symmetry

MDPI

Article

Comparison of Affine and Rational Quadratic Spline Coupling and Autoregressive Flows through Robust Statistical Tests

Andrea Coccaro [1,*], Marco Letizia [1,2,*], Humberto Reyes-González [1,3,4,*] and Riccardo Torre [1,*]

[1] INFN, Sezione di Genova, Via Dodecaneso 33, 16146 Genova, Italy
[2] MaLGa—DIBRIS, Università degli Studi di Genova, Via Dodecaneso 35, 16146 Genova, Italy
[3] Dipartimento di Fisica, Università degli Studi di Genova, Via Dodecaneso 33, 16146 Genova, Italy
[4] Institut für Theoretische Teilchenphysik und Kosmologie, RWTH Aachen, 52074 Aachen, Germany
* Correspondence: andrea.coccaro@ge.infn.it (A.C.); marco.letizia@edu.unige.it (M.L.); humberto.reyes@rwth-aachen.de (H.R.-G.); riccardo.torre@ge.infn.it (R.T.)

Abstract: Normalizing flows have emerged as a powerful brand of generative models, as they not only allow for efficient sampling of complicated target distributions but also deliver density estimation by construction. We propose here an in-depth comparison of coupling and autoregressive flows, both based on symmetric (affine) and non-symmetric (rational quadratic spline) bijectors, considering four different architectures: real-valued non-Volume preserving (RealNVP), masked autoregressive flow (MAF), coupling rational quadratic spline (C-RQS), and autoregressive rational quadratic spline (A-RQS). We focus on a set of multimodal target distributions of increasing dimensionality ranging from 4 to 400. The performances were compared by means of different test statistics for two-sample tests, built from known distance measures: the sliced Wasserstein distance, the dimension-averaged one-dimensional Kolmogorov–Smirnov test, and the Frobenius norm of the difference between correlation matrices. Furthermore, we included estimations of the variance of both the metrics and the trained models. Our results indicate that the A-RQS algorithm stands out both in terms of accuracy and training speed. Nonetheless, all the algorithms are generally able, without too much fine-tuning, to learn complicated distributions with limited training data and in a reasonable time of the order of hours on a Tesla A40 GPU. The only exception is the C-RQS, which takes significantly longer to train, does not always provide good accuracy, and becomes unstable for large dimensionalities. All algorithms were implemented using TensorFlow2 and TensorFlow Probability and have been made available on GitHub.

Keywords: machine learning; generative models; density estimation; normalizing flows

Citation: Coccaro, A.; Letizia, M.; Reyes-González, H.; Torre, R. Comparison of Affine and Rational Quadratic Spline Coupling and Autoregressive Flows through Robust Statistical Tests. *Symmetry* **2024**, *16*, 942. https://doi.org/10.3390/sym16080942

Academic Editor: Marcin Michalak

Received: 12 June 2024
Revised: 15 July 2024
Accepted: 17 July 2024
Published: 23 July 2024

1. Introduction

The modern data science revolution has opened a great window of opportunities for scientific and societal advancement. In particular, machine learning (ML) technologies are being applied in a wide variety of fields from finance to astrophysics. It is thus crucial to carefully study the capabilities and limitations of ML methods in order to ensure their systematic usage. This is particularly pressing when applying ML to scientific research, for instance in a field such as high-energy physics (HEP), where one often deals with complicated high-dimensional data and high levels of precision are needed.

In this paper, we focus on normalizing flows (NFs) [1–4], a class of neural density estimators that for one, offers a competitive approach to generative models, such as generative adversarial networks (GANs) [5] and variational autoencoders (VAEs) [6,7], for the generation of synthetic data and, for another, opens up a wide range of applications due to its ability to directly perform density estimation. Even though we have in mind applications of NFs to HEP, in this paper, we remain agnostic with respect to the applications and only performed a general comparative study of the performances of coupling and autoregressive NFs when used to learn high-dimensional multi-modal target distributions. Nevertheless,

it is worth mentioning some of the applications of NFs to HEP can also be extended to several other fields of scientific research.

While applications of the generative direction of NFs is rather obvious in a field such as HEP, which bases its foundations on Monte Carlo simulations, it is interesting to mention some of the possible density estimation applications. The ability to directly learn the likelihood, or the posterior in a Bayesian framework, has applications ranging from analysis, inference, reinterpretation, and preservation to simulation-based likelihood-free inference [8–13], unfolding of HEP analyses [14], generation of effective priors for Bayesian inference [15–20], systematic uncertainty estimation and parametrization, generation of effective proposals for sequential Monte Carlo [21–28], numerical integration based on importance sampling algorithms [29–32], and probabilistic programming applied to fast inference [33,34].

The basic principle behind NFs is to perform a series of invertible bijective transformations on a simple base probability density function (PDF) to approximate a complicated PDF of interest. The optimal parameters of the transformations, often called "bijectors", are derived from training neural networks (NNs) that directly take the negative log-likelihood of the true data computed with the NF distribution as the loss function. As it turns out, PDFs are everywhere in HEP: from the likelihood function of an experimental or a phenomenological result to the distribution that describes a particle-collision process. Thus, NFs have found numerous applications in HEP: they have been used for numerical integration and event generation [35–42], anomaly detection [43–45], detector unfolding [46,47], etc.

The growing interest in NFs implies the urgency of testing state-of-the-art architectures against complex data to ensure their systematic usability and to assess their expected performances. The purpose of this work was then to evaluate the performance of NFs against generic complicated distributions of increasing dimensionality. By performing this study, we aimed to make a concrete step forward in the general understanding of the realistic performances and properties of NFs, especially in high-precision scenarios. This work comprises a substantial upgrade with respect to our early study of Ref. [48], as we now have included more NF architectures, extended the dimensionality of the distributions, and significantly improved the testing strategy.

Our strategy was the following. We implemented in PYTHON, using TENSORFLOW2 with TENSORFLOW PROBABILITY, four of the most commonly used NF architectures of the coupling and autoregressive type: real-valued non-volume preserving (RealNVP) [49], masked autoregressive flow (MAF) [50], coupling rational quadratic spline (C-RQS) [51], and autoregressive rational quadratic spline (A-RQS) [51].

We tested these NF architectures considering correlated mixture of Gaussian (CMoG) multi-modal distributions with dimensionalities ranging from 4 to 400. We also performed a small-scale hyperparameter scan, explicitly avoiding the fine-tuning of the models and generating the best result for each NF architecture and target distribution.

The performances were measured by means of different test statistics for two-sample testing built from known distance measures: the sliced Wasserstein distance, the dimension-averaged one-dimensional Kolmogorov–Smirnov statistic, and the Frobenius norm of the difference between correlation matrices. The analyses were performed by comparing a test sample, drawn from the original distribution, with an NF-generated one. Moreover, all test-statistics calculations were cross-validated, and an error was assigned both to the evaluation procedure, with repeated calculations of the metrics on different instances of the test and NF-generated samples, and to the training procedure, with repeated calculations on models trained with different instances of the training sample.

This paper is organized as follows. In Section 2, we describe the concept of NFs in more detail, focusing on the coupling and autoregressive types. In Section 3, we introduce the specific NF architectures under investigation. In Section 4, we present the metrics used in our analysis, and in Section 5, we discuss our results. Finally, we provide our concluding remarks in Section 6, with emphasis on the several prospective research avenues that we plan to follow.

2. Normalizing Flows

Normalizing Flows are made of series of bijective, continuous, and invertible transformations that map a simple *base* PDF to a more complicated *target* PDF. The purpose of NFs is to estimate the unknown underlying distribution of some data of interest and to allow for the generation of samples approximately following the same distribution. Since the parameters of both the base distribution and the transformations are known, one can *generate* samples from the target distribution by drawing samples from the base distribution and then applying the proper transformation. This is known as the *generative direction* of the flow. Furthermore, since the NF transformations are invertible, one can also obtain the probability density of the true samples via inverse transformations from the target to the base PDF. This is known as the *normalizing direction* of the flow. It is called "normalizing" because the base distribution is often Gaussian even though this is not a requirement, and this is also the origin of the name "normalizing flows".

The basic idea behind NFs is the change of variable formula for a PDF. Let $X, Y \in \mathbb{R}^D$ be random variables with PDFs $p_X, p_Y : \mathbb{R}^D \to \mathbb{R}$. Let us define a bijective map $\mathbf{g} : X \to Y$, with inverse $\mathbf{f} = \mathbf{g}^{-1}$. The two densities are then related by the well known formula

$$p_Y(y) = p_X(\mathbf{g}^{-1}(y)) \mid \det \mathrm{J}_g \mid^{-1} = p_X(\mathbf{f}(y)) \mid \det \mathrm{J}_f \mid \tag{1}$$

where $\mathrm{J}_f = \frac{\partial \mathbf{f}}{\partial y}$ is the Jacobian of $\mathbf{f}(y)$, and $\mathrm{J}_g = \frac{\partial \mathbf{g}}{\partial x}$ is the Jacobian of $\mathbf{g}(x)$. (Throughout the paper we always interpret X as the base distribution and Y as the target distribution, i.e., the data. We also always model flows in the generative direction, from base to data).

Let us now consider a set of parameters $\{\phi\}$ characterizing the chosen base density p_X (typically the mean vector and covariance matrix of a multivariate Gaussian) and parametrize the map $\mathbf{g}$ by another set of parameters $\{\theta\}$. One can then perform a maximum likelihood estimation of the parameters $\Phi = \{\phi, \theta\}$ given some measured data $\mathcal{D} = \{y^I\}_{I=1}^N$ distributed according to the unknown PDF p_y. The log-likelihood of the data is given by the following expression:

$$\begin{aligned} \log p(\mathcal{D} \mid \Phi) &= \sum_{I=1}^{N} \log p_Y(y^I \mid \Phi) \\ &= \sum_{I=1}^{N} \log p_X(\mathbf{f}_\theta(y^I) \mid \theta, \phi) + \log \mid \det \mathrm{J}_f \mid \end{aligned} \tag{2}$$

where we make the dependence of $\mathbf{f}$ on θ explicit through the notation $\mathbf{f}_\theta$. Then, the best estimate of the parameters Φ is given by

$$\hat{\Phi} = \arg\max_{\Phi} \log p(\mathcal{D} \mid \Phi) \tag{3}$$

Once the parameters $\hat{\Phi}$ have been estimated from the data, the approximated target distribution can be sampled by applying the generative map $\mathbf{g}$ to samples obtained from the base PDF. The normalizing direction $\mathbf{f}$ can instead be used to perform density evaluation by transforming the new data of interest into sample generated by the base PDF, which is easier to evaluate.

Beside being invertible, the map $\mathbf{g}$ should satisfy the following properties:

- It should be sufficiently expressive to appropriately model the target distribution;
- It should be computationally efficient, meaning that both $\mathbf{f}$ (for training, this means computing the likelihood) and $\mathbf{g}$ (for generating samples), as well as their Jacobian determinants, must be easily calculable.

The composition of invertible bijective functions is also an invertible bijective function. Thus, $\mathbf{g}$ can be generalized to a set of N_t transformations as $\mathbf{g} = g_{N_t} \circ g_{N_t-1} \circ \dots g_1$ with inverse $\mathbf{f} = f_1 \circ \dots f_{N_t-1} \circ f_{N_t}$ and $\det \mathrm{J}_f = \prod_{n=1}^{N_t} \det \mathrm{J}_{f_n}$, where each $f_n = g_n^{-1}$ depends on a y_n intermediate random variable. This is a standard strategy to increase the flexibility of the overall transformation.

Typically, but not mandatorily, NF models are implemented using NNs to determine the parameters of the bijectors. The optimal values are obtained by minimizing a loss function corresponding to minus the log-likelihood defined as in Equation (2). This is a natural approach when the samples from the target density are available but the density itself cannot be evaluated, a common occurrence in fields, such as HEP, that heavily rely on Monte Carlo simulations. (Approaches beyond maximum likelihood, which use different loss functions, have also been considered in the literature, such as in Refs [52–57]. In this paper we always use the maximum likelihood approach and minus the log-likelihood as loss function.) This makes the models extremely flexible, with a usually stable training, at the cost of a potentially large number of parameters. Nonetheless, the flow transformation must be carefully designed; for instance, even if a given map and its inverse, with their respective Jacobians, are computable, one direction might be more efficient than the other, leading to models that favor sampling over evaluation (and training) or vice versa. Among the wide and growing variety of NF architectures available (see Ref. [58] for an overview), we focus in this work on *coupling* [4] and *autoregressive* flows [59], arguably the most widely used implementations of NFs, particularly in HEP.

2.1. Coupling Flows

Coupling flows, originally introduced in Ref. [4], are made of stacks of so-called coupling layers, in which each sample with dimension D is partitioned into two samples A and B with dimensions d and $D-d$, respectively. The parameters of the bjiector transforming the sample A are modeled by a NN that uses B as input, effectively constructing the $p(y_d|x_{d-D})$ conditional probability distributions. At each coupling layer in the stack, different partitionings are applied, usually by shuffling the dimensions before partitioning, so that all dimensions are properly transformed.

In other words, starting from a disjoint partition of a random variable $Y \in \mathbb{R}^D$ such that $(y^A, y^B) \in \mathbb{R}^d \times \mathbb{R}^{D-d}$ and a bijector $\mathbf{h}(\,\cdot\,;\theta) : \mathbb{R}^d \to \mathbb{R}^d$, a coupling layer maps $\mathbf{g} : X \to Y$ as follows:

$$\begin{aligned} y^A &= \mathbf{h}(x^A; \Theta(x^B))\,, \\ y^B &= x^B \end{aligned} \tag{4}$$

where the parameters θ are defined by a generic function $\Theta(x^B)$ only defined on the $\mathbb{R}^{D-d}$ partition, generally modeled by an NN. The function $\Theta(x^B)$ is called a *conditioner*, while the bijectors $\mathbf{h}$ and $\mathbf{g}$ are called *coupling function* and *coupling flow*, respectively. The necessary and sufficient condition for the coupling flow $\mathbf{g}$ to be invertible is that the coupling function $\mathbf{h}$ is invertible. In this case, the inverse transformation is given by

$$\begin{aligned} x^A &= \mathbf{h}^{-1}(y^A; \Theta(x^B))\,, \\ x^B &= y^B \end{aligned} \tag{5}$$

Notice that despite the presence of a NN, whose inverse is unknown, to parametrize the conditioner, the invertibility of $\mathbf{h}$ is guaranteed by the fact that such a conditioner is a function of the unchanged dimensions only. The Jacobian of $\mathbf{g}$ is then a two-block triangular matrix. The dimensions $\{1 : d\}$ are given by the Jacobian of $\mathbf{h}$, and the dimensions $\{d : D\}$ is the identity matrix. Thus, the Jacobian determinant is simply the following:

$$\det \mathrm{J}_g = \prod_{i=1}^{d} \frac{\partial h_i}{\partial x_i^A}. \tag{6}$$

Note that the choice of the partition is arbitrary. The most common choice is to split the dimensions in half, but other partitions are possible [58]. Obviously, even when dimensions are halved, the way in which they are halved is not unique and can be implemented in several different ways, for instance through random masks, or through random shuffling before dividing the first and last half.

2.2. Autoregressive Flows

Autoregressive flows, first introduced in Ref. [59], can be viewed as a generalization of coupling flows. Now, the transformations of each dimension i are modeled by an autoregressive DNN according to the previously transformed dimensions of the distribution, resulting in the $p(y_i|y_{1:i-1})$ conditional probability distributions, where $y_{1:i-1}$ is a shorthand notation to indicate the list of variables $y_1, \dots y_{i-1}$. After each autoregressive layer, the dimensions are permuted to ensure the expressivity of the bijections over the full dimensionality of the target distribution.

Let us consider a bijector $\mathbf{h}(\,\cdot\,;\theta) : \mathbb{R} \to \mathbb{R}$, parametrized by θ. We can define an autoregressive flow function $\mathbf{g}$ such that

$$\begin{aligned} y_1 &= x_1\,, \\ y_i &= \mathbf{h}(x_i; \Theta_i(y_{1:i-1}))\,, \quad i = 2, \dots, D \end{aligned} \tag{7}$$

The resulting Jacobian of $\mathbf{g}$ is again a triangular matrix, whose determinant is easily computed as

$$\det \mathrm{J}_g = \prod_{i=1}^{D} \frac{\partial h_i}{\partial x_i}. \tag{8}$$

where $\partial h_i / \partial x_i$ are the diagonal terms of the Jacobian.

Given that the structure of the bijector is similar to that of the coupling flow, also, in this case, the bijector is referred to as a *coupling function*. Note that Θ_j can also be alternatively determined with the precedent untransformed dimensions of X [59] such that

$$\begin{aligned} y_1 &= x_1\,, \\ y_i &= \mathbf{h}(x_i; \Theta_i(x_{1:i-1}))\,, \quad i = 2, \dots, D \end{aligned} \tag{9}$$

The choice of variables used to model the conditioner may depend on whether the NF is intended for sampling or density estimation. In the former case, Θ is usually chosen to be modeled from the base variable X so that the transformations in the generative direction would only require one forward pass through the flow. The transformations in the normalizing direction would instead require D iterations trough the autoregressive architecture. This case is referred to as *inverse autoregressive flow* [59] (Notice that in Ref. [58], parametrizing the flow in the normalizing direction (the opposite of our choice), apparently uses the inverse of our formulas for direct and inverse flows. Our notation (and nomenclature) is consistent with that of Ref. [50]) and corresponds to the transformations in Equation (9). Conversely, in the case of density estimation, it is convenient to parametrize the conditioner using the target variable Y since transformations would be primarily in the normalizing direction. This case is referred to as *direct autoregressive flow* and corresponds to the transformations in Equation (7). In any case, when training the NFs, one always needs to perform the normalizing transformations to estimate the log-likelihood of the data, as in Equation (2). In our study, we only consider the direct autoregressive flow described by Equation (7).

3. Architectures

In the previous section, we describe NFs, focusing on the two most common choices for parametrizing the bijector $\mathbf{g}$ in terms of the coupling function $\mathbf{h}$. The only missing ingredient to make NFs concrete, remains the explicit choice of $\mathbf{h}$. For this study, we have chosen four of the most popular implementations of coupling and autoregressive flows: the real-valued non-volume preserving (RealNVP) [49], the masked autoregressive flow (MAF) [50], and the coupling and autoregressive rational-quadratic neural spline flows (C-RQS and A-RQS) [51]. (Reference [51] refers to coupling and autoregressive RQS flows as RQ-NSF (C) and RQ-NSF (AR), where RQ-NSF stands for rational-quadratic neural spline flow, and A and C for autoregressive and coupling, respectively). We discuss them in turn in the following subsections and give additional details about our specific implementation in Appendices A.1–A.4.

3.1. The RealNVP

The RealNVP [49] is a type of coupling flow whose coupling functions **h** are affine functions with the following form:

$$\begin{aligned} y_i &= x_i\,, & i = 1,\ldots,d\,,\\ y_i &= x_i e^{s_{i-d}(x_{1:d})} + t_{i-d}(x_{1:d})\,, & i = d+1,\ldots,D \end{aligned} \tag{10}$$

where the s and t functions, defined on $\mathbb{R}^d \to \mathbb{R}^{D-d}$, respectively correspond to the scale and translation transformations modeled by NNs. The product in Equations (10) is intended elementwise for each i so that, x_{d+1} is multiplied by s_1, x_{d+2} by s_2 and so on up to x_D, which is multiplied by s_{D-d}. The Jacobian of this transformation is a triangular matrix with diagonal $\mathrm{diag}(\mathbb{I}_d, \mathrm{diag}(\exp(s_{i-d}(x_{1:d}))))$ with $i = d+1,\ldots,D$, so that its determinant is independent of t and is simply given by

$$\det \mathrm{J} = \prod_{i=1}^{D-d} e^{s_i(x_{1:d})} = \exp\left(\sum_{i=1}^{D-d} s_i(x_{1:d})\right) \tag{11}$$

The inverse of Equation (10) is given by

$$\begin{aligned} x_i &= y_i\,, & i = 1,\ldots,d\,,\\ x_i &= (y_i - t_{i-d}(y_{1:d}))e^{-s_{i-d}(y_{1:d})}\,, & i = d+1,\ldots,D \end{aligned} \tag{12}$$

A crucial property of the affine transformation (10) is that its inverse (12) is again an affine transformation depending only on s and t, and not on their inverse. This implies that the s and t functions can be arbitrarily complicated (indeed they are parametrized by a DNN), still leaving the RealNVP equally efficient in the forward (generative) and backward (normalizing) directions.

3.2. The MAF

The MAF algorithm was developed starting from the masked autoencoder for distribution estimation (MADE) [60] approach for implementing an autoregressive neural network through layers masking (see Appendix A.2).

In the original MAF implementation [50], the bijectors are again affine functions described as

$$\begin{aligned} y_1 &= x_1\,,\\ y_i &= x_i e^{s_{i-1}(y_{1:i-1})} + t_{i-1}(y_{1:i-1})\,, & i = 2,\ldots,D \end{aligned} \tag{13}$$

The functions s and t are now defined on $\mathbb{R}^{D-1} \to \mathbb{R}^{D-1}$. The determinant of the Jacobian is simply

$$\det \mathrm{J} = \prod_{i=1}^{D-1} e^{s_i(y_{1:i})} = \exp\left(\sum_{i=1}^{D-1} s_i(y_{1:i})\right) \tag{14}$$

and the inverse transformation is

$$\begin{aligned} x_1 &= y_1\,,\\ x_i &= (y_i - t_{i-1}(y_{1:i-1}))e^{-s_{i-1}(y_{1:i-1})}\,, & i = 2,\ldots,D\,. \end{aligned} \tag{15}$$

As in the case of the RealNVP, the affine transformation guarantees that the inverse transformation only depends on s and t and not on their inverse, allowing for the choice of arbitrarily complicated functions without affecting computational efficiency.

3.3. The RQS Bijector

The bijectors in a coupling or masked autoregressive flow are not restricted to affine functions. It is possible to implement more expressive transformations as long as they remain invertible and computationally efficient. This is the case of the so-called rational-quadratic neural spline flows [51].

The spline bijectors are made of K bins, where in each bin one defines a monotonically increasing rational-quadratic function. The binning is defined on an interval $\mathbb{B} = [-B, B]$,

outside of which the function is set to the identity transformation. The bins are defined by a set of $K+1$ coordinates $\{(x_i^{(k)}, y_i^{(k)})\}_{k=0}^{K}$, called *knots*, strictly monotonically increasing between $\{(x_i^{(0)}, y_i^{(0)}) = (-B, -B)$ and $\{(x_i^{(K)}, y_i^{(K)}) = (B, B)$. We use the bracket index notation to denote knots' coordinates, which are defined for each dimension of the vectors x_i and y_i. It is possible to construct a rational-quadratic spline bijector with the desired properties with the following procedure [61].

Let us define the quantities

$$\begin{aligned} h_i^{(k)} &= x_i^{(k+1)} - x_i^{(k)}\,, \\ \Delta_i^{(k)} &= (y_i^{(k+1)} - y_i^{(k)})/h_i^{(k)}\,. \end{aligned} \tag{16}$$

Obviously, $\Delta_i^{(k)}$ represents the variation of y_i with respect to the variation of x_i within the k-th bin. Moreover, since we assumed strictly monotonically increasing coordinates, $\Delta_i^{(k)}$ is always positive or zero. We are interested in defining a bijector $\mathbf{g}(x_i)$ and mapping the $\mathbb{B}$ interval to itself, such that $\mathbf{g}(x_i^{(k)}) = y_i^{(k)}$, and with derivatives $d_i^{(k)} = dy_i^{(k)}/dx_i^{(k)}$ satisfying the following conditions:

$$\begin{aligned} d_i^{(k)} = d_i^{(k+1)} &= 0 \quad \text{for } \Delta_i^{(k)} = 0\,, \\ d_i^{(k)}, d_i^{(k+1)} &> 0 \quad \text{for } \Delta_i^{(k)} > 0 \end{aligned} \tag{17}$$

Such condition is necessary and also sufficient, in the case of a rational quadratic function, to ensure monotonicity [61]. Moreover, for the boundary knots, we set $d_i^{(0)} = d_i^{(K)} = 1$ to match the linear behavior outside the $\mathbb{B}$ interval.

For $x_i \in [x_i^{(k)}, x_i^{(k+1)}]$, we define

$$\theta_i = (x_i - x_i^{(k)})/h_i^{(k)} \tag{18}$$

such that $\theta_i \in [0, 1]$. Then, for x_i in each of the intervals $[x_i^{(k)}, x_i^{(k+1)}]$ with $k = 0, \ldots, K-1$, we define

$$y_i = P_i^{(k)}(\theta_i)/Q_i^{(k)}(\theta_i) \tag{19}$$

with the functions P and Q defined by

$$\begin{aligned} P_i^{(k)}(\theta_i) &= \Delta_i^{(k)} y_i^{(k+1)} \theta_i^2 + \Delta_i^{(k)} y_i^{(k)} (1-\theta_i)^2 \\ &\quad + (y_i^{(k)} d_i^{(k+1)} + y_i^{(k+1)} d_i^{(k)}) \theta_i (1-\theta_i)\,, \\ Q_i^{(k)}(\theta_i) &= \Delta_i^{(k)} + (d_i^{(k+1)} + d_i^{(k)} - 2\Delta_i^{(k)}) \theta_i (1-\theta_i) \end{aligned} \tag{20}$$

The ratio in Equation (19) can then be written in the simplified form

$$y_i = y_i^{(k)} + \frac{(y_i^{(k+1)} - y_i^{(k)})(\Delta_i^{(k)} \theta_i^2 + d_i^{(k)} \theta_i (1-\theta_i))}{\Delta_i^{(k)} + (d_i^{(k+1)} + d_i^{(k)} - 2\Delta_i^{(k)}) \theta_i (1-\theta_i)} \tag{21}$$

The Jacobian $\mathrm{J}_{\mathbf{g}} = \partial y_i / \partial x_j$ is then diagonal, with entries given by

$$\frac{(\Delta_i^{(k)})^2 (d_i^{(k+1)} \theta_i^2 + 2\Delta_i^{(k)} \theta_i (1-\theta_i) + d_i^{(k)} (1-\theta_i)^2)}{(\Delta_i^{(k)} + (d_i^{(k+1)} + d_i^{(k)} - 2\Delta_i^{(k)}) \theta_i (1-\theta_i))^2} \tag{22}$$

for $i = 1, \ldots, D$. The inverse of the transformation (19) can also be easily computed by solving the quadratic Equations (19) with respect to x_i.

In practice, B and K are hyperparameters, while $\{(x_i^{(k)}, y_i^{(k)})\}_{k=0}^{K}$ and $\{d_i^{(k)}\}_{k=1}^{K-1}$ are $2(K+1)$ plus $K-1$ parameters, modeled by an NN, which determine the shape of the spline function. The different implementations of the RQS bijector, in the context of coupling and autoregressive flows, are determined by the way in which such parameters are computed. We briefly describe them in turn in the following two subsections.

3.4. The C-RQS

In the coupling flow case (C-RQS), one performs the usual partitioning of the D dimensions in the two sets composed of the first d and last $D-d$ dimensions. The first d dimensions are then kept unchanged $y_i = x_i$ for $i = 1, \ldots, d$, while the parameters describing the RQS transformations of the other $D-d$ dimensions are determined from the inputs of the first d dimensions, denoted by $x_{1:d}$. Schematically, we could write

$$\begin{aligned} x_i^{(k)} &= x_i^{(k)}(x_{1:d})\,, \\ y_i^{(k)} &= y_i^{(k)}(x_{1:d})\,, \\ d_i^{(k)} &= d_i^{(k)}(x_{1:d}) \end{aligned} \tag{23}$$

for $i = d+1, \ldots, D$.

A schematic description of our implementation of the C-RQS is given in Appendix A.3.

3.5. The A-RQS

The RQS version of the MAF, which we call A-RQS, is instead obtained by leaving unchanged the first dimension $y_1 = x_1$ and determining the parameters of the transformation of the i-th dimension from the output of all preceding dimensions, denoted by $y_{1:i-1}$. Schematically, this is given by

$$\begin{aligned} x_i^{(k)} &= x_i^{(k)}(y_{1:i-1})\,, \\ y_i^{(k)} &= y_i^{(k)}(y_{1:i-1})\,, \\ d_i^{(k)} &= d_i^{(k)}(y_{1:i-1}) \end{aligned} \tag{24}$$

for $i = 2, \ldots, D$.

4. Non-Parametric Quality Metrics

We assessed the performance of our trained models using three distinct metrics: the dimension-averaged 1D Kolmogorov–Smirnov (KS) two-sample test statistic $\overline{D}_{y,z}$, the sliced Wasserstein distance (SWD) $\overline{W}_{y,z}$, and the Frobenius norm (FN) of the difference between the correlation matrices of two samples $\|C\|_F$. With a slight abuse of nomenclature, we refer to these three different distance measures with vanishing optimal value simply as KS, SWD, and FN, respectively. Each of these metrics served as a separate test statistic in a two-sample test, where the null hypothesis assumed that both samples originate from the same target distribution. For each metric, we established its distribution under the null hypothesis by drawing both samples from the target distribution. We then compared this distribution with the test-statistic calculated from a two-sample test between samples drawn from the target and NF distributions to assign each model a p-value for rejecting the null hypothesis.

To quantify the uncertainty on the test-statistics computed for the test vs NF-generated samples, we performed the tests 10 times using differently seeded target- and NF-generated samples. We calculated p-values based on the mean test statistic and its ± 1 standard deviation.

For model comparison and to assess the uncertainty on the training procedure, we trained 10 instances of each model configuration, defined by a set of hyperparameter values. Rather than selecting the single best-performing instance to represent the best architecture, we averaged the performances across these instances and identified the architecture with the best average performance. After selecting this top-performing model, which we called the "best" model, we reported both its average and peak performances.

When computing the test-statistics distributions under the null hypothesis and evaluating each model's p-values, we found that the discriminative power of the KS metric was larger than that of the SWD and FN ones. For this reason, we used the result of the KS-statistic to determine the best model and then showed results also for the other two statistics. Nevertheless, even though the best model could vary depending on the metric used, no qualitative difference in the conclusions would arise from choosing FN or

SWD as the ranking metric; i.e., results were not identical but were consistent among the three metrics.

It is important to stress that despite our insistence on using non-parametric quality metrics, we actually know the target density, and we used this information for bootstrapping uncertainties and computing p-values. In real-world examples, the target density is generally not known, and depending on the number of available samples, our procedure for evaluation needs to be adapted or may end up being unusable. Nevertheless, this well-defined statistical approach is crucial for us since we aim to draw rather general conclusions, which strongly depend on the ability to estimate the uncertainties and should rely on robust statistical inference.

In the following, we briefly introduce the three aforementioned metrics. To do so, we employ the following notation: we indicate with N the number of D-dimensional points in each sample and use capital indices I, J to run over N and lowercase indices i, j to run over D (We warn the reader not to confuse the dimensionality$D$ with the KS test-statistic $D_{x,y}$.). We also use Greek letters indices α, β to run over slices (random directions).

- **Kolmogorov–Smirnov test**
  The KS test is a statistical test used to determine whether or not two 1D samples are drawn from the same *unknown* PDF. The null hypothesis is that both samples come from the same PDF. The KS test statistic is given by

$$D_{y,z} = \mathbf{sup}_x \mid F_y(x) - F_z(x) \mid , \tag{25}$$

  where $F_{y,z}(x)$ are the empirical distributions of each of the samples $\{y_I\}$ and $\{z_I\}$, and **sup** is the supremum function. For characterizing the performances of our results, we computed the KS test-statistic for each of the 1D marginal distributions along the D dimensions and took the average as follows:

$$\overline{D}_{y,z} = \frac{1}{D} \sum_{i=1}^{D} D^i_{y,z} \, . \tag{26}$$

  The actual test statistic that we consider in this paper is the scaled version of $\overline{D}_{y,z}$, given by

$$t_{\mathrm{KS}} = \sqrt{\frac{N}{2}} \overline{D}_{y,z} \tag{27}$$

  The $\sqrt{N/2}$ factor comes from the known $\sqrt{m \cdot n/(m+n)}$ factor in the scaled KS statistic with different-sized samples of sizes m and n, respectively.
  Notice that even though the test statistic

$$\sqrt{\frac{m \cdot n}{m+n}} D_{y,z} \tag{28}$$

  is asymptotically distributed according to the Kolmogorov distribution [62–66], the same is not true for our t_{KS} statistic due to correlations among dimensions. Nevertheless, our results seem to suggest that the asymptotic distribution of t_{KS} for large D (that means when the average is taken over many dimensions) has a reasonably universal behavior, translating into almost constant rejection lines (solid gray lines with different thicknesses) in the upper panels of Figure 1.
- **Sliced Wasserstein Distance**
  The SWD [67,68] is a distance measure for comparing two multidimensional distributions based on the 1D Wasserstein distance [69,70]. The latter distance between two univariate distributions is given as a function of their respective empirical distributions as follows

$$W_{y,z} = \int_{\mathbb{R}} dx \mid F_y(x) - F_z(x) \mid \tag{29}$$

  Intuitively, the difference between the WD and the KS test statistic is that the latter considers the maximum distance, while the former is based on the integrated distance. Our implementation of the SWD is defined as follows. For each model and each dimensionality D, drew $2D$ random directions $\hat{v}^i_\alpha$, with $i = 1, \ldots D$ and $\alpha = 1, \ldots, 2D$,

uniformly distributed on the surface of the unit N-sphere (This can be done by normalizing the D-dimensional vector obtained by sampling each components from an independent standard normal distribution [71]). Given two D-dimensional samples $\{\mathbf{y}_I\} = \{y_I^i\}$ and $\{\mathbf{z}_I\} = \{z_I^i\}$, we considered the $2D$ projections

$$\{y_I^\alpha\} = \left\{\sum_{i=1}^{D} y_I^i \hat{v}_i^\alpha\right\}, \qquad \{z_I^\alpha\} = \left\{\sum_{i=1}^{D} z_I^i \hat{v}_i^\alpha\right\} \tag{30}$$

and computed the corresponding $2D$ Wasserstein distances as follows:

$$W_{y,z}^\alpha = \int_{\mathbb{R}} dx \mid F_{y^\alpha}(x) - F_{z^\alpha}(x) \mid \tag{31}$$

was

$$\overline{W}_{y,z} = \frac{1}{2D} \sum_{\alpha=1}^{2D} W_{y,z}^\alpha \,. \tag{32}$$

In analogy with the scaled KS test statistic, we defined the scaled SWD test statistic as follows:

$$t_{\mathrm{SWD}} = \sqrt{\frac{N}{2}} \overline{W}_{y,z} \tag{33}$$

- **Frobenius norm**
  The FN of a matrix M is given by

$$\|M\|_F = \sqrt{\sum_{i,j} \mid m_{ij} \mid^2}, \tag{34}$$

where m_{ij} are the elements of M. By defining $C = C_y - C_z$, where C_y, C_z are the two $N \times N$ correlation matrices of the samples $\{\mathbf{y}_I\} = \{y_I^i\}$ and $\{\mathbf{z}_I\} = \{z_I^i\}$, its FN, given by

$$\|C\|_F = \sqrt{\sum_{i,j} \mid c_{y,ij} - c_{z,ij} \mid^2} \tag{35}$$

is a distance measure between the two correlation matrices. In analogy with the previously defined test statistics, we defined the scaled FN test statistic as follows:

$$t_{\mathrm{FN}} = \sqrt{\frac{N}{2}} \frac{\|C\|_F}{D}, \tag{36}$$

where we also divided by the number of dimensions D to remove the approximately linear dependence of the FN distance on the dimensionality of the samples.

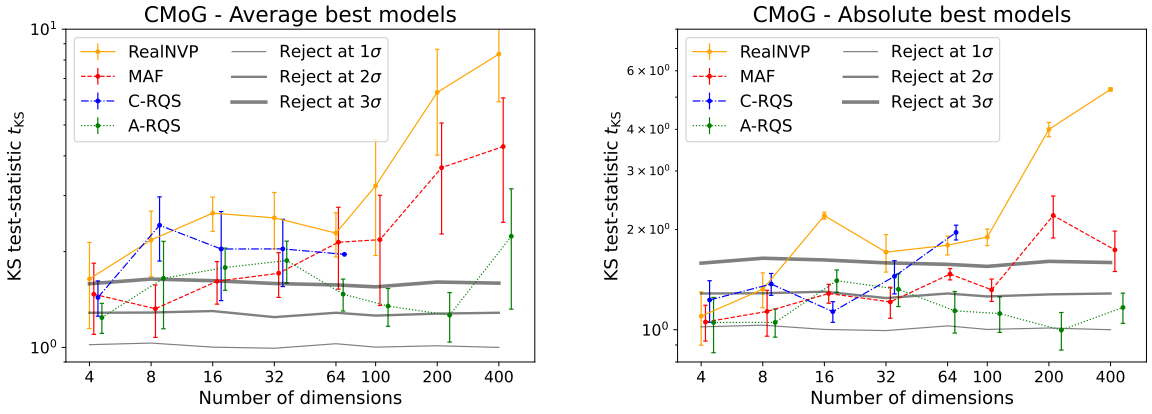

Figure 1. *Cont.*

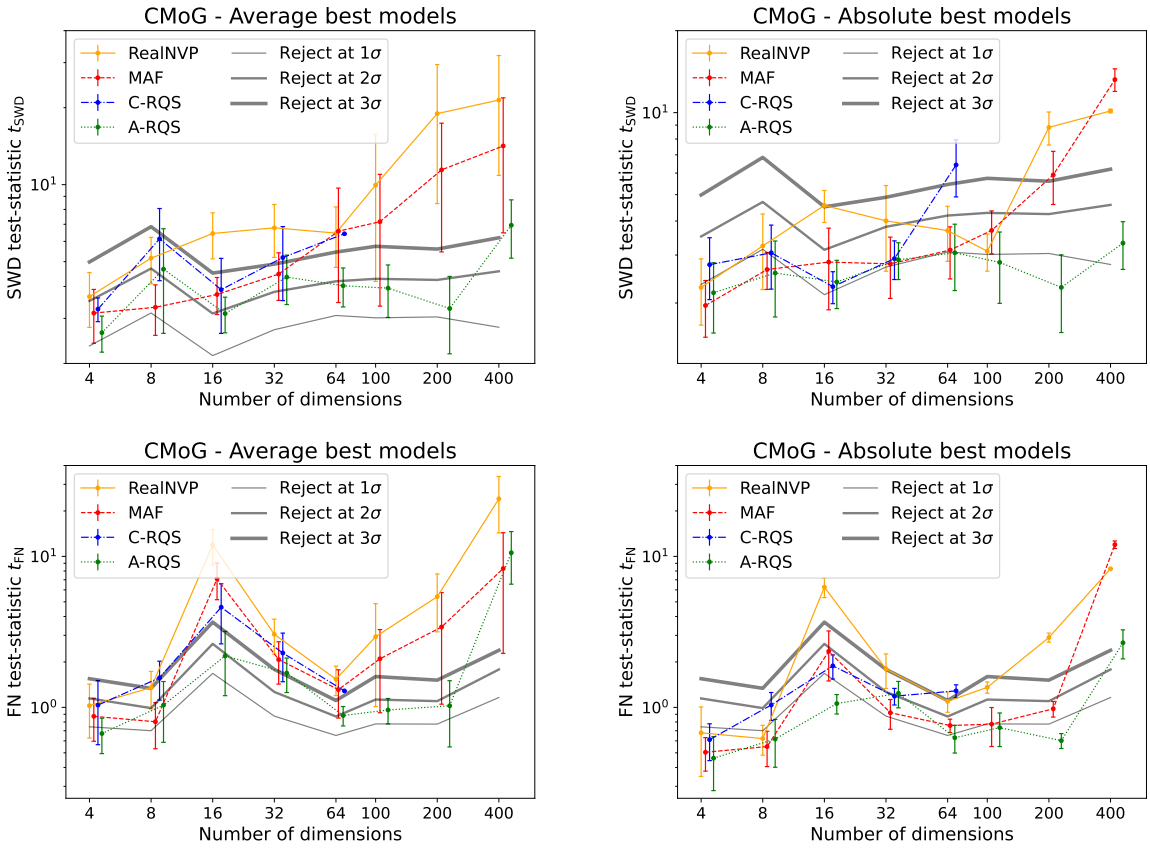

Figure 1. Performance comparison between the average (**left panel**) and absolute (**right panel**) best models obtained with RealNVP, MAF, C-RQS, and A-RQS architectures when learning the CMoG distributions. The figures show the value of the test statistic with its uncertainty, computed as explained in the text. The KS, SWD, and FN test statistics, as defined in Section 4, are shown in the upper, middle, and lower panel, respectively. The gray lines with different widths represent, from thinner to ticker, the $1, 2, 3\sigma$ thresholds for the test statistics, as obtained from the test-statistic distributions under the null hypothesis, evaluated with 10^4 pseudo-experiments.

5. Testing the Normalizing Flows

We tested the four architectures discussed above on CMoG distributions defined as a mixture of $n = 3$ components and $D = 4, 8, 16, 32, 64, 100, 200, 400$ dimensional multivariate Gaussian distributions with diagonal covariance matrices, parametrized by means randomly generated in the $[0, 10]$ interval and standard deviations randomly generated in the $[0, 1]$ interval (The values for the means and standard deviations were chosen so that the different components could generally be resolved). The components were mixed according to an n dimensional categorical distribution (with random probabilities). This meant that a different probability was assigned to each component, while different dimensions of the same component multivariate Gaussian were assigned the same probability. The resulting multivariate distributions had random order-one off-diagonal elements in the covariance matrix and multi-modal 1D marginal distributions (see, for illustration, Figures A1 and A2).

In our analysis, we considered a training set of 10^5 points, a validation set of 3×10^4 points, and a test set equal in size to the training set, with 10^5 points. It is important to note that the chosen size of the test set corresponds to the most stringent condition for evaluating the NF models. This is because the NF cannot be expected to approximate the real target distribution more accurately than the uncertainty determined by the size of the training sample.

In practical terms, the most effective NF would be indistinguishable from the target distribution when tested on a sample size equivalent to the training set. In our analysis, we

found that models tested with 10^5 samples often led to rejection at the $2\sigma/3\sigma$ level, at least with the most powerful KS test. However, this should not be viewed as a poor outcome. Rather, it suggests that one needs to utilize a test set as large as the training set to efficiently discern the NF from the true model, while smaller samples are effectively indistinguishable from those generated with the target distribution.

An alternative approach, which we did not adopt due to computational constraints, involves calculating the sample size required to reject the null hypothesis at a given confidence level. This approach offers a different but equally valid perspective, potentially useful for various applications. Nevertheless, our approach was efficient for demonstrating that NFs can perform exceptionally well on high-dimensional datasets and for comparing, among each other, the performances of different NF architectures.

For each of the four different algorithms described above and for each value of N, we performed a small scan over some of the free hyperparameters. Details on the choice of the hyperparameters are reported in Appendix B. All models were trained on Nvidia A40 GPUs.

The performances of the best NF architectures are reported in Figure 1 and Tables A2 and A3.

Figure 1 shows the values of the three test statistics (vertical panels) for the average (left panels) and absolute (right panels) best models obtained with the four different architectures. The three gray lines with different thicknesses represent the values of the test statistics corresponding to 1σ, 2σ, and 3σ rejection (p-values of 0.68, 0.95, and 0.99, respectively) of the null hypothesis that the two samples (test and NF-generated) are drawn from the same PDF. These rejection lines were obtained through 10^4 pseudo-experiments. The curve for the best C-RQS models stops at 64D since the training becomes unstable and the model does not converge. The situation could likely be improved by adding regularization and by fine-tuning the hyperparameters. However, to allow for a fair comparison with the other architectures, where regularization and fine-tuning are not necessary for a reasonable convergence, we avoided pushing C-RQS beyond 64D. Also notice that the uncertainty shown in the point at 64D for the C-RQS is artificially very small since only a small fraction of the differently seeded runs converged. This uncertainty should therefore be considered unreliable.

All plots in Figure 1 include uncertainties. As already mentioned, the best model was chosen as the one with best architecture on average, and therefore, over 10 different trainings were performed with differently seeded training samples. For the selection of the best model, the left plots show the performances averaged over the 10 trainings, with error bands representing the corresponding standard deviations, while the right plots show the performances of the absolute best instance among the 10 trained replicas, with the error band representing the standard deviation over the 10 replicas generated for testing (test and NF-generated samples). In other words, we can say that the uncertainties shown in the left plots are the standard deviations due to repeated training, while the uncertainties shown in the right plots are the standard deviations due to repeated generation/evaluation (testing).

Figure 1 clearly highlights the distinct characteristics that establish the A-RQS as the top-performing algorithm:

- Its performances are almost independent of the data dimensionality;
- The average best model is generally not rejected at 3σ level when evaluated with a number of points equal to the number of training points;
- The absolute best model is generally not rejected at 2σ level when evaluated with a number of points equal to the number of training points;
- The uncertainties due to differently seeded training and testing are generally comparable, while for all other models, the uncertainty from training is generally much larger than the one from evaluation.

All values shown in Figure 1 are reported in Tables A2 and A3 for the average and absolute best models, respectively. In the tables, we also show the total number of trainable parameters, the average number of epochs, training time, and prediction time. It is interesting to look at the training and prediction times. Indeed, while for the coupling flows, even though training time is much larger than prediction time, both times grow with a similar

rate; for the autoregressive flows, the prediction time grows faster than does the training time, which is almost constant. This is because, as we have already mentioned, the MAF is a "direct flow", very fast for density estimation and therefore for training (single pass through the flow) but slower for generation and therefore for testing (N passes through the flow, with N the dimensionality of the target distribution). Still, testing was reasonably fast, considering that each test actually consisted of 10 tests with three metrics and 10^5 points per sample. All trainings/tests took less than a few hours (sometimes, especially in small dimensionality, a few minutes), which means that all models, expect the C-RQS in large dimensionalities, are fairly fast both in training and inference (Notice that even though the training/testing times do not go beyond a few hours, we trained and tested 10 replicas of four architectures in eight different dimensionalities (apart from C-RQS) and with a few different values of the hyperparameters for a total of about 1360 runs (see Table A1). This took several months of GPU time, showing how resource demanding is to reliably estimate uncertainties of ML models, even in relatively simple cases.) Another interesting number in the tables is the total number of trainable parameters. Such number makes clear how the autoregressive architectures are more expressive than are the simple coupling ones, giving better results with relatively fewer parameters. It is also clear from the table that the improvement stepping from a simple linear affine bijector to a rational quadratic spline based on the same architecture is much larger for autoregressive architectures and less evident for the simplest coupling ones. The large number of parameters needed to obtain reasonable results from C-RQS may be the origin of its training instability at large dimensionality.

6. Conclusions and Outlook

Normalizing flows have shown many potential applications in a wide variety of research fields including HEP, both in their normalizing and generative directions. However, to ensure a standardized usage and to match the required precision, their application to high-dimensional datasets need to be properly evaluated. This paper makes a step forward in this direction by quantifying the ability of coupling and autoregressive flow architectures to model distributions of increasing complexity and dimensionality.

Our strategy consisted in performing statistically robust tests utilizing different figures of merits and including estimates of the variances induced both by the training and the evaluation procedures.

We focused on the most widely used NF architectures in HEP, the coupling (RealNVP and C-RQS) and the autoregressive flows (MAF and A-RQS), and compared them against generic multimodal distributions of increasing dimensionality.

As the main highlight, we found that the A-RQS is highly capable of precisely learning all the high-dimensional complicated distributions it was tested against, always within a few hours of training on a Tesla A40 GPU and with limited training data. Moreover, the A-RQS architecture, showed great generalization capabilities, obtaining almost constant results over a very wide range of dimensionalities, ranging from 4 to 400.

As for the other tested architectures, our results show that reasonably good results can be obtained with all of them but the C-RQS, which ended up being the least capable in generalizing to large dimensionality, with unstable and longer trainings, especially in high dimensionality.

Our analysis was performed implementing all architectures in TensorFlow2 with TensorFlow Probability using Python. The code is available in Ref. [72], while a general-purpose user-friendly framework for NFs in TensorFlow2 named NF4HEP is under development and can be found in Ref. [73]. Finally, a code for statistical inference and two-sample tests, implementing the metrics considered in this paper (and others) in TensorFlow2, is available in Ref. [74].

We stress that the intention of this study was to secure generic assessments of how NFs perform in high dimensions. For this reason, the target distributions were chosen independently of any particular experimentally driven physics dataset. An example of application to a physics dataset, in the direction of building an unsupervised DNNLikeli-

hood [75], is presented in Ref. [76]. Nonetheless, these studies represent the firsts of a series to come. Let us briefly mention, in turn, the research directions we aim to follow starting from the present paper.

- Development of reliable multivariate quality metrics, including approaches based on machine learning [77,78]. We note the importance of performing statistically meaningful tests on generative models, ideally including uncertainty estimation. A thorough study of different quality metrics against high dimensional data is on its way. Moreover, new results [79–81] suggest that classifier-based two-sample tests have the potential to match the needs of the HEP community when paired with a careful statistical analysis. These tests can leverage different ML models to provide high flexibility and sensitivity together with short training times, especially when based on kernel methods [81]. On the other hand, further studies are needed to investigate their efficiency and scalability to high dimensions.
- Study of the dependence of the NF performances on the size of the training sample [82]. In the present paper, we always kept the number of training points to 10^5. It is clear that such a number is fairly large in small dimensionality, such as $N = 4$ dimensions, and undersized for large dimensionality, such as $N \geq 100$. It is important to study the performances of the considered NF architectures in the case of scarce or very abundant data and to assess the dependence of the final precision on the number of training samples. This can also be related to developing techniques to infer the uncertainty of the NF models.
- Studies on how NFs can be used for statistical augmentation. For instance, NFs can be used for normalizing direction to build effective priors and proposals to enhance (in terms of speed and convergence time) known sampling techniques, such as Markov chain Monte Carlo, whose statistical properties are well established.
- The ability to preserve and distribute pretrained NF-based models. A final issue that needs to be addressed to ensure a widespread use of NFs in HEP is the ability to preserve and distribute pretrained NF-based models. This is, for the time being, not an easy and standard task, and support from the relevant software developers in the community is crucial to achieving this goal.

Author Contributions: Conceptualization, A.C., M.L., H.R.-G. and R.T.; Data curation, A.C., M.L., H.R.-G. and R.T.; Formal analysis, A.C., M.L., H.R.-G. and R.T.; Investigation, A.C., M.L., H.R.-G. and R.T.; Methodology, A.C., M.L., H.R.-G. and R.T.; Resources, A.C., M.L., H.R.-G. and R.T.; Software, H.R.-G. and R.T.; Validation, A.C., M.L., H.R.-G. and R.T.; Writing—original draft, A.C., M.L, H.R.-G. and R.T.; Writing—review & editing, A.C., M.L., H.R.-G. and R.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the Italian PRIN grant 20172LNEEZ. M.L. acknowledges the financial support of the European Research Council (grant SLING 819789). H.R.-G. is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762—TRR 257: Particle Physics Phenomenology after the Higgs Discovery.

Data Availability Statement: The NF models produced by the authors and used to generate data for this study are made openly available by the authors on GitHub https://github.com/NF4HEP/NormalizingFlowsHD.

Acknowledgments: We thank Luca Silvestrini for pushing us toward the study of NFs and for fruitful discussions. We are thankful to the developers of TensorFlow2, which made it possible to perform this analysis. We thank the IT team of INFN Genova and in particular Mirko Corosu for useful support. H.R.G. acknowledges the hospitality of Sabine Kraml at LPSC Grenoble and the discussions on normalizing flows held there with the rest of the SModelS Collaboration.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Implementation of NF Architectures

Appendix A.1. The RealNVP

We are given a collection of vectors $\{y_i^I\}$ with $i = 1, \ldots, D$ representing the dimensionality and $I = 1, \ldots, N$ the number of samples representing the unknown PDF p_Y. For all samples $\mathbf{y}^I$, we consider the half partitioning given by the two sets $\hat{\mathbf{y}}^I = \{y_1^I, \ldots, y_{D/2}^I\}$ and $\tilde{\mathbf{y}}_I = \{y_{D/2+1}^I, \ldots, y_D^I\}$ (For simplicity, we assume D is even and therefore $D/2$ integer. In case $D = 2d + 1$ is odd the "half-partitioning" could be equally done by taking the first $d + 1$ and the last d dimensions, or vice versa. This does not affect our implementation). We then use the $\hat{\mathbf{y}}^I$ samples as inputs to train a fully connected MLP (a dense DNN) giving as output the vectors of t_i and s_i, with $i = 1, \ldots, D/2$ in Equation (10). These output vectors are provided by the DNN through two output layers, which are dense layers with linear and tanh activation functions for t_i and s_i, respectively, and are used to implement the transformation in Equation (12), which outputs the (inversely) transformed samples. Moreover, in order to transform all dimensions and to increase the expressivity of the model, we use a series of such RealNVP bijectors, feeding the output of each bijector as input for the next one and inverting the role of the two partitions at each step. After the full transformation is performed, one obtains the final $\{x_i^I\}$ with $i = 1, \ldots, D$ vectors and the transformation Jacobian (the product of the inverse of Equation (11) for each bijector). With these ingredients, and assuming a normal base distribution p_X, one can compute the negative of the log-likelihood in Equation (2), which is used as loss function for the DNN optimization.

As is clear from the implementation, the RealNVP NF, i.e., the series of RealNVP bijectors, is trained in the normalizing direction, taking data samples as inputs. Nevertheless, since the s_i and t_i vectors only depend, at each step, on untransformed dimensions, once the DNN is trained, they can be used both to compute the density by using Equation (12) and to generate new samples, with equal efficiency, by using Equation (10). This shows that the RealNVP is equally efficient in both the normalizing and generative directions.

Appendix A.2. The MAF

As in the case of the RealNVP, for the MAF, the forward direction represents the normalizing direction. In this case, the vectors s_i and t_i of dimension $D - 1$ describing the affine bijector in Equation (13) are parametrized by an autoregressive DNN with D inputs and $2(D - 1)$ outputs, implemented through the MADE [60] masking procedure according to the TensorFlow Probability implementation (see Ref. [83]). The procedure is based on binary mask matrices that define which connections (weights) are kept and which are dropped to ensure the autoregressive property. (The binary mask matrices are simple transition matrices between pairs of layers of dimension (K', K), with K' being the number of nodes in the forward layer (closer to the output) and K being the number of nodes in the backward layer (closer to the input). Obviously $K = D$ is for the input layer, and $K' = 2(D - 1)$ is for the output layer.) Mask matrices are determined from numbers (degrees) assigned to all nodes in the DNN: each node in the input layer is numbered sequentially from 1 to D; each node in each hidden layer is assigned a number between 1 and D, possibly with repetition; the first half output nodes (representing s_i) are numbered sequentially from 1 to $D - 1$, and the same is done for the second half (representing t_i). Once all degrees are assigned, the matrix elements of the mask matrices are 1 if two nodes are connected and 0 if they are "masked", i.e., not connected. The mask matrices are determined by connecting the nodes of each layer with index k with all nodes in the preceding layer that have an index smaller or equal than k. As for the RealNVP, a series of MAF bijectors is used, by feeding each with the $\{x_i^I\}$, with $i = 1, \ldots, D$, according to Equation (15) computed from the previous one. The last bijector computes the final $\{x_i^I\}$, with $i = 1, \ldots, D$, according to Equation (15) and the transformation Jacobian (the product of the inverse of Equation (14) for each bijector), used to compute and optimize the log-likelihood as defined in Equation (2).

The efficiency of the MAF in the normalizing and generative directions is not the same as in the case of the RealNVP. Indeed, computing the log-likelihood for density estimation requires a single forward pass of $\{y_i\}$ through the NF. However, generating samples requires a start from $\{x_i\}$, randomly generated from the base distribution. Then, one needs the following procedure to compute the corresponding $\{y_i\}$:

- Define the first component of the required y_i^{input} as $y_1^{\text{output}} = y_1^{\text{input}} = x_1$, where y_i^{input} is the NF input;
- Start with a $y_i^{\text{input}} = x_i$ and pass it through the NF to determine y_2^{output} as a function of y_1^{output};
- Update y_i^{input} with $y_2^{\text{input}} = y_2^{\text{output}}$ and pass through the NF to determine y_3^{output} as a function of y_1^{output} and y_2^{output};
- Iterate until all the y_i^{output} components are computed.

It is clear to see that the procedure requires D to pass through the NF to generate a sample, and so the generation in the MAF is D times less efficient than is the density estimation. The inverse autoregressive flow (IAF) [59] is an implementation similar to the MAF that implements generation in the forward direction (obtained by exchanging x and y in Equations (13) and (15). In the case of IAF, computing the log-likelihood (which is needed for training) requires D steps, while generation only requires a single pass through the flow. The IAF is therefore much slower in training and much faster in generating new samples.

Table A1. Hyperparameter values used in our analysis. The last row shows the total number of runs for each architecture, with the 10 replicas and the different dimensionalities being taken into account.

Hyperparameter Values				
Hyperpar.	MAF	RealNVP	A-RQS	C-RQS
Number of bijectors	5, 10	5, 10	2	5, 10
Number of hidden layers	3×128 3×256	3×128 3×256	3×128 3×256	3×128 3×256
Number of spline knots	–	–	8, 12	8, 12
Total number of runs	320	320	320	400

Appendix A.3. The C-RQS

The C-RQS parameters are determined by the following procedure [51].

1. A dense DNN takes $x_1, \ldots, x_d$ as inputs and outputs an unconstrained parameter vector θ_i of length $3K - 1$ for each $i = d + 1, \ldots, D$ dimension.
2. The vector θ_i is partitioned as $\theta_i = [\theta_i^w, \theta_i^h, \theta_i^d]$, where θ_i^w and θ_i^h have length K, while θ_i^d has length $K - 1$.
3. The vectors θ_i^w and θ_i^h are each passed through a softmax and multiplied by $2B$; the outputs are interpreted as the widths and heights of the K bins, which must be positive and span the $\mathbb{B}$ interval. Then, the cumulative sums of the K bin widths and heights, each starting at $-B$, yield the $K + 1$ knot parameters $\{(x_i^{(k)}, y_i^{(k)})\}_{k=0}^{K}$.
4. The vector θ_i^d is passed through a softplus function and is interpreted as the values of the derivatives $\{d_i^{(k)}\}_{k=1}^{K-1}$ at the internal knots.

As for the RealNVP, in order to transform all dimensions, a series of RQS bijectors is applied, inverting the role of the two partitions at each step.

Appendix A.4. The A-RQS

In the autoregressive implementation, we follow the same procedure used in the MAF implementation and described in Appendix A.2, but instead of obtaining the $2(D-1)$ outputs determining the affine parameters, we obtain the $3K-1$ parameters needed to compute the values of the knots parameters and derivatives. Once these are determined, the procedure follows steps 2 to 4 of the C-RQS implementation described in the previous subsection.

Appendix B. Hyperparameters

For all models, we used a total of 10^5 trainings, 3×10^4 validations, and 10^5 test points. We employed ReLu activation function with no regularization. All models were trained for up to 1000 epochs, with the ADAM optimizer, with the initial learning rate set to 10^{-3}. (For unstable trainings in large dimensionality, when the training with this initial learning rate failed with a "nan" loss, we reduced the learning rate by a factor $1/3$ and retried until either the training succeeded or the learning rate was smaller than 10^{-6}.) The learning rate was then reduced by a factor of 0.5 after 50 epochs without improvement better than 10^{-4} on the validation loss. Early stopping was used to terminate the learning after 100 epochs without the same amount of improvement. The batch size was set to 256 for RealNVP and to 512 for the other algorithms. For the two neural spline algorithms, we also set the range of the spline equal to $[-16, 16]$. The values of all hyperparameters on which we performed a scan are reported in Table A1.

Appendix C. Results Summary Tables

Table A2. Values of the most relevant hyperparameters and metrics for the average best models obtained for the CMoG distributions. The number of training epochs and the training and prediction times are averages. The columns KS, Sliced WD, and Frobenius Norm contain the values of the corresponding test statistics shown in the left panels of Figure 1. The row corresponding to the best model for each dimension, that is, the one with the minimum KS test statistic, is shown in bold.

Results for Average Best Models										
Hidden Layers	**# of Bijec.**	**Algorithm**	**Spline Knots**	**Trainable Parameters**	**KS**	**Sliced WD**	**Frobenius Norm**	**# of Epochs**	**Training Time (s)**	**Prediction Time (s)**
4D										
3×128	10	MAF	–	346,960	1.5 ± 0.4	3.1 ± 0.7	0.9 ± 0.3	442	5301	17
3×256	5	RealNVP	–	666,900	1.6 ± 0.5	3.7 ± 0.9	1.0 ± 0.4	616	8777	8
3×128	**2**	**A-RQS**	**8**	**91,064**	$\mathbf{1.2 \pm 0.1}$	$\mathbf{2.6 \pm 0.4}$	$\mathbf{0.7 \pm 0.2}$	**670**	**7606**	**54**
3×128	5	C-RQS	8	196,710	1.4 ± 0.2	3.3 ± 0.3	1.0 ± 0.5	483	12,346	26
8D										
3×128	**5**	**MAF**	**–**	**181,200**	$\mathbf{1.3 \pm 0.2}$	$\mathbf{3.3 \pm 0.7}$	$\mathbf{0.8 \pm 0.3}$	**713**	**5757**	**11**
3×128	10	RealNVP	–	346,960	2.2 ± 0.5	5.2 ± 1.1	1.4 ± 0.4	340	8242	12
3×128	2	A-RQS	12	140,592	1.6 ± 0.5	4.7 ± 2.1	1.0 ± 0.4	477	5516	315
3×128	5	C-RQS	8	227,660	2.4 ± 0.6	6.1 ± 1.9	1.6 ± 0.5	294	11,500	27
16D										
3×128	**5**	**MAF**	**–**	**196,640**	$\mathbf{1.6 \pm 0.2}$	$\mathbf{3.7 \pm 0.6}$	$\mathbf{7.1 \pm 1.9}$	**479**	**3410**	**16**
3×128	5	RealNVP	–	181,200	2.6 ± 0.3	6.4 ± 1.3	12 ± 3	366	4263	8
3×128	2	A-RQS	12	214,880	1.8 ± 0.3	3.1 ± 0.5	2.2 ± 1.0	327	3705	65
3×128	10	C-RQS	8	579,120	2.0 ± 0.6	3.9 ± 1.3	4.6 ± 2.0	558	64,056	53
32D										
3×128	**5**	**MAF**	**–**	**227,520**	$\mathbf{1.7 \pm 0.3}$	$\mathbf{4.5 \pm 0.9}$	$\mathbf{2.1 \pm 0.6}$	**595**	**4193**	**24**
3×128	10	RealNVP	–	393,280	2.6 ± 0.5	6.8 ± 1.6	3.0 ± 0.8	676	16,946	12
3×128	2	A-RQS	8	264,384	1.9 ± 0.3	4.4 ± 1.0	1.7 ± 0.4	375	4705	97
3×256	10	C-RQS	12	2,798,560	2.0 ± 0.5	5.2 ± 1.7	2.3 ± 0.8	750	75,606	83

Table A2. *Cont.*

Hidden Layers	# of Bijec.	Algorithm	Spline Knots	Trainable Parameters	KS	Sliced WD	Frobenius Norm	# of Epochs	Training Time (s)	Prediction Time (s)
Results for Average Best Models										
64D										
3 × 128	10	MAF	–	578,560	2.1 ± 0.6	7 ± 3	1.3 ± 0.5	537	5289	89
3 × 256	10	RealNVP	–	1,564,800	2.3 ± 0.4	6.5 ± 1.7	1.5 ± 0.3	711	13,871	16
3 × 128	**2**	**A-RQS**	**8**	**462,464**	**1.5 ± 0.2**	**4.0 ± 0.7**	**0.9 ± 0.1**	**523**	**6197**	**332**
3 × 256	5	C-RQS	12	2,139,360	2.4 ± 0.9	6.4 ± 2.7	2.6 ± 0.8	813	36,608	53
100D										
3 × 128	10	MAF	–	717,520	2.2 ± 0.8	7 ± 4	2.1 ± 1.2	778	7824	144
3 × 256	10	RealNVP	–	1,703,400	3.2 ± 1.3	10 ± 6	2.9 ± 1.9	991	23,500	19
3 × 128	**2**	**A-RQS**	**12**	**994,904**	**1.4 ± 0.2**	**4.0 ± 0.9**	**1.0 ± 0.2**	**588**	**6219**	**1027**
200D										
3 × 128	10	MAF	–	1,103,520	3.7 ± 1.4	12 ± 6.0	3.4 ± 2.4	612	6149	393
3 × 256	10	RealNVP	–	2,088,400	6.3 ± 2.3	19 ± 11	5.4 ± 2.2	1000	22,704	25
3 × 128	**2**	**A-RQS**	**12**	**1,923,504**	**1.3 ± 0.2**	**3.3 ± 1.1**	**1.0 ± 0.5**	**703**	**9943**	**4900**
400D										
3 × 128	10	MAF	–	1,875,520	4.3 ± 1.8	14 ± 8	8 ± 6	600	4612	1242
3 × 256	0	RealNVP	–	2,858,400	8.4 ± 2.4	21 ± 11	24 ± 10	824	23,705	38
3 × 128	**2**	**A-RQS**	**8**	**2,542,304**	**2.2 ± 0.9**	**6.9 ± 1.8**	**11 ± 4**	**796**	**9970**	**9738**

Table A3. Values of the most relevant hyperparameters and metrics for the absolute best models obtained for the CMoG distributions. The number of training epochs and the training and prediction times are averages. The columns KS, Sliced WD, and Frobenius Norm contain the values of the corresponding -statistics shown in the right panels of Figure 1. The row corresponding to the best model for each dimension, that is, the one with the minimum KS test statistic, is shown in bold.

Hidden Layers	# of Bijec.	Algorithm	Spline Knots	Trainable Parameters	KS	Sliced WD	Frobenius Norm	# of Epochs	Training Time (s)	Prediction Time (s)
Results for Absolute Best Models										
4D										
3 × 128	**10**	**MAF**	**–**	**346,960**	**1.1 ± 0.1**	**2.0 ± 0.5**	**0.5 ± 0.1**	**442**	**5301**	**17**
3 × 256	5	RealNVP	–	666,900	1.1 ± 0.2	2.3 ± 0.6	0.7 ± 0.3	616	8777	8
3 × 128	2	A-RQS	8	91,064	1.1 ± 0.2	2.2 ± 0.6	0.5 ± 0.2	670	7606	54
3 × 128	5	C-RQS	8	196,710	1.2 ± 0.2	2.8 ± 0.7	0.6 ± 0.2	483	12,346	26
8D										
3 × 128	5	MAF	–	181,200	1.1 ± 0.2	2.7 ± 0.4	0.6 ± 0.1	713	5757	11
3 × 128	10	RealNVP	–	346,960	1.3 ± 0.2	3.2 ± 1.0	0.6 ± 0.1	340	8242	12
3 × 128	**2**	**A-RQS**	**12**	**140,592**	**1.1 ± 0.1**	**2.6 ± 0.8**	**0.6 ± 0.2**	**477**	**5516**	**315**
3 × 128	5	C-RQS	8	227,660	1.4 ± 0.1	3.1 ± 0.8	1.0 ± 0.2	294	11,500	27
16D										
3 × 128	5	MAF	–	196,640	1.3 ± 0.1	2.8 ± 0.9	2.4 ± 0.9	479	3410	16
3 × 128	5	RealNVP	–	181,200	2.2 ± 0.1	4.6 ± 0.6	6.3 ± 0.9	366	4263	8
3 × 128	2	A-RQS	12	214,880	1.4 ± 0.1	2.4 ± 0.5	1.1 ± 0.2	327	3705	65
3 × 128	**10**	**C-RQS**	**8**	**579,120**	**1.1 ± 0.1**	**2.3 ± 0.3**	**1.9 ± 0.3**	**558**	**64,056**	**53**
32D										
3 × 128	**5**	**MAF**	**–**	**227,520**	**1.2 ± 0.1**	**2.8 ± 0.7**	**0.9 ± 0.2**	**595**	**4193**	**24**
3 × 128	10	RealNVP	–	393,280	1.7 ± 0.2	4.0 ± 1.4	1.8 ± 0.5	676	16,946	12
3 × 128	2	A-RQS	8	264,384	1.3 ± 0.1	2.9 ± 0.4	1.2 ± 0.2	375	4705	97
3 × 256	10	C-RQS	12	2,798,560	1.4 ± 0.2	2.9 ± 0.5	1.2 ± 0.1	750	75,606	83
64D										
3 × 128	10	MAF	–	578,560	1.5 ± 0.1	3.1 ± 0.7	0.8 ± 0.1	537	5289	89
3 × 256	10	RealNVP	–	1,564,800	1.8 ± 0.1	3.7 ± 0.8	1.1 ± 0.2	711	13,871	16
3 × 128	**2**	**A-RQS**	**8**	**462,464**	**1.1 ± 0.2**	**3.1 ± 0.8**	**0.6 ± 0.1**	**523**	**6197**	**332**
3 × 256	5	C-RQS	12	2,139,360	1.5 ± 0.1	3.7 ± 0.5	1.8 ± 0.1	813	36,608	53

Table A3. *Cont.*

Results for Absolute Best Models										
Hidden Layers	**# of Bijec.**	**Algorithm**	**Spline Knots**	**Trainable Parameters**	**KS**	**Sliced WD**	**Frobenius Norm**	**# of Epochs**	**Training Time (s)**	**Prediction Time (s)**
100D										
3×128	10	MAF	–	717,520	1.3 ± 0.1	3.7 ± 0.7	0.8 ± 0.2	778	7824	144
3×256	10	RealNVP	–	1,703,400	1.9 ± 0.1	3.1 ± 0.5	1.4 ± 0.1	991	23,500	19
3×128	**2**	**A-RQS**	**12**	**994,904**	$\mathbf{1.1 \pm 0.2}$	$\mathbf{2.8 \pm 0.8}$	$\mathbf{0.8 \pm 0.3}$	**588**	**6219**	**1027**
200D										
3×128	10	MAF	–	1,103,520	2.2 ± 0.3	5.9 ± 1.3	1.0 ± 0.1	612	6149	393
3×256	10	RealNVP	–	2,088,400	4.0 ± 0.2	8.8 ± 1.2	2.9 ± 0.2	1000	22,704	25
3×128	**2**	**A-RQS**	**12**	**1,923,504**	$\mathbf{1.0 \pm 0.1}$	$\mathbf{2.3 \pm 0.7}$	$\mathbf{0.6 \pm 0.1}$	**703**	**9943**	**4900**
400D										
3×128	10	MAF	–	1,875,520	1.7 ± 0.2	13.2 ± 1.3	12.0 ± 0.7	600	4612	1242
3×256	10	RealNVP	–	2,858,400	5.3 ± 0.1	10.1 ± 0.2	8.3 ± 0.1	824	23,705	38
3×128	**2**	**A-RQS**	**8**	**2,542,304**	$\mathbf{1.2 \pm 0.1}$	$\mathbf{3.3 \pm 0.7}$	$\mathbf{2.7 \pm 0.6}$	**796**	**9970**	**9738**

Appendix D. Correlation Matrix

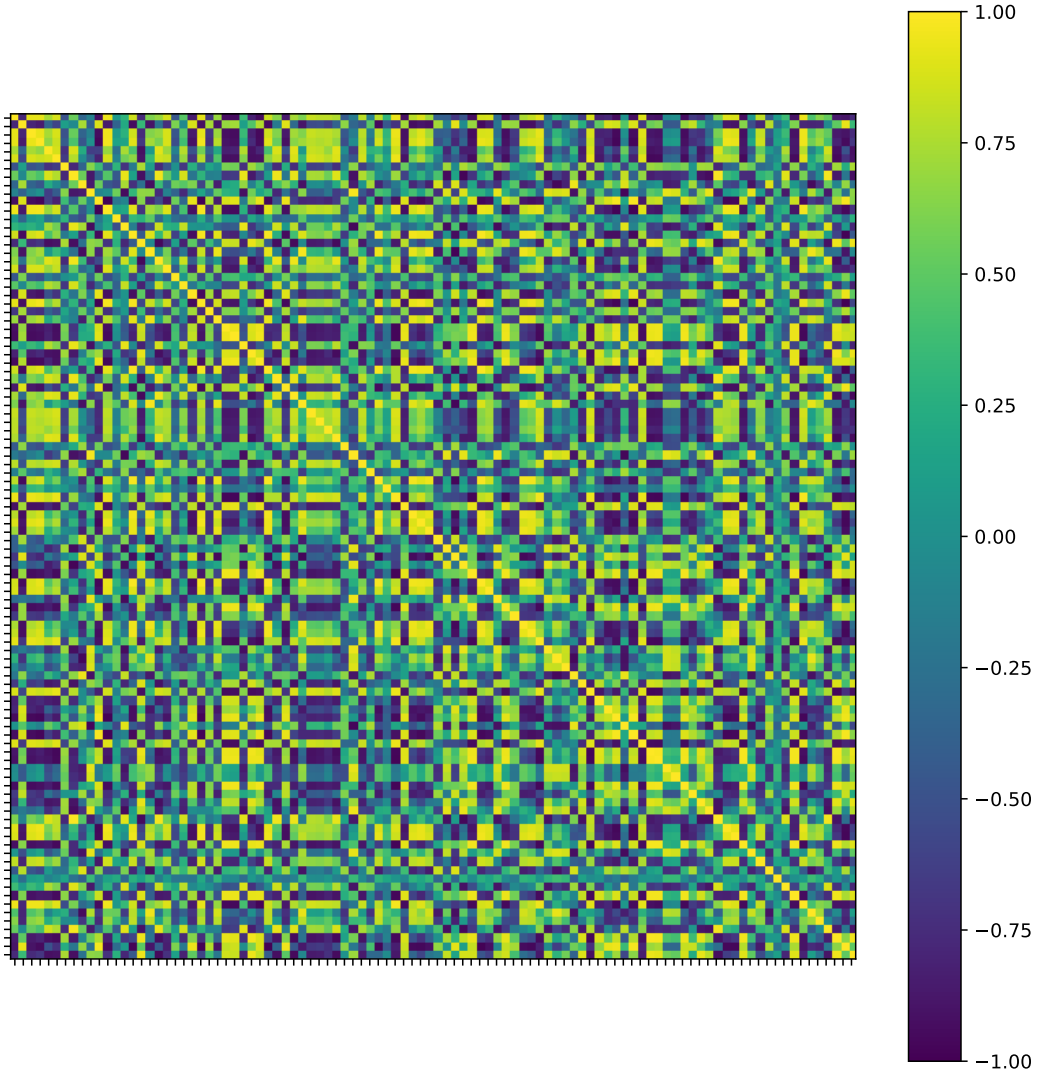

Figure A1. Visual representation of the correlation matrix of our CMoG model in $N = 100$ dimensions. Despite the different multivariate Gaussian components being uncorrelated, the resulting mixture model features random, order-one, off-diagonal elements in the full correlation matrix.

Appendix E. Corner Plots

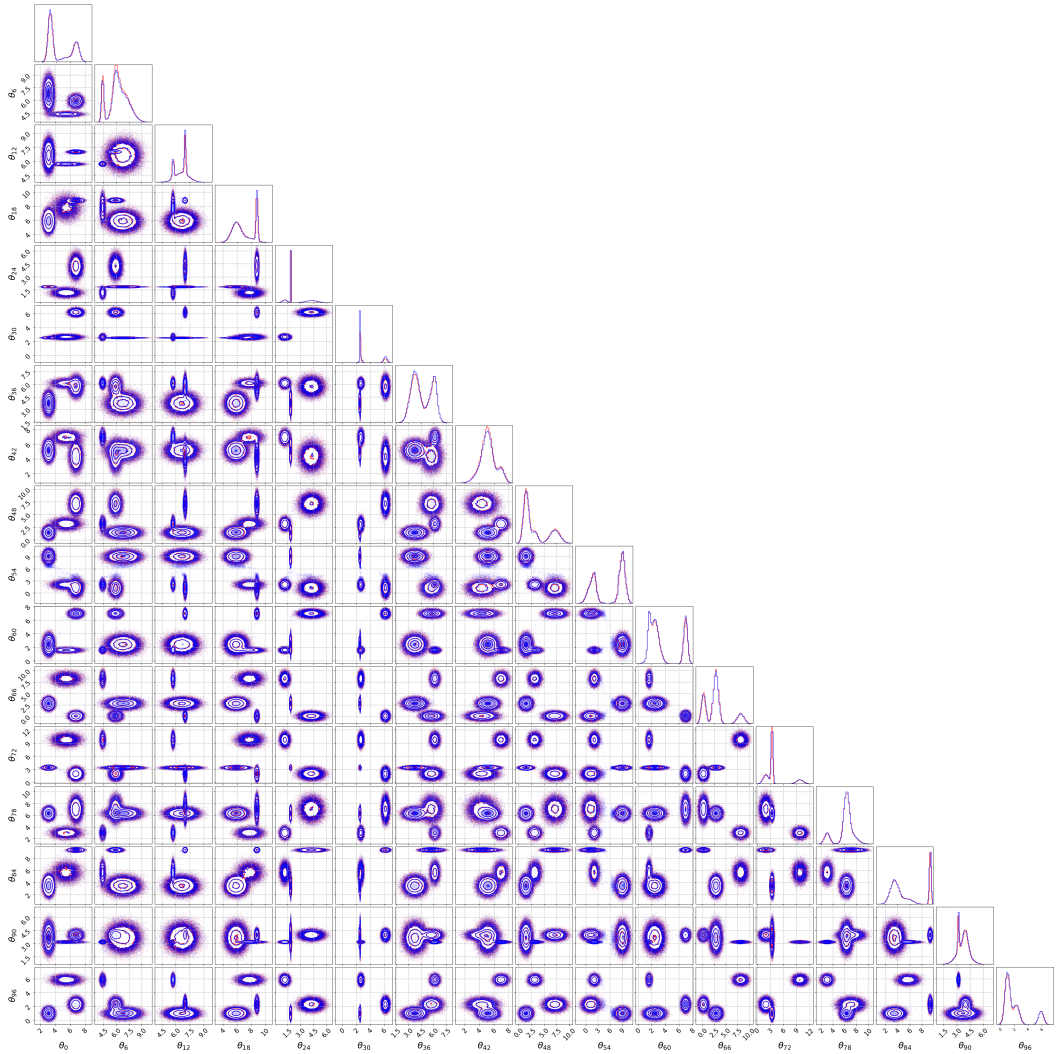

Figure A2. Visual representation of the 1D and 2D marginal distributions for 17 randomly chosen dimensions of the $N = 100$ dimensional CMoG distribution obtained with 10^5 points. The red and blue curves and points represent the test samples and the NF-generated samples obtained with the A-RQS best model, respectively. Given the high dimensionality, the non-trivial structure of the distribution, the limited number of training samples, and the low level of tuning of the hyperparameters, the result can be considered very accurate.

References

1. Tabak, E.; Vanden-Eijnden, E. Density estimation by dual ascent of the log-likelihood. *Commun. Math. Sci.* **2010**, *8*, 217–233. [CrossRef]
2. Tabak, E.; Turner, C. A family of nonparametric density estimation algorithms. *Commun. Pure Appl. Math.* **2013**, *66*, 145–164. [CrossRef]
3. Rezende, D.J.; Mohamed, S. Variational Inference with Normalizing Flows. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015. [CrossRef]
4. Dinh, L.; Krueger, D.; Bengio, Y. NICE: Non-linear Independent Components Estimation. *arXiv* **2015**, arXiv:1410.8516.
5. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*; Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., Weinberger, K., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2014; Volume 27.
6. Kingma, D.P.; Welling, M. Auto-Encoding Variational Bayes. In Proceedings of the 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, 14–16 April 2014.

7. Rezende, D.J.; Mohamed, S.; Wierstra, D. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 21–26 June 2014; Xing, E.P., Jebara, T., Eds.; Association for Computing Machinery: New York, NY, USA, 2014; Volume 32, pp. 1278–1286.
8. Fan, Y.; Nott, D.J.; Sisson, S.A. Approximate Bayesian computation via regression density estimation. *Stat* **2013**, *2*, 34–48. [CrossRef]
9. Papamakarios, G.; Murray, I. Fast ϵ-free Inference of Simulation Models with Bayesian Conditional Density Estimation. In *Advances in Neural Information Processing Systems*; Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2016; Volume 29.
10. Brehmer, J.; Louppe, G.; Pavez, J.; Cranmer, K. Mining gold from implicit models to improve likelihood-free inference. *Proc. Nat. Acad. Sci. USA* **2020**, *117*, 5242–5249.[CrossRef] [PubMed]
11. Green, S.R.; Simpson, C.; Gair, J. Gravitational-wave parameter estimation with autoregressive neural network flows. *Phys. Rev. D* **2020**, *102*, 104057. [CrossRef]
12. Villar, V.A. Amortized Bayesian Inference for Supernovae in the Era of the Vera Rubin Observatory Using Normalizing Flows. In Proceedings of the 36th Conference on Neural Information Processing Systems, New Orleans, LA, USA, 28 November–9 December 2022.
13. Campagne, J.E.; Lanusse, F.; Zuntz, J.; Boucaud, A.; Casas, S.; Karamanis, M.; Kirkby, D.; Lanzieri, D.; Li, Y.; Peel, A. JAX-COSMO: An End-to-End Differentiable and GPU Accelerated Cosmology Library. *arXiv* **2023**, arXiv:2302.05163.
14. Bellagente, M. Go with the Flow: Normalising Flows Applications for High Energy Physics. Ph.D. Thesis, Heidelberg University, Heidelberg, Germany, 2022.
15. Zoran, D.; Weiss, Y. From learning models of natural image patches to whole image restoration. In Proceedings of the 13rd International Conference on Computer Vision, Barcelona, Spain, 6–13 November 2011; pp. 479–486.
16. Green, S.R.; Gair, J. Complete parameter inference for GW150914 using deep learning. *Mach. Learn. Sci. Technol.* **2021**, *2*, 03LT01. [CrossRef]
17. Glüsenkamp, T. Unifying supervised learning and VAEs—Automating statistical inference in (astro-)particle physics with amortized conditional normalizing flows. *arXiv* **2020**, arXiv:2008.05825.
18. Kodi Ramanah, D.; Wojtak, R.; Ansari, Z.; Gall, C.; Hjorth, J. Dynamical mass inference of galaxy clusters with neural flows. *Mon. Not. Roy. Astron. Soc.* **2020**, *499*, 1985–1997. [CrossRef]
19. Cheung, D.H.T.; Wong, K.W.K.; Hannuksela, O.A.; Li, T.G.F.; Ho, S. Testing the robustness of simulation-based gravitational-wave population inference. *Phys. Rev. D* **2022**, *106*, 083014. [CrossRef]
20. Ruhe, D.; Wong, K.; Cranmer, M.; Forré, P. Normalizing Flows for Hierarchical Bayesian Analysis: A Gravitational Wave Population Study. *arXiv* **2022**, arXiv:2211.09008.
21. Gu, S.S.; Ghahramani, Z.; Turner, R.E. Neural Adaptive Sequential Monte Carlo. In *Advances in Neural Information Processing Systems*; Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2015; Volume 28.
22. Paige, B.; Wood, F. Inference Networks for Sequential Monte Carlo in Graphical Models. In Proceedings of the 33rd International Conference on International Conference on Machine Learning, New York, NY, USA, 20–22 June 2016; Volume 48, pp. 3040–3049.
23. Foreman, S.; Izubuchi, T.; Jin, L.; Jin, X.Y.; Osborn, J.C.; Tomiya, A. HMC with Normalizing Flows. In Proceedings of the The 38th International Symposium on Lattice Field Theory (LATTICE2021), Virtual, 26–30 July 2021. [CrossRef]
24. Hackett, D.C.; Hsieh, C.C.; Albergo, M.S.; Boyda, D.; Chen, J.W.; Chen, K.F.; Cranmer, K.; Kanwar, G.; Shanahan, P.E. Flow-based sampling for multimodal distributions in lattice field theory. *arXiv* **2021**, arXiv:2107.00734.
25. Singha, A.; Chakrabarti, D.; Arora, V. Conditional normalizing flow for Markov chain Monte Carlo sampling in the critical region of lattice field theory. *Phys. Rev. D* **2023**, *107*, 014512. [CrossRef]
26. Caselle, M.; Cellini, E.; Nada, A.; Panero, M. Stochastic normalizing flows for lattice field theory. In Proceedings of the 39th International Symposium on Lattice Field Theory (LATTICE2022), Bonn, Germany, 8–13 August 2022. [CrossRef]
27. Matthews, A.G.D.G.; Arbel, M.; Rezende, D.J.; Doucet, A. Continual Repeated Annealed Flow Transport Monte Carlo. In Proceedings of the International Conference on Machine Learning, Baltimore, MD, USA, 17–23 July 2022.
28. Cranmer, K.; Kanwar, G.; Racanière, S.; Rezende, D.J.; Shanahan, P. Advances in machine-learning-based sampling motivated by lattice quantum chromodynamics. *Nat. Rev. Phys.* **2023**, *5*, 526–535. [CrossRef]
29. Papamakarios, G.; Murray, I. Distilling Intractable Generative Models. In Proceedings of the Probabilistic Integration Workshop at the Neural Information Processing Systems Conference, Montreal, BC, Canada, 7–12 December 2015.
30. Gabrié, M.; Rotskoff, G.M.; Vanden-Eijnden, E. Adaptive Monte Carlo augmented with normalizing flows. *Proc. Nat. Acad. Sci. USA* **2022**, *119*, e2109420119. [CrossRef] [PubMed]
31. Pina-Otey, S.; Gaitan, V.; Sánchez, F.; Lux, T. Exhaustive neural importance sampling applied to Monte Carlo event generation. *Phys. Rev. D* **2020**, *102*, 013003. [CrossRef]
32. Gao, C.; Höche, S.; Isaacson, J.; Krause, C.; Schulz, H. Event Generation with Normalizing Flows. *Phys. Rev. D* **2020**, *101*, 076002. [CrossRef]
33. Le, T.A.; Baydin, A.G.; Wood, F. Inference Compilation and Universal Probabilistic Programming. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, USA, 20–22 April 2017; Singh, A., Zhu, J., Eds.; Proceedings of Machine Learning Research: New York, NY, USA, 2017.

34. Papamakarios, G.; Nalisnick, E.; Rezende, D.J.; Mohamed, S.; Lakshminarayanan, B. Normalizing Flows for Probabilistic Modeling and Inference. *J. Mach. Learn. Res.* **2022**, *22*, 1–64. [CrossRef]
35. Butter, A.; Heimel, T.; Hummerich, S.; Krebs, T.; Plehn, T.; Rousselot, A.; Vent, S. Generative Networks for Precision Enthusiasts. *arXiv* **2021**, arXiv:2110.13632.
36. Verheyen, R. Event Generation and Density Estimation with Surjective Normalizing Flows. *SciPost Phys.* **2022**, *13*, 047. [CrossRef]
37. Krause, C.; Shih, D. CaloFlow: Fast and Accurate Generation of Calorimeter Showers with Normalizing Flows. *arXiv* **2021**, arXiv:2106.05285.
38. Krause, C.; Shih, D. CaloFlow II: Even Faster and Still Accurate Generation of Calorimeter Showers with Normalizing Flows. *arXiv* **2021**, arXiv:2110.11377.
39. Gao, C.; Isaacson, J.; Krause, C. i-flow: High-dimensional Integration and Sampling with Normalizing Flows. *Mach. Learn. Sci. Technol.* **2020**, *1*, 045023. [CrossRef]
40. Heimel, T.; Winterhalder, R.; Butter, A.; Isaacson, J.; Krause, C.; Maltoni, F.; Mattelaer, O.; Plehn, T. MadNIS—Neural multi-channel importance sampling. *SciPost Phys.* **2023**, *15*, 141. [CrossRef]
41. Heimel, T.; Huetsch, N.; Maltoni, F.; Mattelaer, O.; Plehn, T.; Winterhalder, R. The MadNIS Reloaded. *arXiv* **2023**, arXiv:2311.01548.
42. Ernst, F.; Favaro, L.; Krause, C.; Plehn, T.; Shih, D. Normalizing Flows for High-Dimensional Detector Simulations. *arXiv* **2023**, arXiv:2312.09290.
43. Nachman, B.; Shih, D. Anomaly Detection with Density Estimation. *Phys. Rev. D* **2020**, *101*, 075042. [CrossRef]
44. Golling, T.; Klein, S.; Mastandrea, R.; Nachman, B. FETA: Flow-Enhanced Transportation for Anomaly Detection. *arXiv* **2022**, arXiv:2212.11285.
45. Caron, S.; Hendriks, L.; Verheyen, R. Rare and Different: Anomaly Scores from a combination of likelihood and out-of-distribution models to detect new physics at the LHC. *SciPost Phys.* **2022**, *12*, 077. [CrossRef]
46. Bellagente, M.; Butter, A.; Kasieczka, G.; Plehn, T.; Rousselot, A.; Winterhalder, R.; Ardizzone, L.; Köthe, U. Invertible Networks or Partons to Detector and Back Again. *SciPost Phys.* **2020**, *9*, 074. [CrossRef]
47. Backes, M.; Butter, A.; Dunford, M.; Malaescu, B. An unfolding method based on conditional Invertible Neural Networks (cINN) using iterative training. *arXiv* **2022**, arXiv:2212.08674.
48. Reyes-Gonzalez, H.; Torre, R. Testing the boundaries: Normalizing Flows for higher dimensional data sets. *J. Phys. Conf. Ser.* **2023**, *2438*, 012155. [CrossRef]
49. Dinh, L.; Sohl-Dickstein, J.; Bengio, S. Density estimation using Real NVP. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017. [CrossRef]
50. Papamakarios, G.; Pavlakou, T.; Murray, I. Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
51. Durkan, C.; Bekasov, A.; Murray, I.; Papamakarios, G. Neural Spline Flows. In Proceedings of the 33rd International Conference on Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; Curran Associates, Inc.: Red Hook, NY, USA, 2019.
52. Noé, F.; Köhler, J.; Wu, H. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science* **2018**, *365*, eaaw1147. [CrossRef]
53. Midgley, L.I.; Stimper, V.; Simm, G.N.C.; Scholkopf, B.; Hernández-Lobato, J.M. Flow Annealed Importance Sampling Bootstrap. *arXiv* **2022**, arXiv:2208.01893.
54. Grover, A.; Dhar, M.; Ermon, S. Flow-GAN: Combining Maximum Likelihood and Adversarial Learning in Generative Models. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017.
55. Villani, C. *Topics in Optimal Transportation (Graduate Studies in Mathematics 58)*; American Mathematical Society: Providence, RI, USA, 2003.
56. Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein Generative Adversarial Networks. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017.
57. Tolstikhin, I.O.; Bousquet, O.; Gelly, S.; Schölkopf, B. Wasserstein Auto-Encoders. *arXiv* **2017**, arXiv:1711.01558.
58. Kobyzev, I.; Prince, S.J.; Brubaker, M.A. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *43*, 3964–3979. [CrossRef]
59. Kingma, D.P.; Salimans, T.; Jozefowicz, R.; Chen, X.; Sutskever, I.; Welling, M. Improved Variational Inference with Inverse Autoregressive Flow. In Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, Changsha, China, 20–23 November 2023; Curran Associates, Inc.: Red Hook, NY, USA, 2016; pp. 4743–4751.
60. Germain, M.; Gregor, K.; Murray, I.; Larochelle, H. MADE: Masked Autoencoder for Distribution Estimation. In Proceedings of the Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 7–9 July 2015. [CrossRef]
61. Gregory, J.A.; Delbourgo, R. Piecewise Rational Quadratic Interpolation to Monotonic Data. *IMA J. Numer. Anal.* **1982**, *2*, 123–130. [CrossRef]
62. Kolmogorov-Smirnov, A.; Kolmogorov, A.N.; Kolmogorov, M. Sulla determinazione empírica di uma legge di distribuzione. *Giorn. Dell'inst. Ital. Att.* **1933**, *4*, 89–91.
63. Smirnov, N. On the estimation of the discrepancy between empirical curves of distribution for two independent samples. *Bull. Math. Univ. Moscou.* **1939**, *2*, 2–26.
64. Kolmogoroff, A.N. Confidence Limits for an Unknown Distribution Function. *Ann. Math. Stat.* **1941**, *12*, 461–463. [CrossRef]
65. Smirnov, N.V. Table for Estimating the Goodness of Fit of Empirical Distributions. *Ann. Math. Stat.* **1948**, *19*, 279–281. [CrossRef]

66. KStwobign Distribution, Scipy Python Package. 2023. Available online: https://docs.scipy.org/doc/scipy/tutorial/stats/continuous_kstwobign.html (accessed on 10 January 2024).
67. Rabin, J.; Peyré, G.; Delon, J.; Bernot, M. Wasserstein Barycenter and Its Application to Texture Mixing. In Proceedings of the Third International Conference, SSVM 2011, Scale Space and Variational Methods in Computer Vision, Ein-Gedi, Israel, 29 May–2 June 2011; Bruckstein, A.M., ter Haar Romeny, B.M., Bronstein, A.M., Bronstein, M.M., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 435–446.
68. Bonneel, N.; Rabin, J.; Peyré, G.; Pfister, H. Sliced and Radon Wasserstein Barycenters of Measures. *J. Math. Imaging Vis.* **2015**, *51*, 22–45. [CrossRef]
69. Kantorovich, L.V. On the translocation of masses. *Dokl. Akad. Nauk SSSR* **1942**, *37*, 227–229. [CrossRef]
70. Wasserstein, R.L. Markov Processes Over Denumerable Products of Spaces, Describing Large Systems of Automata. *Probl. Peredachi Inform.* **1969**, *5*, 64–72.
71. Muller, M.E. A Note on a Method for Generating Points Uniformly on N-Dimensional Spheres. *Commun. ACM* **1959**, *2*, 19–20. [CrossRef]
72. Code Repository for This Paper on GitHub. Available online: https://github.com/NF4HEP/NormalizingFlowsHD (accessed on 10 January 2024).
73. NF4HEP Code Repository on GitHub. Available online: https://github.com/NF4HEP/NFTF2_dev (accessed on 10 January 2024).
74. Code Repository for Statistical Inference and Evaluation Metrics on GitHub. Available online: https://github.com/NF4HEP/GenerativeModelsMetrics (accessed on 10 January 2024).
75. Coccaro, A.; Pierini, M.; Silvestrini, L.; Torre, R. The DNNLikelihood: Enhancing likelihood distribution with Deep Learning. *Eur. Phys. J. C* **2020**, *80*, 664. [CrossRef]
76. Reyes-Gonzalez, H.; Torre, R. The NFLikelihood: An unsupervised DNNLikelihood from Normalizing Flows. *arXiv* **2023**, arXiv:2309.09743.
77. Friedman, J. On Multivariate Goodness-of-Fit and Two-Sample Testing. In Proceedings of the Conference on Statistical Problems in Particle Physics, Astrophysics and Cosmology, Stanford, CA, USA, 8–11 September 2003; Lyons, L., Mount, R., Reitmeyer, R., Eds.; U.S. Department of Energy: Washington, DC, USA, 2003; p. 311.
78. Kansal, R.; Li, A.; Duarte, J.; Chernyavskaya, N.; Pierini, M.; Orzari, B.; Tomei, T. On the Evaluation of Generative Models in High Energy Physics. *arXiv* **2022**, arXiv:2211.10295.
79. D'Agnolo, R.T.; Grosso, G.; Pierini, M.; Wulzer, A.; Zanetti, M. Learning multivariate new physics. *Eur. Phys. J. C* **2021**, *81*, 89. [CrossRef]
80. Chakravarti, P.; Kuusela, M.; Lei, J.; Wasserman, L. Model-Independent Detection of New Physics Signals Using Interpretable Semi-Supervised Classifier Tests. *arXiv* **2021**, arXiv:2102.07679.
81. Letizia, M.; Losapio, G.; Rando, M.; Grosso, G.; Wulzer, A.; Pierini, M.; Zanetti, M.; Rosasco, L. Learning new physics efficiently with nonparametric methods. *Eur. Phys. J. C* **2022**, *82*, 879. [CrossRef]
82. Del Debbio, L.; Rossney, J.M.; Wilson, M. Machine Learning Trivializing Maps: A First Step Towards Understanding How Flow-Based Samplers Scale Up. In Proceedings of the 38th International Symposium on Lattice Field Theory (2021), Virtual, 26–30 July 2021; p. 059. [CrossRef]
83. TensorFlow Probability MAF Documentation. Available online: https://www.tensorflow.org/probability/api_docs/python/tfp/bijectors/MaskedAutoregressiveFlow (accessed on 10 January 2024).