LHC Hadronic Jets Generation Using a ConVAE+NF Approach

Breno Orzari, Raphael Cobe, Jefferson Fialho, Thiago Tomei, Nadezda Chernyavskaya, Maurizio Pierini, Raghav Kansal, Javier Duarte

2023-04-12

Abstract

1 Introduction

In the CERN's Large Hadron Collider (LHC) [1,2], approximately one billion collisions between protons (pp) happen every second. Those collisions generate outgoing particles that are measured by dedicated, custom-made particle detectors. The main data stream collected by such a detector is close to 1.4 GB/s. After the upcoming LHC upgrade to the High-Luminosity LHC (HL-LHC) [3] this number will increase by a factor of 3.5–5. In High Energy Physics (HEP), an important part of the data analysis is the simulation of pp collisions that are used to test analysis techniques, estimate the particle detectors' efficiencies and resolutions, make comparisons with collected data, etc. Usually, the number of simulated events is an order of magnitude higher than the number of stored data, such that the statistical uncertainty of the generated processes doesn't dominate overall uncertainty. For simulating collision data, the generation task uses Monte Carlo (MC) event generators. However, the increased granularity of the detectors and the higher complexity of the collision events after the HL-LHC upgrade pose significant challenges which cannot be solved by increments in the computing resources alone [4].

In particular, one of the most common final-state objects in pp collisions is in the form of showers of hadronic particles, the hadronic jets [5]. These can be described by the quantities of its constituent particles, mainly their four-momentum. The Particle Flow (PF) [6] algorithm is used to combine the particles' information obtained by several sub-detectors to precisely describe its properties. One can use jet reconstruction algorithms [7] to cluster the particle constituents into a jet and obtain its relevant properties[1] for physics analysis such as its invariant mass, transverse momentum (p_T), energy, ϕ, η and many other.

Being able to speed-up the process of jets simulation in a particle detector would be of great advantage to mitigate the needed overhead in computing resources. One of the avenues that shows very promising results is to use Machine Learning (ML) algorithms to perform the jets simulation. The goal is to maintain MC methods' efficiency while making the process orders of magnitude faster. Recently, several approaches using Neural Networks (NN) have been studied and applied for this purpose. More specifically, Generative Adversarial Networks (GAN) [8,9] for the generation of jet constituents and, consequently, jets, have already been explored and provided very inspiring results. Another promising technique is the use of Variational Autoencoders (VAE), while already has been explored [10], is yet to demonstrate high fidelity results.

In this work, a new method for the generation of hadronic jets from noise is presented. It is based on a VAE coupled with a Normalizing Flow (NF) and trained in a two-step process. The paper is organized as follows: section 2 contains a literature overview of ML applied to HEP, mainly focusing on generative methods for jets simulation; in section 3 the dataset and the model are described; section 4 details the custom loss function and the evaluation metrics; results are presented and discussed in section 5; finally in section 6 we summarize the work and give an outlook for using the new method.

[1] In the LHC experiments, the coordinate system used is the following: origin is set at the center of the local pp collision region; y axis is defined vertically upward and z axis along the proton beam direction. 3D coordinates are usually given in terms of $\rho = (x^2 + y^2)^{1/2}$, azimuth angle ϕ and pseudorapidity $\eta = -\ln\tan(\theta/2)$, where θ is the polar angle.

2 Related Work

ML techniques have been applied to hadronic jets generation following several distinct approaches. Some of the first attempts [11–15] were performed using convolutional neural networks (CNN) given the impressive computer-vision results provided by those and the fact that an image-like representation of a jet comes naturally from the particle detectors' data. Jet image datasets are still popular amongst researchers, and have also been used by works that implemented other types of generative network architectures, such as graph neural networks (GNN) and normalizing flows (NF) [16–18]. Other jets representations such as vectors of energy deposits or jets characteristics have also been applied [19, 20].

Recently, particle-based jet datasets have also been extensively used by the ML community in the context of jets generation [8–10, 21]. They are composed of the jets constituent particles' properties and give a detailed description of particles distribution inside the jet, which is known as the jet substructure [22, 23]. GANs based on graph networks [8, 9] have provided inspiring results when aiming to generate particle-based jets, retaining great similarity between generated jets and MC simulated jets, while improving the speed of simulation by 4 to 5 orders of magnitude depending on the number of particles inside the generated jet. Some VAE based on convolutional networks have also been implemented using the same dataset for jets fast simulation [21] and for the jets generation as well [10]. However, for the later, there is great room for improvement when comparing the VAE generated jets with MC simulated jets.

The premise of this work is that the VAE by itself is not enough to capture the full characteristics of a hadronic jet, mainly due to the *a priori* choice of the latent space distribution. On the other hand, a combined VAE+NF approach, which has already been applied for image generation [24], time series data prediction [25], among others, has not yet been tried for HEP applications. In this work, that approach is used to improve on the generation capacity of the VAE.

3 Dataset and Model

We use the gluon HLS4ML LHC Jet dataset [26, 27] composed of ∼177k gluon jets, containing the jets' constituent particles tri-momentum (p_x, p_y and p_z). Those were obtained from simulated pp collisions with a center of mass energy of $\sqrt{s} = 13$ TeV that produced jets with $p_T \approx 1$ TeV, using a parameterization of a general purpose LHC experiment-like particle detector to simulate particles' interaction with the detector material. The particle showers were reconstructed using the anti-k$_T$ algorithm [28, 29] with $\Delta R = 0.8$. All of the 177k samples were divided into $\sim 70\%$ for training, and $\sim 15\%$ each for validation and testing. Even though there is no intrinsic ordering to the particles inside a jet, due to this representation of a hadronic jet being rather sparse, the particles were ordered in a list by decreasing p_T and only the first 30 particles were used as input to the ML algorithm. If a jet didn't contain the 30 particles, the rest of the list was filled with zeros (zero-padding). A feature-wise standardization was performed to bound the values of the tri-momentum to be in the range [0.0, 1.0] as input to the network.

3.1 Convolutional VAE

To accommodate this representation of the input dataset, the chosen model is a VAE [30–32] composed of convolution layers (ConVAE). The variational autoencoder is made of three structures: an encoder, which learns to compress the representation of the input data into a lower dimensional representation; the latent space, that stores this lower dimensional representation into values that closely follow a probability distribution defined by the user; and the decoder, which decodes the latent space values to the output. The probability distribution of the latent space values has to be differentiable, easy to implement and to sample from, where the most common choice is the Standard Gaussian, $\mathcal{N}(0, 1)$. The goal of the VAE is to do this process to produce output data that is as similar as possible to the input in such a way that, during the generation step, it is possible to randomly sample from $\mathcal{N}(0, 1)$ and directly insert those values into the decoder to generate new data that resembles the training dataset.

The network was implemented in PyTorch library [33]. In this work, the encoder and decoder structures are mirrored, and only the encoder will be described. It is composed of a given number of consecutive two-

dimensional convolution layers where the last convolution layer is flattened and introduced into two linear layers, where the latter is fed into the latent space. An activation function is applied after each layer, except for the last linear layer of the encoder and the first linear layer of the decoder. The input and output layers of the network have a fixed size of 3×30, the activation function in between each layer, number of convolution layers in encoder and decoder, kernel size, latent dimension size, number of filters and number of nodes in the linear layers between encoder (decoder) and the latent space were set as hyperparameters of the NN to be obtained through optimization. The activation function of the last deconvolution layer of the decoder was chosen to be the Sigmoid [34], to bound the output values in between 0.0 and 1.0, given the feature-wise standardization.

In the encoder architecture, the number of input nodes of the first linear layer is fixed by the size of the last convolution layer, the number of output nodes of the second linear layer is fixed as two times the number of dimensions of the latent space that will provide the means and standard deviations for the reparameterization trick [32], and in the decoder architecture, the number of input nodes in the first linear layer is fixed by the latent space size and the number of output nodes in the second linear layer is fixed by the size of the first deconvolution layer. The stride and padding of the convolution layers were kept as 1 and 0, respectively. The first convolution layer has 1 input filter and $\mathrm{N_{filters}}$ output filters, and the rest of the convolution layers has an input number of filters equal to $2^{\mathrm{layer}-1}\times\mathrm{N_{filters}}$ and an output number of filters equal to $2^{\mathrm{layer}}\times\mathrm{N_{filters}}$.

Hyperparameter optimization was performed using the Optuna package [35], which allows for fast convergence due to an aggressive pruning based on intermediate results and easy parallelization. Each ConVAE was set to go through 300 trials of the optimization, where each trial was executed for 300 epochs of the training. At training time, the batch size was kept fixed at 100 samples and the Adam optimizer [36] with a learning rate (lr) that was set by the hyperparameter optimization was used to update the network parameters at each epoch. During the optimization, the evaluation metric (described in details in section 4) was minimized using the value evaluated on the validation dataset after every 5 epochs. The ConVAE was retrained with the best set of hyperparameters for 1500 epochs, and the best model was chosen as the one that exhibited the smallest value of the metric.

3.2 Normalizing Flows

When the ConVAE is tuned for reconstruction, no constrain is applied over the values of the latent space to follow a given probability distribution. In this case, the probability distribution of those values, $p_\theta(\mathbf{z})$, should be one of the best possible for the input dataset reconstruction. Those reconstruction-imposed distributions are not known and one couldn't do any modifications to the VAE itself to use them. However, when combining that approach with the usage of Normalizing Flows [37, 38], which is a technique that allows training a NN to find the transformation one probability distribution into another, and given that the values of the latent space are accessible, one can find the transformation from those to a simple and easy to sample from probability distribution, such as $p_\theta(\mathbf{z}) \xrightarrow{f} u_\varphi(\mathbf{x})$ where $u_\varphi(\mathbf{x}) = \mathcal{N}(0,1)$. Since these transformations are invertible by construction, it is possible to start from values that follow the simple distribution, pass it through the inverse of the transformation and get values that follow the complex and unknown distribution as $u_\varphi(\mathbf{x}) \xrightarrow{f^{-1}} p_\theta(\mathbf{z})$. The training of the network is done by maximizing the right-hand side of MAYBE CHANGE THE NOTATION OF EQUATIONS 2 AND 3, AS THEY ARE THE SAME IN THE REALNVP PAPER!!!

$$\log(p_\theta(\mathbf{z})) = \log(u_\varphi(f(\mathbf{z}))) + \log\left(\left|\det\left(\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}\right)\right|\right), \tag{1}$$

where $\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}$ is the Jacobian of the transformation.

The NF network used in this work was based on the RealNVP network[2] [39, 40] that makes use of very simple analytic expressions for each intermediate transformation

$$y_{1:d} = x_{1:d}, \tag{2}$$

[2]The implementation of the RealNVP network in this work was performed following the GitHub repository https://github.com/ispamm/realnvp-demo-pytorch.

$$y_{d+1:D} = x_{d+1:D} \cdot \exp(s(x_{1:d})) + t(x_{1:d}), \tag{3}$$

using the coupling layers [41], where only a subset of the input data undergoes the transformations while the rest remains untouched, and the subsets are permuted every epoch to ensure that all input data goes through the flows. Each individual transformation is still complex enough since the parameters s and t are obtained as the output of Multilayer Perceptrons (MLPs), which can be as complex as wanted.

To combine the ConVAE with the NF network, a two-step training was performed (ConVAE+NF). The procedure was as follows: the hyperparameters of the ConVAE were optimized and a model constructed with the best set of them was trained for 1500 epochs. The model that showed smallest metric when comparing the input with reconstructed jets, calculated every 5 epochs, was saved. Values of the latent space of the saved network, when receiving as input the training and validation jet datasets, were extracted and used to generate new latent space training and validation datasets for the use in the NF. A diagram containing the full training procedure is displayed in figure 1.

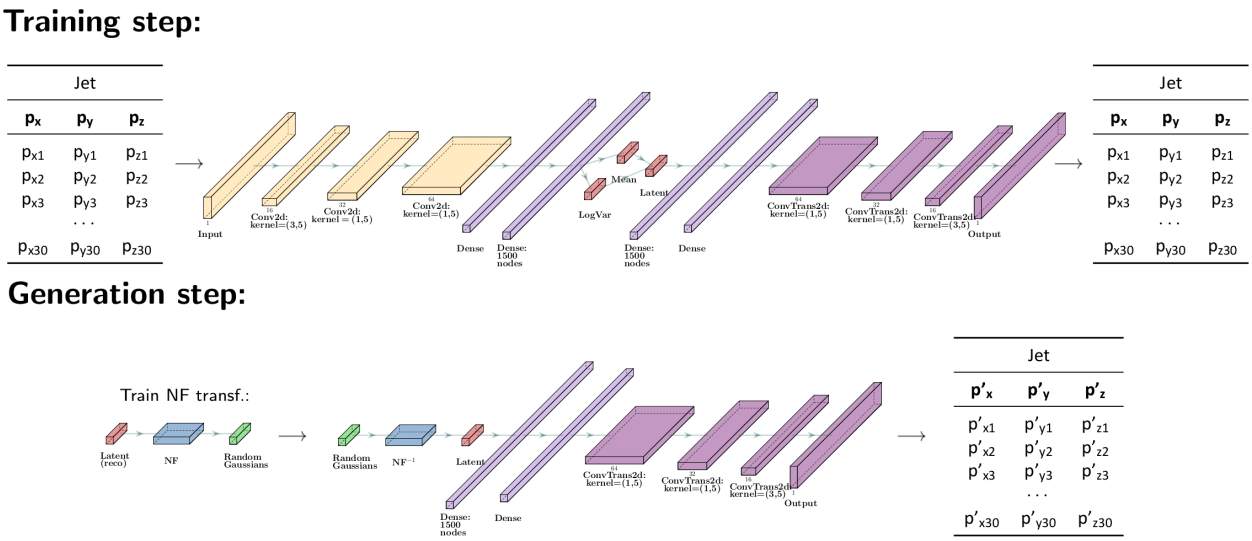

Figure 1: Illustration of ConVAE+NF network training scheme.

The NF network was also implemented using the PyTorch library. The optimizer chosen to update the network parameters after each epoch, aiming to minimize the negative of the mean of expression in equation 1 as loss function, was Adam with a learning rate that was optimized via the Optuna package, and an exponential decay rate valued as $^{lr}/_{10.0}$. The number of flows and the number of layers and nodes of the two MLPs that determined s and t were set as hyperparameters to be optimized, the ReLU [42] activation function was used in between each layer of the MLPs, and for s the hyperbolic tangent was used after its last layer to constrain its values to be in between (-1.0, 1.0). The number of input and output nodes of the s and t networks were set to match the number of latent dimensions of the ConVAE.

The metric was used to evaluate the combination of both networks as the hyperparameter optimization objective as follows: after every four epochs, a number of lat_{dim} values were sampled from $\mathcal{N}(0, 1)$ and fed into the inverse of the NF network, whose output was given as input to the decoder of the ConVAE tuned for reconstruction, and its output was compared with jets from the validation dataset. The optimization occurred for 300 trials with 20 epochs of the NF training per trial, and the best set of hyperparameters was used to train another NF network for 100 epochs where the model that exhibited the smallest metric value, obtained after every epoch, was saved.

4 ConVAE Loss Function and Evaluation Metric

The VAE loss function is given by the following expression [30]:

$$L(\theta, \phi) = -E_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\ p_\theta(\mathbf{x}|\mathbf{z})\right] + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})), \tag{4}$$

where the first term on the right-hand side, also known as the reconstruction term (L_{rec}), measures the distance between the output produced by the network and the input data, and the second term measures the difference between the probability distributions of the output of the encoder values with the latent space values, in which the distribution of the latent space is defined *a priori* by the user, and the distribution of the output of the encoder is inferred as one that is easy to work with, but still complex enough to describe its output, where the usual choice is a multivariate Gaussian.

The minimization of this loss function ensures that the output will be as close as possible to the input, while the values of the latent space closely follow the distribution chosen by the user. In addition, one can add a hyperparameter β [43] to control the importance of each term in the loss. The loss function can be written as:

$$L_{\text{VAE}} = (1-\beta)L_{\text{rec}} + \beta D_{KL}. \tag{5}$$

The reconstruction loss term was customized to ensure good generation capability to the network. The loss function that compares output particles with input dataset particles was chosen to be the chamfer distance [44], also referred to as nearest neighbour distance (NND). This loss function is preferred over the commonly-used Mean Squared Error (MSE) function, as it preserves permutation invariance, which is needed since the reconstructed particles in the jets have no intrinsic ordering. The NND loss is expressed as :

$$L^{\text{NND}} = \sum_k \left[\sum_{i\in\mathcal{J}_k} \min_{j\in\hat{\mathcal{J}}_k} D(\vec{p}_i, \vec{\hat{p}}_j) + \sum_{j\in\hat{\mathcal{J}}_k} \min_{i\in\mathcal{J}_k} D(\vec{p}_i, \vec{\hat{p}}_j)\right], \tag{6}$$

where i and j are indices that go through the particles in the input and output samples, respectively; k is the index on a given jet, where $\mathcal{J}_k$ represents a given dataset; hat distinguishes between input (without) and output (with) objects; and $D(\vec{p}_i, \vec{\hat{p}}_j)$ is the Euclidean distance between input and output particles, treating each as a vector in the p_x, p_y and p_z space. The first term finds the closest output particle to a given input particle, the second term finds the closest input particle to a given output particle, and their distances are summed.

The L^{NND} alone was not enough to provide good quality of the generated jets. Therefore, physics inspired knowledge had to be included in the form of additional distances between the input and output jets properties. The best results were achieved using a combination of the jets mass and p_T terms using MSE as a distance function. The additional loss term was added as follows :

$$L^{\text{J}} = \sum_k \left[\gamma_{p_T}\text{MSE}(p_{T_k}, \hat{p}_{T_k}) + \gamma_m \text{MSE}(m_k, \hat{m}_k)\right]. \tag{7}$$

where k is the index on a given jet; hat distinguishes between input (without) and output (with) objects; and parameters γ_{p_T} and γ_m were used to weight each contribution. Finally, the combined reconstruction loss is given by

$$L_{\text{rec}} = \alpha L^{\text{NND}} + \gamma L^{\text{J}}, \tag{8}$$

in which α and γ were used to weight the importance of each contribution to the loss function. The full set of loss parameters (α, β, γ, γ_{p_T} and γ_m) was also optimized with Optuna.

It is important to note here that the hyperparameter optimization procedure was carried out in two steps: the first where only the loss parameters and the optimizer learning rate were being optimized while the set of network's parameters was fixed at some reasonable choices established after manual tests, and the second where all the network's parameters were optimized given the just found best set of loss parameters. This was performed to ensure convergence of the optimization procedure in a feasible amount of time given the amount of resources available.

The evaluation metric chosen to quantitatively measure the generation capabilities of the distinct models was the Earth Mover's Distance (EMD; analogous to the 1-Wasserstein's distance) [45], calculated for each histogram of the input and output jets mass, energy, p_T, η and ϕ, and summed to get the EMD_{sum}. The choice of the variables was based on interesting quantities for jet-based searches in LHC experiments [46,47].

5 Results and Discussion

From figure 2 it is possible to notice that there is a clear tendency of EMD_{sum} to be minimized and reach a plateau as a function of the epochs, which is also stressed out by the behavior of each separate component.

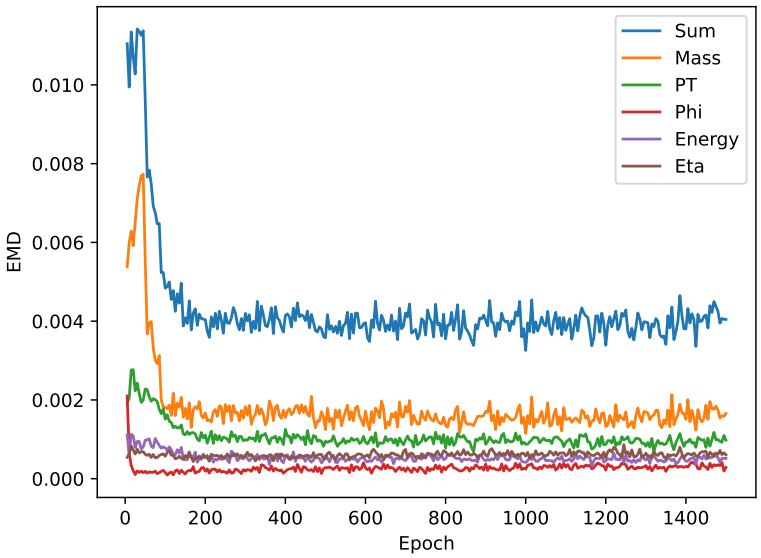

Figure 2: Evolution of EMD_{sum} and each of its components during training calculated every 5 epochs, EMD_{sum} (blue), EMD_{mass} (orange), EMD_{p_T} (green), EMD_{ϕ} (red), $\text{EMD}_{\text{energy}}$ (purple) and EMD_{η} (brown).

This metric was used as the hyperparameter optimization objective, the choice of the best model, and was ultimately used in between distinct approaches to determine the model with best generation capacity. As already described, the Hyperparameter optimization was performed using the Optuna framework and it was executed in steps to ease the convergence of the method due to limited time and computational resources. Below are displayed the ranges of each hyperparameter set used during the optimization for both types of NN.

In the ConVAE case, the set of parameters to be optimized was: the learning rate of the training optimizer (Adam); the loss parameters α, β, γ, $\gamma_{p_{\text{T}}}$ and γ_m; and the architecture parameters as the number of latent dimensions, number of filters, size of kernel related to number of particles to act over, number of nodes in linear layers, number of convolution layers and activation function. The 2D kernel size was kept as (3, kernel size) for the first (last) convolution (deconvolution) layer of the encoder (decoder), where the number 3 refers to the number of particle features of the input dataset, and it was set as (1, kernel size) otherwise.

Parameters $\gamma_{p_{\text{T}}}$ and γ_m ranges were extracted from the dataset, since, as it can be observed from the jet variables histograms NOW THE HISTOGRAMS ONLY APPEAR AFTER THIS PART; SHOULD I NOT MENTION THEM HERE?, jets p_T and mass have distinct scales, which would imply in different magnitude in the loss function. $\gamma_{p_{\text{T}}}$ was set to be optimized in a range 20% smaller and 20% higher than unity, while γ_m assumed values 20% smaller and 20% higher than the ratio of the mean of the jets p_T and jets mass. As stated before, for the loss hyperparameters optimization, the network architecture hyperparameters was fixed at reasonable choices, which are displayed in table 1. For both ConVAE networks (ConVAE and ConVAE+NF approaches) almost all the ranges of the hyperparameters for the optimization were the same:

- lr: $[10^{-5}, 10^{-1}]$ in log scale;
- α: [0.1, 1.0];
- γ: [0.1, 1.0];
- $\gamma_{p_{\text{T}}}$: [0.8, 1.2];

- γ_m: [9.986, 14.98];
- lat_{dim}: [10, 300] in multiples of 10;
- Number of filters: [5, 100] in multiples of 5;
- Kernel size: [1, 8] in multiples of 1;
- Number of linear nodes: [100, 3000] in multiples of 100;
- Number of convolution layers: [1, 4] in multiples of 1;
- Activation function: one of ReLU, GeLU, LeakyReLU, SELU or ELU.

The only exception was for the loss hyperparameter β that, due to the difference in the magnitudes of L_{rec} and D_{KL}, needed to be much closer to 1.0 for the ConVAE tuned for generation, where the optimization range was [0.9, 1.0] in log scale, and much closer to 0.0 for the ConVAE tuned for reconstruction, where the optimization range was [0.0, 0.1] also in log scale.

lat_{dim}	Filters	Kernel	N_{linear}	Layers	Act. Func.
30	50	5	1500	3	ReLU

Table 1: Set of common architecture hyperparameters for ConVAE loss parameters optimization.

Since the number of hyperparameters of the RealNVP network is much smaller than for the ConVAE, its hyperparameter optimization was performed only in one step. The set of hyperparameters that were optimized was: the learning rate of the training optimizer (Adam); and the network architecture hyperparameters as the number of flows, number of linear layers for the s and t MLPs and the number of nodes in each layer of those MLPs. For both networks, the one used together with the ConVAE and the one used by itself, the ranges of the hyperparameter optimization were:

- lr: [10^{-5}, 10^{-1}] in log scale;
- Number of flows: [5, 100] in multiples of 5;
- Number of linear layers of s and t MLPs: [1, 4] in multiples of 1;
- Number of nodes in each linear layer: [50, 400] in multiples of 50 (optimized for each layer);

The first approach was to optimize the hyperparameters, train and test the performance of a ConVAE without further modifications. The final set of optimized hyperparameters is disposed in table 2.

lr	α	β	γ	$\gamma_{p_{\text{T}}}$	γ_m	lat_{dim}	Filters	Kernel	N_{linear}	Layers	Act. Func.
6.1×10^{-4}	0.449	0.998	0.118	0.817	11.7	190	75	3	1100	4	ReLU

Table 2: Set of optimal hyperparameters for the ConVAE tuned for generation.

After hyperparameter optimization and saving the best model during training, the value of EMD_{sum} for this network was 0.0033. Figure 3 shows the comparisons of the distributions of input and output jets mass, p_T, energy, η and ϕ after hyperparameter optimization. Qualitatively, it is possible to notice that the comparisons for jets energy, η and ϕ are very similar, with differences only at the extremes of the histograms, while for p_T there is a slight difference at the peak of the distribution and small discrepancies at both tails. The greatest discrepancy is when comparing jets mass, where, from the ratio of the histograms, only generated jets with masses in the range 80 GeV $\lessapprox j_{\text{mass}} \lessapprox$ 175 GeV are reasonable.

The second approach was the study of the ConVAE+NF model. The complete sets of optimized hyperparameters are displayed in tables 3 and 4. After both networks were trained, at the testing stage, randomly

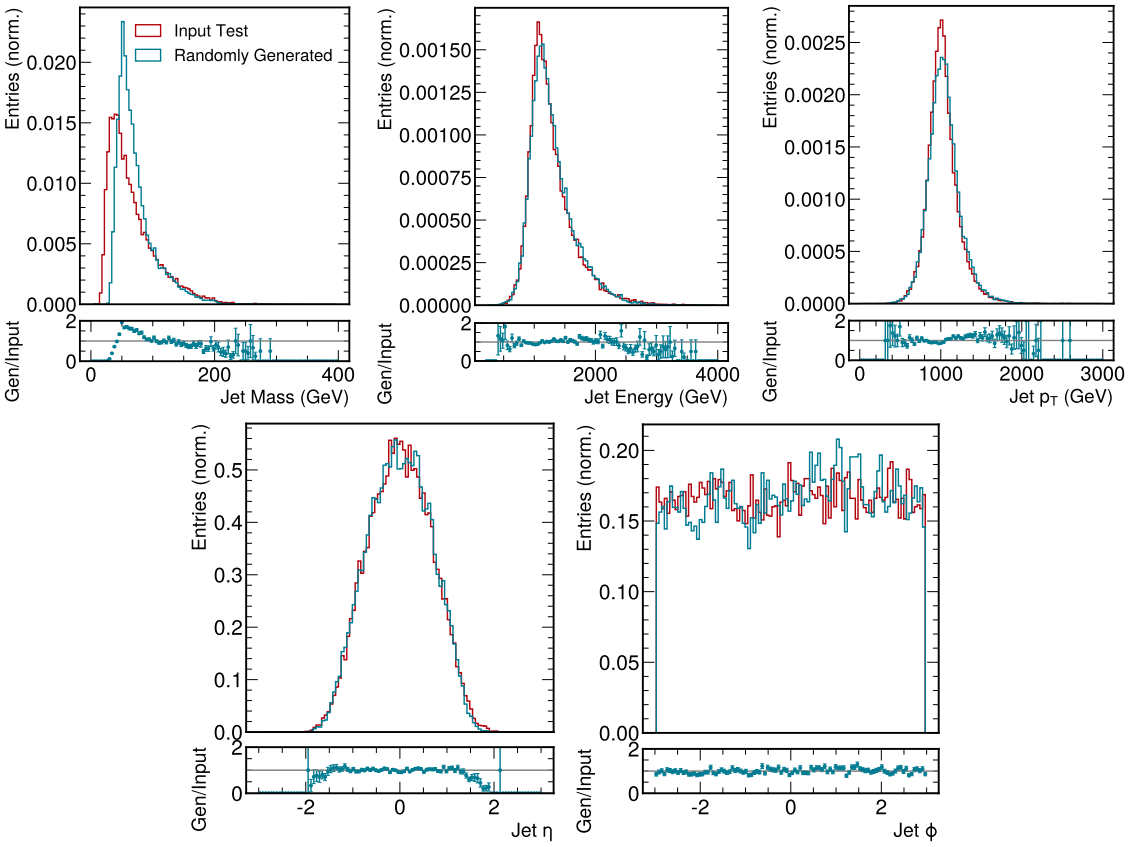

Figure 3: Comparison of input jets variable distributions (red) with randomly generated jets from the ConVAE model (blue). From left to right, top to bottom: mass, energy, p_T, η and ϕ distributions are displayed. On the subplots the ratio $^{\text{gen}}/_{\text{input}}$ is shown.

generated jets were compared with jets from the test dataset, providing an EMD_{sum} of 0.0026. The histograms of the jet variables are displayed in figure 4.

From the EMD_{sum}, it is possible to notice a great improvement on the generative capabilities of the network. Looking at the mass histogram, which was troubling for the ConVAE tuned for generation, it is possible to see that the randomly generated jets have great similarity with the input jets, not agreeing only for masses smaller than 25 GeV. However, specially when looking to η, ϕ and p_T histograms ratios, it is noticeable that a structure arose and is causing some discrepancies between randomly generated and input jets.

lr	α	β	γ	γ_{p_T}	γ_m	lat_{dim}	Filters	Kernel	N_{linear}	Layers	Act. Func.
5.3×10^{-4}	0.425	0.046	0.121	1.15	11.8	180	50	2	3000	4	ReLU

Table 3: Set of optimal hyperparameters for ConVAE tuned for reconstruction.

For comparison, a RealNVP network was also optimized, in the same way as the NF network for the ConVAE+NF approach, and trained to receive as input a flattened jet[3], transform it into $\mathcal{N}(0,1)$, and invert the transformation back to flattened jets. Table 5 contains the set of best parameters after hyperparameter optimization for this network. The value of EMD_{sum} for this case was 0.0049 and the comparison histograms

[3]Instead of the input jet be on the format 3×30 representing each particle's p_x, p_y and p_z, it was flattened to a 1×90 vector containing all particles p_x, then all p_y and, finally, all p_z.

lr	N_{flows}	Layers	N_{linear}
2.9×10^{-4}	55	4	[400, 350, 400, 400]

Table 4: Set of optimal hyperparameters for the NF to be used on the ConVAE+NF approach.

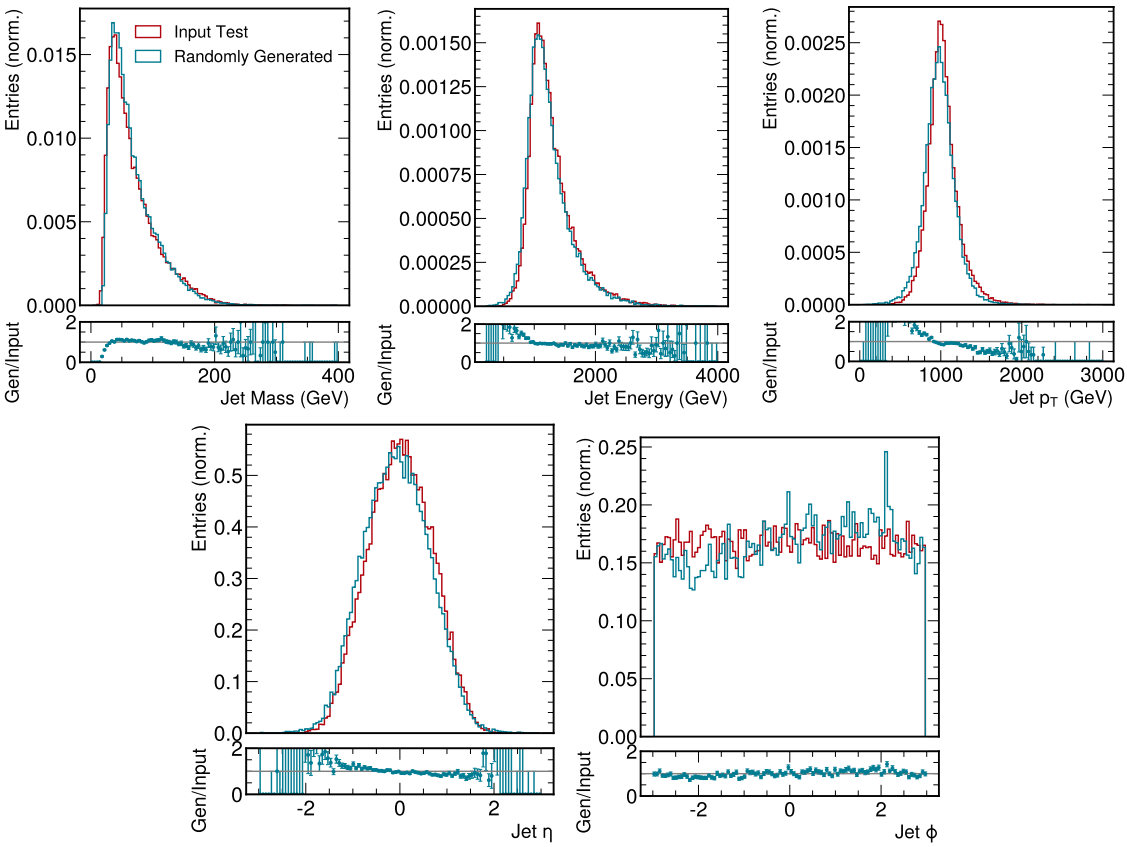

Figure 4: Comparison of input jets variable distributions (red) with randomly generated jets from the ConVAE+NF model (blue). From left to right, top to bottom: mass, energy, p_T, η and ϕ distributions are displayed. On the subplots the ratio $^{gen}/_{input}$ is shown.

are in figure 5 for the jets variables. This approach clearly shows that there is a big advantage in combining the inference capability of the ConVAE, with the invertibility of the NF network.

lr	N_{flows}	Layers	N_{linear}
3.4×10^{-4}	65	4	[150, 200, 350, 150]

Table 5: Set of optimal hyperparameters for the NF.

Following the work done in [8], another set of metrics was calculated and compared to the MPGAN results. W_1^M, W_1^P and W_1^{EFP} correspond to the average 1-Wasserstein distance between input and generated jets mass, constituents η^{rel}, ϕ^{rel} and p_T^{rel}, and the energy-flow polynomials (EFPs) which are a set of variables that capture the distribution of the particles inside the jets (substructure). The computation of the EFPs was performed using the JetNet package [48], where the EFPs used correspond to the set of loopless multigraphs with 4 vertices and 4 edges [8]. MAYBE CHANGE THIS. IT IS STILL TOO SIMILAR TO THE MPGAN PAPER!!!; FPND is the ParticleNet [49] version of the Fréchet Inception Distance (FID) [50], mainly used for comparing images in computer-vision works; and coverage (COV) and minimum matching

distance (MMD) which measure the fraction of samples in X that were matched to Y and the average distance between matched samples, respectively. TOO SIMILAR TO MPGAN PAPER, NEED TO CHANGE THE WAY IT IS WRITTEN!!! From table 6, it is possible to observe that, in between the models implemented in this work, the ConVAE+NF approach is better than the other two in almost every metric. However, the comparison with the MPGAN shows a very distinct picture in which the GAN outperforms the VAE in every single metric. The advantage of the ConVAE+NF approach, however, is that it is approximately two times faster ((18.30 ± 0.04) μs vs 35.7 μs as the average time to generate a jet; timing measured using a NVIDIA Tesla T4 GPU) and, given that its generated jets can be of use for some physics analysis applications, it might be useful in the future.

Model	W_1^M ($\times 10^{-3}$)	W_1^P ($\times 10^{-3}$)	W_1^{EFP} ($\times 10^{-5}$)	FPND	COV↑	MMD
ConVAE	9.1 ± 0.6	8.5 ± 0.8	576 ± 1035	43.6	0.32	0.036
ConVAE+NF	$\mathbf{4.5 \pm 0.5}$	$\mathbf{5.3 \pm 0.4}$	$\mathbf{197 \pm 247}$	**34.3**	**0.38**	0.034
NF	12.7 ± 0.7	8.5 ± 0.8	8.6k $\pm$ 13.2k	93.6	**0.38**	**0.033**
MPGAN	0.7 ± 0.2	0.9 ± 0.3	0.7 ± 0.2	0.12	0.56	0.037

Table 6: Set of metrics used in [8] to compare the performance of distinct ConVAE and NF approaches. In bold are the best values of the metrics when comparing the methods implemented in this work; last line exhibits the performace of the MPGAN for the same gluon jet dataset.

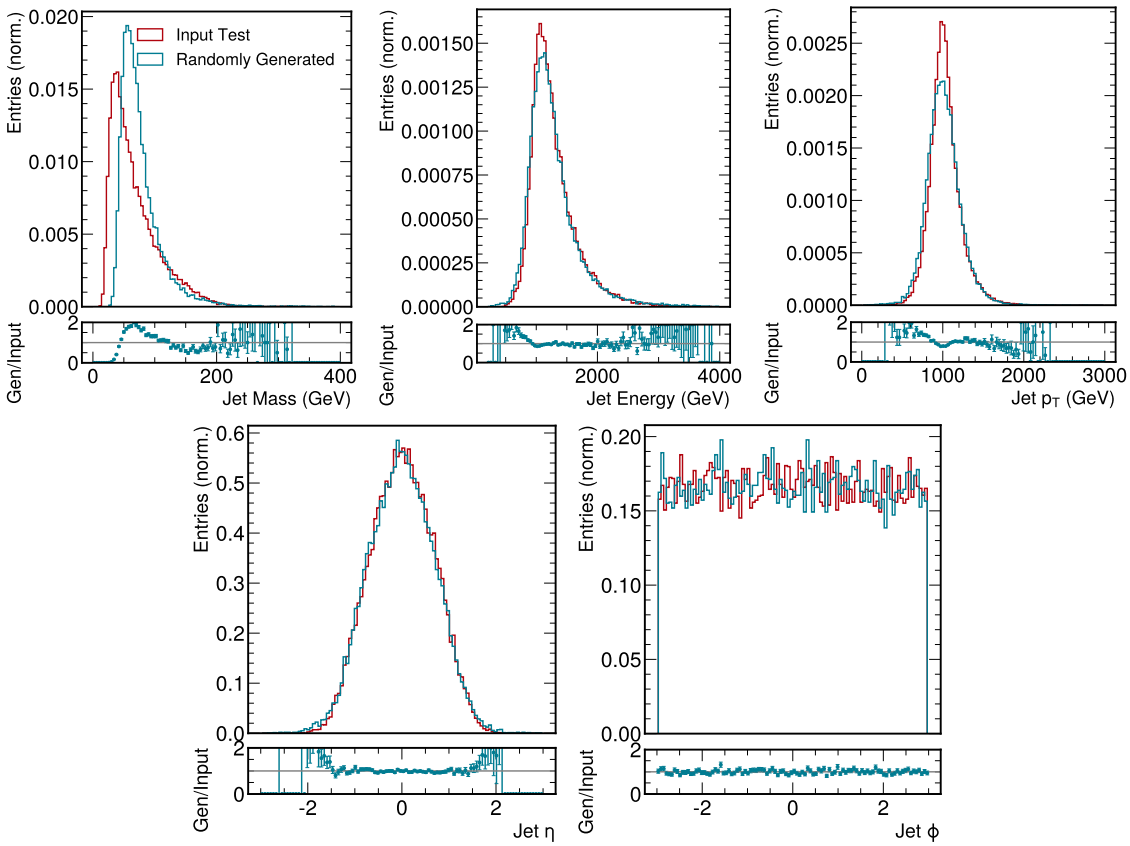

Figure 5: Comparison of input jets variable distributions (red) with randomly generated jets from the NF only model (blue). From left to right, top to bottom: mass, energy, p_T, η and ϕ distributions are displayed. On the subplots the ratio $^{\text{gen}}/_{\text{input}}$ is shown.

From the comparisons of the networks above, it is clear that the mixed VAE+NF approach exhibited an

improvement over the VAE only when looking into low and high-level hadronic jets variables. We attribute this to the fact that not only the Standard Gaussian distribution is not the optimal probability distribution for the elements in the latent space dimension, which was altered by the NF network, but, also, the decompression of those values could be improved by using the same parameters of the ConVAE tuned for reconstruction. There is still, however, some discrepancies in between the ConVAE+NF generated jets with input jets, mainly related to the substructure variables, and with the results of the MPGAN. The core difference between the two approaches is in the architecture chosen for the MPGAN that is a GNN. For the purpose of working with jets, their constituents' representation in terms of an unordered set of particles is much more natural, intrinsically preserves permutation invariance, and most probably is the cause of the large differences between ConVAE and MPGAN results.

6 Summary and Outlook

We presented a novel technique in the context of hadronic jets generation in simulated *pp* collisions, based on a machine learning approach that combined a Convolutional Variational Autoencoder with Normalizing Flows. From the values of EMD_{sum} obtained, there was a clear improvement with respect to the previous work [10], not only by the application of hyperparameter optimization to the ConVAE, but also with the usage of the ConVAE+NF technique. The generation of gluon jets was performed using the ConVAE+NF and compared with a standard ConVAE, NF alone, and the MPGAN, in which the ConVAE+NF stand out against the first two, but needs changes to be comparable to the later.

There is, however, an improvement in comparison with the MPGAN since the ConVAE+NF showed a decrease of almost two times in the time to generate each jet. This observation by itself can already offer advantages in physics applications where large amounts of simulated samples are needed but perfect accuracy is not a strict requirement. It is worth noting, though, that the MPGAN architecture is based on a graph neural network, which has a much more natural representation for a particle-based jet dataset. For a future work, a GraphVAE+NF can be implemented, trained and tested to check for improvements in the jet generation capabilities using the VAE architecture.

Acknowledgements

CHECK THE ACKNOWLEDGEMENTS!!!

B. O. and T. T. are supported by grant 2018/25225-9, São Paulo Research Foundation (FAPESP). B. O. was also partially supported by grants 2019/16401-0 and 2020/06600-3, São Paulo Research Foundation (FAPESP). R. K. was partially supported by an IRIS-HEP fellowship through the U.S. National Science Foundation (NSF) under Cooperative Agreement OAC-1836650. R. K. was additionally supported by the LHC Physics Center at Fermi National Accelerator Laboratory, managed and operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy (DOE). J. D. is supported by the DOE, Office of Science, Office of High Energy Physics Early Career Research program under Award No. DE-SC0021187, the DOE, Office of Advanced Scientific Computing Research under Award No. DE-SC0021396 (FAIR4HEP), and the NSF HDR Institute for Accelerating AI Algorithms for Data Driven Discovery (A3D3) under Cooperative Agreement OAC-2117997. M. P. and N. C. were supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement No. 772369).

References

References

[1] "LHC Design Report Vol.1: The LHC Main Ring", doi:10.5170/CERN-2004-003-V-1.

[2] O. Bruning, H. Burkhardt, and S. Myers, "The Large Hadron Collider", *Prog. Part. Nucl. Phys.* **67** (2012) 705–734, doi:10.1016/j.ppnp.2012.03.001.

[3] O. Aberle et al., "High-Luminosity Large Hadron Collider (HL-LHC): Technical design report". CERN Yellow Reports: Monographs. CERN, Geneva, 2020. doi:10.23731/CYRM-2020-0010.

[4] C. O. Software and Computing, "CMS Phase-2 Computing Model: Update Document", technical report, CERN, Geneva, 2022.

[5] A. Banfi, "Hadronic Jets". 2053-2571. Morgan Claypool Publishers, 2016. doi:10.1088/978-1-6817-4073-7, ISBN 978-1-6817-4073-7.

[6] CMS Collaboration, "The CMS Particle Flow Algorithm", *EPJ Web Conf.* **191** (2018) 02016, doi:10.1051/epjconf/201819102016.

[7] G. P. Salam, "Towards jetography", *The European Physical Journal C* **67** (may, 2010) 637–686, doi:10.1140/epjc/s10052-010-1314-6.

[8] R. Kansal et al., "Particle cloud generation with message passing generative adversarial networks", 2021. doi:10.48550/ARXIV.2106.11535, https://arxiv.org/abs/2106.11535.

[9] E. Buhmann, G. Kasieczka, and J. Thaler, "Epic-gan: Equivariant point cloud generation for particle jets", 2023. doi:10.48550/ARXIV.2301.08128, https://arxiv.org/abs/2301.08128.

[10] B. Orzari et al., "Sparse data generation for particle-based simulation of hadronic jets in the lhc", 2021. doi:10.48550/ARXIV.2109.15197, https://arxiv.org/abs/2109.15197.

[11] L. de Oliveira, M. Paganini, and B. Nachman, "Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis", *Computing and Software for Big Science* **1** (sep, 2017) doi:10.1007/s41781-017-0004-6.

[12] M. Paganini, L. de Oliveira, and B. Nachman, "CaloGAN: Simulating 3d high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks", *Physical Review D* **97** (jan, 2018) doi:10.1103/physrevd.97.014021.

[13] Vallecorsa, Sofia, Carminati, Federico, and Khattak, Gulrukh, "3d convolutional gan for fast simulation", *EPJ Web Conf.* **214** (2019) 02010, doi:10.1051/epjconf/201921402010.

[14] E. Buhmann et al., "Decoding photons: Physics in the latent space of a BIB-AE generative network", *EPJ Web of Conferences* **251** (2021) 03003, doi:10.1051/epjconf/202125103003.

[15] K. Dohi, "Variational autoencoders for jet simulation", 2020. doi:10.48550/ARXIV.2009.04842, https://arxiv.org/abs/2009.04842.

[16] A. Hariri, D. Dyachkova, and S. Gleyzer, "Graph generative models for fast detector simulations in high energy physics", 2021. doi:10.48550/ARXIV.2104.01725, https://arxiv.org/abs/2104.01725.

[17] C. Krause and D. Shih, "Caloflow: Fast and accurate generation of calorimeter showers with normalizing flows", 2021. doi:10.48550/ARXIV.2106.05285, https://arxiv.org/abs/2106.05285.

[18] C. Krause and D. Shih, "Caloflow ii: Even faster and still accurate generation of calorimeter showers with normalizing flows", 2021. doi:10.48550/ARXIV.2110.11377, https://arxiv.org/abs/2110.11377.

[19] ATLAS Collaboration, "Deep generative models for fast shower simulation in ATLAS", technical report, CERN, Geneva, 2018. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNOTES/ATL-SOFT-PUB-2018-001.

[20] R. D. Sipio, M. F. Giannelli, S. K. Haghighat, and S. Palazzo, "DijetGAN: a generative-adversarial network approach for the simulation of QCD dijet events at the LHC", *Journal of High Energy Physics* **2019** (aug, 2019) doi:10.1007/jhep08(2019)110.

[21] M. Touranakou et al., "Particle-based fast jet simulation at the LHC with variational autoencoders", *Machine Learning: Science and Technology* **3** (jul, 2022) 035003, doi:10.1088/2632-2153/ac7c56.

[22] S. Marzani, G. Soyez, and M. Spannowsky, "Looking inside jets: an introduction to jet substructure and boosted-object phenomenology", volume 958. Springer, 2019. doi:10.1007/978-3-030-15709-8.

[23] R. Kogler et al., "Jet Substructure at the Large Hadron Collider: Experimental Review", *Rev. Mod. Phys.* **91** (2019), no. 4, 045003, doi:10.1103/RevModPhys.91.045003, arXiv:1803.06991.

[24] R. Morrow and W.-C. Chiu, "Variational autoencoders with normalizing flow decoders", 2020. doi:10.48550/ARXIV.2004.05617, https://arxiv.org/abs/2004.05617.

[25] X.-B. Jin et al., "Pfvae: A planar flow-based variational auto-encoder prediction model for time series data", *Mathematics* **10** (2022), no. 4, doi:10.3390/math10040610.

[26] M. Pierini, J. M. Duarte, N. Tran, and M. Freytsis, "Hls4ml lhc jet dataset (30 particles)", January, 2020. doi:10.5281/zenodo.3601436, https://doi.org/10.5281/zenodo.3601436.

[27] E. A. Moreno et al., "JEDI-net: a jet identification algorithm based on interaction networks", *Eur. Phys. J. C* **80** (2020), no. 1, 58, doi:10.1140/epjc/s10052-020-7608-4, arXiv:1908.05318.

[28] M. Cacciari, G. P. Salam, and G. Soyez, "FastJet user manual", *The European Physical Journal C* **72** (mar, 2012) doi:10.1140/epjc/s10052-012-1896-2.

[29] M. Cacciari, G. P. Salam, and G. Soyez, "The anti-k_t jet clustering algorithm", *Journal of High Energy Physics* **2008** (apr, 2008) 063–063, doi:10.1088/1126-6708/2008/04/063.

[30] D. P. Kingma and M. Welling, "Auto-encoding variational bayes", 2013.

[31] C. Doersch, "Tutorial on variational autoencoders", 2016.

[32] D. P. Kingma and M. Welling, "An introduction to variational autoencoders", *CoRR* **abs/1906.02691** (2019) arXiv:1906.02691.

[33] A. Paszkeet al., "Pytorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

[34] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning", in *From Natural to Artificial Neural Computation*, J. Mira and F. Sandoval, eds., p. 195. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[35] T. Akiba et al., "Optuna: A next-generation hyperparameter optimization framework", 2019. doi:10.48550/ARXIV.1907.10902, https://arxiv.org/abs/1907.10902.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", 2014. doi:10.48550/ARXIV.1412.6980, https://arxiv.org/abs/1412.6980.

[37] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing flows: An introduction and review of current methods", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **43** (nov, 2021) 3964–3979, doi:10.1109/tpami.2020.2992934.

[38] G. Papamakarios et al., "Normalizing flows for probabilistic modeling and inference", doi:10.48550/ARXIV.1912.02762.

[39] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp", 2016. doi:10.48550/ARXIV.1605.08803, https://arxiv.org/abs/1605.08803.

[40] J. Pomponi and S. Scardapane, "Using a Normalizing Flow to Generate Image Embeddings". https://github.com/ispamm/realnvp-demo-pytorch/blob/master/Normalizing_Flow_for_Embedding_Generation.ipynb.

[41] L. Dinh, D. Krueger, and Y. Bengio, "Nice: Non-linear independent components estimation", 2014. doi:10.48550/ARXIV.1410.8516, https://arxiv.org/abs/1410.8516.

[42] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, eds., volume 15 of *Proceedings of Machine Learning Research*, pp. 315–323. PMLR, Fort Lauderdale, FL, USA, 11–13 Apr, 2011.

[43] I. Higgins et al., "β-VAE: Learning basic visual concepts with a constrained variational framework", in *5th International Conference on Learning Representations*. 2017.

[44] H. Fan, H. Su, and L. J. Guibas, "A point set generation network for 3D object reconstruction from a single image", in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 2463. 6, 2017. arXiv:1612.00603. doi:10.1109/CVPR.2017.264.

[45] Y. Rubner, C. Tomasi, and L. Guibas, "The earth mover's distance as a metric for image retrieval", *Int. J. Comput. Vis.* **40** (11, 2000) 99, doi:10.1023/A:1026543900054.

[46] ATLAS Collaboration, "Search for New Phenomena in Dijet Events using 139 fb^1 of pp collisions at $\sqrt{s}$ = 13TeV collected with the ATLAS Detector",.

[47] A. M. Sirunyan et al., "Search for high mass dijet resonances with a new background prediction method in proton-proton collisions at $$ \sqrt{s} $$ = 13 TeV", *Journal of High Energy Physics* **2020** (may, 2020) doi:10.1007/jhep05(2020)033.

[48] R. Kansal, "jet-net/jetnet: v0.0.3", October, 2021. doi:10.5281/zenodo.5597893, https://doi.org/10.5281/zenodo.5597893.

[49] H. Qu and L. Gouskos, "Jet tagging via particle clouds", *Physical Review D* **101** (mar, 2020) doi:10.1103/physrevd.101.056019.

[50] M. Heusel et al., "Gans trained by a two time-scale update rule converge to a local nash equilibrium", doi:10.48550/ARXIV.1706.08500.