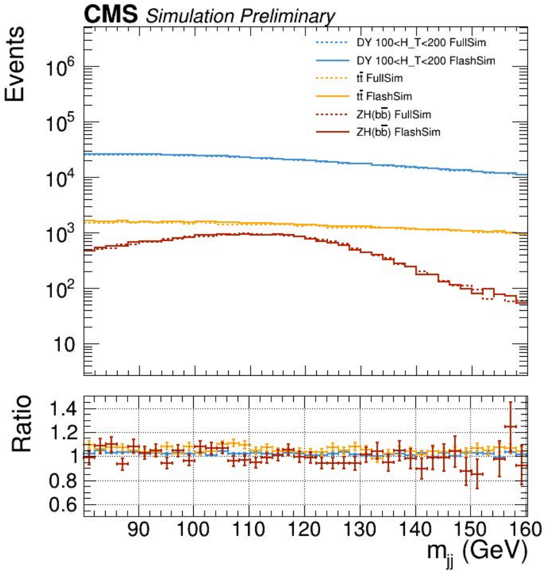




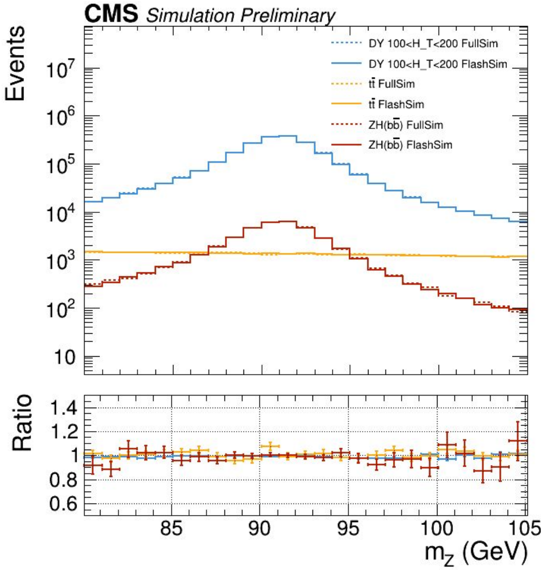
MET


CMS Simulation Preliminary
FullSim
FlashSim
Calo p_T (GeV)
Chs p_T (GeV)
Tk p_T (GeV)
Raw p_T (GeV)
Puppi p_T (GeV)
p_T (GeV)
50
100
150
200

Z(ll)H(bb)

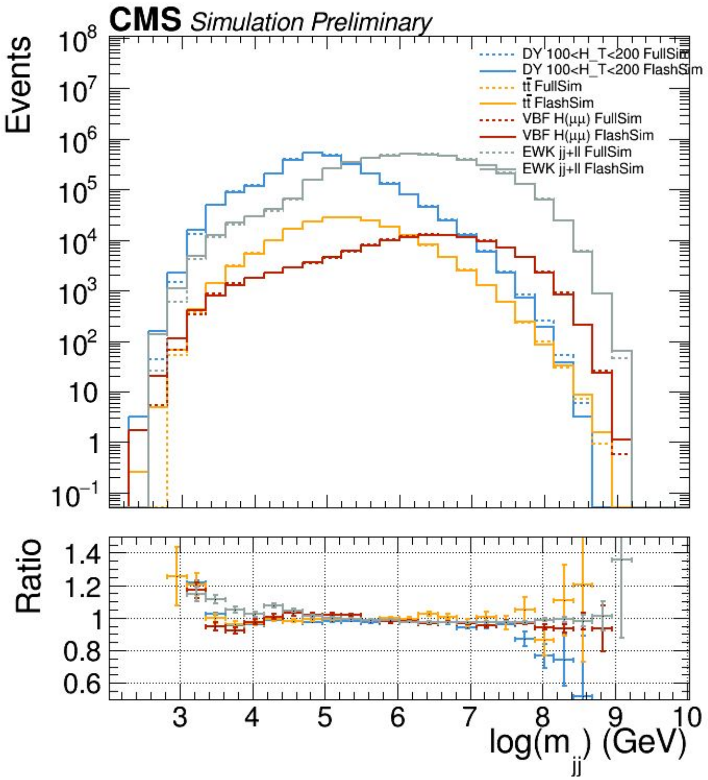


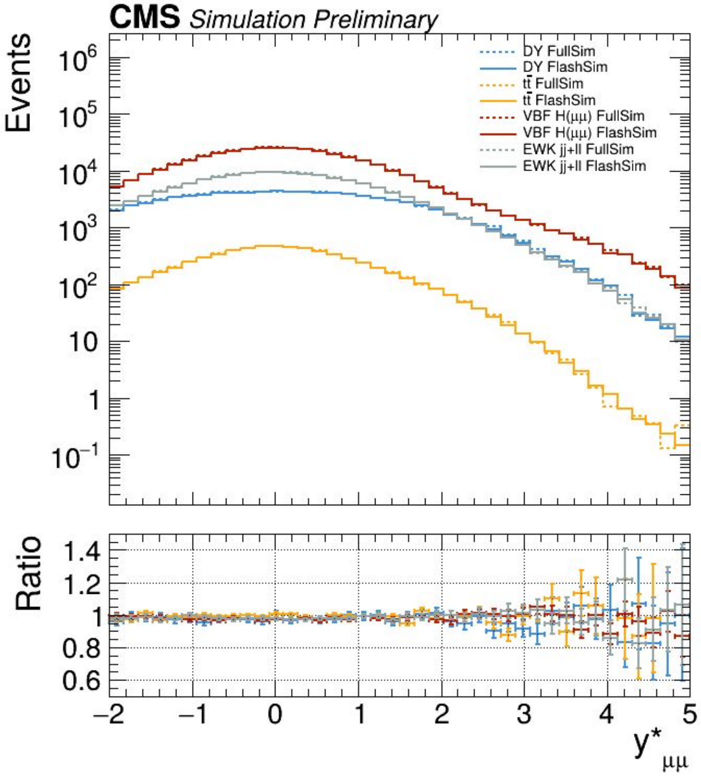
CMS Simulation Preliminary
Events
DY 100<H_T<200 FullSim
DY 100<H_T<200 FlashSim
tt̄ FullSim
tt̄ FlashSim
ZH(bb̄) FullSim
ZH(bb̄) FlashSim
Ratio
m_jj (GeV)



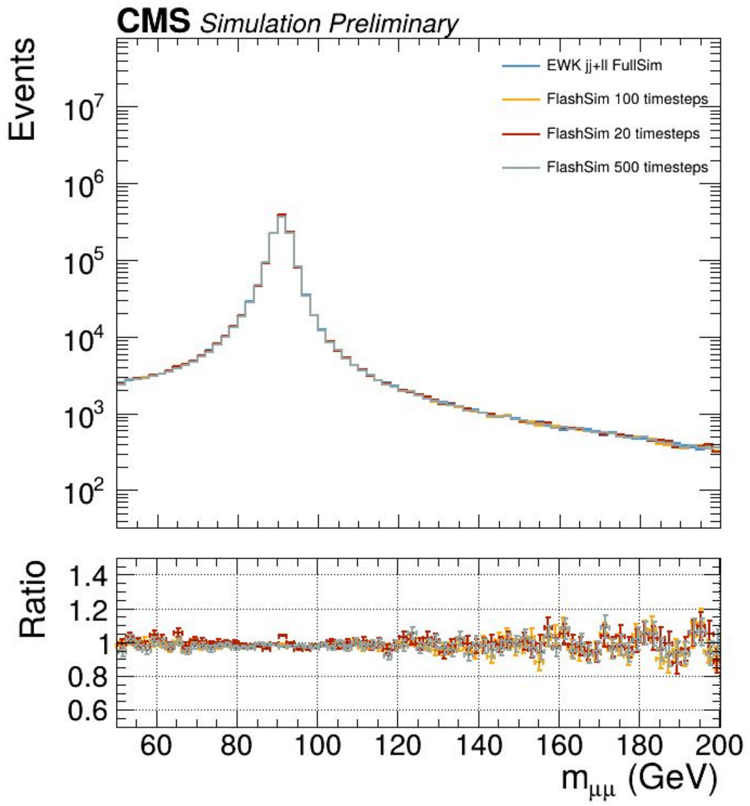
CMS Simulation Preliminary
Events
DY 100<H_T<200 FullSim
DY 100<H_T<200 FlashSim
tt̄ FullSim
tt̄ FlashSim
ZH(bb̄) FullSim
ZH(bb̄) FlashSim
Ratio
m_Z (GeV)

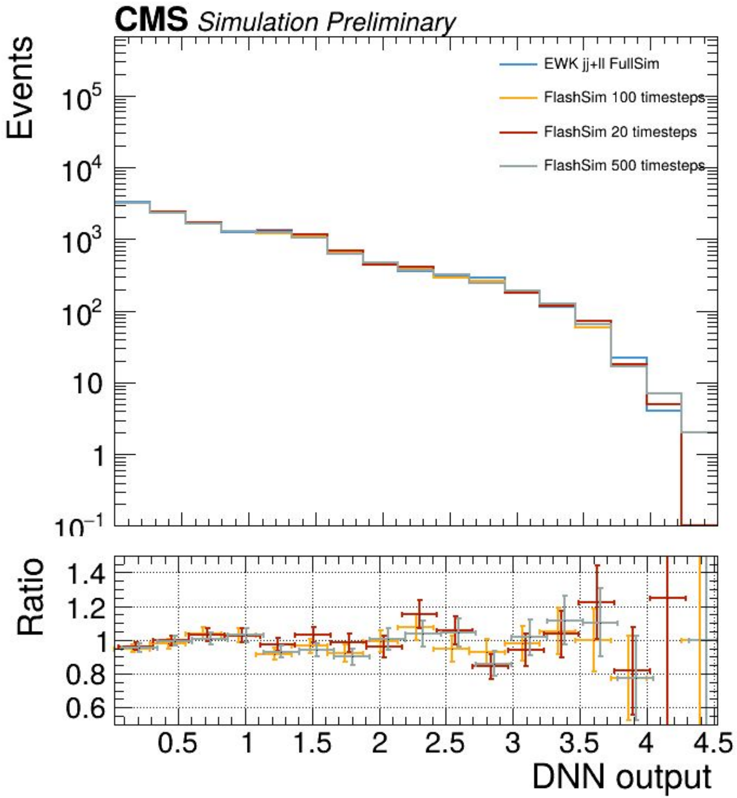
VBF Higgs to mumu





ODE integration accuracy





Flow Matching as a solution

Main idea:

Learn vector field u, approximation of v

u is the field going from noise to data under a Gaussian assumption

t=0: p(z) = N(0,1)

t=1: p(z) = N(x, sigma_min)

$$p_t(z|x) = \mathcal{N}(z|tx, (t\sigma_{\min} - t + 1)^2),$$

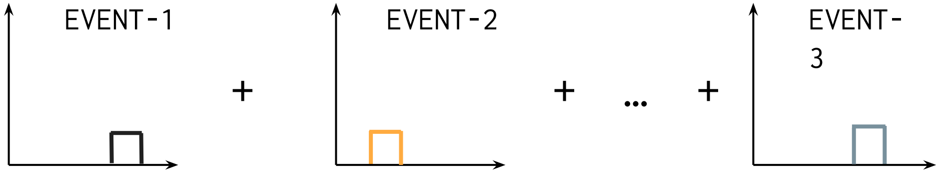
$$u_t(z|x) = \frac{x - (1 - \sigma_{\min})z}{1 - (1 - \sigma_{\min})t},$$

y = NN(x)

Loss = || u - y ||

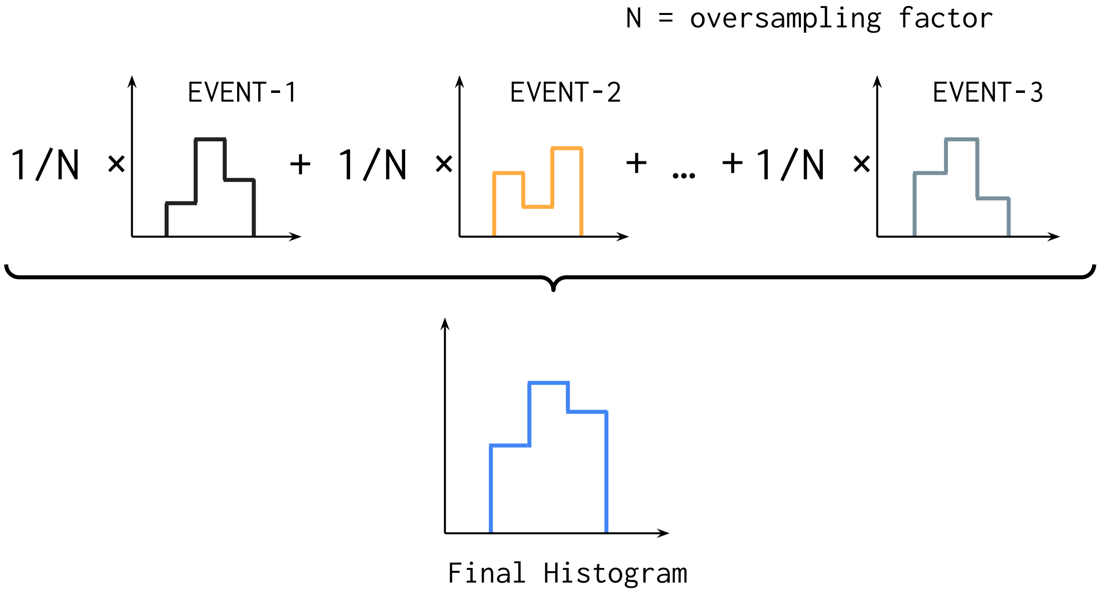
Oversampling: statistical treatment

Usually, a histogram is filled with events (and their weights)



Oversampling → the final histogram is given by the weighted sum of *sub-histograms* filled with the **distributions of events sharing the same GEN**

Note: the final uncertainty is larger than just calling `TH1::Fill()`



Oversampling

- Non-oversampled case
  - w statistical weight associated with the MC event
  - For the *i*-th bin of an histogram, the probability of being in this bin and the associated uncertainty are

$$p_i = \frac{\sum_{j\in\text{bin}} w_j}{\sum_{k\in\text{sample}} w_k} \qquad \sigma_i = \frac{\sqrt{\sum_{j\in\text{bin}} w_j^2}}{\sum_{k\in\text{sample}} w_k}$$

- Oversampled case
  - A *fold* is the set of RECO events sharing the same GEN

$$p_i = \frac{\sum_{j\in\text{bin}} \sum_{l\in\text{fold}\in\text{bin}} w_{jl}}{N\sum_{k\in\text{sample}} w_k} = \frac{\sum_{j\in\text{bin}} \sum_{l\in\text{fold}\in\text{bin}} w_{jl}/N}{\sum_{k\in\text{sample}} w_k} \equiv \frac{\sum_{j\in\text{bin}} w_j p_j^{\text{fold}}}{\sum_{k\in\text{sample}} w_k}$$

$$\sigma_i = \frac{\sqrt{\sum_{j\in\text{bin}} (w_j p_j^{\text{fold}})^2}}{\sum_{k\in\text{sample}} w_k}$$

Discrete Flows: *transforms*

How do we transform the variables?
Various ways to do it (as long as the transformation is invertible!)

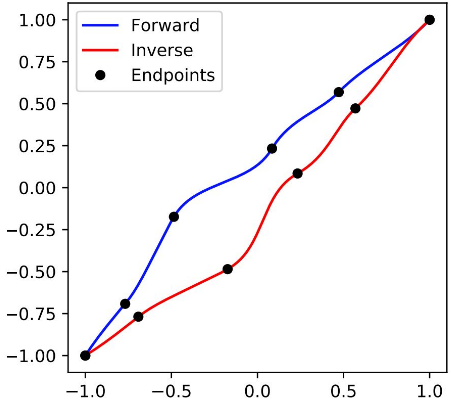
Each model is made up of multiple conditioner+transformation blocks

This gives us an expressive final transformation with good correlations
between variables

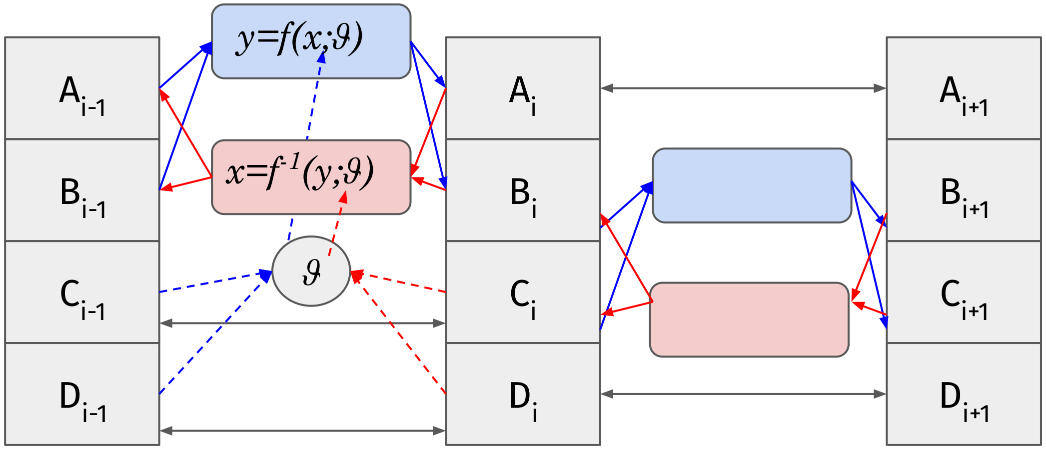
Affine:

$$\tau(z_i; \boldsymbol{h}_i) = \alpha_i z_i + \beta_i$$

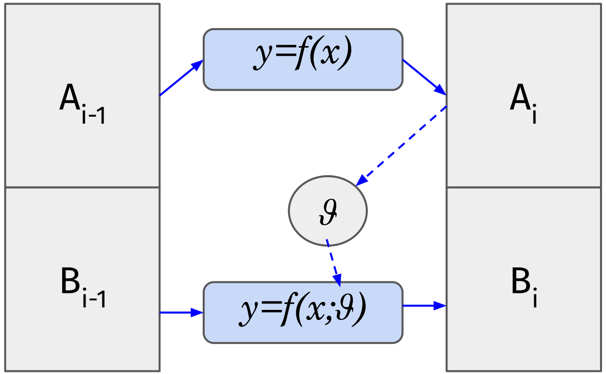
Splines:

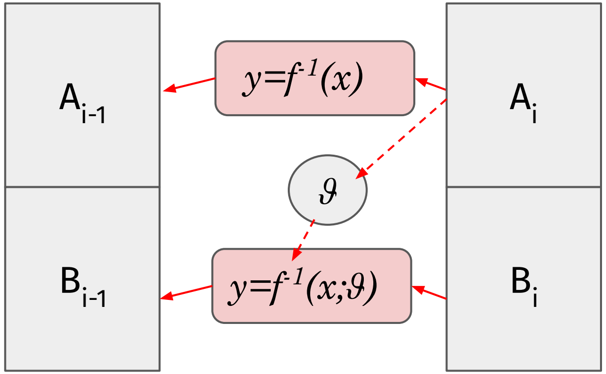


Coupling flow



Autoregressive flow





Some technical info

FlashSim uses the following packages and tools

- RDataFrame
- numpy
- pytorch
- `torchcfm` and `nflow`

Data transfer RDF <--> numpy not yet fully efficient (would benefit from RDF "custom batch process" capabilities)