SYSTEMATIC EVALUATION OF GENERATIVE MACHINE LEARNING CAPABILITY TO SIMULATE DISTRIBUTIONS OF OBSERVABLES AT THE LARGE HADRON COLLIDER

Jan Gavranovič[*1,2] **and Borut Paul Kerševan**[†1,2]

[1]Jožef Stefan Institute, Jamova 39, Ljubljana, 1000, Slovenia
[2]Faculty of Mathematics and Physics, Jadranska 19, Ljubljana, 1000, Slovenia

March 21, 2024

ABSTRACT

Monte Carlo simulations are a crucial component when analysing the Standard Model and New physics processes at the Large Hadron Collider. This paper aims to explore the performance of generative models for complementing the statistics of classical Monte Carlo simulations in the final stage of data analysis by generating additional synthetic data that follows the same kinematic distributions for a limited set of analysis-specific observables to a high precision. Several normalizing flow architectures are adapted for this task and their performance is systematically evaluated using a well-known benchmark sample containing the Higgs boson production beyond the Standard Model and the corresponding irreducible background. The applicability of normalizing flows under different model parameters and a restricted number of initial events used in training is investigated. The best performing model is then chosen for further evaluation with a set of statistical procedures and a simplified physics analysis. We demonstrate that the the number of events used in training coupled with the flow architecture are crucial for the physics performance of the generative models. By implementing and performing a series of statistical tests and evaluations we show that a machine-learning-based generative procedure can can be used to generate synthetic data that matches the original samples closely enough and that it can therefore be incorporated in the final stage of a physics analysis with some given systematic uncertainty.

Keywords High energy physics · LHC · machine learning · normalizing flows · Monte Carlo simulations

1 Introduction

In data analysis of new physics searches at the Large Hadron Collider (LHC) [1] experiments, the use of Monte Carlo (MC) simulation is essential to accurately describe the kinematics of the known *background* processes in order to determine an eventual discrepancy with the measured data and attribute such a deviation to a certain new physics *signal* hypothesis. Describing LHC data precisely through MC simulations involves several key steps (see e.g. [2]). First, particles are generated based on a calculated differential cross-section, a process referred to as "event generation". This generation is carried out using MC generator tools like Pythia [3] or Sherpa [4]. Next, these particles are simulated as they pass through the detector volume and interact with the detector's materials. This stage is typically performed using the Geant4 toolkit [5] and is known as "detector simulation". During this step, the model also incorporates various factors that accurately represent the collision environment, such as the response to multiple simultaneous proton collisions in the LHC, often referred to as "pile-up" events. These interactions in the detector are converted into simulated electronic response of the detector electronics ("digitization" step) at this stage, the MC simulated data matches the real recorded collision data as closely as possible. Subsequently, the response of the detector trigger system

[*]jan.gavranovic@ijs.si
[†]borut.kersevan@ijs.si

is modeled, and then the simulated data undergoes the same complex reconstruction procedure as the measured data, involving the reconstruction of basic analysis objects, such as electrons or jets, followed by reconstruction of more involved event kinematics.

Eventually, the final data analysis optimizes the data selection ("filtering") procedure to maximize measurement accuracy and new physics discovery potential (statistical significance) and determines the potential presence of new physics using statistical tests on the final data selection (for a nice overview see Ref. [6]). The filtering and final statistical analysis are based on comparing the MC signal and background predictions with the real data by using several $\mathcal{O}(10)$ kinematic variables. Obviously, the statistics of the simulated events limits the prediction accuracy of the background and signal events - ideally, the number of simulated events would exceed the data predictions by several orders of magnitude to minimize the impact of finite MC statistics on the systematic uncertainty of the final measurement. While the simulated background events can typically be shared between several analyses at a LHC experiment, the simulation of a chosen signal process, and the subsequent choice of the relevant kinematic observables is very analysis-specific.

1.1 High-Luminosity LHC and the need for more computing power

A fundamental problem with the standard MC simulation procedure described above is the need for large computational resources, which restricts the physics analysis discovery process due to the speed and high cost of CPU and disk space needed to simulate and store billions of MC events describing high energy particle collisions [7]. The full procedure of MC event simulation of one of the detectors at the LHC may take several minutes per event and produce $\mathcal{O}(1$ MB/event$)$ of data of which only $\mathcal{O}(1$ kB/event$)$ of high-level features, i.e. reconstructed kinematic observables, is used in a specific final analysis of a physics process of interest.

With the current Run 3 and, in the future, the High-Luminosity LHC (HL-LHC), it is expected that the experiments will require even more computing power for MC simulation to match the precision requirements of physics analysis, which increase proportionally to the size of the collected datasets that will come with the increase in integrated luminosity. Taken together, the data and MC requirements of the HL-LHC physics programme are formidable, and if the computing costs are to be kept within feasible levels, very significant improvements will need to be made both in terms of computing and storage, as stated in Ref. [8] for the ATLAS detector [9]. The majority of resources will indeed be needed to produce simulated MC events in the simulation chain from physics modelling (event generation) to detector simulation, reconstruction, and production of analysis formats.

1.2 Machine learning for fast event generation

To address the limitation posed by insufficient MC statistics in constraining the potential of physics analysis, a promising approach is the utilization of Machine Learning (ML), specifically focusing on deep generative modelling. For this purpose, a "fast simulation" approach is being investigated (i.e. [10]), which aims to replace the computationally intensive parts of the MC simulation chain with faster ML solutions. The general idea is to create large numbers of events at limited computing cost using a learning algorithm that was trained on a comparatively smaller set of MC-simulated events. Several different approaches exist, trying to replace different MC stages in simulation chain with faster ML solutions. Commonly used ML approaches include Generative Adversarial Networks (GAN), Variational Autoencoders (VAE), normalizing flows, and diffusion models [11].

The study presented in this paper specifically focuses on normalizing flows [12]. The data space in the final stage of a physics analysis is relatively low-dimensional, typically $\mathcal{O}(10)$ observables, and the precision requirements are high. This is a perfect setting for normalizing flows, which have a needed complexity and necessary transformations to model such data samples in a computationally feasible way.

Focusing on the normalizing flow architectures, there have been many recent examples involving event generation [13, 14, 15, 16], replacing the most computationally intensive parts of the detector simulation, such as the calorimeter response [17, 18, 19, 20], jet modelling [21], anomaly detection [22, 23], background estimation [24], Bayesian uncertainties [25], reweighting [26], end-to-end simulation [27] and many more.

The article focuses on the approach of developing a generative ML procedure for a finite set of analysis-specific reconstructed kinematic observables. The generative ML algorithm is thus trained on a set of MC-simulated and reconstructed events using the kinematic distributions used in the final analysis. The requirement is to be able to extend the statistics of the existing MC using this procedure by several orders of magnitude with the generation being fast enough that the events can be produced on-demand without the need for expensive data storage. In other words, the ML algorithm will learn to model the multi-dimensional distributions with a density $p(\mathbf{x})$ of $\mathcal{O}(10)$ observables $\mathbf{x}$ that can

be used in the final stage of a given physics analysis, thus the approach can also be interpreted as smoothing ("kriging") of the multi-dimensional kinematic distributions derived from a finite learning sample.

The ML procedures used in particle physics are often built upon the most recent advances in ML tools used for commercial purposes, whereby the goals and precision requirements in commercial applications are very different from the ones used in a particle physics analysis. While ML approaches can achieve a *visually impressive* agreement between the training MC sample and the events generated using the derived ML procedure (for example with GAN [28] and VAE [29]), one still needs to *systematically evaluate* [30, 31] whether the agreement is good enough in terms of the requirements of the corresponding physics data analysis. This paper aims to provide a systematic evaluation of: what are the actual precision requirements in terms of statistical analysis of a new physics search, and in view of these evaluates the performance of a few custom implementations of the recently trending ML tools and a set of statistical tools for their evaluation.

2 Training MC dataset

The study in this paper uses the publicly available simulated LHC-like HIGGS dataset [32] of a new physics beyond the Standard model (BSM) Higgs boson production and a background process with identical decay products in the final state and very similar kinematic features, to illustrate the performance of ML data generation in high-dimensional feature spaces. The HIGGS dataset MC simulation uses the DELPHES toolkit [33] to simulate the detector response of a LHC experiment.

The advantage of using this benchmark dataset is that it is publicly available and often used in evaluations of new ML tools by computer science researchers and developers, and referenced even in groundbreaking works such as Ref. [34].

The signal process is the fusion of two gluons gg into a heavy neutral Higgs boson H^0 that decays into heavy charged Higgs bosons $H^\pm$ and a W boson. The $H^\pm$ then decays to a second W boson and to a light Higgs boson h^0 that decays to b quarks. The whole signal process can be described as:

$$gg \to H^0 \to W^\mp H^\pm \to W^\mp W^\pm h^0 \to W^\mp W^\pm b\bar{b}\,.$$

The background process, which features the same intermediate state state $W^\mp W^\pm b\bar{b}$ without the Higgs boson production is the production and decay of a pair of top quarks, $gg \to t\bar{t} \to W^\mp W^\pm b\bar{b}$, to a semi-leptonic final state. Events were generated assuming 8 TeV collisions of protons at the LHC with masses set to $m_{H^0} = 425$ GeV and $m_{H^\pm} = 325$ GeV.

Observable final state decay products include electrically charged leptons ℓ (electrons or muons) and particle jets j. The dataset contains semi-leptonic decay modes, where the first W boson decays into $\ell\nu$ and the second into jj giving us decay products $\ell\nu b$ and jjb. Further restrictions are also set on transverse momentum p_T, pseudorapidity η, type and number of leptons, and number of jets. Events that satisfy the above requirements are characterized by a set of observables (features) that describe the experimental measurements using the detector data. These basic kinematic observables were labeled as "low-level" features and are:

- jet p_T, η and azimuthal angle ϕ,
- jet b-tag,
- lepton p_T, η and ϕ,
- missing energy E_m,

which gives us 21 features in total. More complex derived kinematic observables can be obtained by reconstructing the invariant masses of the different intermediate states of the two processes. These are the "high-level" features and are:

$$m_{jj},\ m_{jjj},\ m_{l\nu},\ m_{jl\nu},\ m_{b\bar{b}},\ m_{Wb\bar{b}},\ m_{WWb\bar{b}}\,.$$

Ignoring azimuthal angles ϕ due to the detector symmetry (giving a uniform distribution), and focusing only on continuous features finally results in an 18-dimensional feature space.

2.1 Data preprocessing

Data preprocessing (also known as feature scaling) is a crucial step in the ML pipeline. It is used to transform the data into a format that is more suitable for the ML algorithms. We present two different data preprocessing methods that were used in this study. The data splitting into independent sets is presented in the Appendix A.

A quantile transform (also known as a Gauss rank transformation) maps a variable's distribution to another probability distribution [35]. This method transforms features to follow a normal distribution. It reduces the impact of outliers making it a robust preprocessing scheme. For the final analysis we have chosen to implement and use this transformation over the logit normal transformation discussed above.

3 Review of used ML methods

A short description of normalizing flows is presented in this section in order to give a relevant context to the ML approaches evaluated in this paper. The description closely follows the reviews from Refs. [12, 36]. Preliminary results of the evaluation of the normalizing flows on the HIGGS dataset will be presented in this section. In the next section we will use the best performing normalizing flow model for further evaluation with a set of statistical tools.

Let $\mathbf{u} \in \mathbb{R}^D$ be a random vector with a known probability density function $p_{\mathbf{u}}(\mathbf{u}) : \mathbb{R}^D \to \mathbb{R}$. Distribution $p_{\mathrm{u}}(\mathbf{u})$ is called a base distribution and is usually chosen to be something simple, such as a normal distribution. Given data $\mathbf{x} \in \mathbb{R}^D$, one would like to know the distribution $p_{\mathrm{x}}(\mathbf{x})$ it was drawn from. The solution is to express $\mathbf{x}$ as a transformation T of a random variable $\mathbf{u}$, distributed according to a distribution $p_{\mathrm{u}}(\mathbf{u})$ in such a way that

$$\mathbf{x} = T(\mathbf{u}) \,, \quad \mathbf{u} \sim p_{\mathrm{u}}(\mathbf{u}) \,, \tag{1}$$

where T is implemented using ML components, such as a neural network. The transformation T must be a diffeomorphism, meaning that it is invertible and both T and T^{-1} are differentiable. Under these conditions, the density $p_{\mathrm{x}}(\mathbf{x})$ is well-defined and can be calculated using the usual change-of-variables formula

$$p_{\mathrm{x}}(\mathbf{x}) = p_{\mathrm{u}}\left(T^{-1}\left(\mathbf{x}\right)\right) \left|\mathrm{det} J_T\left(T^{-1}\left(\mathbf{x}\right)\right)\right|^{-1} = p_{\mathrm{u}}\left(T^{-1}\left(\mathbf{x}\right)\right) \left|\mathrm{det} J_{T^{-1}}\left(\mathbf{x}\right)\right| \,, \tag{2}$$

where J_T is a $D \times D$ Jacobian matrix of partial derivatives.

Invertible and differentiable transformations are composable, which allows one to construct a flow by chaining together different transformations. This means that one can construct a complicated transformation T with more expressive power by composing many simpler transformations:

$$T = T_K \circ \ldots \circ T_1 \quad \text{and} \quad T^{-1} = T_1^{-1} \circ \ldots \circ T_K^{-1} \,. \tag{3}$$

A flow is thus referring to the trajectory of samples from the base distribution as they get sequentially transformed by each transformation into the target distribution. This is known as forward or generating direction. The word normalizing refers to the reverse direction, taking samples from data and transforming them to the base distribution, which is usually normal. This direction is called inverse or normalizing direction and is the direction of the model training. Flows in forward and inverse directions are then, respectively,

$$\begin{aligned} \mathbf{z}_k &= T_k(\mathbf{z}_{k-1}) \quad \text{for} \quad k = 1, \ldots, K \,, \\ \mathbf{z}_{k-1} &= T_k^{-1}(\mathbf{z}_k) \quad \text{for} \quad k = K, \ldots, 1 \,, \end{aligned} \tag{4}$$

where $\mathbf{z}_0 = \mathbf{u}$ and $\mathbf{z}_K = \mathbf{x}$.

The log-determinant of a flow is given by

$$\log \left|\det J_T\left(\mathbf{z}_0\right)\right| = \log \left| \prod_{k=1}^{K} \det J_{T_k}\left(\mathbf{z}_{k-1}\right) \right| = \sum_{k=1}^{K} \log \left|\det J_{T_k}\left(\mathbf{z}_{k-1}\right)\right| \,. \tag{5}$$

A trained flow model provides event sampling capability by Eq. (1) and density estimation by Eq. (2).

The best description of the unknown probability density $p_{\mathrm{x}}(\mathbf{x})$ is obtained by fitting a parametric flow model $p_{\mathrm{x}}(\mathbf{x}; \boldsymbol{\theta})$ with free parameters $\boldsymbol{\theta}$ to a target distribution by using a maximum likelihood estimator computing the average log-likelihood over N data points

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^{N} \log p_{\mathrm{x}}(\mathbf{x}_n; \boldsymbol{\theta}) \,. \tag{6}$$

The latter defines the loss function of the ML algorithm and is thus the quantity optimized using gradient-based methods while training the ML procedure. This can be done because the exact log-likelihood of input data is tractable in flow-based models. In order to keep the computing load at a manageable level, averaging is performed over batches of data and not on the whole dataset, as is customary in practically all ML procedures.

Rewriting Eq. (6) in terms of variables $\mathbf{u}$ using Eq. (2) and introducing a parametric description of the distribution $p_{\mathrm{u}}(\mathbf{u}; \boldsymbol{\psi})$ from Eq. (1), one gets

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^{N} [\log p_{\mathrm{u}}\left(T^{-1}(\mathbf{x}_n; \boldsymbol{\phi}); \boldsymbol{\psi}\right) - \log \left|\mathrm{det} J_{T^{-1}}(\mathbf{x}_n; \boldsymbol{\phi})\right|] \,, \tag{7}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\phi}, \boldsymbol{\psi}\}$ are the parameters of the target and base distributions, respectively. The parameters $\boldsymbol{\psi}$ of the base distribution are usually fixed, for example, the zero mean and the unit variance of a normal distribution. From Eq.

(7), one can see that in order to fit the flow model parameters one needs to compute the inverse transformation T^{-1}, the Jacobian determinant, the density $p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})$ and be able to differentiate through all of them. For sampling the flow model, one must also compute T and be able to sample from $p_{\mathrm{u}}(\mathbf{u};\boldsymbol{\psi})$.

For applications of flow models, the Jacobian determinant should have at most $\mathcal{O}(D)$ complexity, which limits the flow-model-based design.

3.1 Coupling models

The main principle of finding a set of transformations optimally suited to be used in flow-based ML generative models, introduced by Ref. [37], focus on a class of transformations that produce Jacobian matrices with determinants reduced to the product of diagonal terms. These classes of transformations are called coupling layers.

A coupling layer splits the feature vector $\mathbf{z}$ into two parts at index d and transforms the second part as a function of the first part, resulting in an output vector $\mathbf{z}'$. In the case of RealNVP model [38] the implementation is a follows:

$$\begin{aligned}\mathbf{z}'_{\leq d} &= \mathbf{z}_{\leq d}\,,\\ \mathbf{z}'_{>d} &= \mathbf{z}_{>d}\cdot\exp\left(\boldsymbol{\sigma}(\mathbf{z}_{\leq d})\right)+\boldsymbol{\mu}(\mathbf{z}_{\leq d})\,.\end{aligned} \tag{8}$$

This affine transformation of the form $\boldsymbol{s}\cdot\mathbf{z}+\boldsymbol{t}$, consisting of separate scaling ($\boldsymbol{s}=\exp(\boldsymbol{\sigma})$) and translation ($\boldsymbol{t}=\boldsymbol{\mu}$) operations, is implemented by distinct neural networks f. These operations depend on the variables $\mathbf{z}_i$ in the other half of the block ($i\leq d$), i.e. $\boldsymbol{\mu}=f_{\boldsymbol{\mu}}(\mathbf{z}_{\leq d})$ and $\boldsymbol{\sigma}=f_{\boldsymbol{\sigma}}(\mathbf{z}_{\leq d})$. It is worth noting that this affine transformation possesses a straightforward inverse, eliminating the need to compute the inverses of $\boldsymbol{s}$ and $\boldsymbol{t}$. Furthermore, it exhibits a lower triangular Jacobian with a block-like structure, enabling the determinant to be computed in linear time.

When sequentially stacking coupling layers, the elements of $\mathbf{z}$ need to be permuted between each of the two layers so that every input has a chance to interact with every other input. This can be done with a trained permutation matrix

$$\mathbf{z}'=\mathbf{W}\mathbf{z} \tag{9}$$

as in the Glow model [39]. The idea is thus to alternate these invertible linear transformations with coupling layers.

In order to further speed up and stabilize flow training, batch normalization is introduced at the start of each coupling layer as described in Ref. [38]. One block of such a flow is schematically presented in Fig. 1.

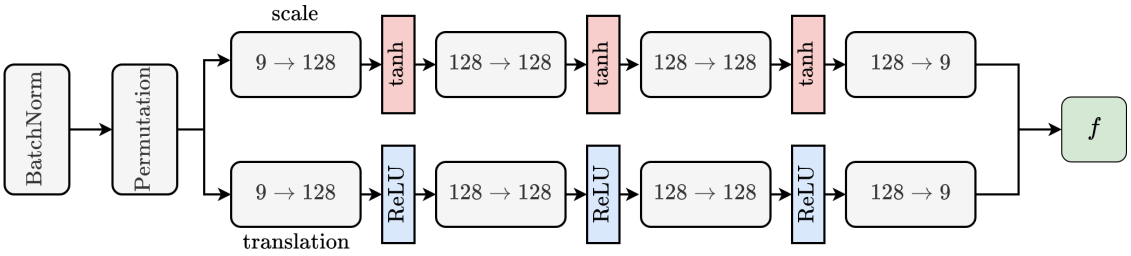

Figure 1: The building block of a coupling layer in a flow. The block consists of a coupling layer with batch normalization and learned permutations. The scaling and translation networks have the same architecture but differ in activation functions. Scaling network uses tanh functions, whereas the translation network uses ReLU functions.

The expressive power of a flow can be increased by composing multiple blocks of coupling layers, batch normalizations and permutations. The number of blocks in the flow is the main hyper-parameter of such a model. Fig. 2 shows the dependence of validation loss, i.e. the value of the loss function from Eq. (6) obtained on an validation sub-sample of the HIGGS dataset, on the number of blocks in a flow. Models with more blocks are harder to train, and show signs of over-fitting earlier. A list of all other hyper-parameters is given in Table 3 in the Appendix A.

3.2 Autoregressive models

Using the chain rule of probability, one can rewrite any joint distribution over D variables (as discussed in Ref. [40]) in the form of a product of conditional probabilities

$$p(\mathbf{z})=\prod_{i=1}^{D}p_i\left(\mathbf{z}_i;c_i(\mathbf{z}_{<i})\right)\,, \tag{10}$$

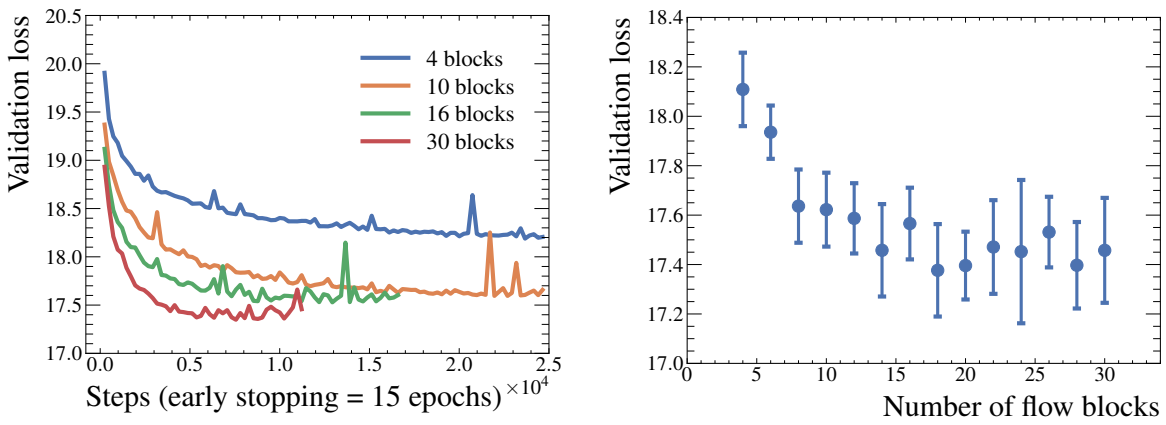

Figure 2: Validation loss as a function of training steps (epochs) and the number of flow blocks. Early stopping was employed to account for over-fitting. All other hyper-parameters were kept constant. Models were trained on 2.5×10^5 events. The uncertainties presented in the right plot were estimated by repeated training and validation using random sampling of events. One can observe rapidly diminishing gains by using more than 10 blocks.

where c_i is some conditional or context on inputs. If $p_i\left(\mathbf{z}_i; c_i(\mathbf{z}_{<i})\right)$ is conditioned on a mixture of Gaussians (MOG), one gets a RNADE model from Ref. [41]:

$$p_i(\mathbf{z}_i; \mathbf{z}_{<i}) = \sum_{c=1}^{C} \boldsymbol{\alpha}_{i,c} \mathcal{N}(\mathbf{z}_i; \boldsymbol{\mu}_{i,c}, \boldsymbol{\sigma}^2_{i,c}) \,. \tag{11}$$

The mixture model parameters are calculated using a neural network that returns a vector of outputs $(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i, \boldsymbol{\alpha}_i) = f_i(\mathbf{z}_{<i}; \boldsymbol{\theta}_i)$, as illustrated on Fig. 3.

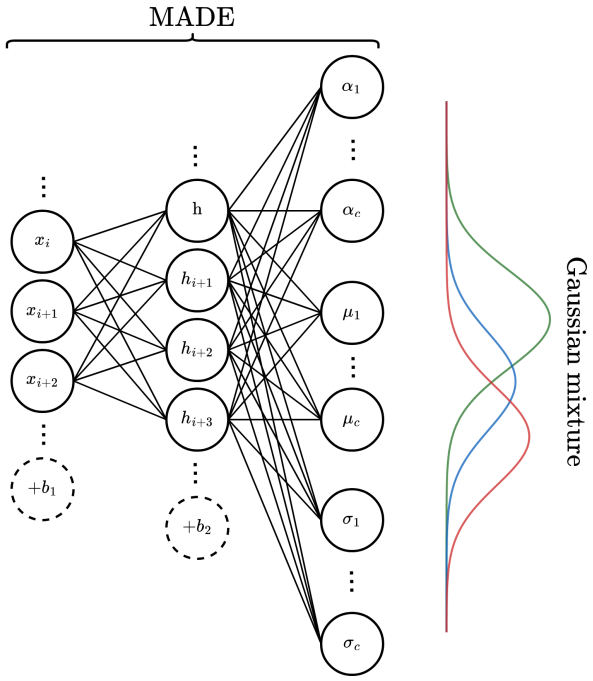

Figure 3: Mixture of Gaussians (MOG) output of a neural network implemented with MADE.

Specifically, the mixture of Gaussian parameters for the conditionals is calculated in the following sequence:

$$\begin{aligned}
\mathbf{h}(\mathbf{z}) &= \text{ReLU}(\mathbf{W}^\top \mathbf{z} + \mathbf{b}) \,, \\
\boldsymbol{\alpha}(\mathbf{z}) &= \text{softmax}(\mathbf{W}_\alpha^\top \mathbf{h}(\mathbf{z}) + \mathbf{b}_\alpha) \,, \\
\boldsymbol{\mu}(\mathbf{z}) &= \mathbf{W}_\mu^\top \mathbf{h}(\mathbf{z}) + \mathbf{b}_\mu \,, \\
\boldsymbol{\sigma}(\mathbf{z}) &= \text{ELU}(\mathbf{W}_\sigma^\top \mathbf{h}(\mathbf{z}) + \mathbf{b}_\sigma) + 1 + \varepsilon \,,
\end{aligned} \tag{12}$$

where $\mathbf{W}$ are the weight matrices, and $\mathbf{b}$ the bias vectors. ReLU, softmax and ELU are the activation functions. The event sampling generative step is performed simply as sampling from a Gaussian mixture. As a substantial simplification, a single neural network with D inputs and D outputs for each parameter vector can be used instead of using D separate neural networks (f_i) for each parameter. This is done with a MADE network [42] that uses a masking strategy to ensure the autoregressive property from Eq. (10). Adding Gaussian components to a MADE network increases its flexibility. The model was proposed in Ref. [43] and is known as MADEMOG. The MADE network was implemented using residual connections from [44] as described in Ref. [45]. MADE networks can also be used as building blocks to construct a Masked Autoregressive Flow or MAF [43] by sequentially stacking MADE networks. In this case, the i-th conditional is given by a single Gaussian

$$p_i\left(\mathbf{z}_i; c_i(\mathbf{z}_{<i})\right) = \mathcal{N}\left(\mathbf{z}_i; \boldsymbol{\mu}_i, (\exp(\boldsymbol{\sigma}_i))^2\right) \,, \tag{13}$$

where $\boldsymbol{\mu}_i = f_{\boldsymbol{\mu}_i}(\mathbf{z}_{<i})$ and $\boldsymbol{\sigma}_i = f_{\boldsymbol{\sigma}_i}(\mathbf{z}_{<i})$ are MADE networks. The generative sampling in this model is straightforward:

$$\mathbf{z}_i = \mathbf{u}_i \cdot \exp(\boldsymbol{\sigma}_i) + \boldsymbol{\mu}_i \,, \quad \text{where} \quad \mathbf{u}_i \sim \mathcal{N}(0,1) \tag{14}$$

with a simple inverse

$$\mathbf{u}_i = (\mathbf{z}_i - \boldsymbol{\mu}_i) \exp(-\boldsymbol{\sigma}_i) \,, \tag{15}$$

which is the flow model training direction. Due to the autoregressive structure, the Jacobian is triangular (the partial derivatives $\partial x_i / \partial u_j$ are identically zero when $j > i$), hence its determinant is simply the product of its diagonal entries. One block of such a flow is presented in Fig. 4. A further option is to stack a MADEMOG on top of a MAF, which is then labeled as a MAFMADEMOG.

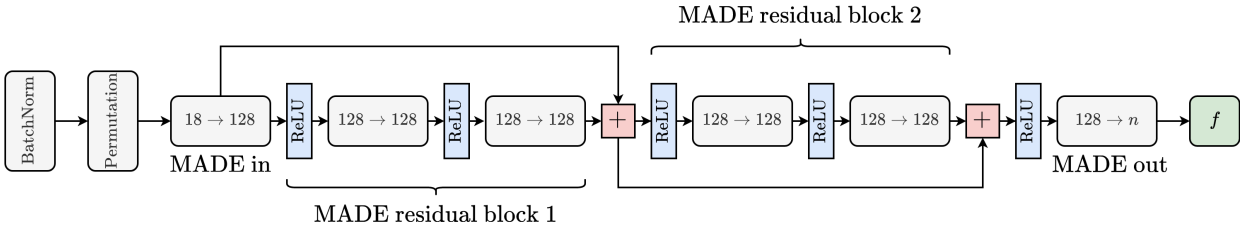

Figure 4: Block of a used MAF model. The architecture follows the one outlined in Ref. [46].

Training curves for all three types of models are shown in Fig. 5, where one can see that the MADEMOG and MAFMADEMOG achieve similar performance, in both cases significantly better than the simpler MAF. The dependence on the number of Gaussians is also shown, where one can see that the model performance saturates at a relatively low number of chosen Gaussian mixtures.

To maximize the physics potential we have further trained the MADEMOG model, which has shown promising performance with a simple architecture at the good sampling speed (see appendix B for ML event generation times). To combat the problems introduced in the this section (mainly training instability and overfitting) we have used a GELU activation function [47] and cosine learning scheduler with warm restarts [48]. We have also used Gaussian feature scaling introduced in section 2.1. The network has 250M trainable parameters and managed to train for 30 epochs before converging on all available MC background events after data partitioning. This setup was eventually shown to be the most performant and was used in the final studies in this paper.

3.3 Spline transformations

At the end of the flow block, one needs to choose a specific transformation f. So far, affine transformations of Eq. (8) and Eq. (14) were used, which were combined with sampling from Gaussian conditionals as in Eq. (11). Rather than relying on basic affine transformations, this approach can be extended to incorporate spline-based transformations.

A spline is defined as a monotonic piecewise function consisting of K segments (bins). Each segment is a simple invertible function (e.g. linear or quadratic polynomial, as in Ref. [49]). In this paper, rational quadratic splines are implemented, as proposed in Ref. [46] and used by Ref. [50]. Rational quadratic functions are differentiable and are also analytically invertible, if only monotonic segments are considered. The spline parameters are then determined from neural networks in an autoregressive way. The resulting autoregressive model, replacing affine transformations of Eq. (14) and its inverse Eq. (15) with equivalent expressions for splines, is labeled as RQS in the studies of this paper. Symbolically, the generative step is thus:

$$\mathbf{z}_i = \mathrm{RQS}_i\left(\mathbf{u}_i, K_i, \boldsymbol{\theta}_i\right), \quad \text{where} \quad \mathbf{u}_i \sim \mathcal{N}(0,1) \tag{16}$$

with an inverse

$$\mathbf{u}_i = \mathrm{RQS}_i^{-1}\left(\mathbf{z}_i, \mathbf{u}_i, K_i, \boldsymbol{\theta}_i\right), \tag{17}$$

where the K represents the bin parameters and $\boldsymbol{\theta}$ denotes the remaining model (spline) parameters.

Fig. 6 shows the validation loss for different number of spline bins. The validation loss does not seem to decrease substantially with the increasing number of bins, however, the quality of generated samples does, in fact, improve with more bins. For the studies performed in this paper, rational splines in 32 bins were eventually chosen. As one can observe from the Fig. 5 and Fig. 6, the performance of this algorithm is comparable to the one of the MADEMOG, however the computing requirements are noticeably higher. Therefore, the MADEMOG was chosen as the final algorithm for further studies, as already stated.

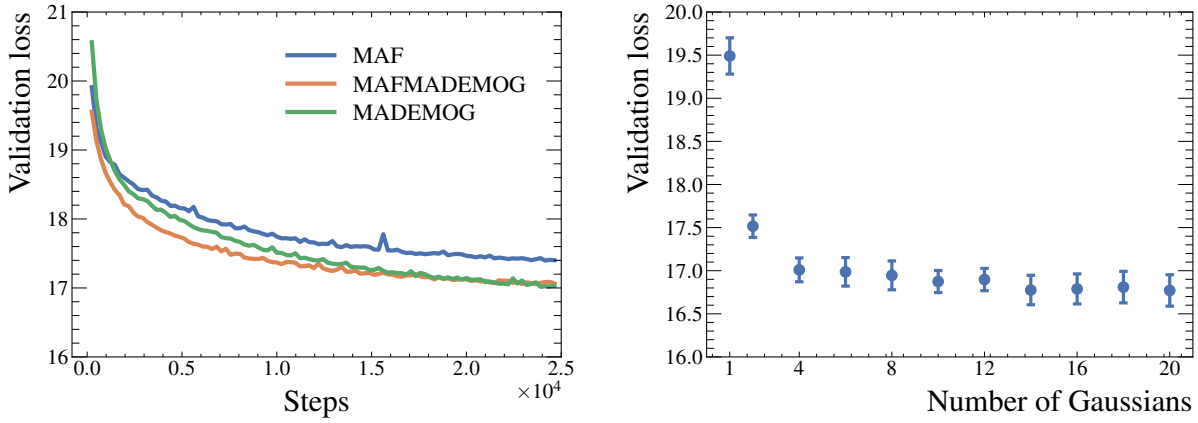

Figure 5: Validation loss as a function of training steps (epochs) for different types of models and the dependance of validation loss on the number of Gaussian components. Models were trained on 2.5×10^5 events.

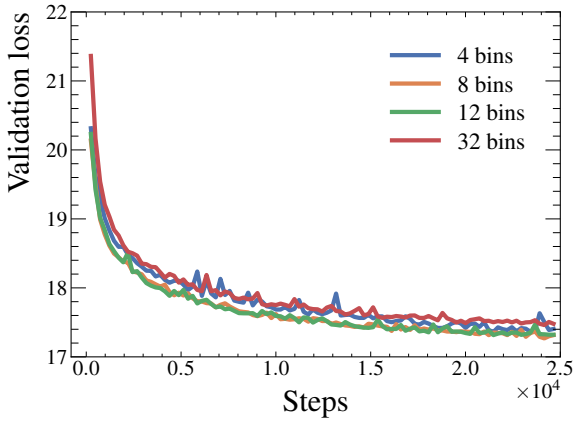

Figure 6: Validation loss as a function of training steps and number of spline bins.

4 Performance evaluation of the ML techniques in a physics analysis at the LHC

After defining the representative evaluation dataset and generative ML procedures, one can focus on the main objective of this paper, which is to systematically evaluate the performance of such tools with a finite-sized training sample in terms of requirements based on a physics analysis at the LHC and HL-LHC.

As described in the introduction, a typical particle physics data analysis with the final selection and the statistical evaluation procedure uses an order of $\mathcal{O}(10)$ reconstructed kinematic observables (which are very often used to construct the 'ultimate' selection variable using ML techniques). The corresponding available statistics of MC signal and background events used to construct the predicted distributions of these kinematic observables, is however often of the order $\mathcal{O}(10^6)$ events or less due to the computing resource constraints and severe filtering requirements of a new physics search. Consequently, the ultimate kinematic region, where one searches for new physics signal, can even contain orders of magnitude fewer MC events, since one is looking for unknown new processes at the limits of achievable kinematics, which translates to "tails" of kinematic distributions.

4.1 ML-generated distributions of observables

Histograms of event distributions of kinematic observables used in the HIGGS sample, where the events were ML-generated with the best-performing MADEMOG algorithm, are shown in Fig. 7, with the detailed MADEMOG configuration given in Table 1. Generated distributions are compared with distributions of MC events on which the model was trained. One can observe that the model reliably reproduces the original distributions at least on the this (visual) level of comparison, confirming that the chosen MADEMOG is adequate for these studies. Further plots are shown in the Appendices.

In order to give a more detailed insight in the quality of reproduction, histograms containing binned ratios of these ML-generated events and MC-simulated events are shown in Fig. 8. It is evident that the deviations between the ML and MC are for all cases most pronounced in the tails where the event count becomes very low. Performance in these tail regions diminishes due to the fact that the learning algorithm did not see enough rare events in the tails of distributions and could not reproduce a reliable distribution in that region.

Hyper-parameter list	
Parameter	Value
Hidden layer size	2048
Flow blocks	60
Residual connecection	every 2 blocks
Gaussian mixtures	32
Activation function	GELU
Batch size	1024
Training size	all split background (see Figure 23)
Optimizer	Adam
Max. learning rate	1×10^{-3} reached on epoch start
Min. learning rate	0 reached on epoch end
Learning rate scheduler	cosine annealing with warm restarts
Weight decay	1×10^{-6}
Max. epochs	100
Early stopping	15
Feature scaling	Gauss scaler
GPU	NVIDIA GeForce RTX 4090

Table 1: List of used hyper-parameters for the final MADEMOG analysis model. The network has 250M trainable parameters.

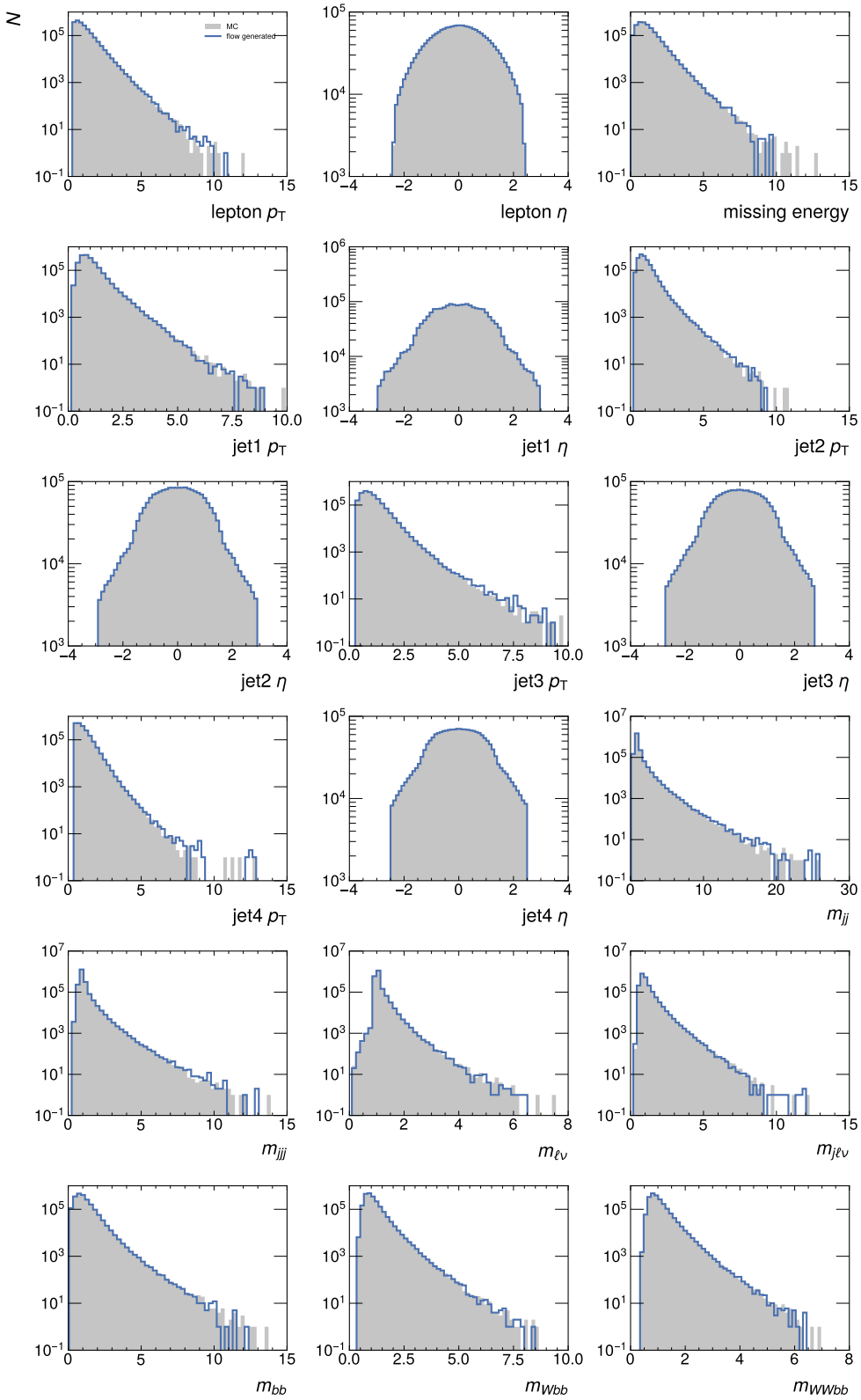

Figure 7: Distributions of ML-generated events using the MADEMOG algorithm, trained on the HIGGS dataset on its kinematic observables. The original MC distribution from this dataset is shown in grey. Visually, the quality of reproduction is very good.

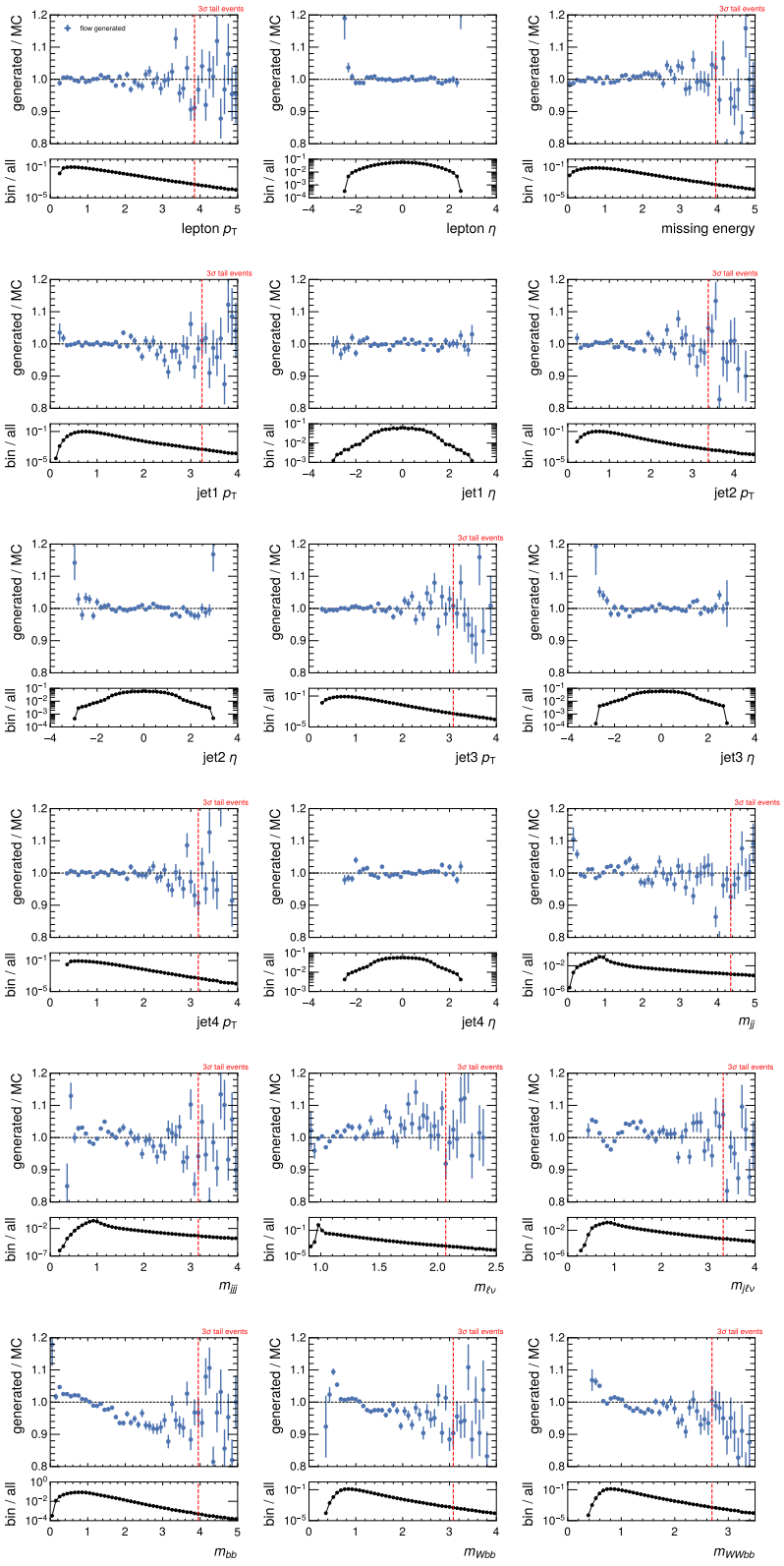

Figure 8: Histograms containing binned ratios of ML-generated events and MC-simulated events. Smaller plots containing the normalized original distribution are given under each ratio plot for easier comparison. Discrepancy between bins for MC events and ML-generated events increases in the tails of the distributions with poor statistics. The vertical dashed red line indicates the region containing the '3σ' tail (i.e. approximately the 10^{-3} fraction of MC events), indicating very poor statistics of MC events available for training the ML algorithm.

4.2 Evaluating the ML generation using divergence measures between probability distributions in one dimension

The probability distributions we can use in statistical tests can in this study be extracted from the of event samples. We have the ML-generated events $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \sim P_{\text{MC}}$ and the MC events $\{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_M\} \sim P_{\text{ML}}$. Determining if the samples come from the same distribution (i.e. if $P_{\text{MC}} = P_{\text{ML}}$) is known as a two-sample test. This can be computed by defining some suitable divergence metric and comparing it to a baseline reference (threshold). For this purpose, we have used several well known divergence measures (or 'tests'): the χ^2 distance, the Kullback-Leibler divergence, the Hellinger distance and the Wasserstein distance [40]. For the baseline reference of a successful test in the presence of statistical uncertainty, we have split the MC events into two parts, and performed the distance calculation on the two parts. All these tests are in practice used as one-dimensional, i.e. their values are calculated per-feature and per-event and then averaged to get the final value. The results are shown in Fig. 9. The results show that the similarity of the ML-generated events is consistent across the different tests and is indeed close to the baseline MC events also using these objective measures. Comparing the obtained results to different generative methods presented in the Ref [31], the values of these tests show that the applied MADEMOG method is in fact quite competitive to other approaches, confirming the representative value of these studies.

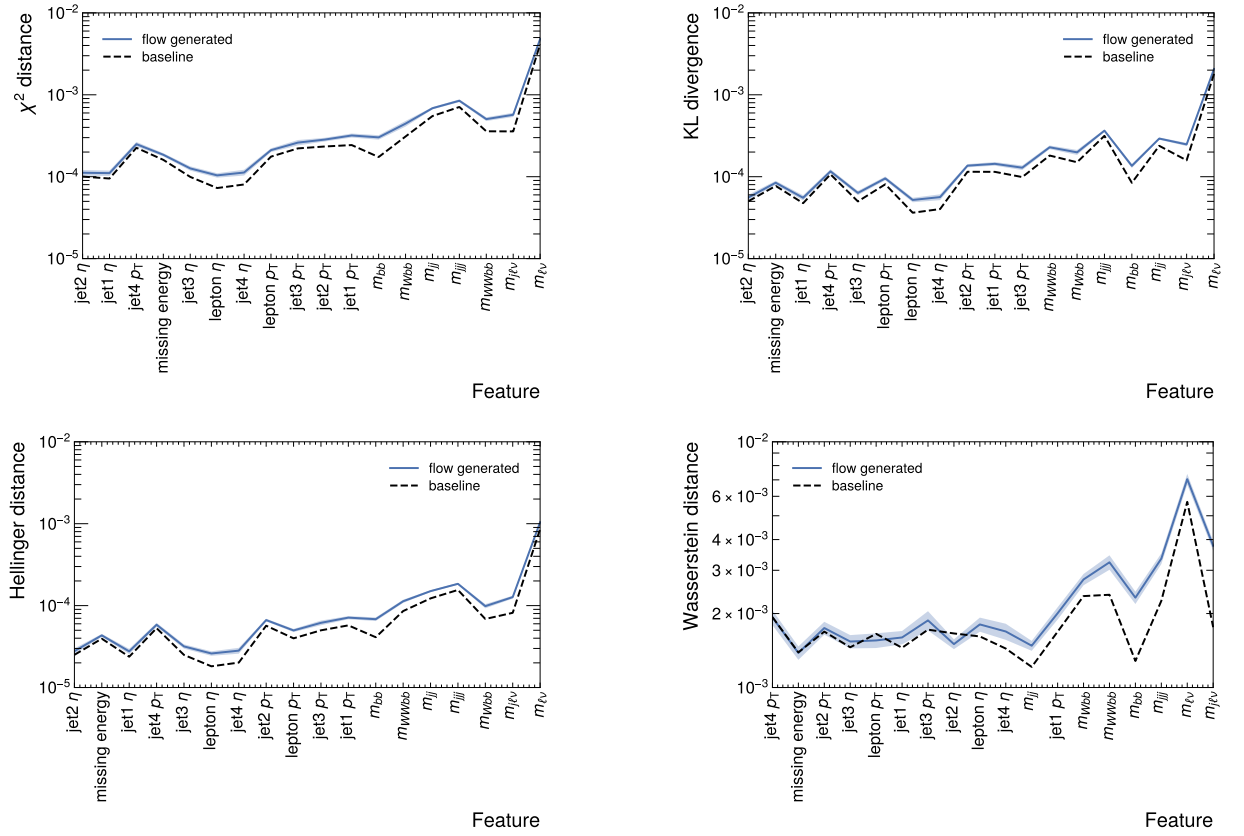

Figure 9: Evaluation of the performance of the MADEMOG generative ML algorithm using different divergence measures. The plots show the statistical distance between the generated and MC distributions.

4.3 Classifier two sample test (C2ST) as a multi-dimensional distribution comparison

As stated above, we would like to asses if samples from P_{ML} and P_{MC} are equivalent. In other words we want to accept or reject the hypothesis $P_{\text{ML}} = P_{\text{MC}}$. A relively new procedure is to use ML tools for this and train a classifier to distinguish between the two samples. If the classifier is unable to distinguish between the two samples, we can conclude that the two samples are equivalent [51]. The goal is thus to train a maximally confused classifier with 50% accuracy and AUC equal to 0.5. To achieve this, we construct a joint dataset with label 0 for MC events and label 1 for ML events:

$$\mathcal{D} = \{(\mathbf{x}'_i, 0)\}_{i=1}^{n} \cup \{\mathbf{x}''_i, 1)\}_{i=1}^{n} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{2n} , \tag{18}$$

where we have assumed same sample sizes for simplicity. We then shuffle the dataset and split it into a training and a holdout set. We train a classifier $f(\mathbf{x};\boldsymbol{\theta}) \approx p(\mathbf{y}=1|\mathbf{x})$ on the training set, the score of which we can interpret as the probability measure for the label $\mathbf{y}=1$ based on inputs $\mathbf{x}$. We can then use classification accuracy or AUC on the holdout set as a measure of the similarity of the two distributions. This procedure is favoured compared to the classical divergence measures we evaluated in the previous section, because it scales very well also to multi-dimensional distributions, both in terms of simplicity and as well as computing requirements.

The obtained training curves for the classifier are shown in Fig. 10 and the ROC curve is shown in Fig. 11. The classifier achieves an AUC close to 0.5 and an accuracy close to 50%, but slightly above, meaning that it can distinguish between the two samples to some extent. The same can be said looking at the validation loss and accuracy while training. We can see a deviation from the ideally confused classifier, but this is expected due to the finite training sample size used in training the generative model, which translates to poor modelling of the events in the tails of the distributions that the classifier can pick up to distinguish between the two samples.

To gain an even more detailed insight into the strengths and weaknesses of our generative procedure, we can extract the density ratio $r(\mathbf{x}) = P(\mathbf{x})/Q(\mathbf{x})$ from the same C2ST procedure. Using the Bayes' rule with a prior $p(\mathbf{y}=1) = \pi = \frac{1}{2}$, and taking into account the complementarity of the two hypotheses, i.e. $p(\mathbf{y}=1|\mathbf{x}) = 1 - p(\mathbf{y}=0|\mathbf{x})$, we can write

$$\begin{aligned} r(\mathbf{x}) = \frac{P(\mathbf{x})}{Q(\mathbf{x})} = \frac{p(\mathbf{x}|\mathbf{y}=1)}{p(\mathbf{x}|\mathbf{y}=0)} = \frac{p(\mathbf{y}=1|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y}=1)} \cdot \frac{p(\mathbf{y}=0)}{p(\mathbf{y}=0|\mathbf{x})p(\mathbf{x})} \\ = \frac{p(\mathbf{y}=1|\mathbf{x})}{p(\mathbf{y}=0|\mathbf{x})} \cdot \frac{\pi}{1-\pi} = \frac{p(\mathbf{y}=1|\mathbf{x})}{1-p(\mathbf{y}=1|\mathbf{x})} = \exp\left[\log \frac{p(\mathbf{y}=1|\mathbf{x})}{1-p(\mathbf{y}=1|\mathbf{x})}\right] , \end{aligned} \tag{19}$$

which we can estimate with a classifier trained on a two-sample dataset from Eq. (18) as:

$$r(\mathbf{x}) = \exp\left\{\sigma^{-1}\left[p(\mathbf{y}=1|\mathbf{x})\right]\right\} \approx \exp\left\{\sigma^{-1}\left[f(\mathbf{x};\boldsymbol{\theta})\right]\right\} , \tag{20}$$

where σ is the sigmoid function and σ^{-1} is it's inverse (the logit function). For the classifier we have used a simple neural network with a sigmoid output, and a binary cross entropy loss function. This density ratio can then be used to search for the failure modes of the ML generative model [31], and thus has further advantages w.r.t. to the previously described tests, since the events in the tails of the $r(\mathbf{x})$ ratio value can be identified as the ones where the ML generative model fails to reproduce the MC events. The obtained distributions of ML-generated events w. r. t. the $r(\mathbf{x})$ ratio value, along with and independent MC sample for reference, are shown in Fig. 12 and the event distributions corresponding to the events in the tails of the classifier ratio score are shown in Fig. 13. The plots in this figure again confirm the insight that the discrepancy between MC and ML events is most pronounced in the tails of the distributions, where the training event count becomes very low.

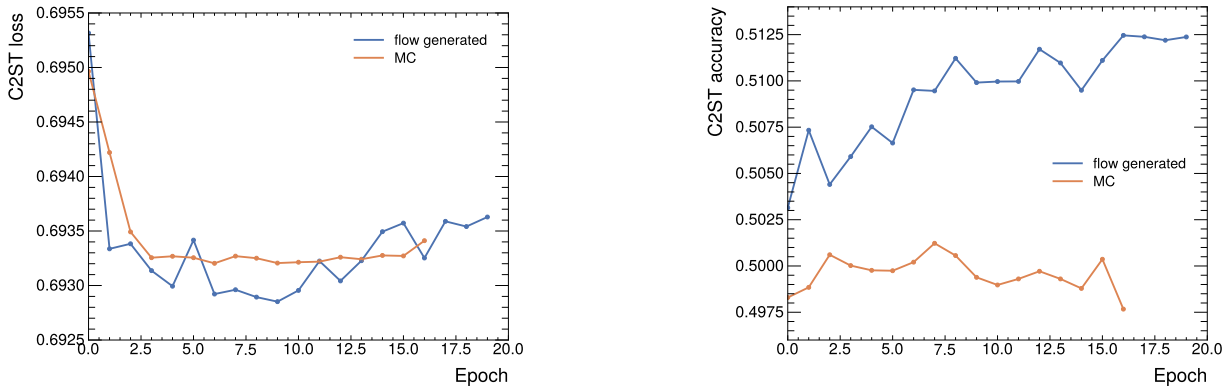

Figure 10: Validation loss and accuracy while training a classifier using a dataset given by Eq. (18).

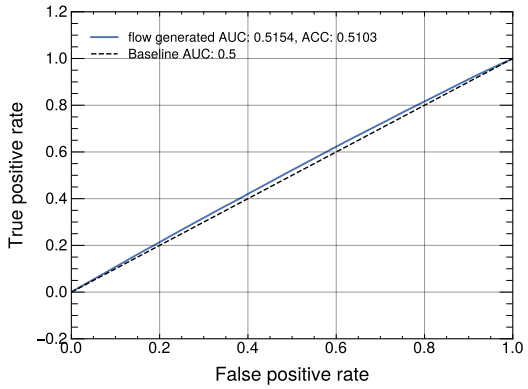

Figure 11: ROC curve for the C2ST classifier.

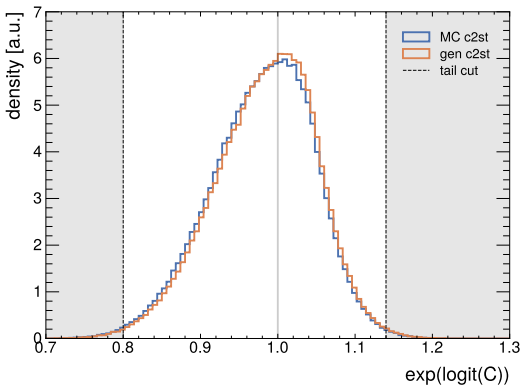

Figure 12: Density ratios of distributions.

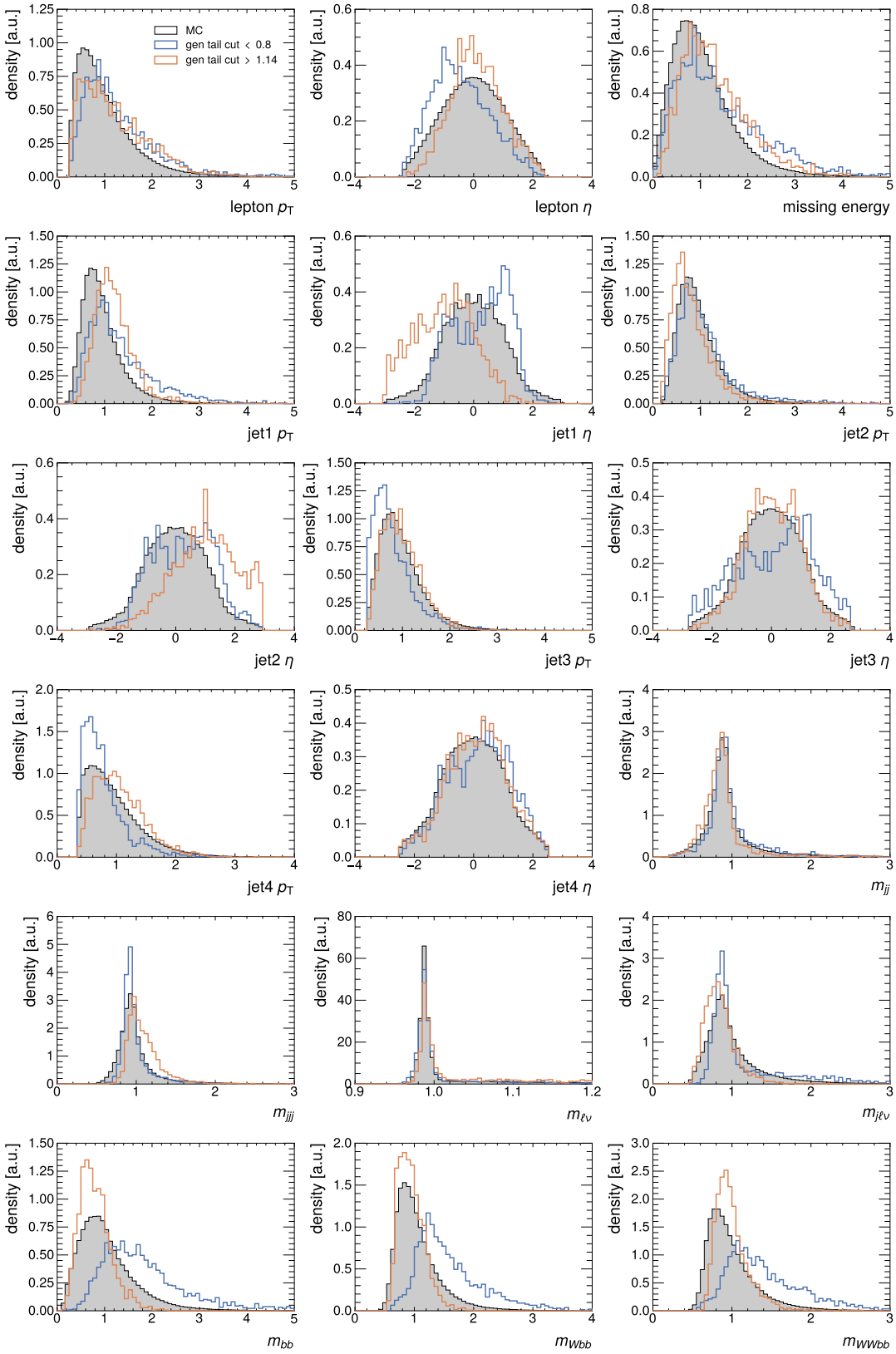

Figure 13: Distributions of events from the tails of the density ratio and comparison to the actual MC shape.

4.4 ML performance evaluation in a physics analysis

In order to provide a relevant quantitative evaluation of the impact of replacing MC-simulated distributions with ML-generated ones in an analysis environment, a simplified analysis setup was constructed that matches a typical analysis of a new physics search in an LHC experiment and its statistical evaluation. In this setup, the HIGGS sample was used for training the ML generative algorithms for the background modelling, with subsets being reserved for validation and testing. In the following studies, the MADEMOG model trained on the HIGGS dataset was used to generate the new background samples as described in the previous sections.

In order to emulate a typical analysis procedure in an LHC experiment, a further step was introduced, namely the ML generative algorithm was trained and applied on background events in the HIGGS sample without any kinematic cuts on its observables, while the statistical analysis studies were done after an additional cut on a new ML classifier, trained to separate signal from background events in the HIGGS sample. The performance of this classifier is shown in Fig. 14, with the final signal selection cut at the value of 0.51. The discrepancy between the classifier scores of the independent MC-simulated and ML-generated samples is on the level of 10% and is shown as a ratio between the two score distributions in Fig. 15 for clarity.

This additional selection step is introduced to match the standard analysis approach, where first a relatively simple baseline selection is applied on the data and MC samples as the first stage of analysis optimisation, at which the agreement of kinematics of the signal and background predictions is generally good enough and the statistics of the MC samples is sufficiently large to train a ML classifier (as well as a generative algorithm). A representative LHC analysis then uses this 'ultimate' classifier to perform a final data selection to achieve an optimal signal-to-background separation in terms of discovery significance. However, the MC-simulated samples in new physics searches tend to have quite low statistics after this final selection, which is then too low to be usable to effectively train a ML generative procedure. Consequently, as also done in this paper, the training of the ML generative algorithm needs to happen before this ultimate selection and the impact of the classifier cut on the agreement between the MC and ML-generated samples needs to remain adequate after the selection step.

To sum up, implementing the required analysis procedure involving a classifier cut on the ML generated samples is also an innovative implementation of the study presented in this paper. The procedure can be summarised as follows:

1. train a generative model on MC background samples and generate large datasets of new events using it,
2. apply the final selection on the ML-generated samples using a neural network classifier,
3. normalise the ML-generated samples to match the integrated luminosity of the data sample after the final selection,
4. perform the statistical analysis.

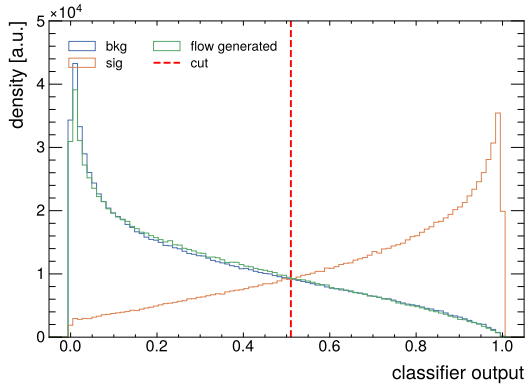

Figure 14: MC signal, MC background and ML background classifier output score.

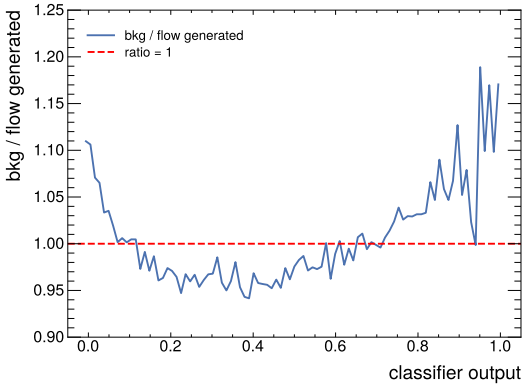

Figure 15: Mismatch between classifier scores for generated ML events and MC events.

By introducing a (ML) classifier cut, one can also see how well the multi-dimensional generation of correlated variables is in fact performing. As a further insight from a different perspective: without subsequent (final) kinematic cuts, each relevant kinematic distribution can just be modelled (smoothed) independently, without a need for a generative procedure. As one can observe in Fig. 16, good agreement is preserved between the MC and ML samples, demonstrating

that the variable correlations are adequately modelled and that the ML generative approach is thus preforming well enough for such an analysis approach.

In this study the the background yield is chosen and then the integrated luminosity with a given cross section is calculated. The evaluation was done in the background range $B \in [10^3, 10^6]$, corresponding to a luminosity increase up to three orders of magnitude ($L \in [10\ \text{fb}^{-1}, 1000\ \text{fb}^{-1}]$). The signal yield ($S$) fraction, denoted $\alpha = S/B$, was set to have a value of 1% or 5% of the background yield in the performed tests. An inclusive relative systematic uncertainty (β), which in a real analysis would comprise both theoretical and experimental uncertainties, was set to β =10%. This systematic uncertainty excludes the systematic uncertainty due to finite simulated sample statistics, which is the focus of this study and is being treated separately and has the expected total Poisson-like $\sim \sqrt{N}$ dependence on the number of ML-generated events for the normalised background prediction, translating into the appropriate multinomial uncertainties in binned distributions.

The data prediction was constructed by adding the MC-simulated samples of background and signal with a chosen signal content, giving a so-called Asimov dataset, which perfectly matches the MC-samples and is often used in the performance validation of a statistical analysis in an LHC experiment. For the purpose of the studies in this paper, the MC-simulated background is then replaced with ML-generated samples, whereby the ML generative procedure was trained on these samples (before the classfier cut, as previously described).

The signal prediction is retained as the MC one because in LHC analyses, the signal simulation, which generally needs a comparatively small number of generated events, is typically not as much of an issue as the background. The final analysis selection is centred around the signal prediction, retaining most of the signal statistics while only selecting the low-statistics tails of the background samples.

This setup aims to provide a good measure of the quality of the ML approach in a LHC analysis environment - ideally the results after this replacement would still perfectly match the results of the original MC setup on the Asimov dataset, however, an agreement within the joint statistical and systematic uncertainty is still deemed as acceptable, and thus a successful implementation of the ML generative procedure.

The achieved statistical agreement certainly depends on the integrated luminosity, i.e. data statistics, assuming a constant presence of a fixed total systematic uncertainty. The dependence on integrated luminosity then translates into a requirement on the quality of ML-generated samples, assuming that an arbitrarily large (and constant) statistics (size) of these ML-generated samples is easily achievable.

With the aim of closely matching a typical statistical analysis, as done in LHC experiments, the HistFactory model from the pyhf [52, 53] statistical tool was used, and different standard procedures of evaluation of the agreement between data and simulation predictions were implemented (profile likelihood calculation, upper limit estimation, background-only hypothesis probability etc.). The statistical analysis was performed using two different possibilities for choosing the optimal variable, resulting in a binned distribution w.r.t. the m_{bb} in the first case and classifier output score in the second case, which aims to give an optimal separation between the shape of the background and signal predictions. In the statistical analysis, the signal presence is evaluated by using a scaling factor μ (signal strength) of the predicted signal normalisation. Statistical error scale factors $\boldsymbol{\gamma} = \{\gamma_i\}$ are used to model the uncertainty in the bins due to the limited statistics of (ML-)simulated samples. By using the fast ML event generation, one aims to push this uncertainty to negligible values, as is indeed done in the study presented below. As already stated, the simulated predictions are given an additional relative uncertainty of β =10% in each bin of the distribution as $\boldsymbol{\beta} = \{\beta_i\}$ to model the systematic uncertainty contribution on the predicted background shape and normalization. In the likelihood calculation, the parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are modelled as nuisance parameters in addition to μ as the main parameter of the likelihood fit.

The binned distributions of samples used in this statistical analysis are shown in Fig. 17 for the two relevant observables. One can see that the agreement between the Asimov data and the simulation prediction using the ML-generated background sample seems to be good, which is an encouraging starting point for a detailed analysis.

In more detail [6], the likelihood is defined as the product over all bins of the Poisson probability to observe N_i events in a particular bin i:

$$L_{\text{phys}}(\text{data}\,|\,\mu) = L_{\text{phys}}(\mu) = \prod_{i \in \text{bins}} \text{Pois}(N_i\,|\,\mu S_i + B_i) = \prod_{i \in \text{bins}} \frac{(\mu S_i + B_i)^{N_i}}{N_i!} e^{-(\mu S_i + B_i)}, \tag{21}$$

where S_i and B_i are the expected signal and background yields, respectively. As already stated, the main parameter of interest is the signal strength, denoted as μ.

The systematic uncertainties are included in the likelihood via a set of nuisance parameters (NP), denoted as $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\gamma})$, that modify the expected background yield, i.e. $\{B_i\} \rightarrow \{B_i(\boldsymbol{\theta})\}$. The overall relative systematic uncertainties β_i

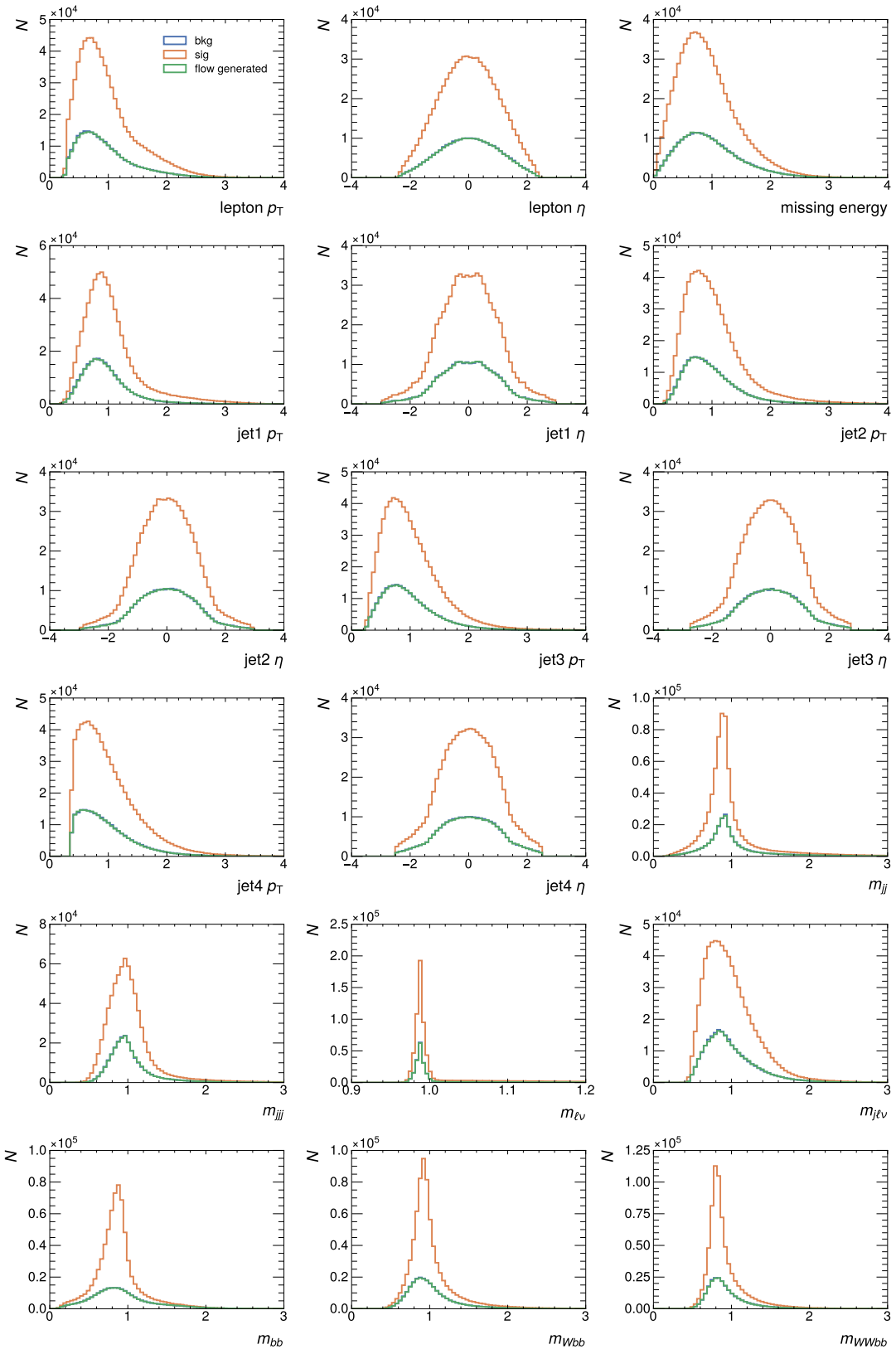

Figure 16: Kinematic distributions of ML (green) and MC-generated (blue) background events after the cut on the classifier from Fig. 14 at an output score of 0.51. The MC-generated signal events are aslo shown for completeness (orange).

can be encoded into Gaussian functions and subsequently into an auxiliary likelihood function $L_{\text{aux}}(\boldsymbol{\beta})$, while the uncertainties on the background predictions due to the limited number of simulated events are accounted for in the likelihood function considering Poisson terms

$$L_{\text{stat}}(\boldsymbol{\gamma}) = \prod_{i \in \text{bins}} \frac{(\gamma_i B_i)^{B_i} e^{-\gamma_i B_i}}{\Gamma(B_i)}, \tag{22}$$

where Γ is the gamma function.

The profile likelihood function can finally be defined as a product of three likelihoods

$$L(\mu, \boldsymbol{\theta}) = L_{\text{phys}}(\mu) \cdot L_{\text{aux}}(\boldsymbol{\beta}) \cdot L_{\text{stat}}(\boldsymbol{\gamma}). \tag{23}$$

A likelihood fit using this likelihood is then performed to determine the value of μ and its uncertainty, as well as the nuisance parameters. The estimates of μ and $\boldsymbol{\theta}$ are obtained as the values of the parameters that maximise the likelihood function $L(\mu, \boldsymbol{\theta})$ or, equivalently, minimise $-\ln L(\mu, \boldsymbol{\theta})$.

This profile likelihood function is also used to construct statistical tests of w.r.t. the hypothesised value of μ. A profile likelihood ratio, $\lambda(\mu)$, is defined as

$$\lambda(\mu) = \frac{L(\mu, \hat{\hat{\boldsymbol{\theta}}}(\mu))}{L(\hat{\mu}, \hat{\boldsymbol{\theta}})}, \tag{24}$$

where $\hat{\mu}$ and $\hat{\boldsymbol{\theta}}$ are the parameters that maximise the overall likelihood, and $\hat{\hat{\boldsymbol{\theta}}}$ are the NP values that maximise the likelihood for a particular value of μ.

The test statistic is then defined as $q_\mu = -2 \ln \lambda(\mu)$, for which the lower values indicate better compatibility between the data and the hypothesised value of μ. The test statistic is used to calculate a p-value that quantifies the agreement

$$p_\mu = \int_{q_\mu^{\text{obs}}}^{\infty} f(q_\mu \,|\, \mu) \, \mathrm{d}q_\mu, \tag{25}$$

where q_μ^{obs} is the value of test statistics observed in the data, and $f(q_\mu \,|\, \mu)$ is the probability density function of the test statistic q_μ under the signal strength assumption, μ. The p-value can be expressed in units of Gaussian standard deviations $Z = \Phi^{-1}(1 - p)$, where Φ^{-1} is the inverse Gaussian CDF. The rejection of the background-only hypothesis ($\mu = 0$) with a significance of at least $Z = 5$ (corresponding to $p_0 = 2.87 \times 10^{-7}$) is considered a discovery.

A test statistic used in this analysis is the one for the positive signal discovery in which the background-only hypothesis with $\mu = 0$ is tested. If the data is compatible with the background-only hypothesis, the nominator and the denominator of the test statistic, $q_0 = -2 \ln \lambda(\mu = 0)$, are similar. The$q_0$ value is then close to 0, and the corresponding p_0 value is 0.5. For this scenario, upper limits on the signal strength are derived at a CL $= 95\%$ confidence level using the CL_s method [54], for which both the signal plus background, p_{S+B}, and background-only, p_B, p-values need to be calculated. For a given set of signal masses or branching ratios, the signal hypothesis is tested for several values of μ. The final confidence level CL_s is computed as the ratio $\text{CL}_\text{s} \equiv \frac{p_{S+B}}{1-p_B}$, which excludes a signal hypothesis at CL $= 95\%$ when giving a value below 5%.

4.5 Results of likelihood tests

Examples of event distributions obtained after the profile likelihood fit to the Asimov data, as described above, are shown on Fig. 18 for the two different variable selections. One can observe a very nice agreement between the fitted prediction and Asimov data.

Figure 19 shows how well the expected value of the signal strength μ is reproduced in the statistical evaluation. The estimated value of μ with its uncertainty is shown w.r.t. the increase in integrated luminosity. It is evident that the statistical estimation quite reliably reproduces the expected value of $\mu = 1$ for a (small) injected signal at the fraction $\alpha = 1\%$ of the background. One can observe a small bias, which of course persists with increasing integrated luminosity for the ML-generated sample. The (biased) values are generally still within the uncertainty, but it is of course clear that background mis-modelling, present when using ML-generated background, leads to biases and possible sensitivity loss in an analysis with relatively tiny signal presence. This evaluated discrepancy between the injected and estimated signal in the final analysis fit can also be interpreted as the presence of a spurious signal [55], which is another common test in the LHC analyses.

Using the derived profile likelihood in a test statistic to determine the signal and background hypothesis probabilities (p-values in LHC jargon) and the eventual CL_s value, as described above, is shown in Figure 20 as a function of

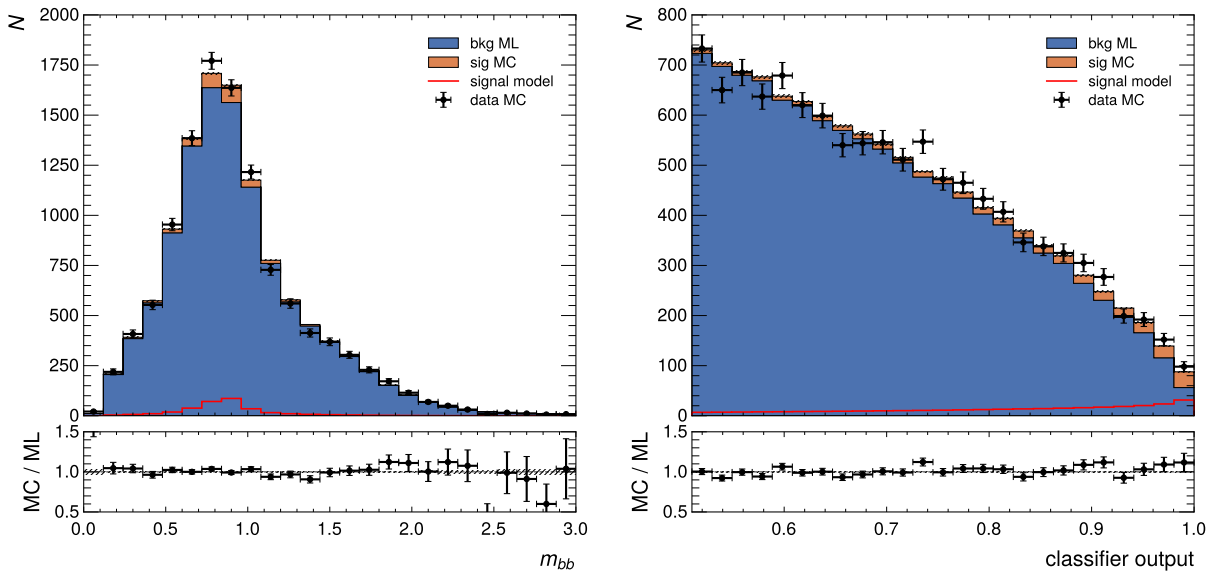

Figure 17: m_{bb} and classifier distributions after the classifier cut. The "MC data" (crosses) represents the Asimov data set composed from MC signal and background prediction and matches quite well the combined MC signal (orange) and ML background (blue) prediction.

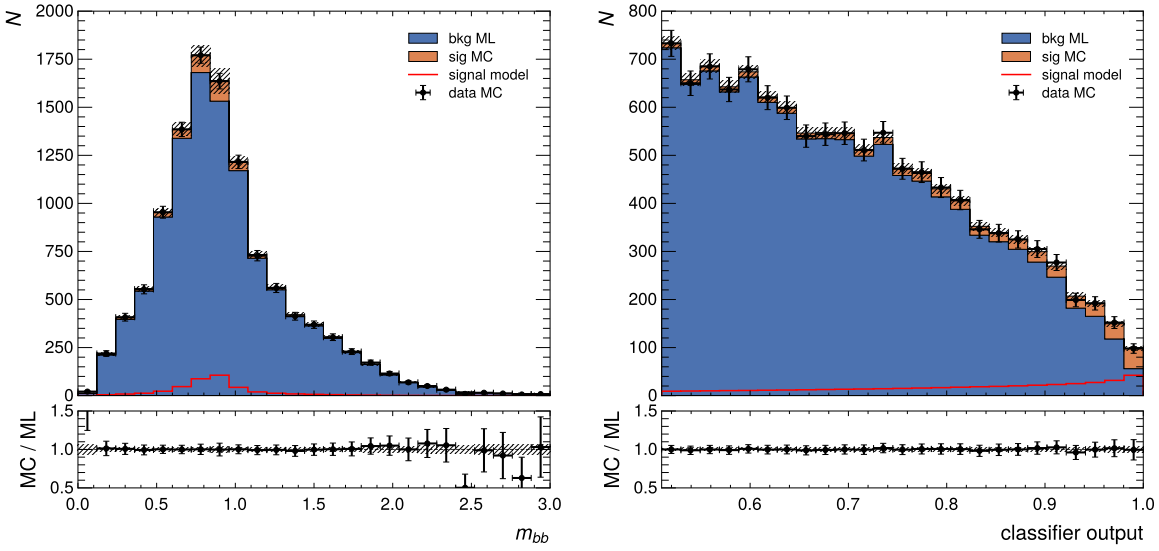

Figure 18: Post-fit distributions for profile likelihood fits to the m_{bb} or the classifier output score.

luminosity. Results for the ideal scenario, using only the MC simulated samples, which match the Asimov dataset, are shown for comparison with the ML-generated background results. One can see that the relevant observables, in particular CL_s, converge as expected with increasing luminosity, whereby the discrepancy between the MC-simulated and ML-generated background predictions then increases with expected luminosity (and thus analysis sensitivity).

4.6 Upper limit estimation

As the final step in this physics analysis study, one aims to evaluate the upper limits on the signal strength, together with the uncertainty estimates using the profile-likelihood-based test statistics, as is done in LHC analyses. The dependence

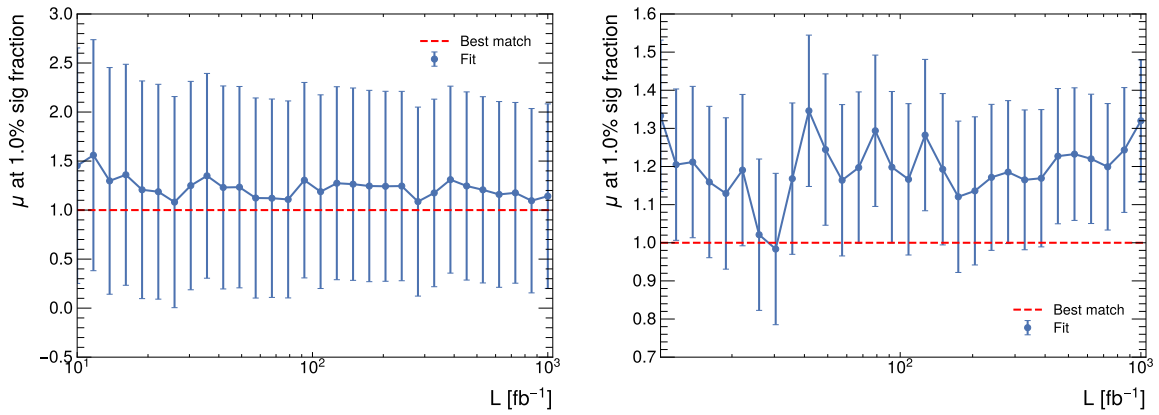

Figure 19: Fitted parameter of interest μ as a function of integrated luminosity L for ML-generated background is shown for the fit using the m_{bb} variable (left) or the classifier output score (right). It is evident that the statistical estimation quite reliably reproduces the expected value of $\mu = 1$ for a (small) injected signal at $\alpha = 1\%$ fraction of the background.

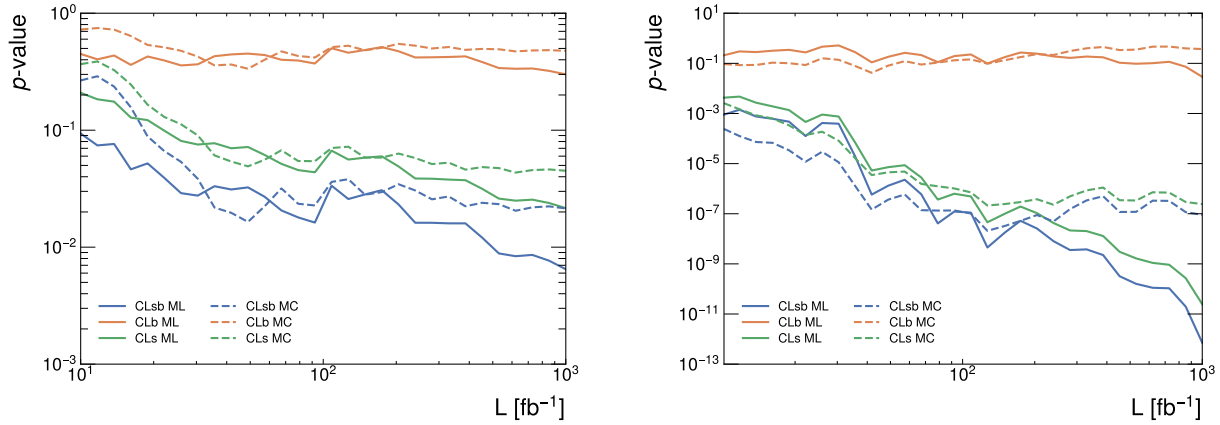

Figure 20: Estimated p-values for signal and background, background and $\mathrm{CL_s}$ for both scenarios (m_{bb} left and classifier score right) at different values of luminosity.

of the extracted upper limit as a function of integrated luminosity is shown in Figures Fig. 21 and Fig. 22 for different values of injected signal fraction and the two fitting scenarios. Again, the ideal (reference) scenario, using the MC simulated samples both for Asimov data construction and simulated predictions is used as a reference, and is in the figures shown together with the derived uncertainty bands. One can observe that the shifts in upper limit estimation are on a scale compatible with the estimated uncertainties for the reference scenario. The use of the classifier score as the variable in the fitted distribution consistently in all tests shows a higher sensitivity to the signal presence, which however also leads to an increased sensitivity to the background mis-modelling using ML-generated events. The discrepancies are nonetheless deemed acceptable in both cases, as one can observe from these statistical tests.

From these results it is evident that the ML-generated samples can indeed be used in a physics analysis as a surrogate model for the background prediction. However, to further minimize the impact of the background ML mis-modelling, one would need to work on implementing techniques that go even beyond the current commercial state-of-the-art approaches, similar to the one used in this paper, and to understand how to optimally adapt them for this use case in high energy physics.

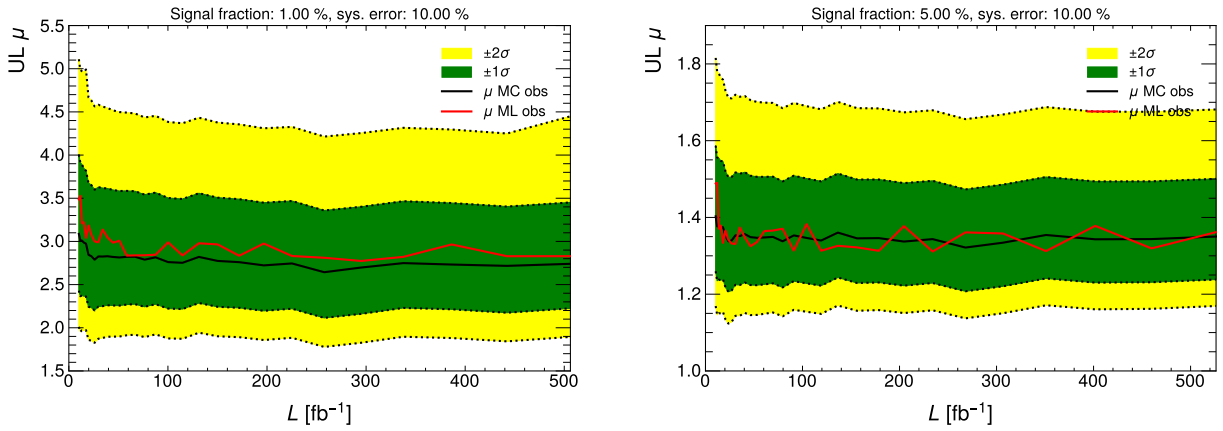

Figure 21: Upper limits on the signal strength μ as a function of integrated luminosity L for the likelihood fit to the m_{bb} distribution.

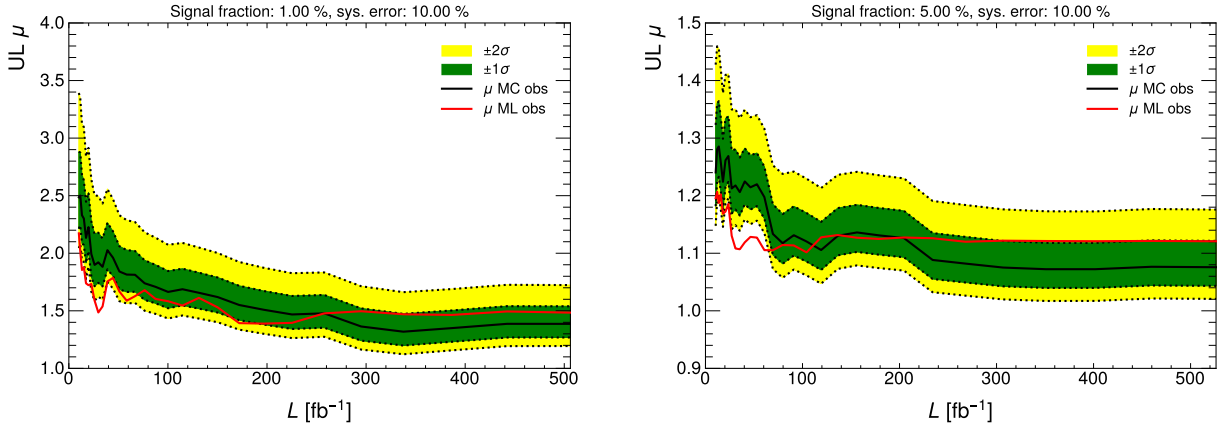

Figure 22: Upper limits on the signal strength (μ) as a function of integrated luminosity L for the likelihood fit to the classifier score distribution.

5 Discussion and outlook

The paper investigates the possibility of using normalizing flows for analysis-specific ML-based generation of events used by the final stage of a given particle physics analysis, where the state-of-the-art generative ML algorithms are trained on the available MC-simulated samples. The extended custom event samples can thus serve to extend the MC statistics, thereby minimizing the analysis uncertainty due to the statistics limitations of MC events or, equivalently, to smooth and minimize the uncertainties on the predicted kinematic event distributions used in the final statistical analysis of the data.

The presented work builds on the ideas of using machine learning for fast simulation and gives better or comparable results to other similar studies mentioned in section 1.2. Furthermore, aside from the achievable accuracy, the advantage of using normalizing flows is that they provide density estimation alongside sampling, which can also be incorporated into the analysis. The downside of this architecture is that it does not provide much flexibility in the design of the method due to the invertibility and differentiability constraints of transformations and the calculation of their determinants. Another downside is the computational cost of the training, which might be prohibitive for some applications using more then $\mathcal{O}(100)$ observables. However, the training can be done on a GPU, which can significantly reduce the time needed for for both training and inference.

The envisaged ML modelling strategy is to learn the D-dimensional distributions of kinematic observables used in a representative physics analysis at the LHC, and produce large amounts of ML-generated events at a low computing

cost (see Appendix B). The input kinematic distributions are derived from the MC-simulated samples, which are statistics-limited by design since they are produced in a very computationally expensive procedure, albeit giving very accurate predictions of the physics performance of the LHC experiments. Furthermore, the study presented in this paper replicates the realistic case of the final event selection in a physics analysis using a cut on a ML-derived discriminating parameter, which defines the final data sets for physics analysis. In this final set, the MC statistics is usually too low to effectively train a ML procedure, thus the training needs to happen at the stage before the final filtering and it is essential that the ML training reproduce the correlations between the observables so that the agreement between the ML-generated and original MC-simulated data is preserved after the final event selection. This paper demonstrates that this can, in fact, be achieved by using the implemented ML techniques, with reasonable precision.

The generative ML approach described in this paper is inherently analysis-specific, meaning that each analysis would require a dedicated training setup. As a demonstrator for this paper, a number of different state-of-the-art normalizing flow architectures with different parameters were implemented. The procedures were not fine-tuned for specific analysis and/or MC dataset to preserve generality, but could potentially achieve even better performance with further optimization of hyper-parameters in Appendix C. The implemented models were trained on the LHC-specific dataset (beyond the Standard model Higgs boson decay) with $\mathcal{O}(10)$ observables. Both coupling layer models and autoregressive models were considered with as discussed in Section 3. All of the models were capable of learning complicated high-dimensional distributions to some degree of accuracy, with autoregressive models having an advantage at the cost of somewhat longer sampling times, which does not pose a problem since our distributions are relatively low-dimensional and the absolute speed-ups are impressive in all cases, i.e. orders of magnitude lower with respect to the standard MC generation procedures used at the LHC.

Performance evaluations using various divergence measures, from χ^2 to Wasserstein distance, classifier two sample test (C2ST) as well as a simplified statistical analysis, matching the procedures used for upper limit estimation of new physics searches at the LHC, show that the generally available MC samples of $\mathcal{O}(10^6)$ events are indeed enough to train such state-of-the-art generative ML models to a satisfactory precision to be used to reduce the systematic uncertainties due to the limited MC statistics. The generative modelling strategy described in this paper could alleviate some of the high CPU and disk size requirements (and costs) of generating and storing simulated events. When trained, these models provide not only fast sampling but also encode all of the distributions in the weights and biases of neural networks, which take up significantly less space than the full MC datasets and can generate analysis-specific events practically on-demand, which is a functionality that goes beyond the scope of this paper but should be studied in a dedicated project.

The listed advantages become even more crucial when considering future LHC computing requirements for physics simulation and analysis, as it is clear that the increase in collision rates will result in much larger data volumes and even more complex events to analyse. Using generative modelling could thus aid in faster event generation as well as future storage requirements coming with the HL-LHC upgrade and beyond.

6 Acknowledgements

The authors would like to acknowledge the support of Slovenian Research and Innovation agency (ARIS) by funding the research project J1-3010 and programme P1-0135.

A Data partitioning

For simplicity we have used 50% split for ML set and holdout set, 80% split for the training and validation sets, and again 50% split for both test sets. Equivalent procedure can also be used for the signal dataset. The procedure is shown in Figure 23. After evaluating the generated samples and confirming they match the MC samples to some accuracy, we can combine the ML-generated background with all the MC events to form the final enlarged dataset for the analysis.

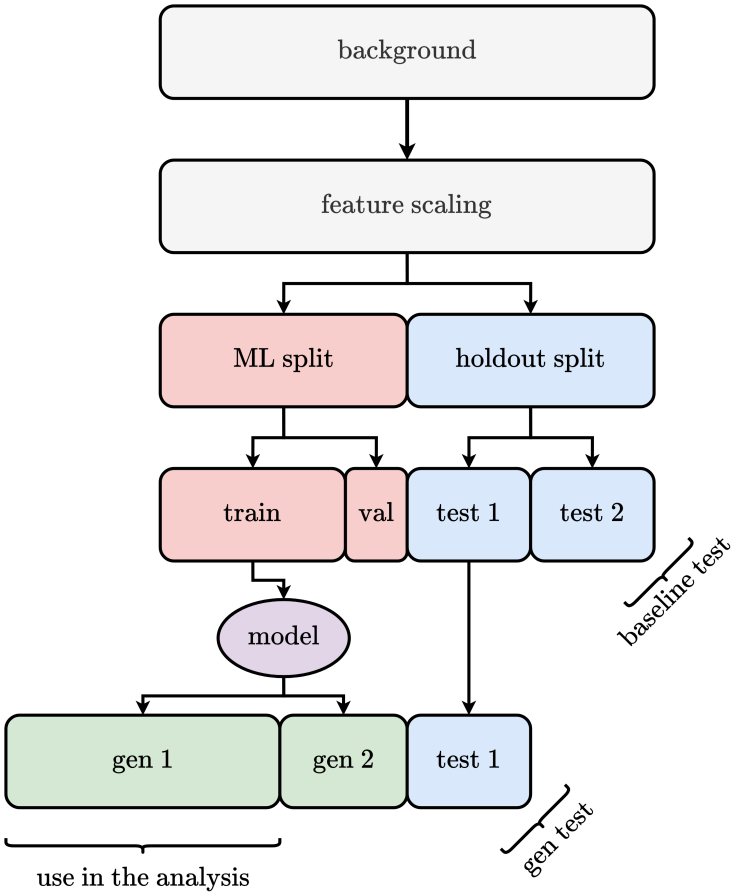

Figure 23: Data split into training, validation and test sets. Splits between the sets are arbitrary and are left to the discretion of the method user.

B Event generation times

The computational time per event on a GPU is given below for all our models. Autoregressive models have longer sampling times because of the modeling constraint that variable i is dependent on all variables preceding variable at index i in an input vector, giving us $\mathcal{O}(D)$ sampling time.

Event generation timing	
Model	Time $[\mu s/\text{event}]$
Glow	1.33 ± 0.02
MAFMADEMOG	11.10 ± 0.08
RQS	103.59 ± 0.94

Table 2: Event generation times using an NVIDIA GeForce RTX 3070 graphics card.

C Hyper-parameters

A table of parameters used in the model selection stage is given in table 3.

Hyper-parameter list	
Parameter	Value
Hidden layer size	128
Flow blocks	10
Activation function	ReLU
Batch size	1024
Training size	2.5×10^5
Optimizer	Adam
Learning rate	3×10^{-4}
Weight decay	1×10^{-7}
Max. epochs	100
Early stopping	15
Feature scaling	logit normal
GPU	NVIDIA GeForce RTX 3070

Table 3: List of used hyper-parameters in the inital model selection stage.

D Generated events in linear scale

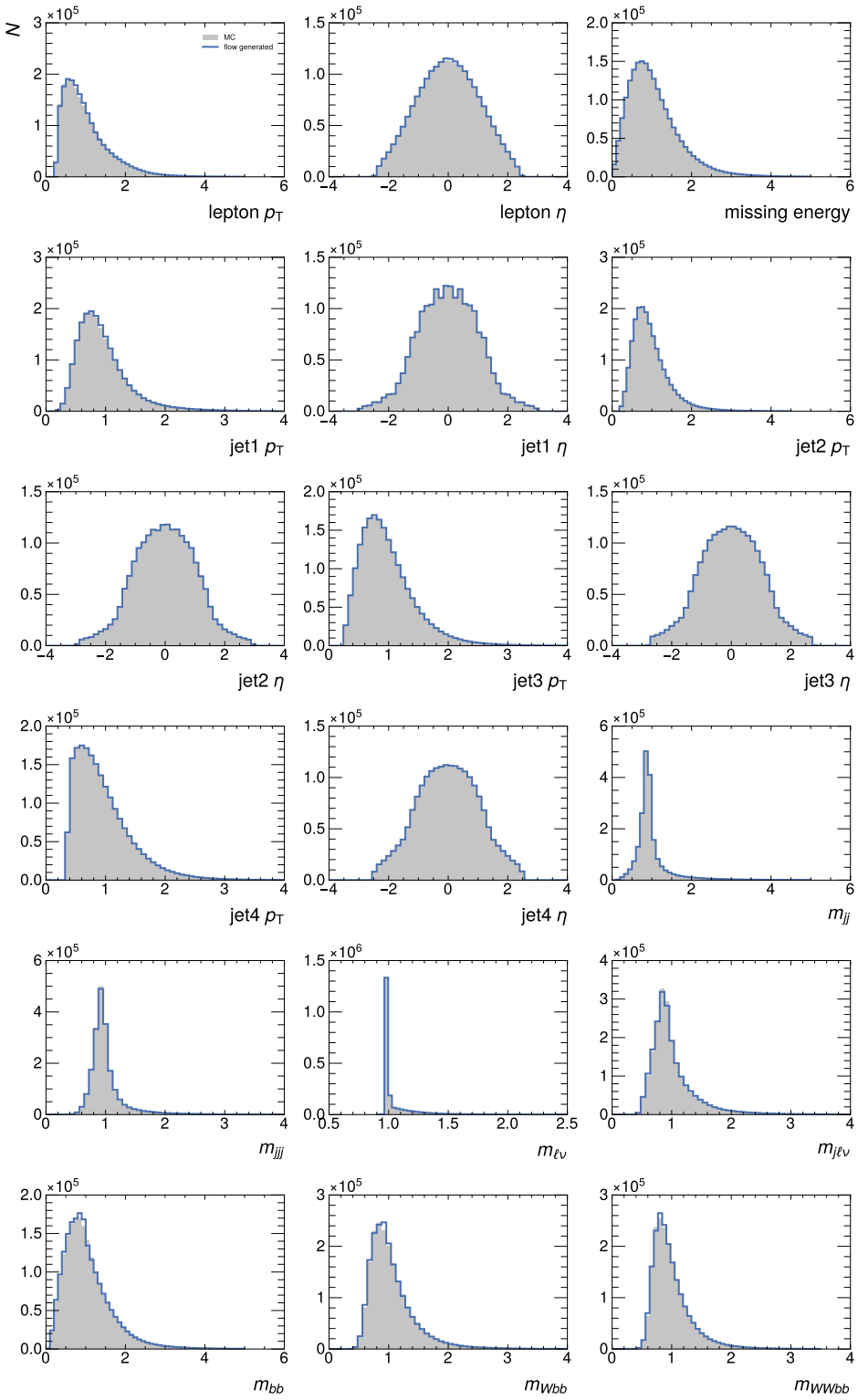

Figure 24: Distributions of generated events. Original MC distribution is shown in grey.

E Corner plot

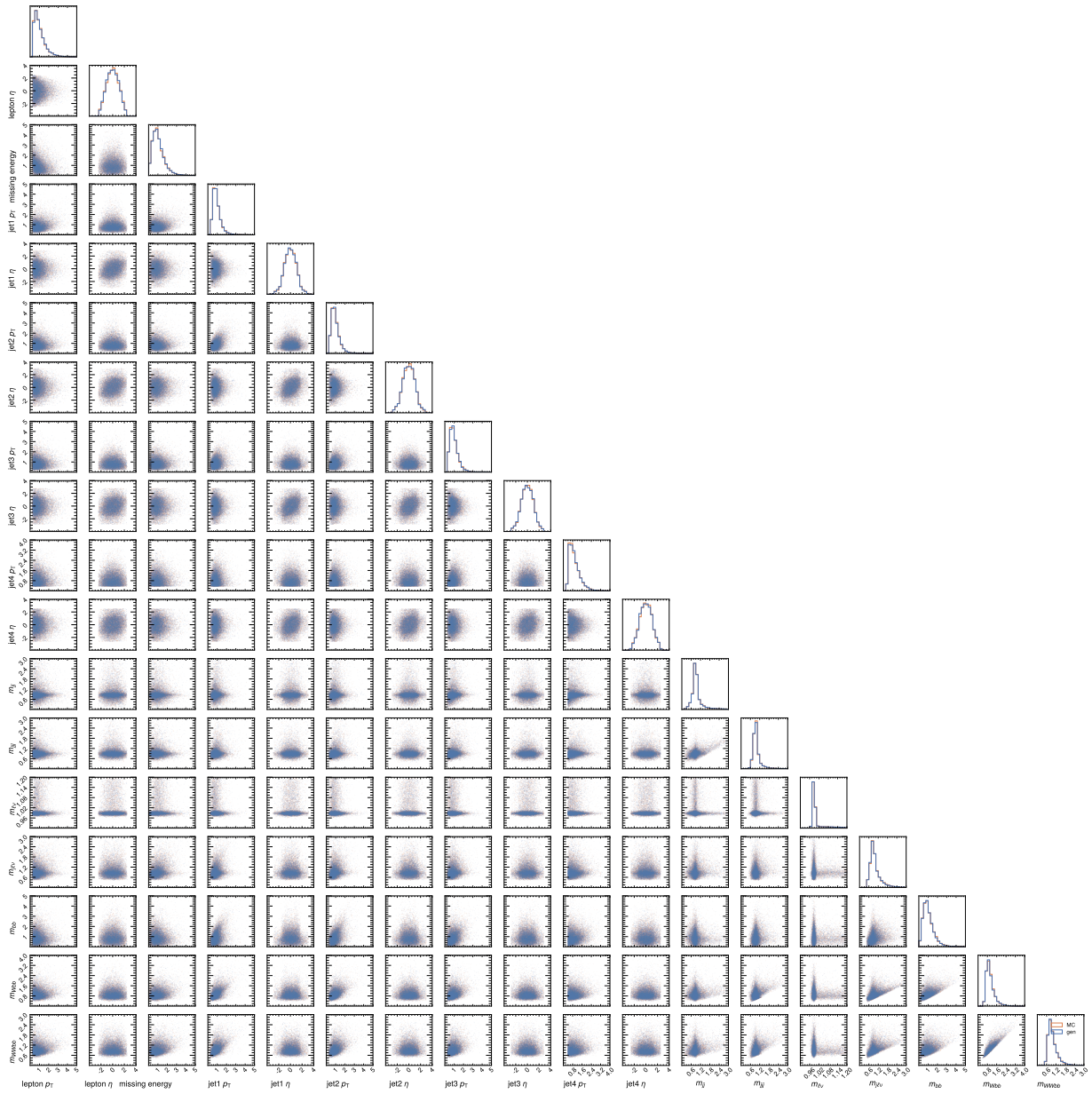

Figure 25: Corner plot of the generated events.

References

[1] Lyndon Evans and Philip Bryant. Lhc machine. *Journal of Instrumentation*, 3(08):S08001, aug 2008. doi: 10.1088/1748-0221/3/08/S08001. URL `https://dx.doi.org/10.1088/1748-0221/3/08/S08001`.

[2] ATLAS Collaboration. The atlas simulation infrastructure. *The European Physical Journal C*, 70(3):823–874, 2010. doi: 10.1140/epjc/s10052-010-1429-9. URL `https://doi.org/10.1140/epjc/s10052-010-1429-9`.

[3] Torbjörn Sjöstrand, Stefan Ask, Jesper R. Christiansen, Richard Corke, Nishita Desai, Philip Ilten, Stephen Mrenna, Stefan Prestel, Christine O. Rasmussen, and Peter Z. Skands. An introduction to PYTHIA 8.2. *Comput. Phys. Commun.*, 191:159, 2015. doi: 10.1016/j.cpc.2015.01.024. URL `https://doi.org/10.1016/j.cpc.2015.01.024`.

[4] Stefan Höche, Frank Krauss, Steffen Schumann, and Frank Siegert. Qcd matrix elements and truncated showers. *Journal of High Energy Physics*, 2009(05):053, may 2009. doi: 10.1088/1126-6708/2009/05/053. URL https://dx.doi.org/10.1088/1126-6708/2009/05/053.

[5] S. Agostinelli et al. GEANT4 – a simulation toolkit. *Nucl. Instrum. Meth. A*, 506:250, 2003. doi: 10.1016/S0168-9002(03)01368-8. URL https://doi.org/10.1016/S0168-9002(03)01368-8.

[6] Glen Cowan, Kyle Cranmer, Eilam Gross, and Ofer Vitells. Asymptotic formulae for likelihood-based tests of new physics. *The European Physical Journal C*, 71:1–19, 2011.

[7] I Bird, P Buncic, F Carminati, M Cattaneo, P Clarke, I Fisk, M Girone, J Harvey, B Kersevan, P Mato, R Mount, and B Panzer-Steindel. Update of the Computing Models of the WLCG and the LHC Experiments. Technical report, 2014. URL https://cds.cern.ch/record/1695401.

[8] P Calafiura, J Catmore, D Costanzo, and A Di Girolamo. ATLAS HL-LHC Computing Conceptual Design Report. Technical report, CERN, Geneva, Sep 2020. URL https://cds.cern.ch/record/2729668.

[9] ATLAS Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008. doi: 10.1088/1748-0221/3/08/S08003. URL https://dx.doi.org/10.1088/1748-0221/3/08/S08003.

[10] ATLAS Collaboration. AtlFast3: the next generation of fast simulation in ATLAS. *Computing and Software for Big Science*, 6(1):1–54, 2022. doi: 10.1007/s41781-021-00079-7. URL https://doi.org/10.1007/s41781-021-00079-7.

[11] Hosein Hashemi and Claudius Krause. Deep generative models for detector signature simulation: An analytical taxonomy, 2023. URL https://arxiv.org/abs/2312.09597.

[12] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference, 2021. URL https://arxiv.org/abs/1912.02762.

[13] Christina Gao, Stefan Höche, Joshua Isaacson, Claudius Krause, and Holger Schulz. Event generation with normalizing flows. *Physical Review D*, 101(7), apr 2020. doi: 10.1103/physrevd.101.076002. URL https://doi.org/10.1103%2Fphysrevd.101.076002.

[14] Christina Gao, Joshua Isaacson, and Claudius Krause. i-flow: High-dimensional integration and sampling with normalizing flows. *Machine Learning: Science and Technology*, 1(4):045023, November 2020. ISSN 2632-2153. doi: 10.1088/2632-2153/abab62. URL http://dx.doi.org/10.1088/2632-2153/abab62.

[15] Anja Butter, Theo Heimel, Sander Hummerich, Tobias Krebs, Tilman Plehn, Armand Rousselot, and Sophia Vent. Generative networks for precision enthusiasts. *SciPost Phys.*, 14:078, 2023. doi: 10.21468/SciPostPhys.14.4.078. URL https://scipost.org/10.21468/SciPostPhys.14.4.078.

[16] Rob Verheyen. Event Generation and Density Estimation with Surjective Normalizing Flows. *SciPost Phys.*, 13:047, 2022. doi: 10.21468/SciPostPhys.13.3.047. URL https://scipost.org/10.21468/SciPostPhys.13.3.047.

[17] Claudius Krause and David Shih. Fast and accurate simulations of calorimeter showers with normalizing flows. *Phys. Rev. D*, 107:113003, Jun 2023. doi: 10.1103/PhysRevD.107.113003. URL https://link.aps.org/doi/10.1103/PhysRevD.107.113003.

[18] Jesse C. Cresswell, Brendan Leigh Ross, Gabriel Loaiza-Ganem, Humberto Reyes-Gonzalez, Marco Letizia, and Anthony L. Caterini. Caloman: Fast generation of calorimeter showers with density estimation on learned manifolds, 2022. URL https://arxiv.org/abs/2211.15380.

[19] Matthew R. Buckley, Claudius Krause, Ian Pang, and David Shih. Inductive caloflow, 2023. URL https://arxiv.org/abs/2305.11934.

[20] Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Claudius Krause, Imahn Shekhzadeh, and David Shih. L2lflows: generating high-fidelity 3d calorimeter images. *Journal of Instrumentation*, 18(10):P10017, oct 2023. doi: 10.1088/1748-0221/18/10/P10017. URL https://dx.doi.org/10.1088/1748-0221/18/10/P10017.

[21] Benno Käch, Dirk Krücker, Isabell Melzer-Pellmann, Moritz Scham, Simon Schnake, and Alexi Verney-Provatas. Jetflow: Generating jets with conditioned and mass constrained normalising flows, 2022. URL https://arxiv.org/abs/2211.13630.

[22] Benjamin Nachman and David Shih. Anomaly detection with density estimation. *Physical Review D*, 101(7), April 2020. ISSN 2470-0029. doi: 10.1103/physrevd.101.075042. URL http://dx.doi.org/10.1103/PhysRevD.101.075042.

[23] Tobias Golling, Gregor Kasieczka, Claudius Krause, Radha Mastandrea, Benjamin Nachman, John Andrew Raine, Debajyoti Sengupta, David Shih, and Manuel Sommerhalder. The interplay of machine learning–based resonant anomaly detection methods, 2023. URL https://arxiv.org/abs/2307.11157.

[24] Suyong Choi, Jaehoon Lim, and Hayoung Oh. Data-driven estimation of background distribution through neural autoregressive flows, 2020. URL https://arxiv.org/abs/2008.03636.

[25] Marco Bellagente, Manuel Haußmann, Michel Luchmann, and Tilman Plehn. Understanding Event-Generation Networks via Uncertainties. *SciPost Phys.*, 13:003, 2022. doi: 10.21468/SciPostPhys.13.1.003. URL https://scipost.org/10.21468/SciPostPhys.13.1.003.

[26] Benjamin Nachman and Ramon Winterhalder. Elsa: enhanced latent spaces for improved collider simulations. *The European Physical Journal C*, 83(9), September 2023. ISSN 1434-6052. doi: 10.1140/epjc/s10052-023-11989-8. URL http://dx.doi.org/10.1140/epjc/s10052-023-11989-8.

[27] Francesco Vaselli, Filippo Cattafesta, Patrick Asenov, and Andrea Rizzi. End-to-end simulation of particle physics events with flow matching and generator oversampling, 2024. URL https://arxiv.org/abs/2402.13684.

[28] Bobak Hashemi, Nick Amin, Kaustuv Datta, Dominick Olivito, and Maurizio Pierini. Lhc analysis-specific datasets with generative adversarial networks, 2019. URL https://arxiv.org/abs/1901.05282.

[29] Sydney Otten, Sascha Caron, Wieske de Swart, Melissa van Beekveld, Luc Hendriks, Caspar van Leeuwen, Damian Podareanu, Roberto Ruiz de Austri, and Rob Verheyen. Event generation and statistical sampling for physics with deep generative models and a density information buffer. *Nature communications*, 12(1):1–16, 2021. doi: https://doi.org/10.1038/s41467-021-22616-z. URL https://doi.org/10.1038/s41467-021-22616-z.

[30] Raghav Kansal, Anni Li, Javier Duarte, Nadezda Chernyavskaya, Maurizio Pierini, Breno Orzari, and Thiago Tomei. Evaluating generative models in high energy physics. *Physical Review D*, 107(7), April 2023. ISSN 2470-0029. doi: 10.1103/physrevd.107.076017. URL http://dx.doi.org/10.1103/PhysRevD.107.076017.

[31] Ranit Das, Luigi Favaro, Theo Heimel, Claudius Krause, Tilman Plehn, and David Shih. How to understand limitations of generative networks. *SciPost Physics*, 16(1), January 2024. ISSN 2542-4653. doi: 10.21468/scipostphys.16.1.031. URL http://dx.doi.org/10.21468/SciPostPhys.16.1.031.

[32] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):1–9, 2014. doi: https://doi.org/10.1038/ncomms5308. URL https://doi.org/10.1038/ncomms5308.

[33] S. Ovyn, X. Rouby, and V. Lemaitre. Delphes, a framework for fast simulation of a generic collider experiment, 2010. URL https://arxiv.org/abs/0903.2225.

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016. http://www.deeplearningbook.org.

[35] T Mark Beasley, Stephen Erickson, and David B Allison. Rank-based inverse normal transformations are increasingly used, but are they merited? *Behavior genetics*, 39:580–595, 2009.

[36] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, 2021. doi: 10.1109/TPAMI.2020.2992934.

[37] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear Independent Components Estimation, 2015. URL https://arxiv.org/abs/1410.8516.

[38] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP, 2017. URL https://arxiv.org/abs/1605.08803.

[39] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative Flow with Invertible 1x1 Convolutions, 2018. URL https://arxiv.org/abs/1807.03039.

[40] Kevin P. Murphy. Probabilistic Machine Learning: Advanced Topics. 2023. http://probml.github.io/book2.

[41] Benigno Uria, Iain Murray, and Hugo Larochelle. RNADE: The real-valued neural autoregressive density-estimator, 2014. URL https://arxiv.org/abs/1306.0186.

[42] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked Autoencoder for Distribution Estimation, 2015. URL https://arxiv.org/abs/1502.03509.

[43] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked Autoregressive Flow for Density Estimation, 2018. URL https://arxiv.org/abs/1705.07057.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision – ECCV 2016*, pages 630–645, Cham, 2016. Springer International Publishing. doi: 10.1007/978-3-319-46493-0_38.

[45] Charlie Nash and Conor Durkan. Autoregressive Energy Machines, 2019. URL https://arxiv.org/abs/1904.05626.

[46] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural Spline Flows, 2019. URL https://arxiv.org/abs/1906.04032.

[47] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs), 2016. URL https://arxiv.org/abs/1606.08415.

[48] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts, 2016. URL https://arxiv.org/abs/1608.03983.

[49] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Trans. Graph.*, 38(5):145:1–145:19, October 2019. ISSN 0730-0301. doi: 10.1145/3341156. URL http://doi.acm.org/10.1145/3341156.

[50] Christina Gao, Joshua Isaacson, and Claudius Krause. i- flow: High-dimensional integration and sampling with normalizing flows. *Machine Learning: Science and Technology*, 1(4):045023, oct 2020. doi: 10.1088/2632-2153/abab62. URL https://dx.doi.org/10.1088/2632-2153/abab62.

[51] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests, 2016. URL https://arxiv.org/abs/1610.06545.

[52] Lukas Heinrich, Matthew Feickert, Giordon Stark, and Kyle Cranmer. pyhf: pure-python implementation of histfactory statistical models. *Journal of Open Source Software*, 6(58):2823, 2021. doi: 10.21105/joss.02823.

[53] Lukas Heinrich, Matthew Feickert, and Giordon Stark. pyhf: v0.7.3. https://github.com/scikit-hep/pyhf/releases/tag/v0.7.3.

[54] Alexander L. Read. Presentation of search results: the CL_S technique. *J. Phys. G*, 28:2693, 2002. doi: 10.1088/0954-3899/28/10/313.

[55] ATLAS Collaboration. Recommendations for the Modeling of Smooth Backgrounds. Technical report, CERN, Geneva, 2020. URL https://cds.cern.ch/record/2743717.