Accelerating Particle Physics Simulations

with Normalizing Flows and Flow Matching

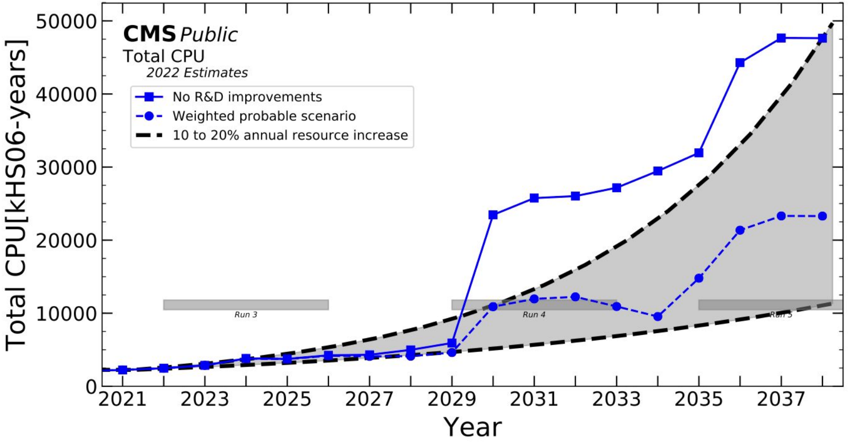
Francesco Vaselli, Filippo Cattafesta, Patrick Asenov, Andrea Rizzi
INFN, Scuola Normale Superiore & University of Pisa

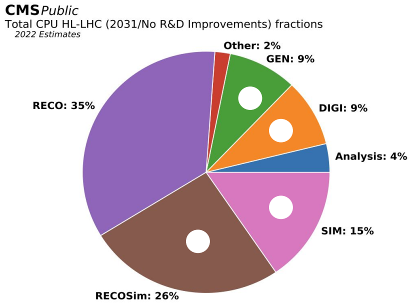
We need Faster simulation frameworks!

Event simulation is a non-negligible fraction of the total projected CPU need

Faster simulation frameworks are a part of the solution to the computing challenges posed by the HL-LHC era

Machine learning is expected to provide both the speed and the accuracy we need



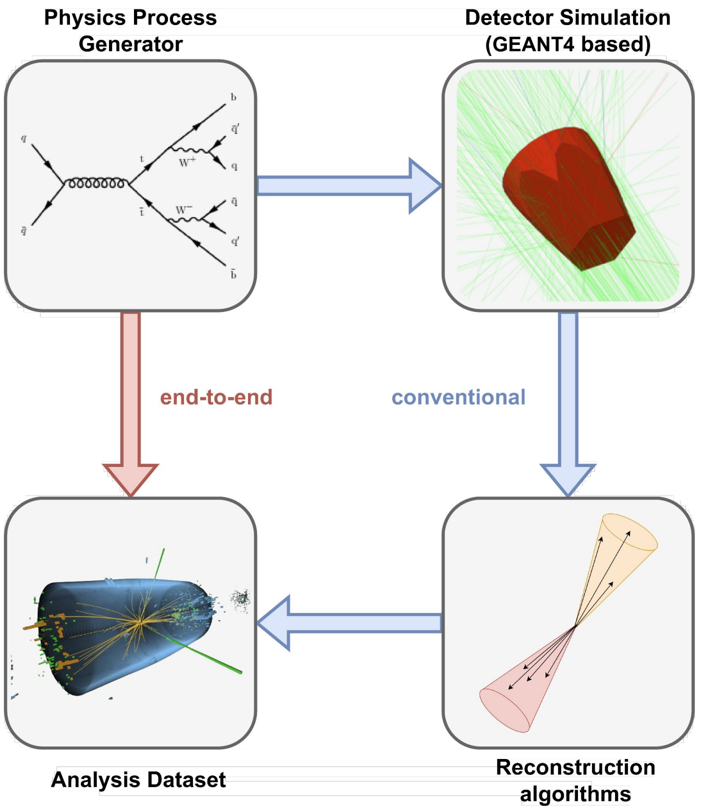


We propose to go *end-to-end*

Main idea: going directly from the generator output objects to the high level analysis objects (jets, muons ...)!

We want something:

- Fast(er): reached ~kHz!
- Not analysis specific
- Depending on Gen (not just a generic event but the event)



We select particle jets as our benchmark

From generator-level jets to analysis level

Build pseudo-realistic dataset with Pythia and physically reasonable response functions

6 generator inputs:

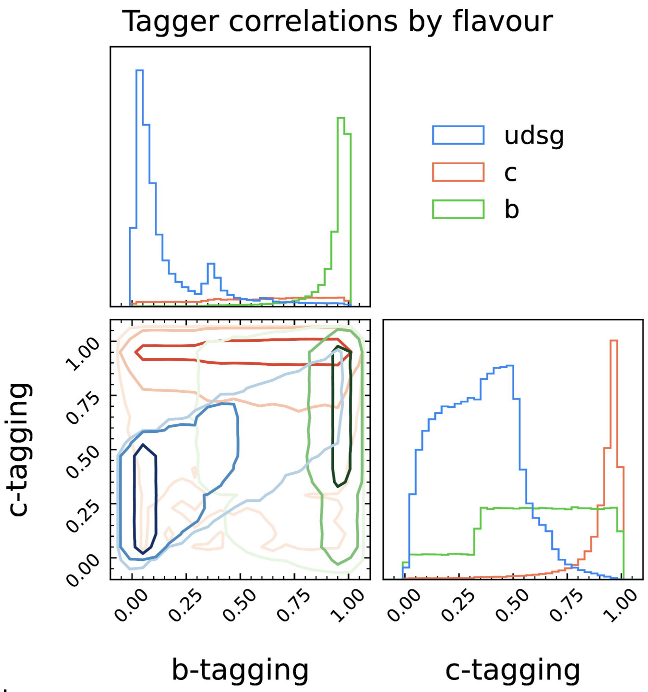
Kinematic, flavour, mass, N muons in jet

16 high-level targets:

Kinematic, mass, b/c-taggers, energy fractions, secondary vertices ...

6 different metrics to evaluate each model:

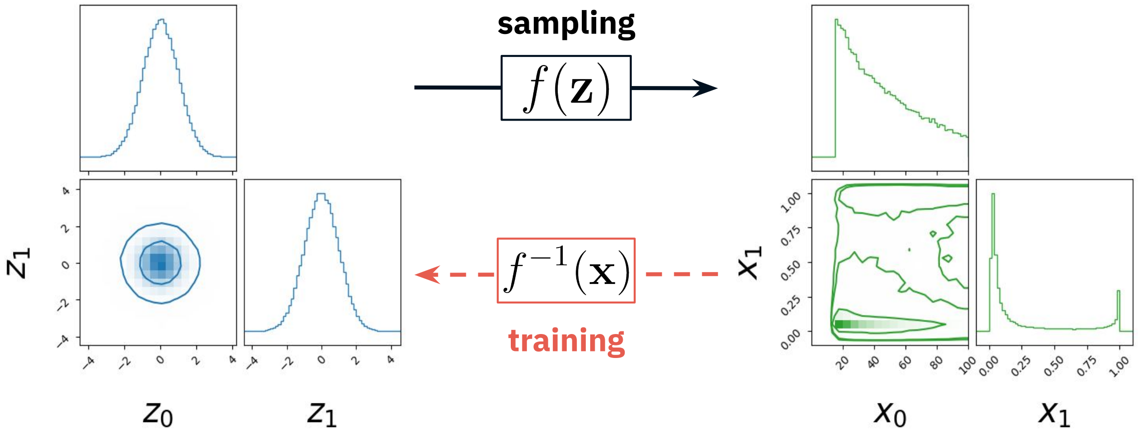
Wasserstein, KS, Covariance Matching, Fréchet, Area Between ROC, c2st



Normalizing Flows are the backbone of our approach!

We learn an invertible transformation, taking us from data *x* to noise *z*

Once *f* has been found we can invert it, start from noise and sample new data from the unknown PDF!



$$\begin{cases} \mathbf{x} = f(\mathbf{z}) \\ p_x(\mathbf{x}) = p_z(\mathbf{z}) \det \left| \dfrac{d\mathbf{z}}{d\mathbf{x}} \right| \end{cases}$$

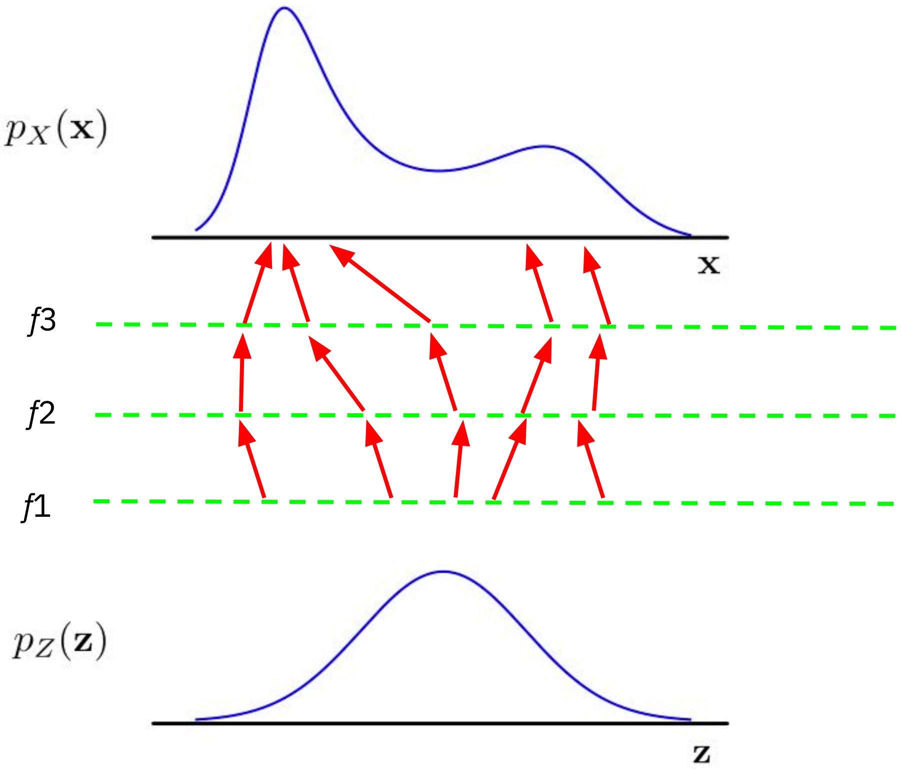
Discrete flows use a series of discrete functions

Each model is made up of multiple
transformation blocks

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

Affine transform:

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



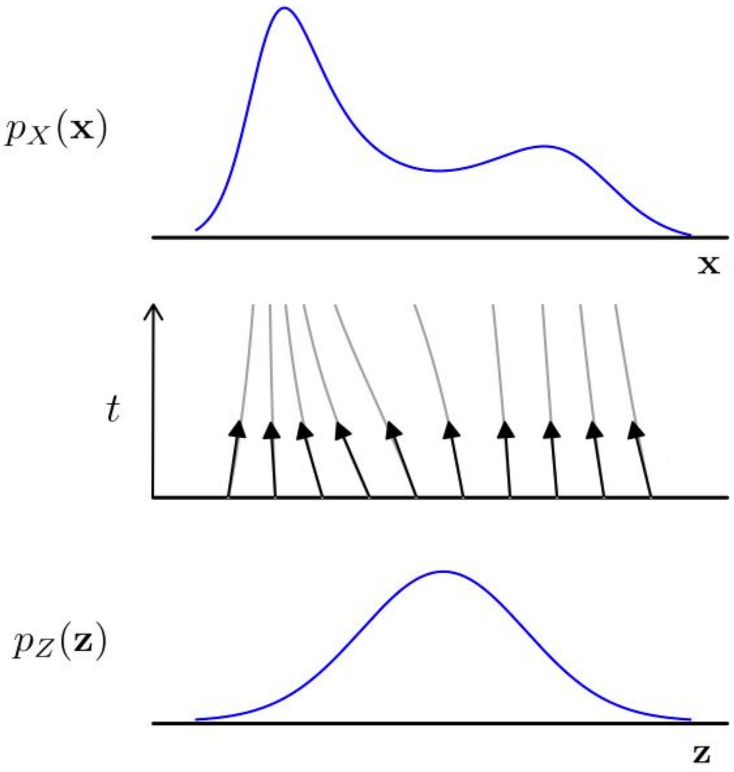
Continuous flows learn a vector field!

We learn a single transformation parametrized by *t*, and then we integrate on it to get the data!

$$f(z) = z + \int_0^1 dt v_t$$

$$f^{-1}(x) = x + \int_1^0 dt v_t$$

Problem:
How do we learn *v*?
Solution:
Flow Matching



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

Continuous flows are the best class of models

Best values across every validation metric

More accurate results with fewer parameters

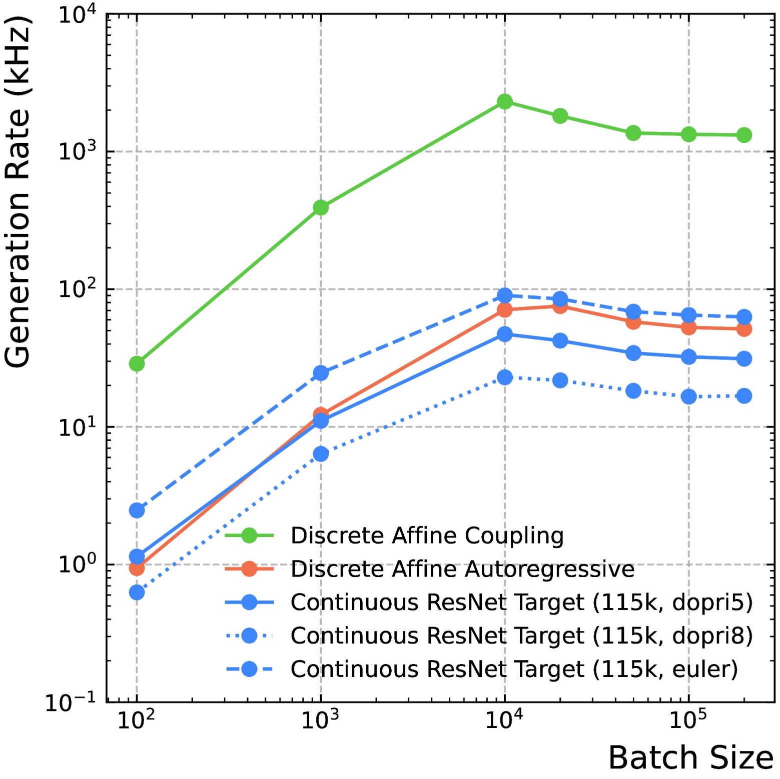
Trained on 500k jets

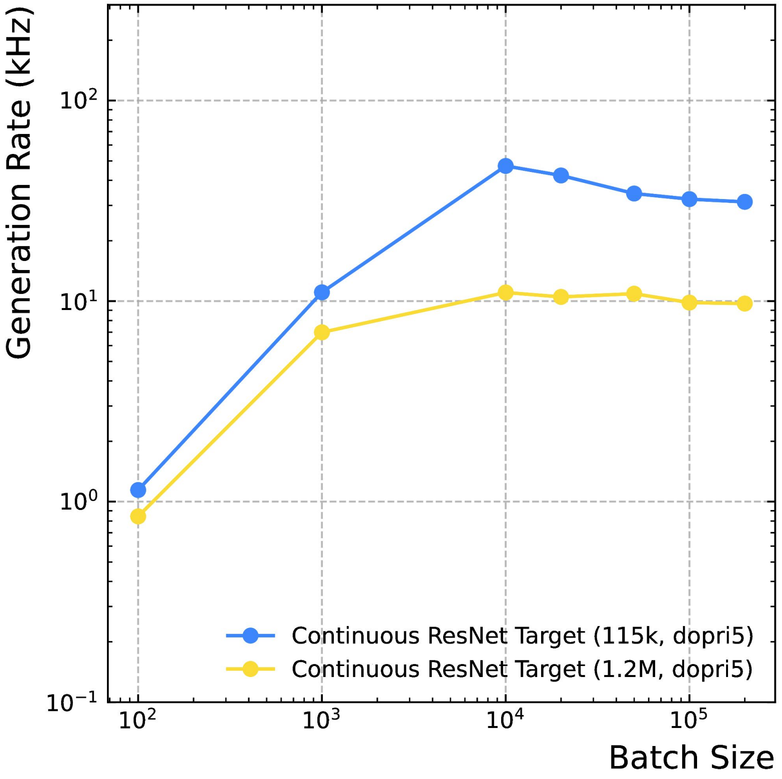
Validated on a separate split of 650k

Heatmap of Models and Metrics

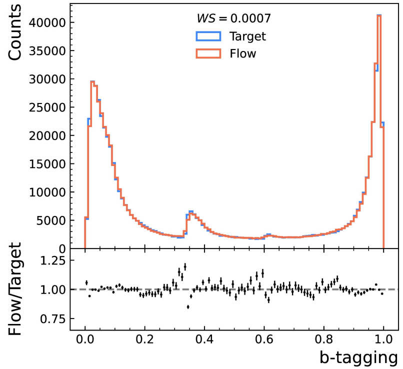
Model	FGD	CM	<KS>	<WS>	ABC	c2st	Avg
CRT (115k)	0	0	0	0	0	0	0
CRT bigger (1.2M)	-0.24	-0.02	-0.02	0.16	0.15	-4.8	0.8
CRB no α (112k)	-0.87	-0.55	0	-0.1	-0.07	-5.3	0.62
CRB (112k)	-0.4	-0.11	-0.02	0.13	-0.45	-0.99	0.02
CMB (88k)	-3.2	-0.37	-0.03	-0.71	-0.29	-7.6	0.5
CMB small (20k)	-2.3	-1.2	0.01	-0.48	-1.1	-5	-0.01
CMB small σ (88k)	-0.99	-0.3	0.02	-0.58	-0.64	-4.4	0.32
DAA gelu (0.9M)	-28	-43	-0.08	-10	-4.2	-5.3	-13
DAA silu (0.9M)	-6.9	-2.3	-0.44	-6.7	-5.1	-1.8	-3.3
DAC gelu (0.7M)	-22	-24	-1.5	-36	-6.6	-13	-13
DAC silu (0.7M)	-5.7	-7.3	-0.1	-5.8	-3.9	-23	0.1

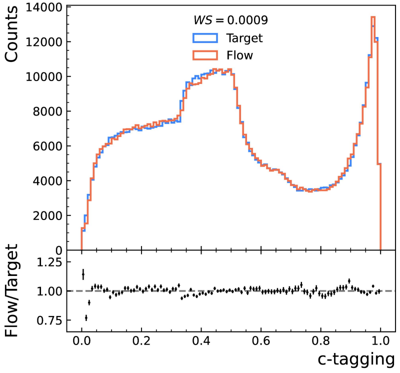
Speed comparison shows promising results

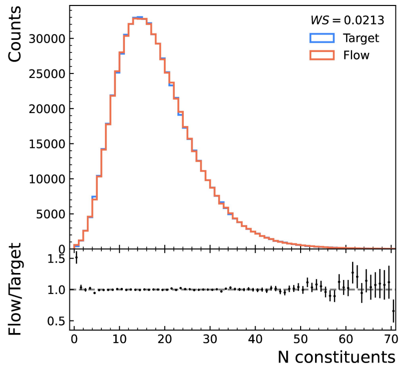


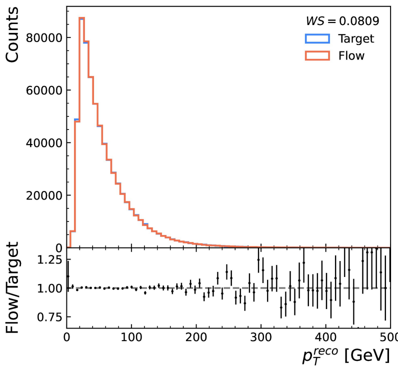


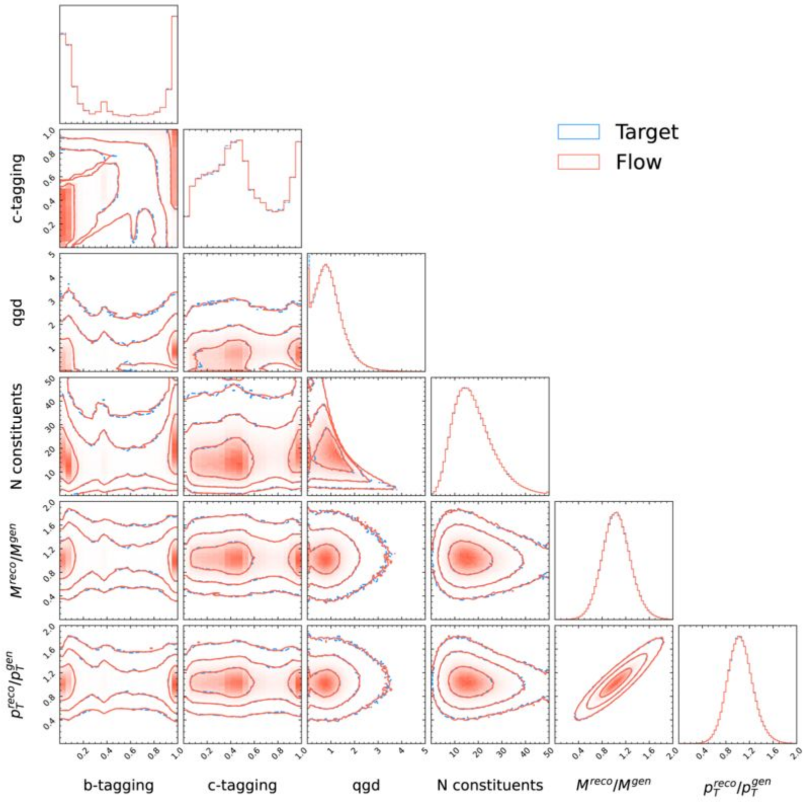
Good convergence achieved on 1d and correlations



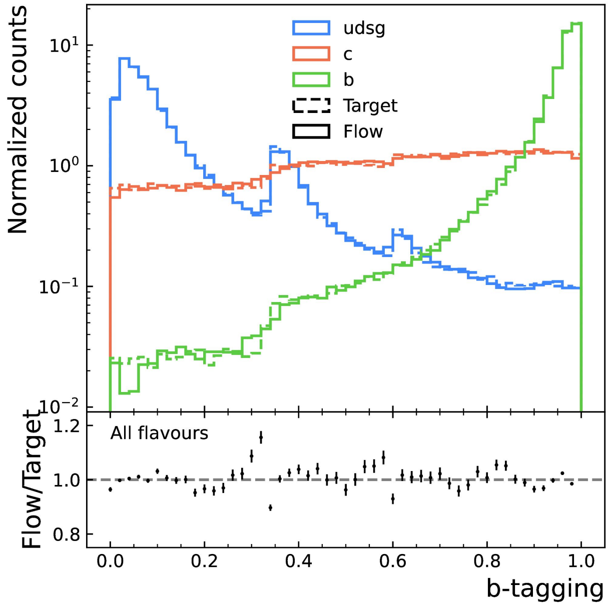


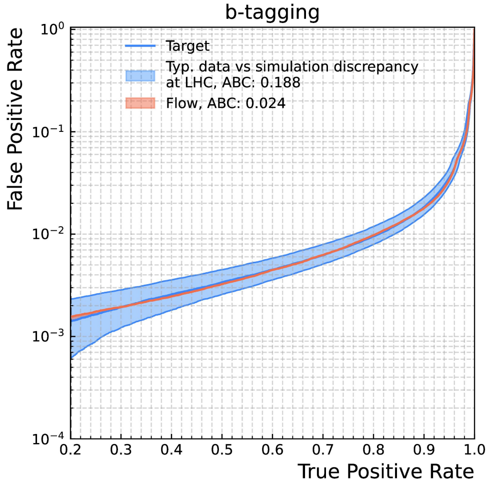


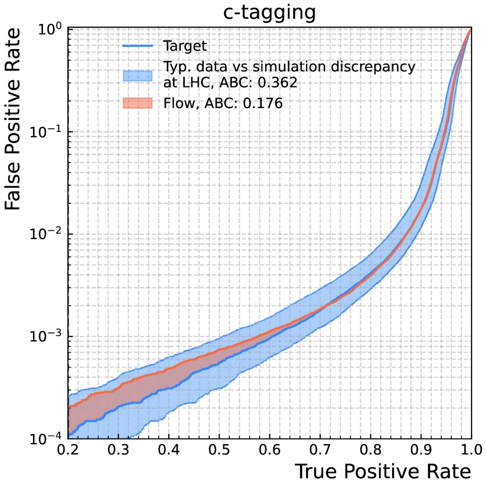




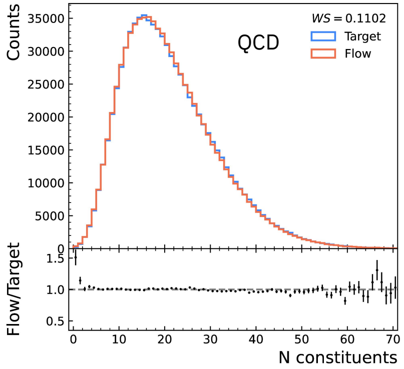
The input information is correctly taken into account

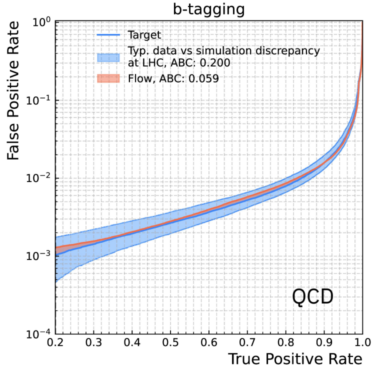


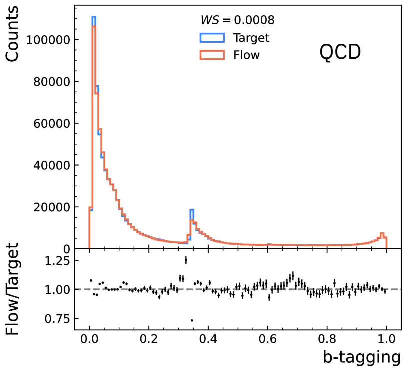


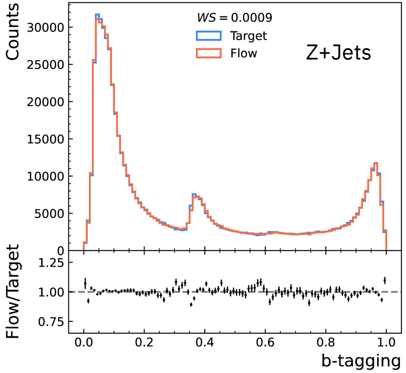


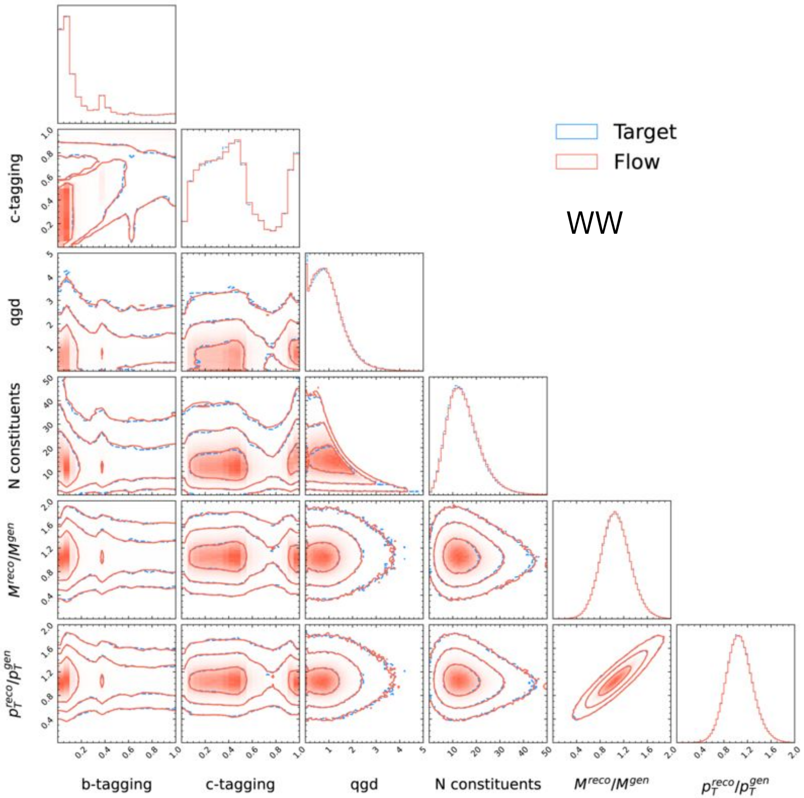
The performance is conserved on other processes (no retrain)!











Increasing the size of a dataset

Method 1 — **ONE-TO-ONE EVENT SIMULATION**

The generation uses a fraction of the CPU resources compared to conventional

Given ~kHz per object, the generator could be a bottleneck

GEN $\longrightarrow$ SIM

GEN $\longrightarrow$ SIM

... ...

GEN $\longrightarrow$ SIM

GEN $\longrightarrow$ SIM

Increasing the size of a dataset

Method 1 — **ONE-TO-ONE EVENT SIMULATION**

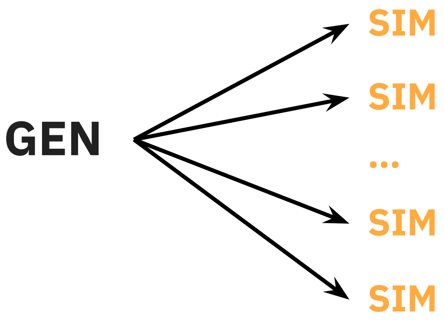
The generation uses a fraction of the CPU resources compared to conventional

Given ~kHz per object, the generator could be a bottleneck

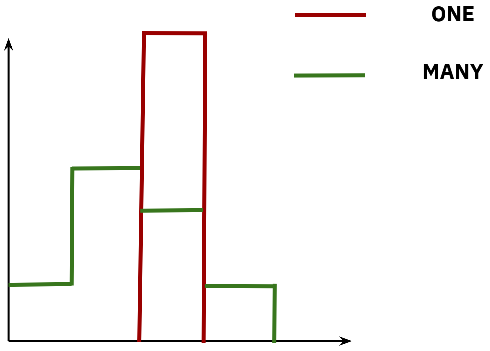
Method 2 — **OVERSAMPLING**

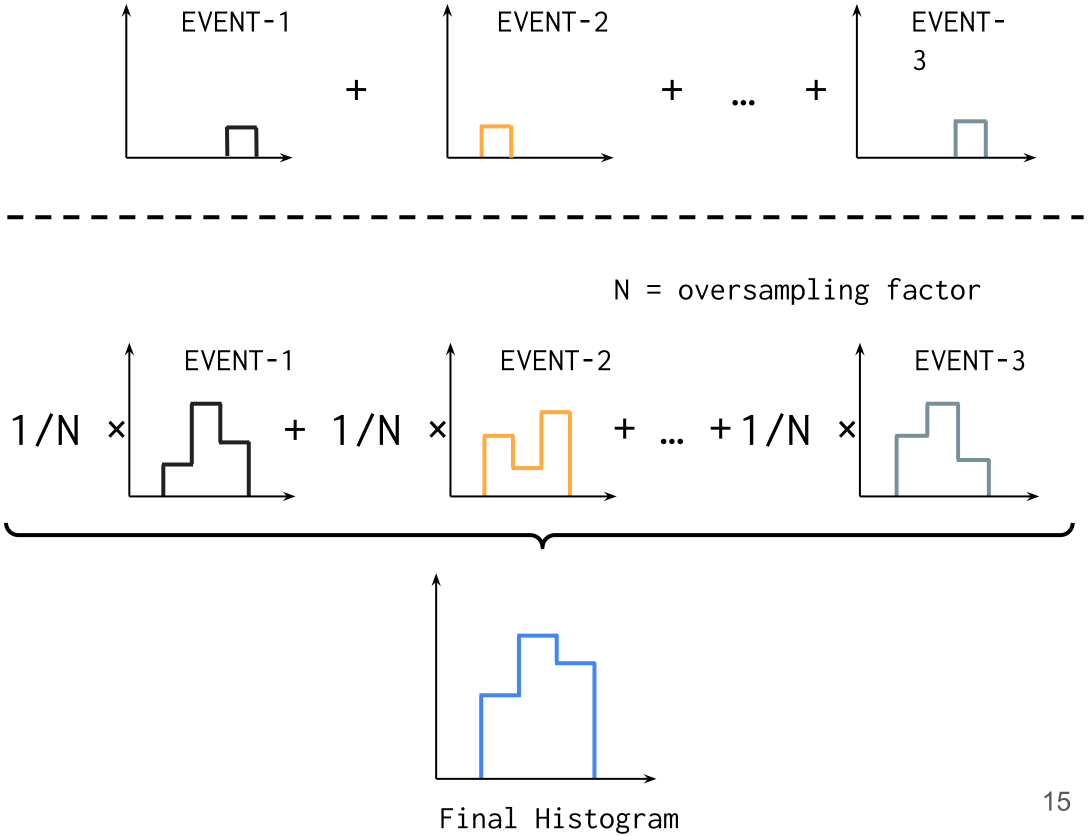
Simulate multiple SIM events using the same GEN event as input

Need to handle correlations!



Oversampling: statistical treatment

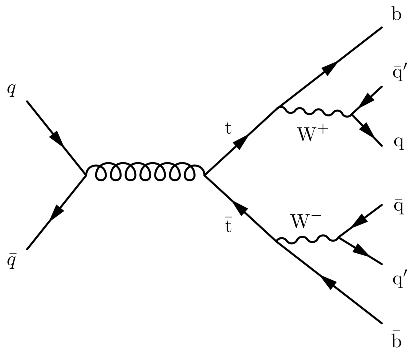


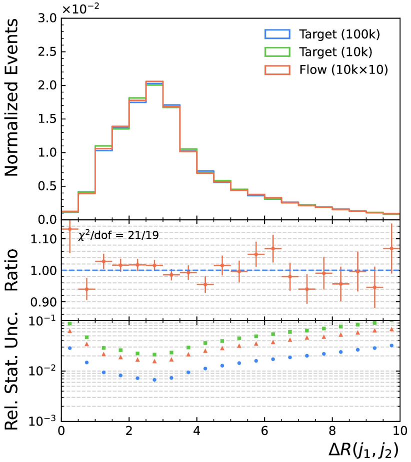


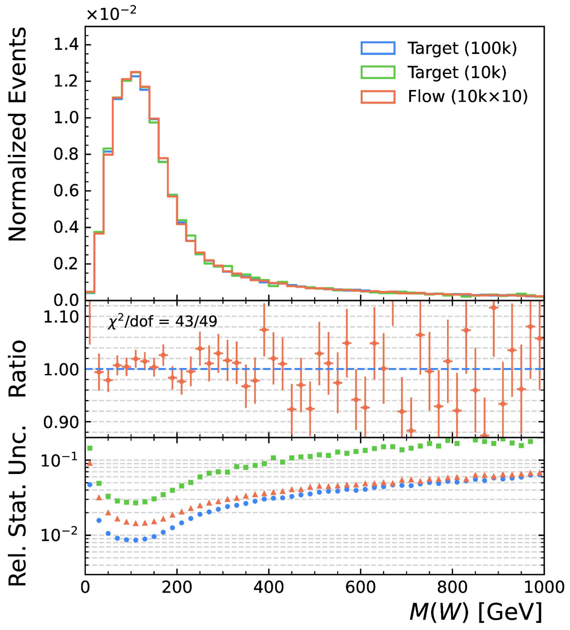
Oversampling → the final histogram is given by the weighted sum of *sub-histograms* filled with the distributions of events sharing the same GEN

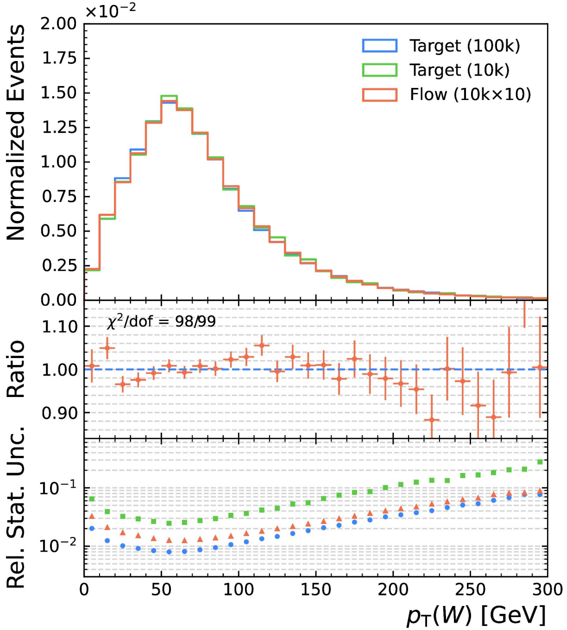
Note: the final uncertainty is larger than just filling the histogram N more times

Results on pseudo-analysis of W mass









Conclusions

Normalizing Flows are a powerful tool for HEP *end-to-end* simulation, with several orders of magnitude of speed-up

If we generate fast enough, we can use oversampling to reduce the uncertainties of the sample!

Paper here: https://arxiv.org/abs/2402.13684

Repo here: https://github.com/francesco-vaselli/FlowSim

If you have any questions, get in touch!

francesco.vaselli@cern.ch

Backup

- timing table
- flow details, loss
- oversampling details
- variables list
- metrics details
- training on more data
- more plots
- Flow matching

Time estimates

Table 2: Comparison of millions of events produced per day on a single 4 GPU computing node in different scenarios and their ratio to a conventional simulation scenario taking 20 seconds per event.

			Millions of events per day on a HPC Node						*Ratio to Conventional sim*				
Generator	**Gen time s/event**	**Fold size**	**Conventional (20s/event)**	**Object sampling speed [kHz] 1**	**5**	**10**	**50**	**100**	**Object sampling speed [kHz] 1**	**5**	**10**	**50**	**100**
Existing	0	1	0.138	17.3	86.4	172.8	864.0	1728.0	125	625	1250	6250	12500
Simple	0.02	1	0.138	15.4	53.2	76.8	119.2	128.0	111	385	556	863	927
		10	0.138	17.1	81.3	153.6	531.7	768.0	123	588	1111	3847	5556
Average	1	1	0.132	2.4	2.7	2.7	2.8	2.8	18	20	21	21	21
		10	0.138	10.6	20.9	23.8	26.8	27.2	77	152	173	195	198
Accurate and slow	20	1	0.069	0.14	0.14	0.14	0.14	0.14	2	2	2	2	2
		10	0.126	1.28	1.4	1.4	1.4	1.4	10	11	11	11	11

Flow details and loss

$$p_x(\mathbf{x}) = p_z(f^{-1}(\mathbf{x})) \det \left| \frac{d\mathbf{z}}{d\mathbf{x}} \right|$$

Invertible transform

Jacobian for Volume Correction

$$\log(p_x(x)) = \log(p_z(f^{-1}(\mathbf{x}))) + \log\left(\det \mathbb{J}_{f^{-1}}(\mathbf{x})\right)$$

$$\mathcal{L}(\phi) = -\mathbb{E}_{p_x^*(\mathbf{x})}[\log(p_z(f^{-1}(\mathbf{x};\ \phi))) + \log\left(\det \mathbb{J}_{f^{-1}}(\mathbf{x};\ \phi)\right)]$$

where ϕ are the parameters of *f(z)*

see https://arxiv.org/abs/2210.02747
and https://arxiv.org/abs/2302.00482

Flow Matching as a solution

Learn vector field u,
approximation of v

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

see https://arxiv.org/abs/2210.02747
and https://arxiv.org/abs/2302.00482

Flow Matching as a solution

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 ⇢ $p(z) = N(x, \sigma_{\min})$

see https://arxiv.org/abs/2210.02747
and https://arxiv.org/abs/2302.00482

Flow Matching as a solution

Learn vector field *u*, approximation of *v*

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

t=0 $\dashrightarrow$ p(*z*) = N(0,1)

t=1 $\dashrightarrow$ p(*z*) = N(*x*, $\sigma_{\min}$)

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

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

see https://arxiv.org/abs/2210.02747
and https://arxiv.org/abs/2302.00482

Flow Matching as a solution

Learn vector field *u*,
approximation of *v*

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

y = NN(x)
Loss = (u - y)**2
Simple regression!

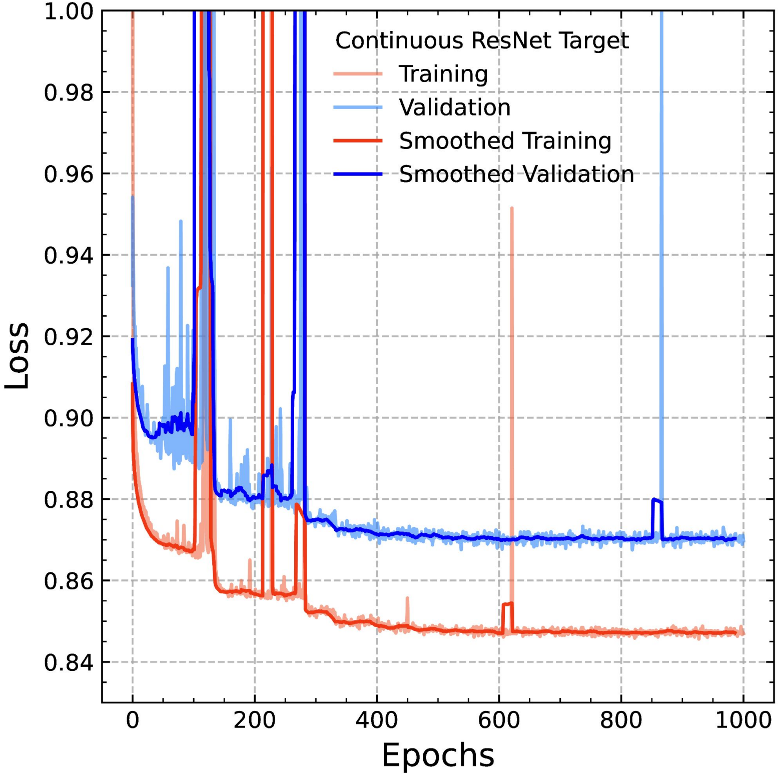
t=0 $\dashrightarrow$ p(z) = N(0,1)

t=1 $\dashrightarrow$ p(z) = N(x, $\sigma_{\min}$)

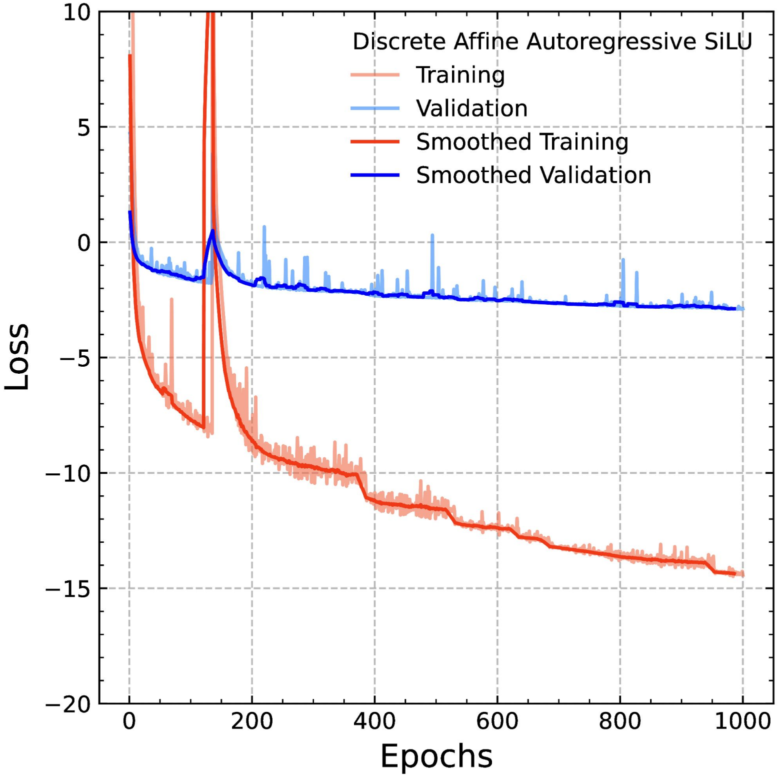
$$p_t(z|x) = \mathcal{N}(z|tx, (t\sigma_{\min} - t + 1)^2),$$

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

Losses



Continuous ResNet Target
Training
Validation
Smoothed Training
Smoothed Validation
Loss
Epochs
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0
200
400
600
800
1000



Discrete Affine Autoregressive SiLU
Training
Validation
Smoothed Training
Smoothed Validation
Loss
Epochs
10
5
0
−5
−10
−15
−20
0
200
400
600
800
1000

Oversampling: statistical treatment

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

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

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

Variables list

Table 1: The two datasets used in this work: one with 6 input generator-level variables and 5 target reco-level variables; an extended one with the same inputs and 16 target reco variables in total.

Generator level variables	**Description**
p_T, η, ϕ, mass	Kinematic properties of the generated jet
jet flavour	Distinguishing b, c jets from light quarks or gluon jets
number of μ in jet	Counting the number of muons within the jet radius
Basic reconstructed variables	**Description**
p_T, η, ϕ, mass	Kinematic properties of the reconstructed jet
b-tagging discriminator	Score in [0,1] mimicking a tagging algorithm
number of constituents	Counting the number of reconstructed jet constituents
Extended dataset variables	**Description (in addition to basic variables)**
Neutral Hadron Fraction (nhf)	fraction of jet energy carried by Neutral Hadrons
Charged Hadron Fraction (chf)	fraction of jet energy carried by Charged Hadrons
Neutral Electromagnetic Fraction (nef)	fraction of jet energy carried by photons and π^0 mesons
Charged Electromagnetic Fraction (cef)	fraction of jet energy carried by electrons
Quark-Gluon discriminator (qgd)	Discriminator score mimicking a quark/gluon tagging algorithm
Jet Identification (jetId)	Discriminator score mimicking a jet Identification algorithm
Number of Charged Particles (ncharged)	Number of reconstructed charged particles
Number of Neutral Particles (nneutral)	Number of reconstructed neutral particles
c-tagging discriminator	Score of c-tagging algorithm, correlated with b-tagging
Number of Secondary Vertices (nSV)	Poisson distributed number of Secondary Vertices in jets

Metrics I

- The 1-d *Wasserstein* score (WS) [36] and the two-sample *Kolmogorov-Smirnov* distance (KS) for comparing 1-d distributions between the target and the samples produced by the model. A WS is assigned to each variable.
- The *Fréchet* distance as a global measure. It is the distance between Multivariate Gaussian distributions fitted to the features of interest, which [36] calls the Fréchet Gaussian Distance (FGD). It is generally called the Fréchet Inception Distance (FID) in image generation tasks:

$$d^2(x, y) = \|\mu_x - \mu_y\|^2 + \mathrm{Tr}(\Sigma_x + \Sigma_y - 2(\Sigma_x \Sigma_y)^{1/2}). \tag{8}$$

- *Covariance matching*: another global metric used to measure how well an algorithm is modelling the correlations between the various target features. Given the covariance matrices of the two samples, target and model, we compute the *Frobenius Norm* of the difference between the two:

$$||\mathrm{Cov}(X_{\mathrm{target}}) - \mathrm{Cov}(X_{\mathrm{model}})||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |c_{ij}^{\mathrm{t}} - c_{ij}^{\mathrm{m}}|^2}. \tag{9}$$

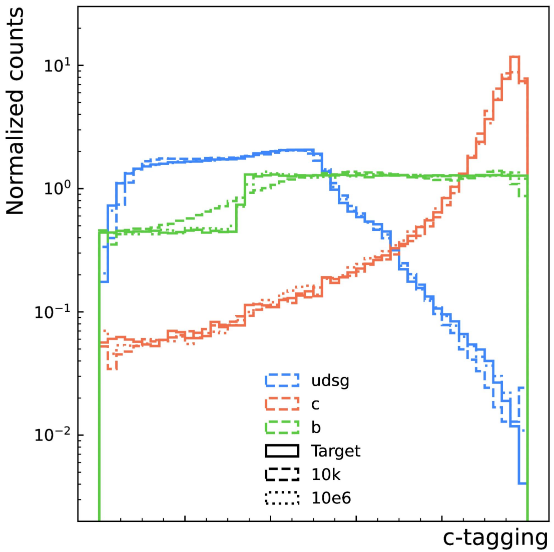
Correlations in the model samples are also visually evaluated through the use of dedicated plots.

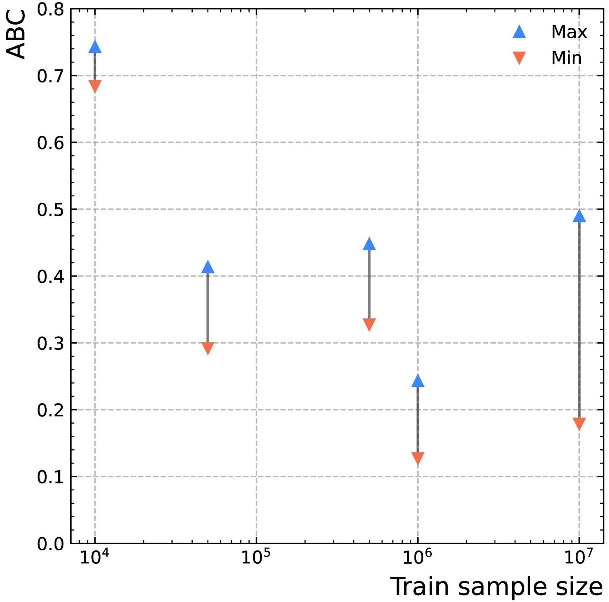
Metrics II

- As b and c-tagging are such important tasks in the study of jets, we compute the *receiver operating characteristic* (ROC) curves for both scores. To quantify the performance of a model, we compute the difference in log-scale between the ROC coming from the model and that from the target distribution. Log-scale is used because the true positive rate (TPR) and false positive rate (FPR) span different orders of magnitude. We call this evaluation metric the *Area Between the Curves* (ABC).
- Finally, we implement a *classifier two-sample test* (c2st): we train a classifier to distinguish between training samples and samples coming from our models, giving as additional input the gen information. The output is the percentage P_{c2st} of samples which were *incorrectly* classified. For the optimal model, it has a maximum value of 0.5. We thus report our results as $0.5 - P_{\mathrm{c2st}}$: in this way the best model has the lowest c2st value. We use a scikit-learn [37] *HistGradientBoostingClassifier* with default parameters as our classifier.

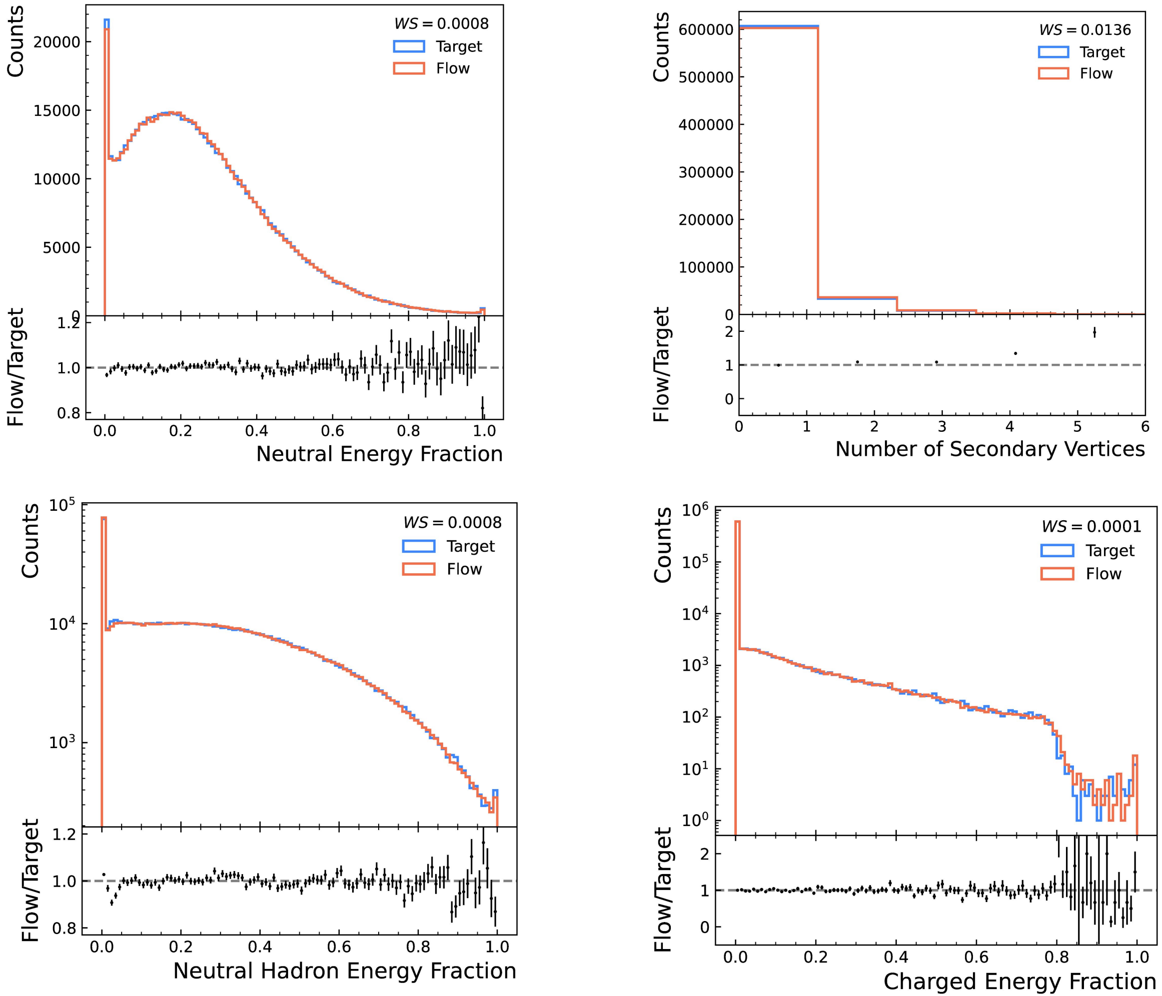
Training on more data

If we vary the training split size from 10k to 10M jets, and we generate 1M, we can see that more training data helps with accuracy, but there is a plateau





More results



More results