Fast and accurate simulations of calorimeter showers with normalizing flows

Claudius Krause[*] and David Shih[†]

NHETC, Department of Physics and Astronomy, Rutgers University, Piscataway, New Jersey 08854, USA

(Received 25 July 2022; accepted 19 April 2023; published 28 June 2023)

We introduce CaloFlow, a fast detector simulation framework based on normalizing flows. For the first time, we demonstrate that normalizing flows can reproduce many-channel calorimeter showers with extremely high fidelity, providing a fresh alternative to computationally expensive GEANT4 simulations, as well as other state-of-the-art fast simulation frameworks based on generative adversarial networks (GANs) or variational autoencoders (VAEs). In addition to the usual histograms of physical features and images of calorimeter showers, we introduce a new metric for judging the quality of generative modeling: the performance of a classifier trained to differentiate real from generated images. We show that GAN-generated images can be identified by the classifier with nearly 100% accuracy, while images generated from CaloFlow are better able to fool the classifier. More broadly, normalizing flows offer several advantages compared to other state-of-the-art approaches (GANs and VAEs), including tractable likelihoods, stable and convergent training, and principled model selection. Normalizing flows also provide a bijective mapping between data and the latent space, which could have other applications beyond simulation, for example, to detector unfolding.

DOI: 10.1103/PhysRevD.107.113003

I. INTRODUCTION

The amazing successes of the LHC physics program in probing nature at its most fundamental level have been made possible only through an enormous accompanying computational effort. This includes not only computation related to the data (acquisition, reconstruction, analysis), but also computation related to detailed and accurate simulation of the Standard Model (SM). (For recent reviews of the current status and future plans of computing at LHC, see [1–5].) In fact, the latter effort (simulation) consumes by far the lion's share of computational resources of the LHC Collaborations. And within that, simulation of the detector response is the single most expensive element of the LHC computational pipeline. (See, e.g., Fig. 1 of [3].) Using GEANT4 [6–8] to simulate the full detector can take minutes per event. This in turn can severely limit analyses that rely on Monte Carlo simulations of SM processes.

In recent years, there has been considerable interest in the potential of machine learning and deep generative modeling to speed up detector simulations [9–17]. The idea is to train a neural network to faithfully reproduce the probability density of simulated events. By sampling from this fitted probability density, one can, in principle, generate realistic calorimeter images while shortcutting the computationally expensive *ab initio* modeling of the detector.

The applications of deep generative modeling to calorimeter simulation have so far almost entirely focused on generative adversarial networks (GANs) [18], as can be seen from the references given above.[1] GANs are the dominant deep learning framework for generative modeling of natural images, achieving stunning performance on a multitude of tasks, including producing realistic images that can even fool most human observers, creating original artworks, photorealistic inpainting, creating "deepfake" videos, face swapping, and face aging. They have also been used for a variety of other tasks in high-energy physics (HEP) [9–15,20–49]. In applications to fast calorimeter simulation, GANs have been demonstrated to be capable of reproducing GEANT4 calorimeter images with reasonable accuracy (at the individual image level, but also, more importantly, at the distributional level), while gaining up to 5 orders of magnitude in computational speed.

However, at the same time, GANs also have their drawbacks (see [50] for a nice recent overview). The GAN loss is famously a nonconvex min-max objective, and while theoretically this objective is optimized when the learned distribution matches the true distribution, because of the inherent instability to the training, they do not necessarily converge to this optimum in a controlled and measurable way. This leads to many well-known problems of GANs,

[*]Claudius.Krause@rutgers.edu
[†]shih@physics.rutgers.edu

[1]The exceptions are as follows: [13], which also considered a variational autoencoder (VAE), and [15,16], which considered a novel architecture called bounded information bottleneck autoencoder (BIB-AE) [19] that combines GANs with VAEs.

most notoriously the issue of "mode collapse," where the GAN will learn to generate only a subset of the data. More generally, it is not at all clear from studies of natural images how faithfully GANs truly reproduce the underlying distribution of the data. In HEP, the need for "realistic" individual events is less important than the need for accurate distributions. Each individual event is often very sparse and not very interpretable. It is only by aggregating a large number of events together and examining their distributions that we learn anything meaningful. This suggests that other approaches besides GANs could have advantages.

In this paper, we explore, for the first time, a completely different approach to deep generative modeling of calorimeter images: density estimation with normalizing flows (for recent reviews and original references, see, e.g., [51,52]). Normalizing flows use neural networks to learn a bijective mapping (with tractable Jacobian) between the data and a latent space described by a simple probability distribution (e.g., uniform or Gaussian). Being bijective (i.e., invertible), this transformation can, in principle, be run in either direction, allowing the probability density of existing data points to be inferred (density estimation with a tractable likelihood) and allowing new samples to be generated that follow the fitted distribution of data. Normalizing flows are capable of fitting complex, multimodal distributions in high-dimensional spaces, far better than previous methods (such as kernel density-estimation and Gaussian mixture models) [53,54]. Because they are parametrized by neural networks, normalizing flows strongly benefit from the expressivity and robustness that come with deep learning.

While normalizing flows have been applied to event generation and phase space integration [55–60], unfolding [61], data-driven background estimation [62], inference [63], and anomaly detection [64] previously in our field, these were all much lower-dimensional spaces than the calorimeter images we will consider in this work. Here we will demonstrate, for the first time, that normalizing flows are capable of describing the very high-dimensional space of GEANT4-generated calorimeter images with extremely high fidelity.

For our study, we will use the calorimeter setup of the original CaloGan paper [9,10] and compare our results to those of the CaloGan. The calorimeter is a simplified version of the ATLAS electromagnetic calorimeter, with three layers of sizes 3×96, 12×12, and 12×6 voxels, respectively. The GEANT4 data correspond to e^+, π^+, and γ perpendicularly incident on the calorimeter with energies uniformly sampled from 1–100 GeV. For several reasons, we chose to start with the simpler, but still very-high-dimensional setup of CaloGan instead of the even-higher-dimensional calorimeters considered in more recent works [e.g., $12 \times 15 \times 7$-dimensional calorimeter of [12] or the $30 \times 30 \times 30$ International Linear Detector (ILD)-prototype detector of [15,16]]. As this paper is the first demonstration of normalizing flows for fast calorimeter simulation, it is meant to be a proof of concept, and reproducing the 504 voxels of the CaloGan setup is already a major leap for normalizing flow-based modeling in our field. Furthermore, demonstrating CaloFlow on a simplified ATLAS electromagnetic calorimeter setup could have more immediate applications (i.e., to the actual ATLAS detector).

By a similar token, while it would have been interesting to compare to other state-of-the-art GAN architectures, such as the Wasserstein generative adversarial network with gradient penalty (WGAN-GP) [65,66], we believe that CaloGan is still indicative of the pros and cons of GAN-based fast simulation. In particular, characteristics such as the failing of the "ultimate" classifier test (explained below) also apply to more recent setups like the BIB-AE [15,16], as was shown in [41]. Finally, from a more practical standpoint, CaloGan made its GEANT4 training data [67] and source code including GEANT4 configuration [68] fully publicly available, so this greatly facilitated the comparison.

For the normalizing flow, we will use a combination of masked autoencoders for distribution estimation (MADE) [69] and neural spline flows [70] to maximize expressive power.

An innovative aspect of our approach is that we train two separate normalizing flows, a smaller one to learn the distribution of energies deposited in the three layers of the calorimeter and a larger one to learn the shower shapes in each layer. The first flow is constructed so that energy conservation is automatically ensured, while the second one learns images with unit-normalized total intensities, guaranteeing that the focus is on shower shapes in each layer and not on just the brightest voxels overall. This two-step generative framework could be useful even beyond normalizing flows and could potentially also improve GAN-based calorimeter simulations.

We will show that CaloFlow improves greatly upon the original CaloGan results and achieves an excellent description of the GEANT4 calorimeter images. We will perform qualitative comparisons of average and nearest-neighbor images, as well as more quantitative comparisons of histograms of important physics features such as energies, shower widths, and sparsity. Finally, we will demonstrate the extremely high fidelity of CaloFlow-generated images using a new quantitative metric: a binary classifier trained to distinguish GEANT4 and generated images.[2] An optimal classifier would be the ultimate metric for generative modeling, as it would be the most powerful test of $p_{\mathrm{real}}(x)$ vs $p_{\mathrm{generated}}(x)$ by the Neyman-Pearson lemma, resulting in random guessing between real and generated images if and only if $p_{\mathrm{real}}(x) = p_{\mathrm{generated}}(x)$. Of course, given finite training data and model capacity, any real-life

[2]Previous studies have focused on weaker classifier tests, such as training a classifier to distinguish between images of different particle types and seeing if there is any difference in performance when switching out GEANT4 for generated images. This is because the strong classifier test always distinguished the GAN vs real samples with nearly 100% accuracy [41,71].

classifier will be suboptimal. We will see plenty of evidence of this suboptimality: in the fact that our classifier scores will depend on preprocessing, data representation, and model architecture. Nevertheless, we believe even an approximately optimal classifier metric can be a very informative metric for generative model quality that provides a unique window into the multivariate correlations between features in a high-dimensional phase space. All in all, we will see that our trained classifier can learn with essentially perfect accuracy to distinguish between GEANT4 and CaloGan images, but has a much more difficult time distinguishing between GEANT4 and CaloFlow images.

We believe deep generative modeling with normalizing flows offers the following advantages over GAN-based simulations:

(i) Training a density estimator is a straightforward objective, unlike GANs which are saddle points. Therefore, the training and convergence are much more stable.

(ii) Since the loss of the density estimator is just the maximum likelihood, model selection is also completely straightforward. With GANs, it is often very challenging to select the "best epoch" since the generator and discriminator (or critic) losses are not so meaningful and often one must resort to subjective or *ad hoc* criteria (see, e.g., [15,17]). With flows, one just selects the epoch with the lowest loss on the validation set and this is more or less guaranteed to give the best results.

(iii) GANs only learn the likelihood implicitly (if at all), while density estimators produce a tractable, differentiable likelihood. This could have other applications beyond just generative modeling, e.g., parameter inference for particle reconstruction.

(iv) Since GANs do not fit the likelihood explicitly, they are prone to mode collapse and other pitfalls such as artifacts in images. We will show that CaloFlow is much more robust against mode collapse and that its images are objectively much closer to the GEANT4 ones.

(v) In a similar vein, since normalizing flows learn a bijective mapping that can be run in either direction, the CaloFlow could have more applications, e.g., to detector unfolding [61] or to understanding uncertainties [60].

The outline of our paper is as follows. In Sec. II, we give an introduction/overview to density estimation with normalizing flows (with further details in Appendix A). Section III contains a brief description of the calorimeter setup, which is taken from [10]. In Sec. IV, we define the specific two-step architecture that we use for CaloFlow and describe the pre- and postprocessing steps involved in training and generation. Finally, Sec. V contains the main results of the paper—average and nearest-neighbor images, histograms of physics features, and the direct classifier metric. We conclude in Sec. VI with a summary and a list of interesting future directions.

II. DENSITY ESTIMATION WITH NORMALIZING FLOWS

Normalizing flows (NFs) [51,52,72] are a special machine learning architecture that learn a bijective transformation between two spaces: the original data space x, where the data are described by an unknown (and usually complicated) probability density $p(x)$, and a "base" or "latent" space z, where the data follow a simple (usually uniform or normal) distribution $\pi(z)$. Under a bijective mapping $x = f^{-1}(z)$, the densities change according to

$$p(x) = \pi(f(x)) \left| \det \frac{\partial f(x)}{\partial x} \right| = \pi(z) \left| \det \frac{\partial f^{-1}(z)}{\partial z} \right|^{-1}. \qquad (1)$$

In the "forward" direction,[3] we start from a sample x in the data and infer its probability density via the first part of Eq. (1). The NF is a density estimator for the data. In the "inverse" direction, we start from a sample z of the base distribution and use the second part of Eq. (1) to map the sample to data space. The NF acts as a generative model in this case. In contrast to GANs, which learn the probability density implicitly, normalizing flows learn $p(x)$ explicitly. This has the advantage of a more stable and convergent training [$-\log p(x)$ is minimized directly] and no propensity for mode collapse in training (for sufficiently expressive f).

As is evident from Eq. (1), a tractable implementation of the bijective mapping requires tractability of the inverse as well as of the Jacobian determinant. NFs achieve this by using "simple" transformations $f(x;\vec{\kappa})$ that are analytically invertible and whose parameters $\vec{\kappa}$ are given by neural networks. By using specific architectures that have the "autoregressive" property (i.e., transformations of coordinate x_i depend only on the previous coordinates $x_1, \ldots, x_{i-1}$), the Jacobian matrix can be made triangular, such that the determinant can be computed in linear time as product of the diagonal entries (instead of in cubic time for a generic matrix). To ensure that the NF can learn complicated, high-dimensional data, a series of these simple bijectors ("blocks") is chained together to form the full bijective mapping between data and the base distribution.

In the machine learning literature, many options have been devised for both the family of transformations $f(x;\vec{\kappa})$ (e.g., affine transformations [74] or splines [70]), as well as the neural network architecture for their parameters $\vec{\kappa}$ (e.g., MADE blocks [69] and coupling layers [53]).[4] In principle,

[3]Of course, forward and inverse are a matter of convention. The choice here agrees with the terminology in the software package NFLOWS [73] that we use.

[4]See Appendix A for a more detailed description of these architectures.

these two components of normalizing flows can be chosen independently of one another. In the HEP literature, the masked autoregressive flow (MAF) architecture [54] (affine transformations with MADE blocks for the parameters) was used in anomaly detection with density estimation (ANODE) [64] for density estimation, and coupling layers with rational quadratic splines (RQS) [70] were used in I-FLOW [55,56] for phase space integration.

In this paper, we will consider a combination of transformations and neural network parametrizations that we believe maximizes the expressivity of the normalizing flows: RQS transformations with MADE blocks for the parameters.

(i) MADE blocks offer superior density-estimation performance compared to coupling layers [54]. The price one pays for this is that the MADE approach is very fast in one direction, as the full set of transformation parameters are given by the outputs of a single pass through the MADE block. The other direction, however, is slow in evaluation, as the parameters for the inverse transformation can only be obtained by looping through all dimensions. (Coupling-layer-based approaches tend to be equally fast in both directions.) Depending on the use case and the available computing resources, one can choose to implement the faster pass for the density-estimation direction, yielding a MAF [54], or implement the faster pass for the sampling direction, yielding an inverse autoregressive flow (IAF) [75].[5] Even though our main application would be sampling (generation of calorimeter showers), we implemented the MAF-style architecture, as it was impossible to store the gradients of the IAF-style architecture while looping through 504 dimensions. Even if the memory constraints could be overcome, training the IAF-style architecture would still take significantly longer than the MAF-style architecture, making it likely prohibitive. Instead, in a future project [76], we explore the combination of a MAF with an IAF to benefit from the best of both, known as probability density distillation [77].

(ii) For the family of transformations $z = f(x)$, we will use the monotonic RQSs from [70,78] to further increase the expressivity of the normalizing flow. These are continuous functions with continuous first derivatives, which are defined in a piecewise manner on intervals in some bounded, square region $[-B, B]$ of x and z space, with the tail bound B being a fixed hyperparameter, not parametrized with a neural network. (Outside of $[-B, B]$ the transformation is taken to be the identity mapping.)

On each interval, the transformation is a rational quadratic function [70,78],

$$z = f(x) = z_0 + \frac{(z_1 - z_0)[s\xi^2 + d_0\xi(1-\xi)]}{s + [d_1 + d_0 - 2s]\xi(1-\xi)}, \quad (2)$$

where $\xi = (x - x_0)/(x_1 - x_0)$, $s = (z_1 - z_0)/(x_1 - x_0)$, and $x_{0(1)}, z_{0(1)}$, and $d_{0(1)}$ are the locations and derivatives at the left (right) boundary of the interval. In total, after imposing continuity of the function and its first derivatives, there are a total of $3(n-1)$ parameters for an n-interval RQS. In practice for numerical stability, the algorithm of [70] uses one more pair of (x, z) locations and renormalizes the ranges to have length $2B$, increasing the number of parameters to a total of $3n - 1$. These were summarized as $\vec{\kappa}$ above. The inverse of the transformation is given by the positive root solution of a quadratic equation, leading to a monotonically increasing x within the boundary of the original distribution. Further details on the implementation and numerical stability can be found at [70].

Figure 1 shows the schematic view of a sample MADE block (this is just an example for illustration purposes; it is not the exact architecture we are using for CaloFlow) on the left. There, a three-dimensional distribution $\vec{x}$ is transformed based on a four-dimensional conditional vector $\vec{c}$. The input layers for the three coordinate and four conditional inputs have six neurons each. Their outputs are summed and then fed into the subsequent hidden layers. There are two such hidden layers of six neurons each and there is an output layer with 15 neurons, giving the parameters $\vec{\kappa}$ for three RQSs with two bins each. The connections inside the network are masked, such that the parameters of transformation i only depend on coordinates $k < i$. Connections are colored to illustrate this: Red connections, yielding the parameters $\vec{\kappa}_1$ of the RQS transforming x_1, only connect back to x_0 and $\vec{c}$; blue connections, yielding the parameters $\vec{\kappa}_2$ of the RQS transforming x_2, only connect back to x_0, x_1, and $\vec{c}$. $\vec{\kappa}_0$ is given by a trainable bias term, with no connection to the bijector or conditional input. On the right, we show an example for a two-bin RQS. We highlight the five parameters $\vec{\kappa}$ coming from the MADE block in green.

III. CALORIMETER SETUP

As described in the Introduction, we base our proof-of-concept study heavily off of the CaloGan setup of [9,10]. The toy calorimeter of CaloGan was a three-layer, liquid argon calorimeter cube with 480 mm side length. The training data consisted of GEANT4 calorimeter images for three particle types (e^+, γ, and π^+). These are electromagnetic showers only (ECAL). The particles are perpendicularly incident

[5]While the original references for MAF [54] and IAF [75] only use affine transformations, we use the terminology for a generic stack of MADE blocks with any type of transformation.

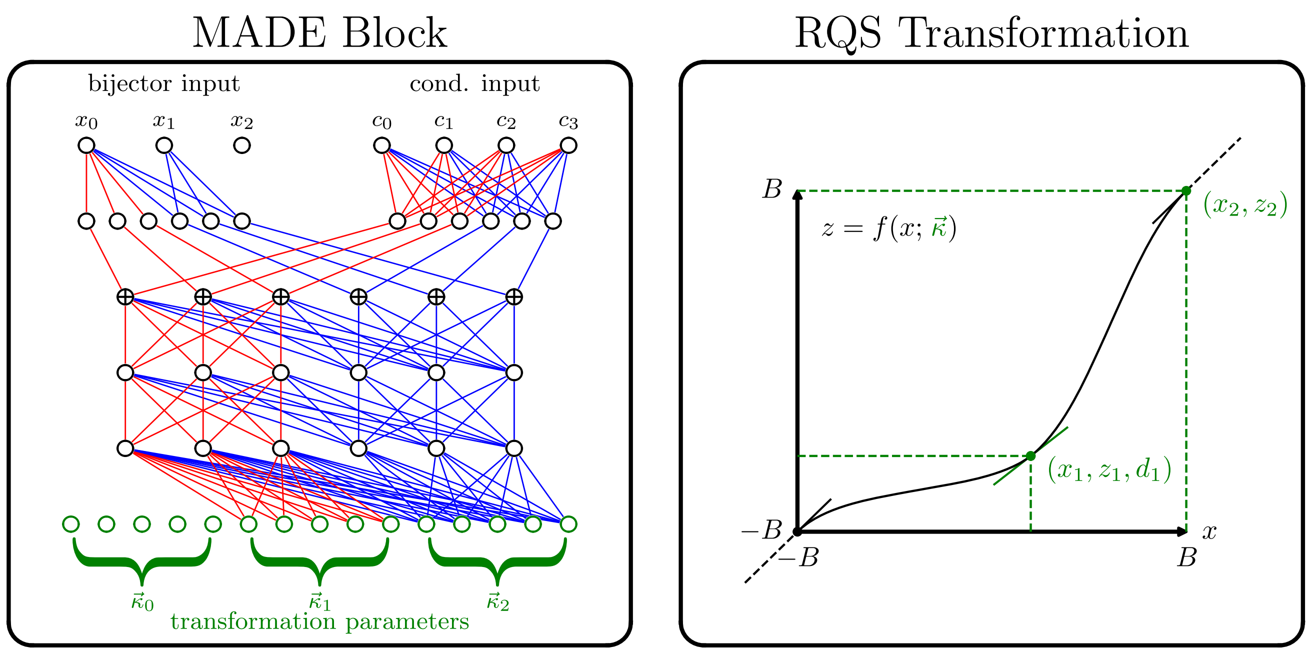

FIG. 1. Left: schematic view of a MADE block. There are three variables x_i to be transformed and there are four additional variables c_i, which the transformation is conditioned on. This example uses input layers with six nodes, two hidden layers with six nodes, and an output layer with 15 nodes for $\vec{\kappa}_i$, where the latter number is given by the required parameters of a two-bin rational quadratic spline transformation. Red connections show that $\vec{\kappa}_1$ only depend on x_0 and $\vec{c}$. Blue connections show that $\vec{\kappa}_2$ only depend on x_0, x_1, and $\vec{c}$. Right: an example for a two-bin rational quadratic spline transformation. Green color indicates the parameters $\vec{\kappa}$ coming from a neural network (NN): all but one of the knot locations and the derivatives of all internal knots. Outside the domain $[-B, B]$, the identity transformation is applied (indicated by the black dashed line).

with energy E_{inc} uniformly sampled from 1 to 100 GeV. The voxel sizes are not uniform (see Table I), yielding a different resolution in the three layers. The first, second, and third layers have resolutions of 3×96, 12×12, and 12×6, respectively. Importantly, the deposited energy is not exactly E_{inc} due to leakage and punchthrough (especially with the pions).

While the original GEANT4 samples that were used to train CaloGan are publicly available at [67], we chose instead to generate our own samples [79] using the GEANT4 code provided with CaloGan [68]. We checked that our GEANT4 samples were indistinguishable from the ones used in the original CaloGan work at the level of histograms and average images, but to be safe we chose not to mix and match the two samples, since they were produced with newer versions of GEANT4, the C compiler, etc.

Since there are a total of 100,000 calorimeter images of each particle type in the dataset [67], we generated 100,000 samples with GEANT4 based on the code provided at [68]. We use 70,000 of them for training and 30,000 for testing. (CaloGan used the full set of 100,000 for training.) For the classifier test of Sec. V D, we generate an additional, independent sample of 100,000 calorimeter images of each particle type with GEANT4. We split this set into sets of 60,000 for training, 20,000 for validation, and 20,000 for testing [79].

TABLE I. Sizes of the calorimeter voxels. The z axis is along the particle propagation direction (corresponding to radial direction in a full detector), the η axis is along the proton beam axis, and ϕ is perpendicular to z and η [10].

Layer	z length (mm)	η length (mm)	ϕ length (mm)	Number of voxels
0	90	5	160	3×96
1	347	40	40	12×12
2	43	80	40	12×6

IV. CaloFlow

Our goal is to learn the full joint probability density $p(\vec{\mathcal{I}}|E_{\text{inc}})$ of the 504 calorimeter voxel intensities $\vec{\mathcal{I}}$, conditioned on the input energy E_{inc}. We will treat each particle type (e^+, γ, and π^+) as a separate density-estimation problem.

The simplest and most direct approach would be to train a single NF on the full calorimeter. Unfortunately, this turned out to be insufficiently precise for the high degree of energy conservation that we require. Training separate NFs on the voxels of each calorimeter layer $\vec{\mathcal{I}}_k$, conditioned on the energy depositions of the voxels in previous calorimeter

TABLE II. Composition of flow I and flow II. "MADE input dimension" refers to bijector and conditional input, see Fig. 1, top left. "Input layer size" refers to the first hidden layer, which merges bijector and conditional input. The size of the output layer is determined by the number of RQS bins and the dimensionality of the data space.

	MADE input dimension	Base distribution	Number of MADE blocks	Layer sizes: Input	Layer sizes: Hidden	Layer sizes: Output	Number of RQS bins
Flow I	$3+1$	3D standard normal	6	64	2×64	69	8
Flow II	$504+4$	504-dimensional standard normal	8	378	1×378	11592	8

layers $\vec{\mathcal{I}}_0, \ldots, \vec{\mathcal{I}}_{k-1}$, also proved to be inadequate. Instead, what worked well was a modular, two-step setup in which one NF (which we call "flow I") first learns the distribution of deposited energies conditioned on the input energy $p_1(E_0, E_1, E_2|E_{\mathrm{inc}})$,[6] and then another NF (which we call "flow II") learns the shower shapes conditioned on the energies $p_2(\vec{\mathcal{I}}|E_0, E_1, E_2, E_{\mathrm{inc}})$. In this setup, flow I and flow II are independent of each other, meaning flow I can be replaced by an improved version without the need to retrain flow II, or vice versa.

A. Flow I: Learning the energy depositions per layer

The distribution of layerwise energy deposits, $p_1(E_0, E_1, E_2|E_{\mathrm{inc}})$, is learned by a NF (flow I) whose specifications are in the top row of Table II and which is sketched in Fig. 2. Note that there are two input layers to the MADE block of the same size, one autoregressive one for the three input dimensions and one "normal" one for the conditioning on the total event energy E_{inc} (see Fig. 1 for a schematic view of the MADE block using this input setup). In between the MADE blocks, we randomly permute all dimensions to capture correlations between them better. We do not use dropout or batch normalization. We note that this architecture easily scales to more complicated setups with more calorimeter layers.

To ensure energy conservation, the energy depositions in the calorimeter layers (E_0, E_1, E_2) are transformed to $\vec{u} \in [0,1]^3$, where

$$u_0 = \frac{E_0 + E_1 + E_2}{E_{\mathrm{inc}}}, \qquad u_1 = \frac{E_0}{E_0 + E_1 + E_2},$$
$$u_2 = \frac{E_1}{E_1 + E_2}. \tag{3}$$

In other words, u_0 is the ratio of the deposited to the incident energy, and u_i with $i > 0$ is the ratio of the energy deposited in layer $i-1$ to the net remaining, available energy. This transformation is invertible provided E_{inc} is given. For later convenience, we define the total deposited energy $\hat{E}_{\mathrm{tot}} \equiv E_0 + E_1 + E_2 = \sum \vec{\mathcal{I}}$.

To better learn distributions that are localized toward the boundaries, we transform $\vec{u}$ one more time, to logit space via

$$u_{\mathrm{logit},i} = \log \frac{\tilde{u}_i}{1 - \tilde{u}_i}, \tag{4}$$

where

$$\tilde{u}_i = \alpha + (1 - 2\alpha) u_i \quad \text{and} \quad \alpha = 10^{-6}. \tag{5}$$

The cutoff α ensures that the boundaries 0 and 1 map to finite values of ± 13.82. We therefore choose the tail bound of the RQS to be $B = 14$. Flow I is trained on the features u_{logit}, which when transformed back to (E_0, E_1, E_2), ensures that $\hat{E}_{\mathrm{tot}} \leq E_{\mathrm{inc}}$.

Before being used an a conditional input to flow I, the incident energy is transformed as

$$\log_{10}(E_{\mathrm{inc}}/10\ \mathrm{GeV}) \in [-1, 1]. \tag{6}$$

Working in log space helped the flow to learn the distribution for small energies better. We train flow I by minimizing the negative log-likelihood $-\log p_1(E_0, E_1, E_2|E_{\mathrm{inc}})$ with a batch size of 175 for 75 epochs using the Adam [83] optimizer with an initial learning rate of 10^{-4}. We use a learning rate schedule that applies an additional factor of 0.5 to the learning rate after the epochs [5, 15, 40, 60]. We use the model state of the flow with the lowest test loss in the following.

B. Flow II: Learning the shower shapes

The distribution of shower shapes $p_2(\vec{\mathcal{I}}|E_0, E_1, E_2, E_{\mathrm{inc}})$ is learned by a second NF (flow II) that acts on the full $288 + 144 + 72$-dimensional space of all voxels and is conditioned on E_{inc}, as well as the E_i whose distribution was learned in flow I. See Fig. 3 for a detailed schematic of flow II and the second row of Table II for the specifications of flow II. In between the MADE blocks, we alternate layerwise order inversions and layerwise order

[6]Using a normalizing flow for this low-dimensional density-estimation problem might be overkill, and simpler alternatives such as kernel density estimation [80,81] or mixture density networks [82] might also prove viable.

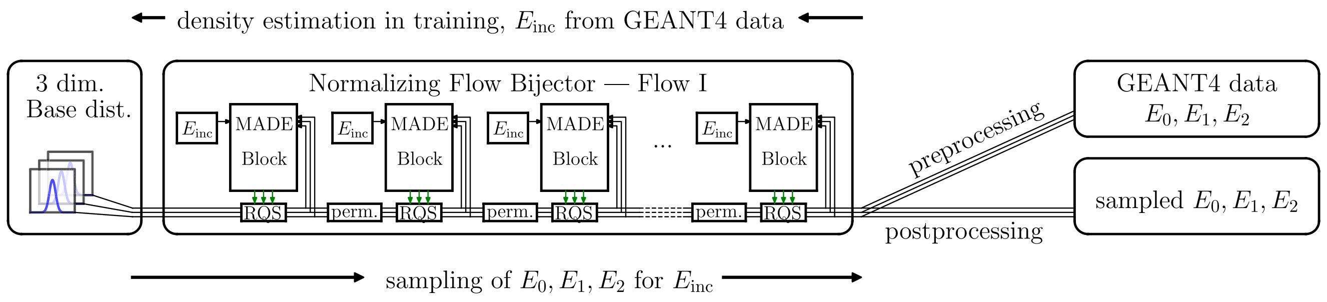
density estimation in training, $E_{\rm inc}$ from GEANT4 data
3 dim. Base dist.
Normalizing Flow Bijector — Flow I
$E_{\rm inc}$
MADE Block
RQS
perm.
GEANT4 data E_0, E_1, E_2
sampled E_0, E_1, E_2
preprocessing
postprocessing
sampling of E_0, E_1, E_2 for $E_{\rm inc}$

FIG. 2. Schematic view of flow I. Random permutations (perm.) are between MADE blocks. Preprocessing (for training and density estimation) and postprocessing (for sampling) are explained in the main text. The green arrows indicate the parameters κ that define the RQS.

permutations of the variables to better capture correlations between them. Layerwise in this context means that variables of calorimeter layer 0 stay in the first 288 positions, variables of calorimeter layer 1 stay in the positions 289–432, and the variables of calorimeter layer 2 stay in the last 72 positions throughout the permutation/inversion. We found that training with a dropout [84,85] probability of 5% enhances the performance.

For the training data, we transform the raw GEANT4 calorimeter images in the following ways.

(1) We found it was essential to first apply uniform random noise in the range [0, 1] keV to all voxels when called for training. The energy distribution of each voxel is sharply peaked at zero, and without the noise regularization, the NF would expend all of its capacity fitting to these sharp (and largely irrelevant) boundaries, while getting wrong the voxels with significant, nonzero energies. To say it another way, without noise regularization, the loss and the gradients would be dominated by the dimmest voxels, and the NF would be prevented from learning how to reproduce the brighter voxels. The noise regularization was key for stabilizing the training and producing a good outcome, especially for the π^+ calorimeter images since they have a large fraction of zero voxels (see sparsity plots).

(2) Next, the voxel energies are normalized so that the voxels in each layer sum to 1 (the energy in each layer is supplied as a conditioning label so it can always be restored). The energy depositions in the different calorimeter layers differ by a large amount, see the $E_i/\hat{E}_{\rm tot}$ histograms of Sec. V B. For example, for e^+, the fraction of deposited energy per layer peaks at about 10% in layer 0, at about 80% in layer 1, and about 0.1% for layer 2. Normalizing each layer to unit intensity helped the flow to learn each layer equally well.

(3) Finally, we transform the (noise-regularized and normalized) voxels to logit space using the transformation of Eqs. (4) and (5). We again use a tail bound of $B = 14$ in the RQS.

These steps define the preprocessed data in Fig. 3.

In Fig. 4, we further illustrate the need for noise regularization, using example plots from training π^+ with and without noise regularization. On the left, we show the training and test loss for the two cases. Without noise regularization, the loss reaches much lower values, suggesting a better fit to data. This, however, cannot be

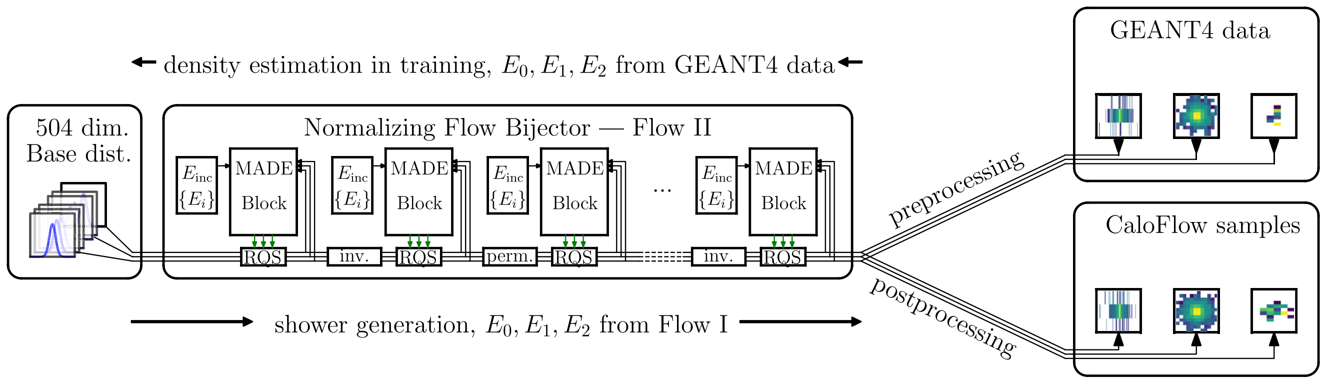
density estimation in training, E_0, E_1, E_2 from GEANT4 data
504 dim. Base dist.
Normalizing Flow Bijector — Flow II
$E_{\rm inc}$ $\{E_i\}$
MADE Block
RQS
inv.
perm.
GEANT4 data
CaloFlow samples
preprocessing
postprocessing
shower generation, E_0, E_1, E_2 from Flow I

FIG. 3. Schematic view of flow II. Inversions (inv.) and random permutations (perm.) are layerwise. Preprocessing (for training and density estimation) and postprocessing (for sampling) are explained in the main text; $\{E_i\}$ is short for the set (E_0, E_1, E_2). The green arrows indicate the parameters κ that define the RQS.

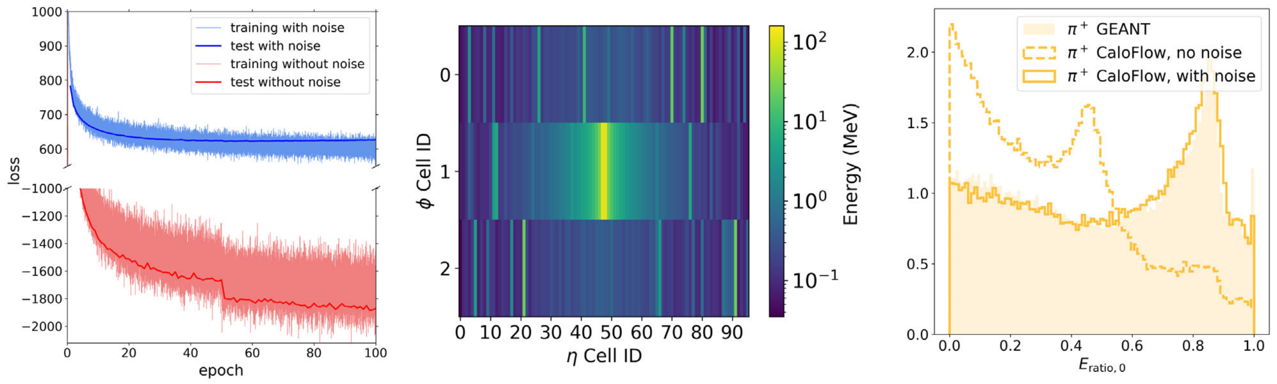

FIG. 4. Influence of noise regularization on training and results. Left: training and test losses. Center: layer 0 average of 100,000 sampled events without noise regularization in training. Right: comparison of $E_{\mathrm{ratio},0}$ when trained with and without noise regularization.

observed in the generated images. Notice also that without the noise regularization the value of the loss is less stable and scatters more. The sudden jump of the loss at epoch 50 comes from multiplying the learning rate by a factor 0.5 as part of our learning rate schedule. In the center, we show the average of layer 0 of 100,000 sampled events. Compared to the plot that uses noise regularization in training (see Fig. 7), we see a less uniform distribution of the voxel energies, coming from the loss being dominated by the 0 voxels. As a result, the $E_{\mathrm{ratio},0}$ plot (Fig. 4, right panel) is also completely off.

The MADE blocks in flow II are conditioned on E_{inc} and E_i, where E_{inc} is encoded as in Eq. (6) and E_i are the layer energies encoded as

$$\log_{10}\left((E_i + 1\ \mathrm{keV})/100\ \mathrm{GeV}\right) + 2 \in [-6, 3]. \quad (7)$$

As before, working in log space helped the flow to learn the distribution for small energies better.

The training is then done by minimizing $L = -\log p_2(\vec{\mathcal{I}}|E_0, E_1, E_2, E_{\mathrm{inc}})$ using 100 epochs of the Adam [83] optimizer with initial learning rate 10^{-4} that is halved after 50 epochs and a batch size of 175. The values of E_i are taken directly from the data. We select the epoch with the lowest test loss for the subsequent sample generation.

C. Sampling from CaloFlow

In generation, we first sample $E_{i=0,1,2}$ from flow I given an input energy E_{inc}. We then use flow II to generate the shower shapes based on the conditionals E_{inc} and E_i. The raw showers are first transformed back to energy space via the sigmoid function. Then, the individual layers are rescaled to have the correct energy E_i.

Note that, while flow II is trained on layerwise unit-normalized shower shapes, this constraint is not imposed as part of the flow (i.e., the generated images still live in the full 504-dimensional space). A well-trained NF will produce images with normalization of approximately one, but there will be some scatter in the result. Therefore, when we generate from flow II, we choose to further renormalize the generated images so they have exact unit normalization in each layer, before rescaling by E_i. This is necessary to enforce the right energies per layer. Alternatively, one could consider a normalizing flow with manifold learning [86] to achieve energy conservation.

As a final step, we set all voxels with energy depositions below 10 keV to zero to ensure the correct sparsity of the shower images. All these steps are what we call post-processing in Fig. 3.

Our specific handling of sparse images (adding noise for training and setting a threshold after generation) is necessary, as the bijective nature of normalizing flows does not allow a mapping of random variables to an absolute 0 for a large range of input values.

V. RESULTS

In this section, we present the results of sampling with CaloFlow, with detailed comparisons to CaloGan and GEANT4. We organize our results in order of more qualitative to more quantitative. We start with comparisons of average images and nearest-neighbor images between CaloFlow and GEANT4. Then we compare histograms of relevant physical quantities from CaloFlow, CaloGan, and GEANT4. Finally, we compare results of training classifiers on the generated (CaloFlow and CaloGan) vs real (GEANT4) images. In our histograms, we exhibit the same quantities that were already used in [10], as well as additional quantities to better assess the quality of the generated shower shapes and voxel level information (i.e., to better probe the performance of flow II).

Since these additional histograms require the full information of all events and not just a marginalized subset, we generate our own CaloGan sample by training CaloGan based on the code [68] and the default hyperparameters (which match the ones defined in [10]), except for the number of epochs in training, where we used 100 instead of 50. We use a different TensorFlow [87] version (v1.14.0 instead of v1.1.0), but the same version of KERAS [88] (v2.0.3) that was used for [10]. As in [10], we did not perform any detailed model selection, but we looked at the histograms of samples based on different epochs and used the generator state that agreed best with the GEANT4 data. These were epochs 80, 50, and 100 for e^+, γ, and π^+, respectively. Our training yielded qualitatively similar results in all histograms compared to the ones shown in [10], except for π^+, where our run seems to model the energy peaks slightly better. In the following, all histograms are based on 100,000 samples: the GEANT4 set that we based the training of CaloFlow and CaloGan on, 100,000 samples we sampled from our trained CaloGan, and 100,000 samples we sampled from CaloFlow.

A. Qualitative comparisons (average and individual images)

We start the comparison by looking at the average of the 100,000 events we use for visualization in Figs. 5–7. We see excellent agreement between CaloFlow and GEANT4 average images; the averages of CaloFlow are smooth and very close to their GEANT4 counterparts. Meanwhile CaloGan has a few voxels with an average deposition of zero—which is only possible if those voxels are always zero. This is a sign of mode collapse, since the GAN did not learn to cover the full available phase space.

Another common method for detecting mode collapse consists of selecting elements of the GEANT4 set at random and looking for their nearest neighbors in the set of generated CaloFlow samples. If mode collapse occurred in training, we would find that some GEANT4 images have very close nearest neighbors, while others have very distant ones. Figures 8–10 show five selected events from the GEANT4 set at incident energies $E_{\mathrm{inc}} = 5$, 10, 20, 50, and 95 GeV and their Euclidean nearest neighbor in a CaloFlow dataset of 2000 samples at the same energies. We define nearest neighbors across all layers simultaneously (nearest in 504-dimensional voxel space), in contrast to the layerwise definition of [10], with the expectation that this provides a more stringent test of mode collapse. Overall, we observe nearest neighbors that are close to the target events in all cases, suggesting that no mode collapse occurred.

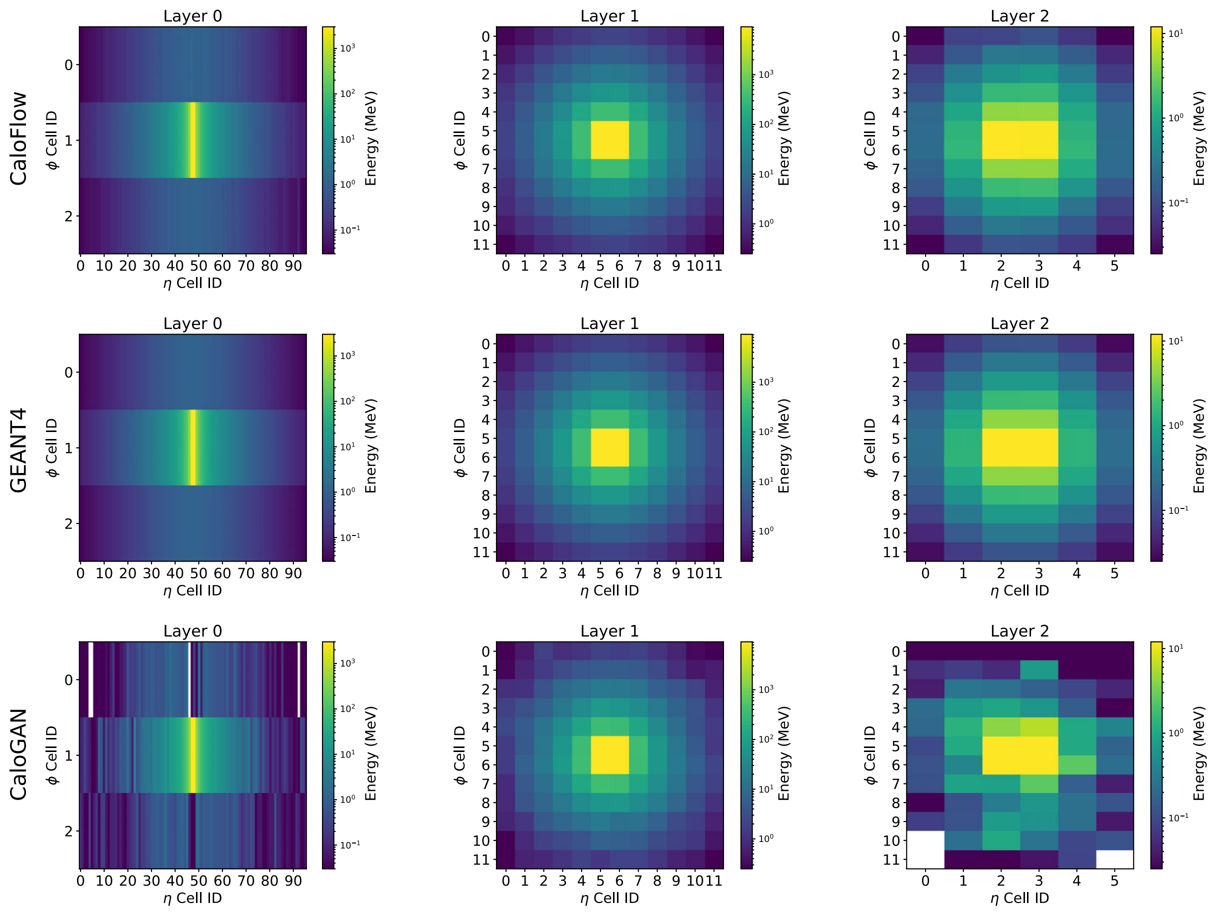

FIG. 5. Average shower shapes for e^+. Columns are calorimeter layers 0–2, top row shows CaloFlow, center row GEANT4, and bottom row CaloGan.

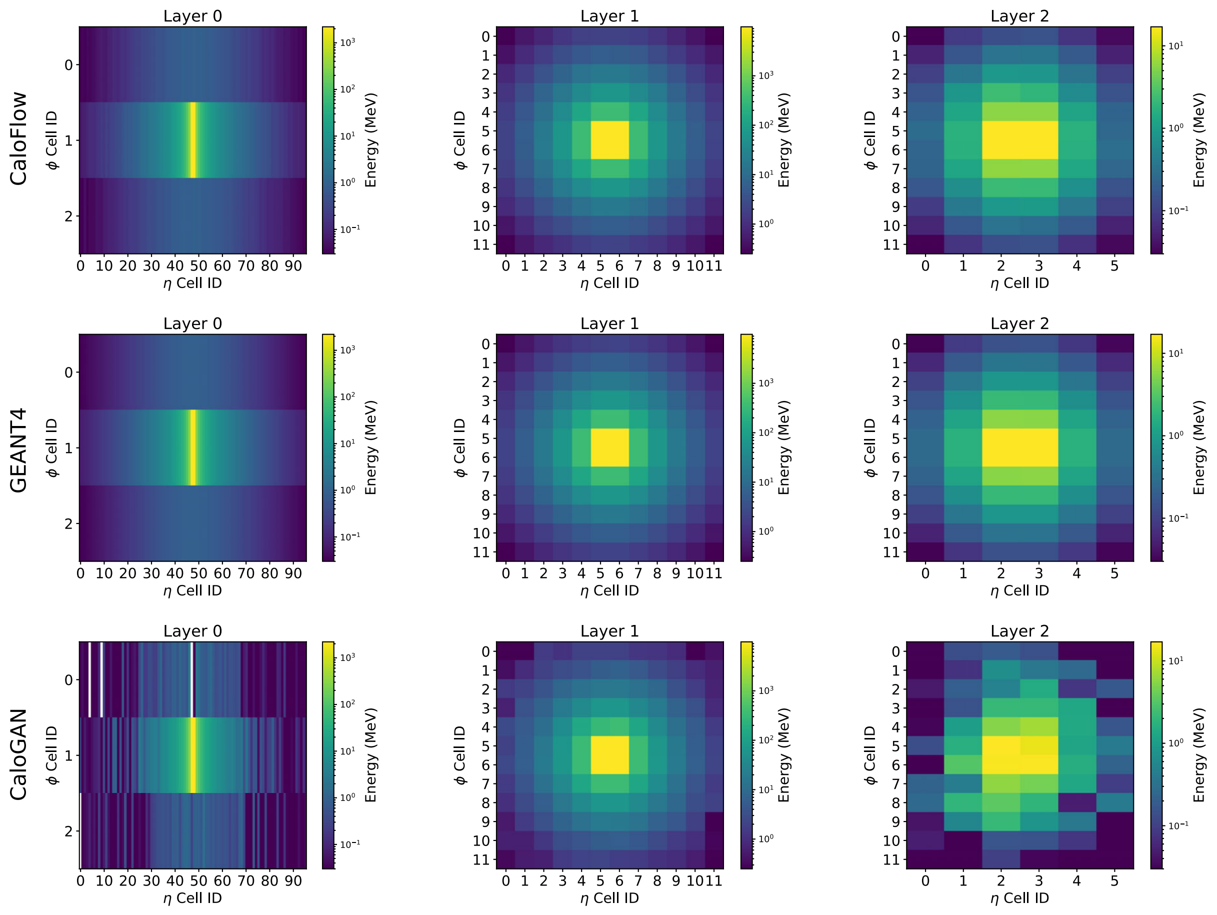

FIG. 6. Average shower shapes for γ. Columns are calorimeter layers 0–2, top row shows CaloFlow, center row GEANT4, and bottom row CaloGan.

B. Flow I histograms

We now have a look at some distributions, starting with the ones that are only sensitive to the results of flow I. In Figs. 11–13 we show histograms for e^+, γ, and π^+, respectively. All of these were also considered in [10].

In general, we observe that the CaloFlow samples are much closer to the GEANT4 samples than the CaloGan samples are, and the overall agreement between CaloFlow and GEANT4 is quite impressive in an absolute sense. Note that all these features are learned by minimizing the log-likelihood, not by adding specific terms to the loss. In more detail:

(i) In the top rows of Figs. 11–13, we show the energy depositions in each calorimeter layer $E_k = \sum \mathcal{I}_k$, as well as the total deposited energy in all three layers $\hat{E}_{\text{tot}} = \sum_{k=0}^{2} E_k$. We see that CaloFlow models the energy distributions extremely well. For π^+, we see that CaloFlow models the peak a lot better than the CaloGan. In $\hat{E}_{\text{tot}}$ we see the advantage of our two-flow approach, as we have, by construction, perfect energy conservation ($\hat{E}_{\text{tot}} \leq E_{\text{inc}}$).

(ii) The second rows of Figs. 11–13 show the ratio of the layer energies E_k to the total deposited energy $\hat{E}_{\text{tot}}$. CaloFlow models these really well for all three particles, and especially the performance increase in $E_2/\hat{E}_{\text{tot}}$ compared to CaloGan is remarkable.

(iii) The third rows of Figs. 11–13 show the layer (depth)-weighted total energy, $l_d = \sum_{k=0}^{2} kE_k$, on the left, the layer-weighted energy normalized to the total energy, $s_d = l_d/\hat{E}_{\text{tot}}$, in the center, and the standard deviation of s_d, called shower depth width σ_{s_d}, on the right. The quantity s_d was called "shower depth" in [10]. In l_d we see CaloFlow better maps out the low-energy region compared to CaloGan. Notice also how well CaloFlow learns the sharp feature in σ_{s_d}.

C. Flow II histograms

We now turn to distributions that are also sensitive to flow II. Figures 14–16 start by showing histograms for e^+, γ, and π^+, respectively, that are sensitive to the events at the voxel level.

(i) In the top two rows, we show the distribution of the brightest two voxels in each layer, normalized to the total energy deposition in that calorimeter layer. We observe an improvement over CaloGan for the π^+ distributions, but also small peaks at low values in layer 2, for all particles.

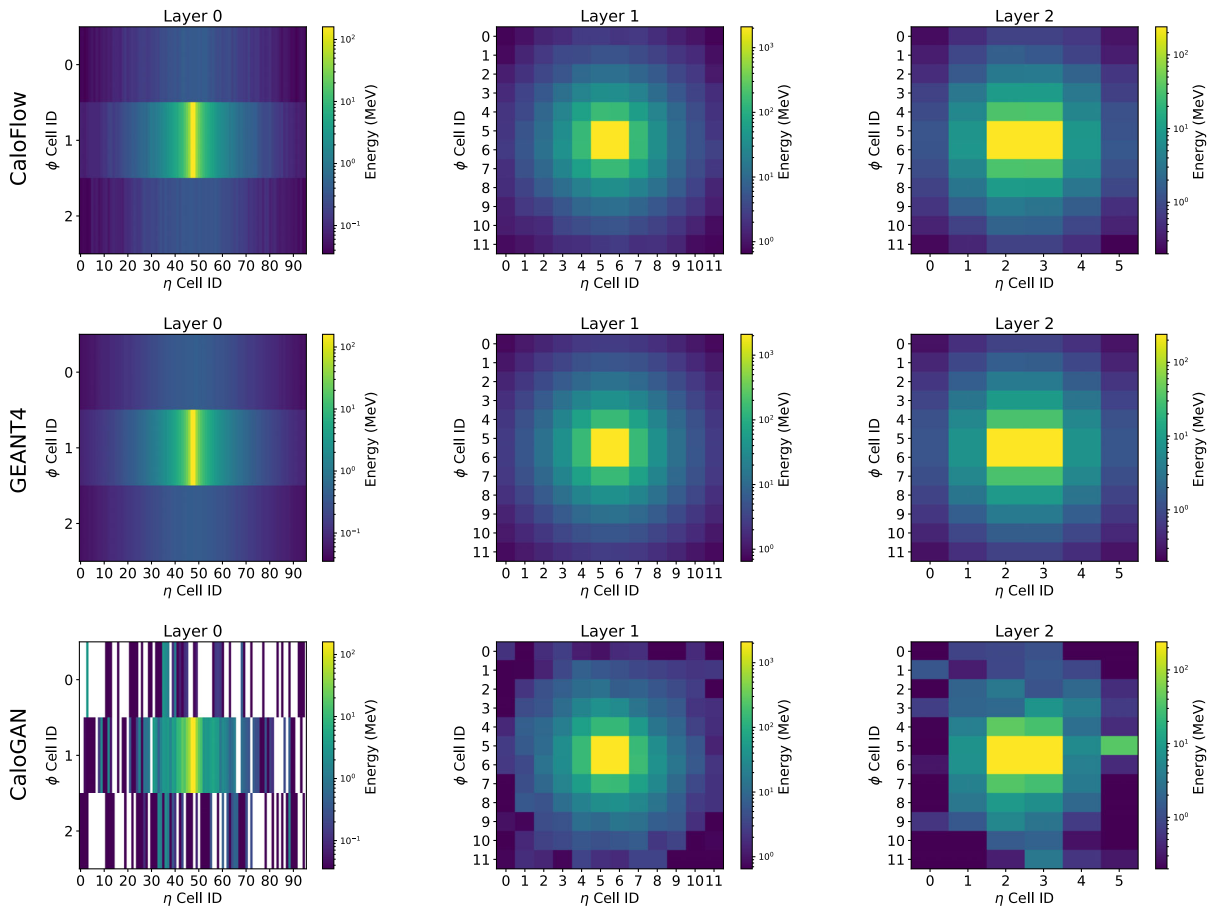

FIG. 7. Average shower shapes for π^+. Columns are calorimeter layers 0–2, top row shows CaloFlow, center row GEANT4, and bottom row CaloGan.

(ii) In the third row, we show the histograms for $E_{\mathrm{ratio},k}$, which is the difference of the brightest and second brightest voxel of layer k, divided by their sum,

$$E_{\mathrm{ratio},k} = \frac{\mathcal{I}_{k,(1)} - \mathcal{I}_{k,(2)}}{\mathcal{I}_{k,(1)} + \mathcal{I}_{k,(2)}}. \tag{8}$$

While $E_{\mathrm{ratio},2}$ for e^+ and γ in CaloFlow is a bit larger than the GEANT4 reference around the peaks of the distributions, we see a large improvement in all three $E_{\mathrm{ratio},k}$ for π^+ over CaloGan.

(iii) In the last row, we show the sparsity of the events in the calorimeter layers. The sparsity is defined as the ratio of the number of voxels with nonzero deposition to the total number of voxels in layer k. Here, CaloFlow improves a lot over CaloGan for all particles and layers.

The second and third items (sparsity and $E_{\mathrm{ratio},k}$) were also considered in [10]. We have added the histograms of the brightest and second brightest voxels as additional probes of the quality of flow II.

We investigate distributions that are sensitive to shower shapes in Figs. 17–19, for e^+, γ, and π^+, respectively.

(i) The top rows show the centroids in ϕ and η direction. The centroids in $\eta(\phi)$ are defined via H (F), which are the locations of the voxel centers in units of millimeters, as well as

$$\langle\eta_k\rangle = \frac{\mathcal{I}_k \odot H}{E_k} \quad \text{and} \quad \langle\phi_k\rangle = \frac{\mathcal{I}_k \odot F}{E_k}. \tag{9}$$

Here, $\odot$ denotes the elementwise multiplication and sum ($a \odot b = \sum_i a_i b_i$, with i the index of the voxels in a layer), and k is the index of the calorimeter layer. The histograms for CaloFlow are all centered, as expected, for incoming particles that are centered and perpendicularly incident. For the CaloGan, however, we observe an asymmetry in the centroids, another hint to possible mode collapse of the GAN.

(ii) In the last row, we show the standard deviation of the η centroid. This was also considered in [10], where it was called layer lateral width. If H is again the location of the voxel center in η direction, σ_k is defined as

$$\sigma_k = \sqrt{\frac{\mathcal{I}_k \odot H^2}{E_k} - \left(\frac{\mathcal{I}_k \odot H}{E_k}\right)^2}. \tag{10}$$

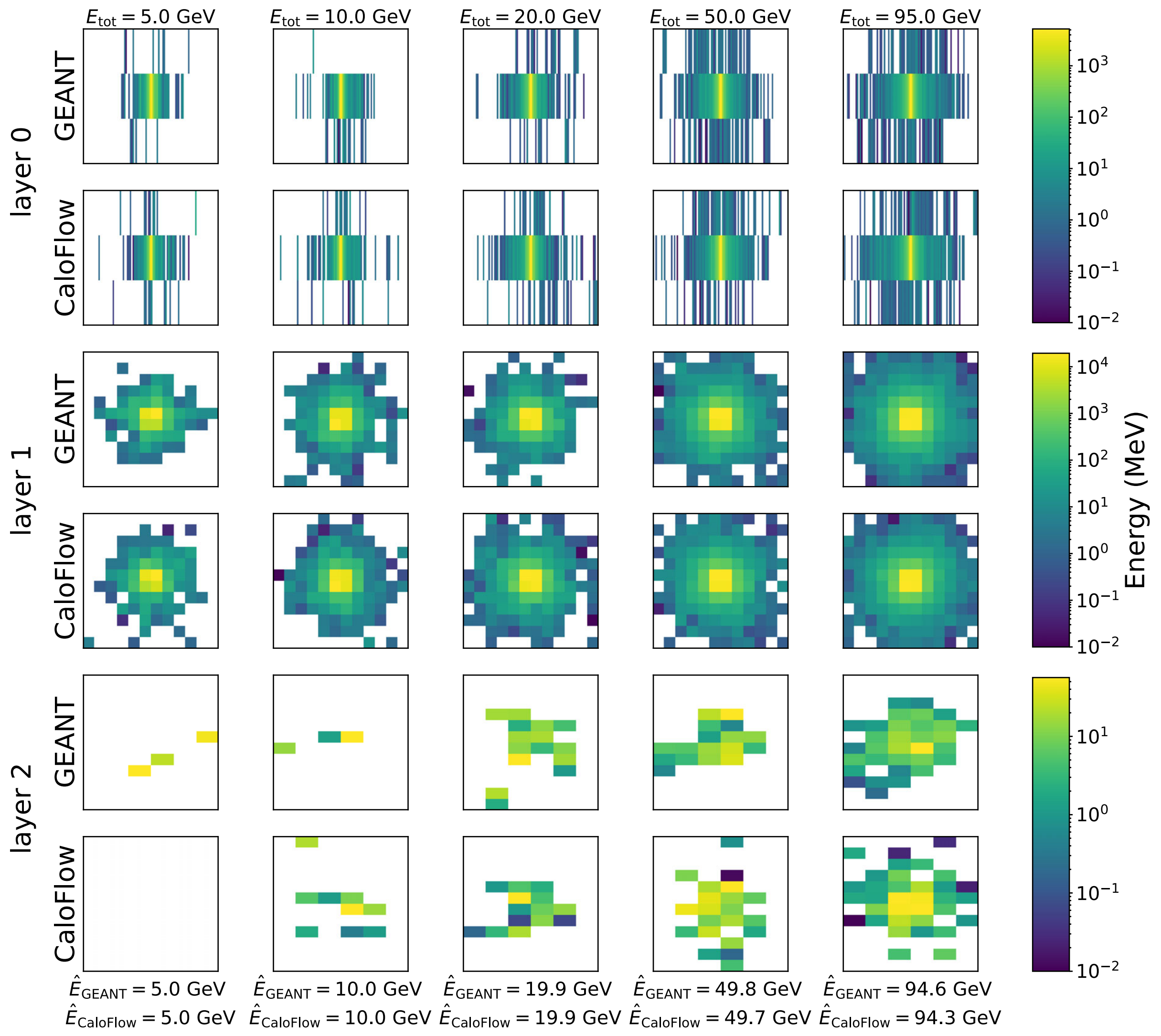

FIG. 8. Five randomly selected e^+ events of GEANT4 and their nearest neighbors in the CaloFlow samples.

For e^+ and γ, CaloFlow follows the GEANT4 distributions well, improving over CaloGan. The improvement is even bigger for the π^+ showers.

D. Classifier metrics

In much of the GAN literature (see, e.g., [10]), a common metric is to train classifiers to distinguish between different categories of data (e.g., e^+ vs π^+) and to see if there is any difference in classifier performance when real data and generated data are interchanged. For example, one might train a classifier on e^+ vs π^+ GEANT4 images and compare this to a classifier trained on e^+ vs π^+ GAN images. If the classifier trained on real images performs similar to the classifier trained on generated images, then this is evidence that the generated images are approximating the real images well. One can repeat this test for different combinations of real and generated data.

The ultimate test of whether $p_{\mathrm{generated}}(x) = p_{\mathrm{data}}(x)$ would be the optimal binary classifier between real and generated images of the *same* type.[7] If the generated and true probability densities are equal, and the classifier is optimal, then according to the Neyman-Pearson lemma it will be no better than random guessing. Compared to evaluating various histograms, this approach has the potential advantage to look at the full 504-dimensional voxel space and all the correlations between them instead of just a multitude of one-dimensional projections.

[7]A common misconception is that this classifier is the same one as in the GAN architecture. Whereas the GAN classifier is only optimized for a few minibatches or epochs in order to give reasonable gradients, the classifiers used here are trained to convergence to approximate the ultimate, Neyman-Pearson classifier. Also, the gradients of the GAN classifier feed back into the training of the GAN generator, whereas here the classifier metric is only trained after the flow generative model is fully trained and optimized.

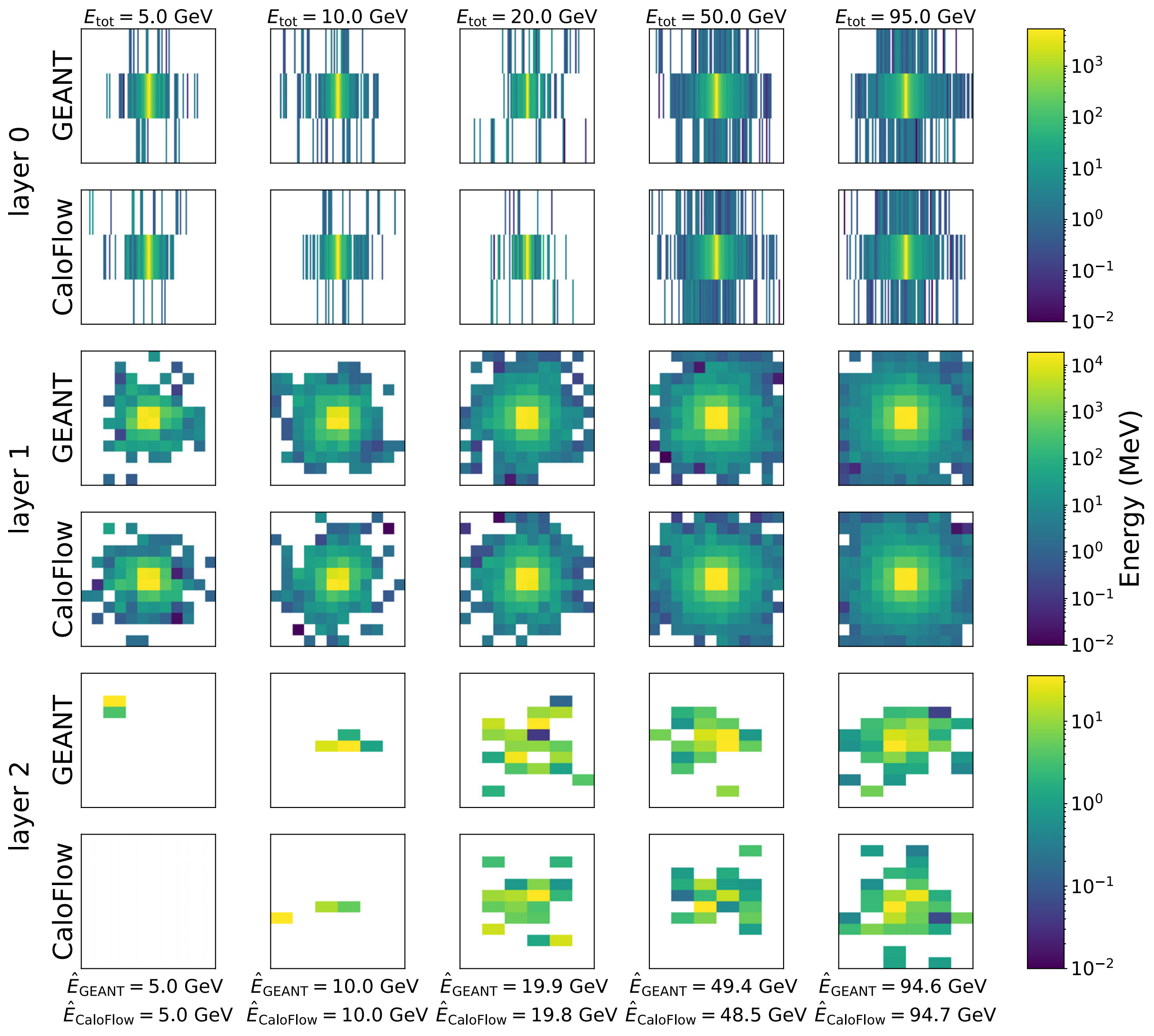

FIG. 9. Five randomly selected γ events of GEANT4 and their nearest neighbors in the CaloFlow samples.

Although this binary classifier test between real and generated samples has been proposed before as a way to evaluate generative model performance [71], one rarely (never?) sees this more direct classifier-based metric used in the GAN literature. The reason appears to be that GAN-generated images are never good enough to fool such a classifier; they always have a "tell" that leads such a classifier to nearly 100% accuracy [41,71].[8]

In this section, we will demonstrate that CaloFlow-generated images are sufficiently high fidelity to fool classifiers trained to distinguish real from generated images of the same type. We will investigate several different versions of the classifier metric based on varying the model architecture and data representation (preprocessing): a deep neural network (DNN) and a convolutional neural network (CNN) on low-level features (LLFs, meaning voxels) and a DNN on high-level features (HLFs, the ones we investigated in Sec. V). While voxel-based classifiers should give the most sensitive metric of shower quality, they may detect differences in irrelevant features or be suboptimal due to noisy or uninformative features. This is why we also include a classifier based on HLFs. We expect this should give a realistic picture of the difference between the samples in the most physically relevant space.

For the voxel-based classifiers, we consider two different approaches to data preprocessing: (1) using the calorimeter samples as they were generated and (2) normalizing the

[8]In the deep neural networks using the classification for tuning and reweighting (DCTRGAN) method of [41], a classifier between real and generated images was proposed—not as a judge of simulation quality, but as a reweighting function that could be applied to generated images to make them better resemble the data. However, a classifier trained on calorimeter images generated from the state-of-the-art BIB-AE architecture [15,16] had to be handicapped (by training it for a single epoch) so that it could not achieve perfect accuracy (which would have rendered the reweighting function useless).

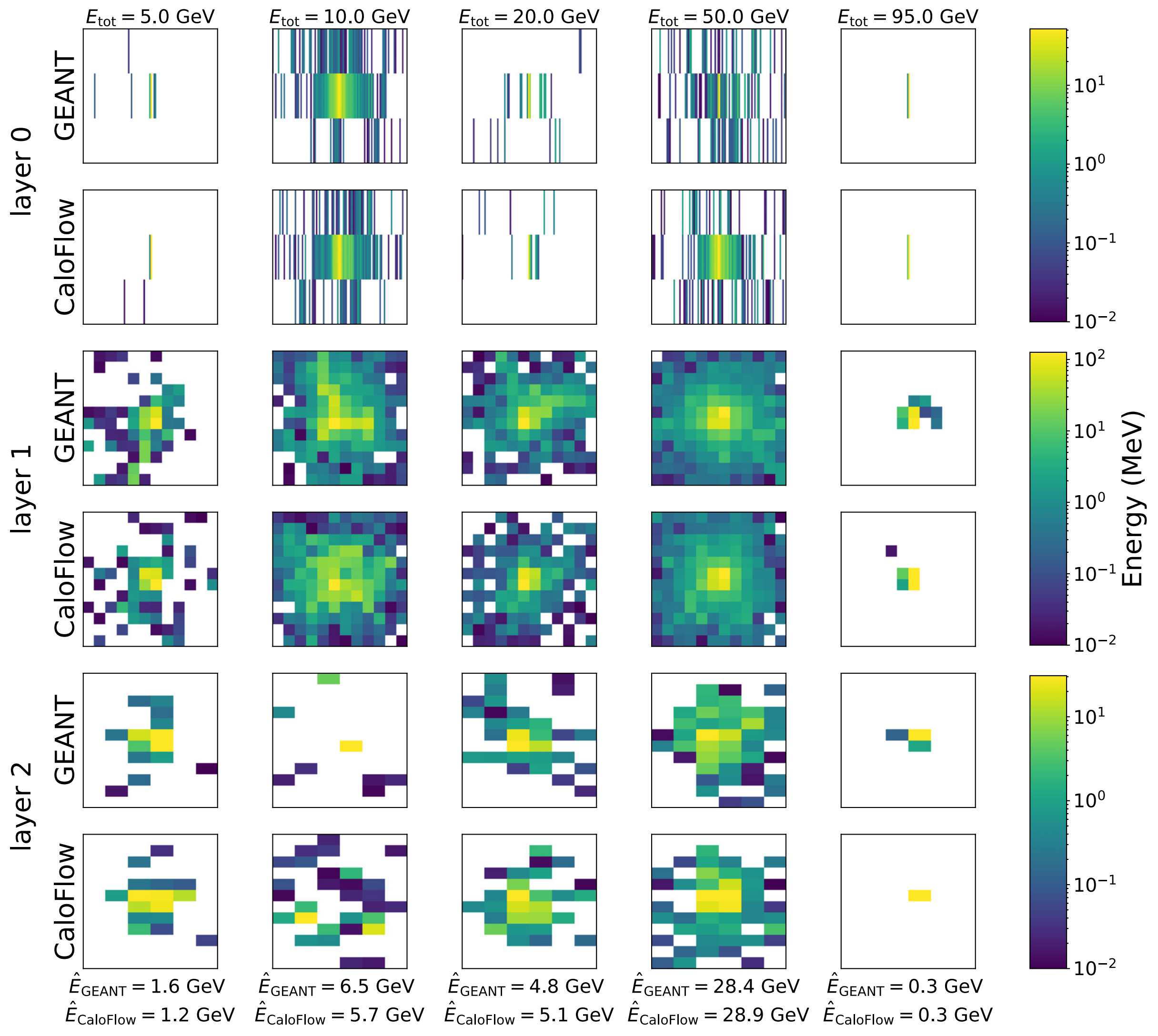

FIG. 10. Five randomly selected π^+ events of GEANT4 and their nearest neighbors in the CaloFlow samples.

voxels such that they sum to 1 in each calorimeter layer. The latter enhances the features in the calorimeter layers that have less energy deposition in them, making it easier for the classifiers to find differences in the datasets. However, these datasets do not correspond to true showers anymore, so the results of the classifiers are biased.[9]

[9]In addition to these results, we investigated the influence of working in logit space by transforming voxel values using Eq. (4) and then dividing it by 10 before feeding it into the classifiers. As this preprocessing step artificially enhances features of dim voxels that likely do not contribute much to the physics analysis (as can be seen by comparing to the high-level classifier results), we do not think that these results necessarily reflect physically meaningful differences between the CaloFlow (or CaloGan) and GEANT4 datasets. For completeness, we report the results of the classifiers with data in logit space in Table V in Appendix C. We also checked if applying a threshold cut of 10 keV to GEANT4 and CaloGan data (the same as in the last step of generating CaloFlow data) has an influence and we found none.

Since the classifiers are not optimal, being limited by finite training data and model capacity, their performance depends on the model architecture and data preprocessing. Strictly speaking, the Neyman-Pearson classifier is then best approximated by whichever model architecture and preprocessing yields the best separation of the datasets. However, we prefer to take a more holistic view of all the various classifier metrics as providing different windows into the generative model quality in the full high-dimensional phase space. For details about the classifier architectures and calibration procedures, see Appendix B.

The output of the classifiers is shown in Table III, where we give the result in terms of two aggregate metrics. The first, AUC, is the area under the receiver operating characteristic (ROC) curve of the classifier. A maximally confused classifier gives $\mathrm{AUC} = 0.5$, whereas a perfect classifier gives $\mathrm{AUC} = 1.0$. The second metric is the Jensen-Shannon divergence (JSD) between the two distributions, which we deduce from the binary cross entropy

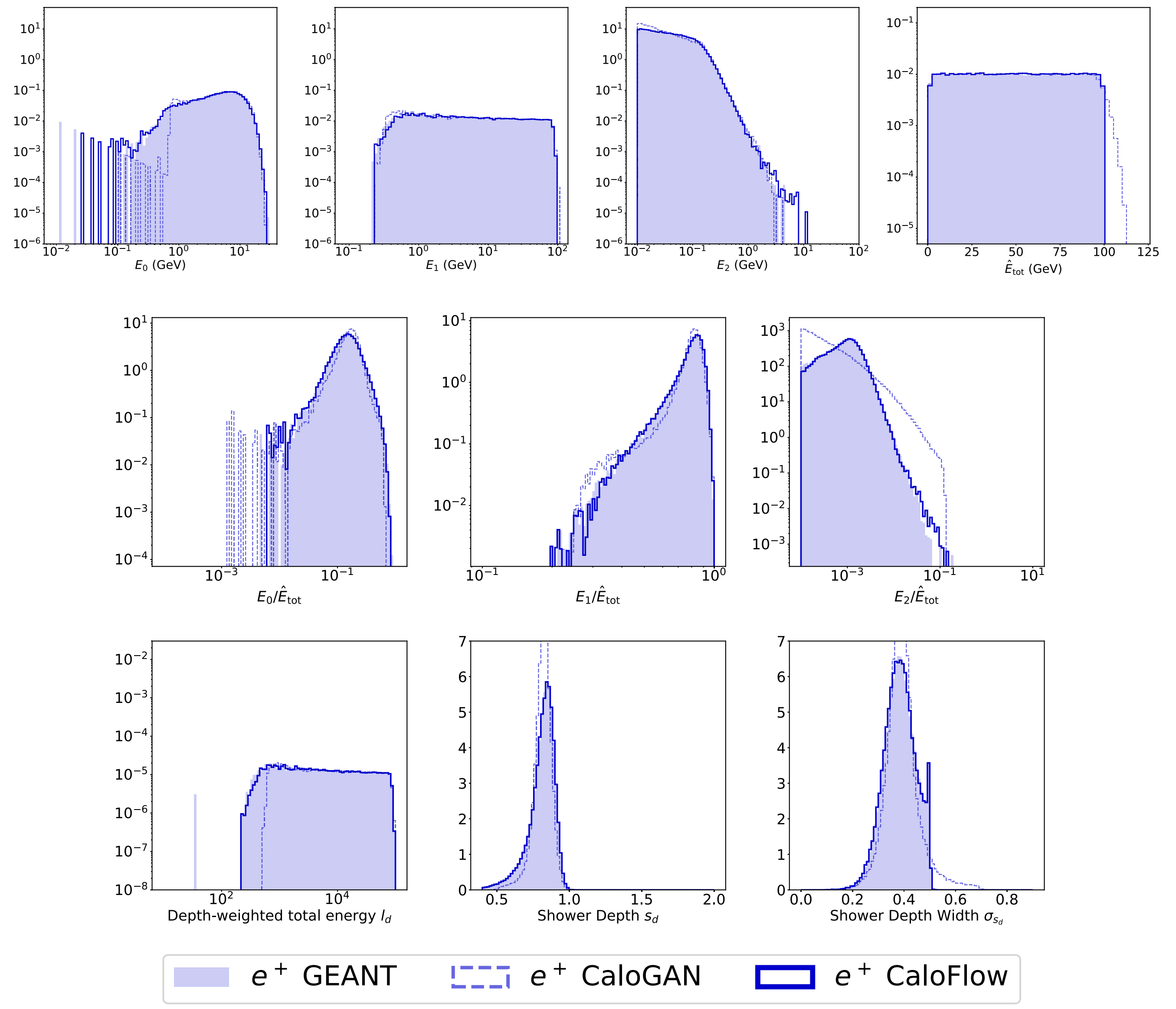

FIG. 11. Distributions that are sensitive to flow I for e^+. Top row: energy deposition per layer and total energy deposition. Center row: layer energy normalized to total energy deposition. Bottom row: weighted energy depositions, see text for detailed definitions.

of the test set at the minimum [89,90]. The JSD is 0 if the two distributions are identical and 1 if they are disjoint.

In all these tests, we see that a classifier can distinguish GEANT4 and CaloGan samples with 100% accuracy, whereas it has a much harder time to distinguish between GEANT4 and CaloFlow, indicating that CaloFlow produced a more realistic dataset. We think that this is, in part, due to CaloGan not sampling the full space, as can be seen from average layer depositions that show voxels with 0 value, as well as centroid correlations that peak off center. All these features act as tells for the classifier.

In general, we observe that the CNN classifier scores are lower (closer to random) than the DNN scores. This suggests that the shower images are not the best representation of the data, perhaps because the individual showers are extremely sparse and generally free of any meaningful substructure. Also, all showers start from the center of the detector, not producing many translation invariant shapes across the entire surface. Further, we observe that normalizing showers produces higher classifier scores; i.e., it makes the generated showers easier to distinguish from the reference showers. This suggests that the generated showers differ from the reference ones largely in the lower-energy voxels and normalizing the showers likely amplifies the role of these in the classifier decision. Whether this is physically relevant for any downstream task is observable dependent. Finally, it is interesting that the high-level classifier produces a lower score for positrons

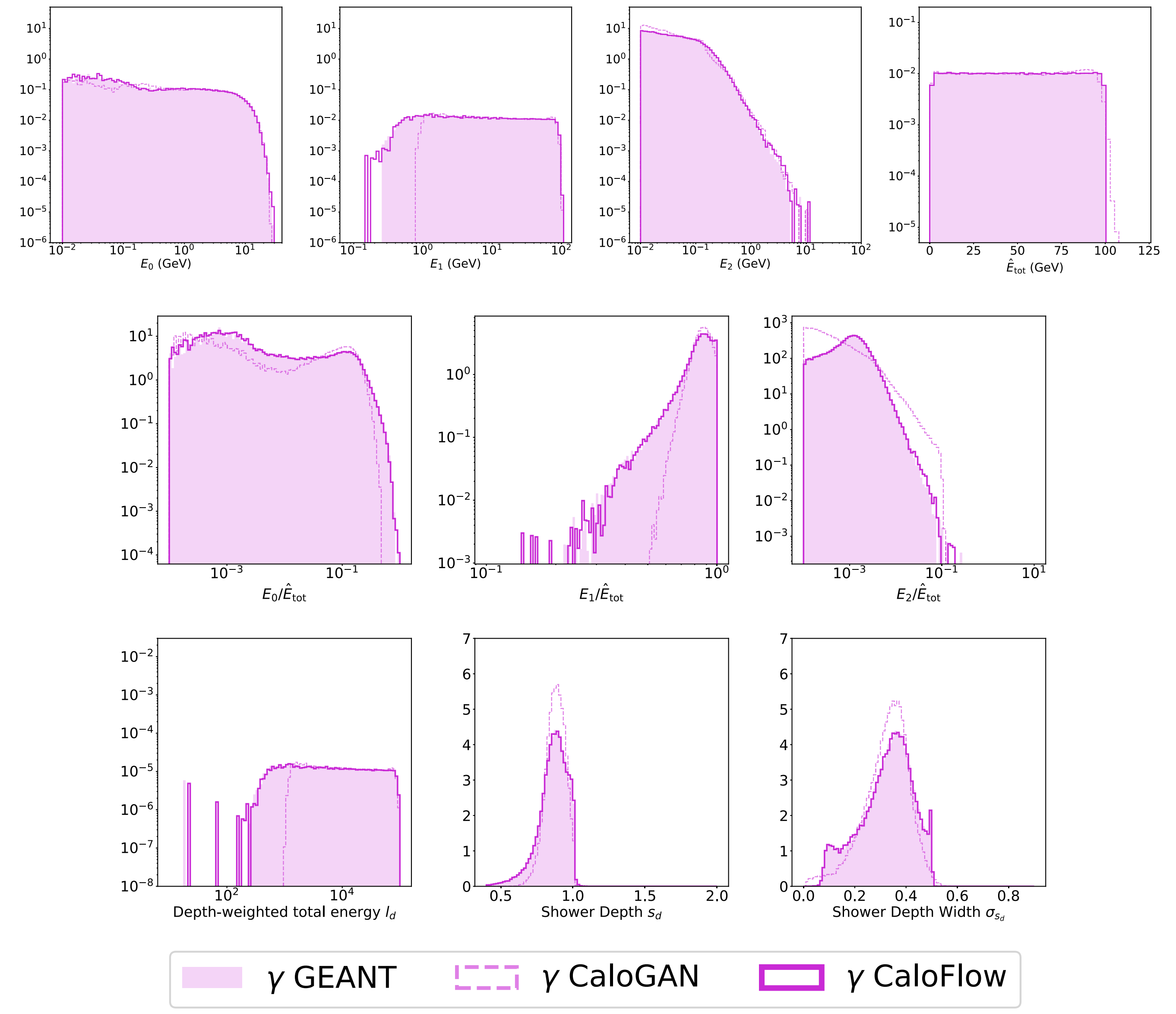

FIG. 12. Distributions that are sensitive to flow I for γ. Top row: energy deposition per layer and total energy deposition. Center row: layer energy normalized to total energy deposition. Bottom row: weighted energy depositions, see text for detailed definitions.

and photons but not for pions. Since the high-level classifier uses much less information than the low-level classifier, we would expect that, if both were optimal, the former always has a strictly lower score than the latter; i.e., using more information can only enhance the classification power. This is consistent with the situation for positrons and photons, but for pions it goes in the opposite direction. We can only attribute this to the suboptimality of the LLF classifiers for pions, whether due to finite training data or model capacity. We suspect the LLF classifiers for pions have a more difficult time learning the (real) differences between CaloFlow and GEANT4 showers because the pion showers have much higher shower-to-shower variance than photon and positron showers. The latter, being electromagnetic, come only from bremsstrahlung, pair production, ionization, Compton- and photoeffect; while the former are hadronic and exhibit a much higher variety due to the various types of QCD interactions with matter.

E. Timing benchmarks

Having generated our own CaloGan sample for this analysis, we are able to perform a head-to-head comparison of the time required for shower generation between CaloGan and CaloFlow. Training times on a TITAN V graphics processing unit (GPU) are about 210 min for CaloGan, about 22 min for CaloFlow flow I, and 82 min for CaloFlow flow II. In Table IV we show the time per shower in

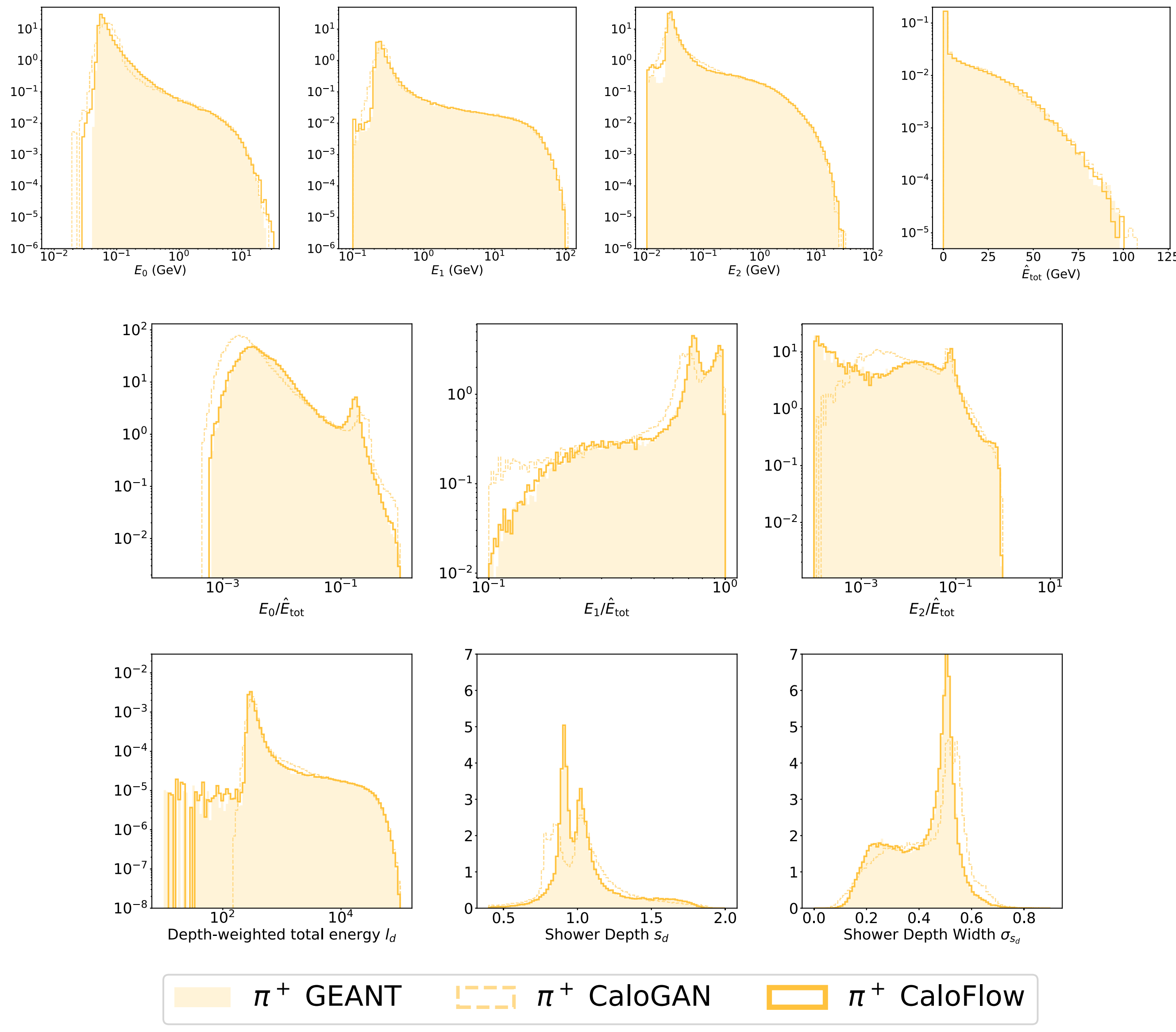

FIG. 13. Distributions that are sensitive to flow I for π^+. Top row: energy deposition per layer and total energy deposition. Center row: layer energy normalized to total energy deposition. Bottom row: weighted energy depositions, see text for detailed definitions.

milliseconds for different batch sizes, also with a TITAN V GPU. For the best CaloGan case, we see a saturation around 0.07 ms, compared to 36 ms for CaloFlow, yielding a relative factor of about 500. Since CaloGan and CaloFlow have roughly the same size—CaloGan has 29.726280×10^6 trainable parameters and flow I and flow II of CaloFlow have a total of 37.914414×10^6 trainable parameters—we believe that the essential difference in generation time must be due to the MAF architecture that needs to loop over the full 504-dimensional voxel space in generation. We are currently investigating [76] the possibility of switching over to a MAF-IAF pair as in Parallel WaveNet [77], which could yield a speedup of CaloFlow of the same order of magnitude as the dimensionality of the data, bringing it in line with CaloGan's speed.

Another source of the difference between CaloGan and CaloFlow generation speeds is indicated by the two columns listed under CaloGan in Table IV. For CaloGan, we report two different timings: one for generating a single batch of size "batch size" and one for generating a total of 100,000 events. The difference in those two timings arises from KERAS-TensorFlow building the graph for the prediction at the beginning of the function call and then reusing it for subsequent batches. CaloFlow is using the pytorch [91] based package nflows [73] and batches are handled using a simple for-loop. We therefore observe no timing difference for CaloFlow when requesting more samples than the batch size. So it is possible that we could further speed up the per-event generation time of CaloFlow by implementing it in KERAS-TensorFlow and requesting more events than the batch size.

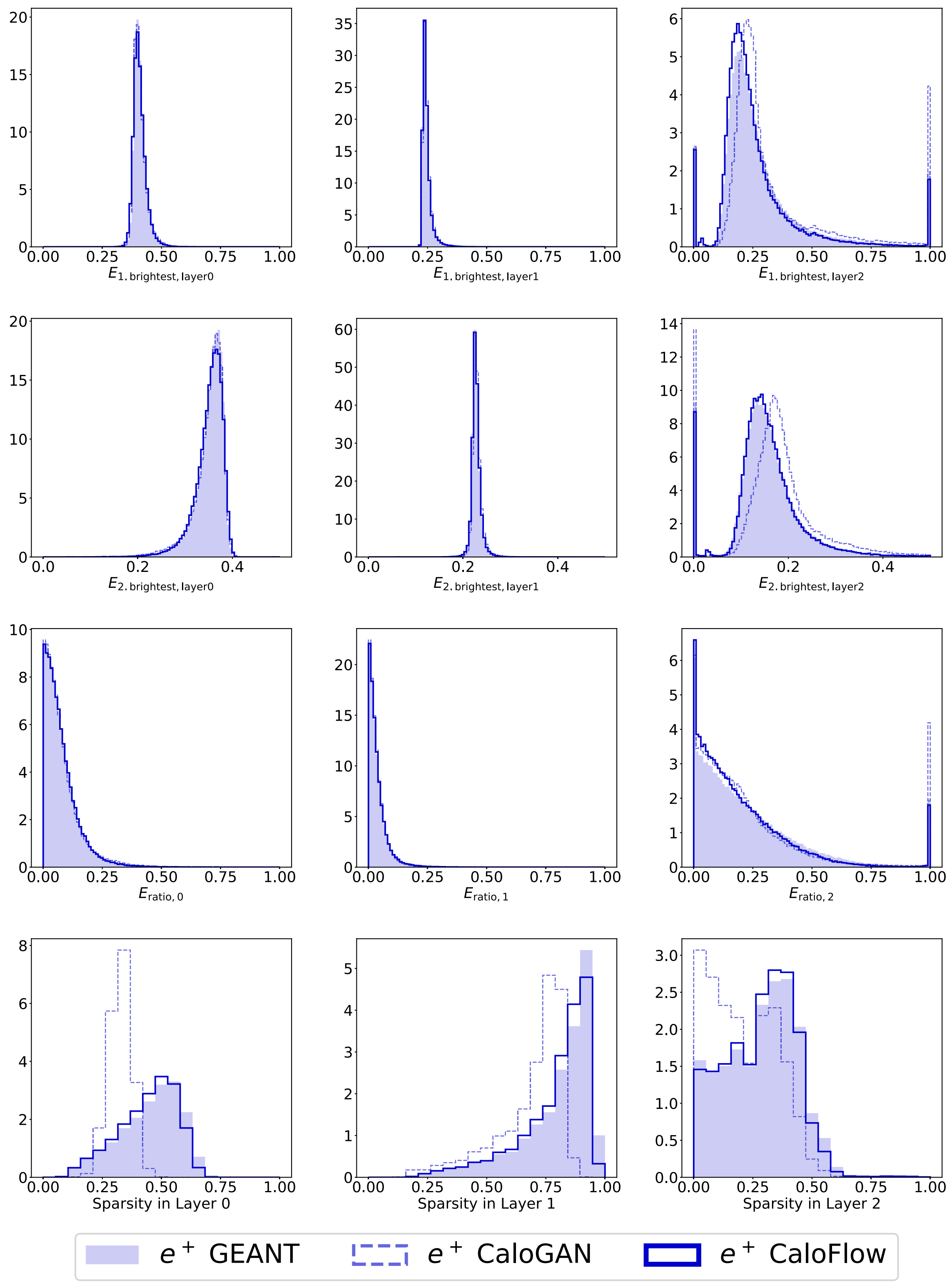

FIG. 14. Distributions that are sensitive to flow II for e^+. Top row: energy of brightest voxel compared to the layer energy. Second row: energy of second brightest voxel compared to the layer energy. Third row: difference of brightest and second brightest voxel, normalized to their sum. Last row: sparsity of the showers, see text for detailed definitions.

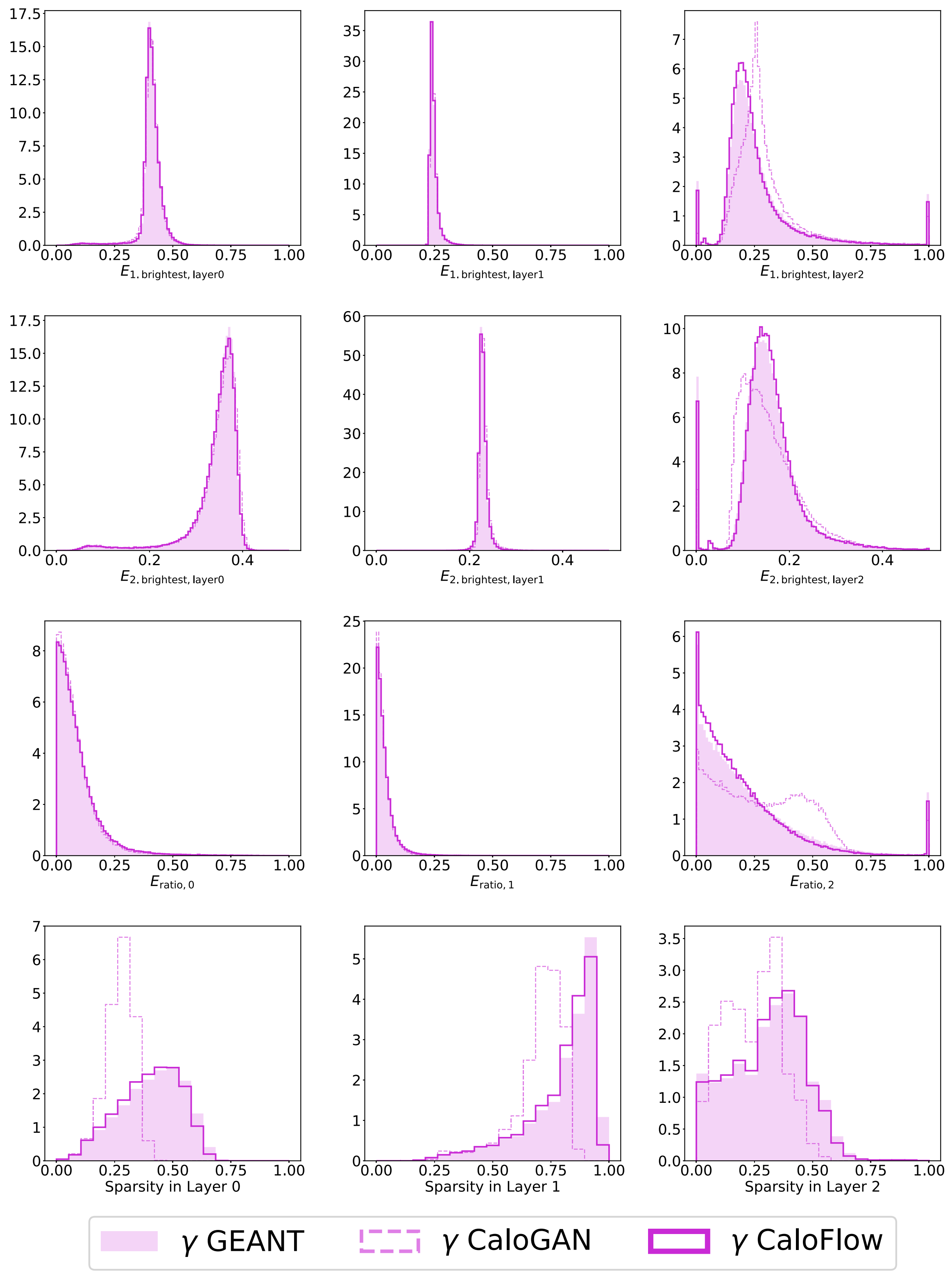

FIG. 15. Distributions that are sensitive to flow II for γ^+. Top row: energy of brightest voxel compared to the layer energy. Second row: energy of second brightest voxel compared to the layer energy. Third row: difference of brightest and second brightest voxel, normalized to their sum. Last row: sparsity of the showers, see text for detailed definitions.

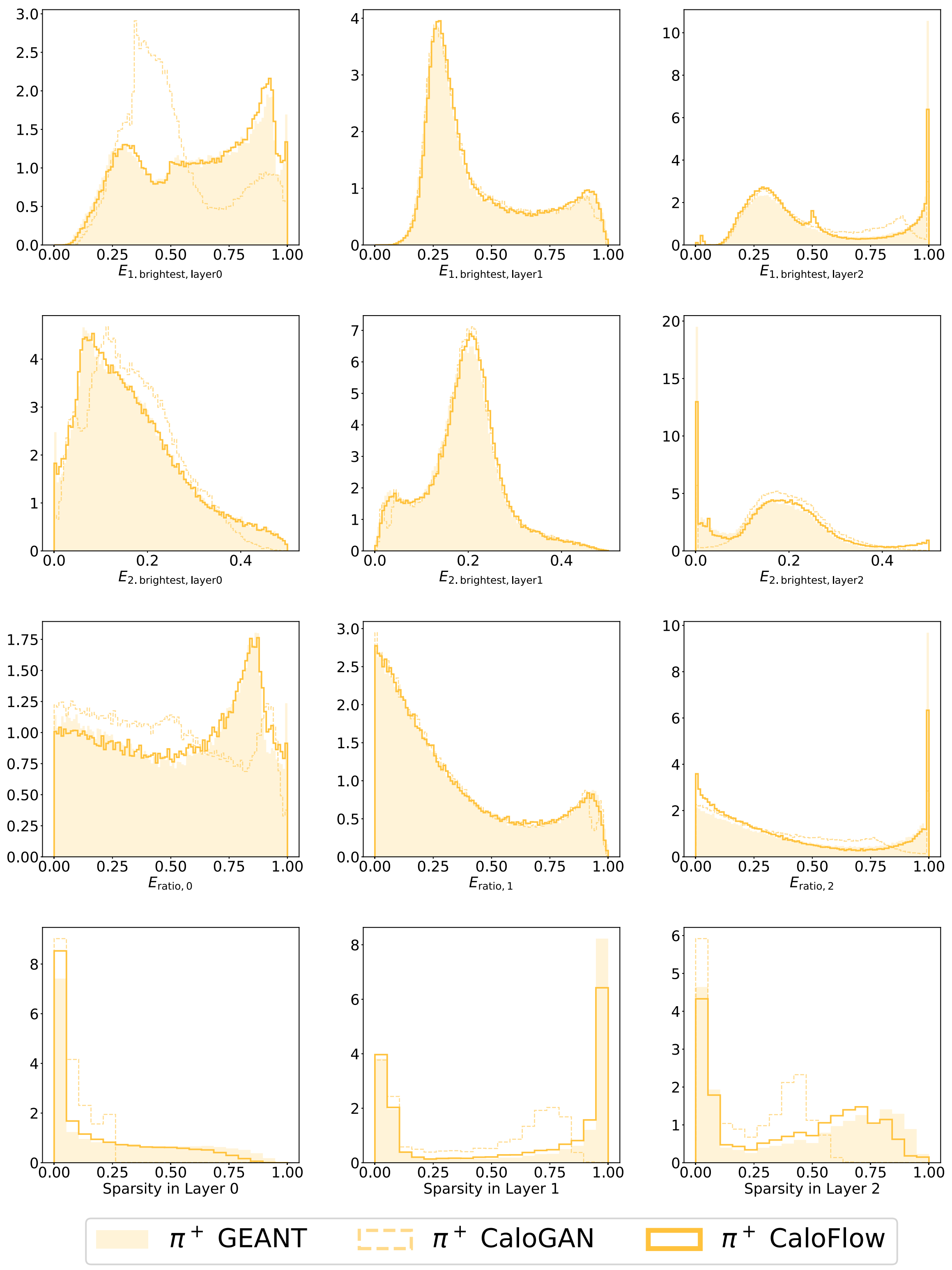

FIG. 16. Distributions that are sensitive to flow II for π^+. Top row: energy of brightest voxel compared to the layer energy. Second row: energy of second brightest voxel compared to the layer energy. Third row: difference of brightest and second brightest voxel, normalized to their sum. Last row: sparsity of the showers, see text for detailed definitions.

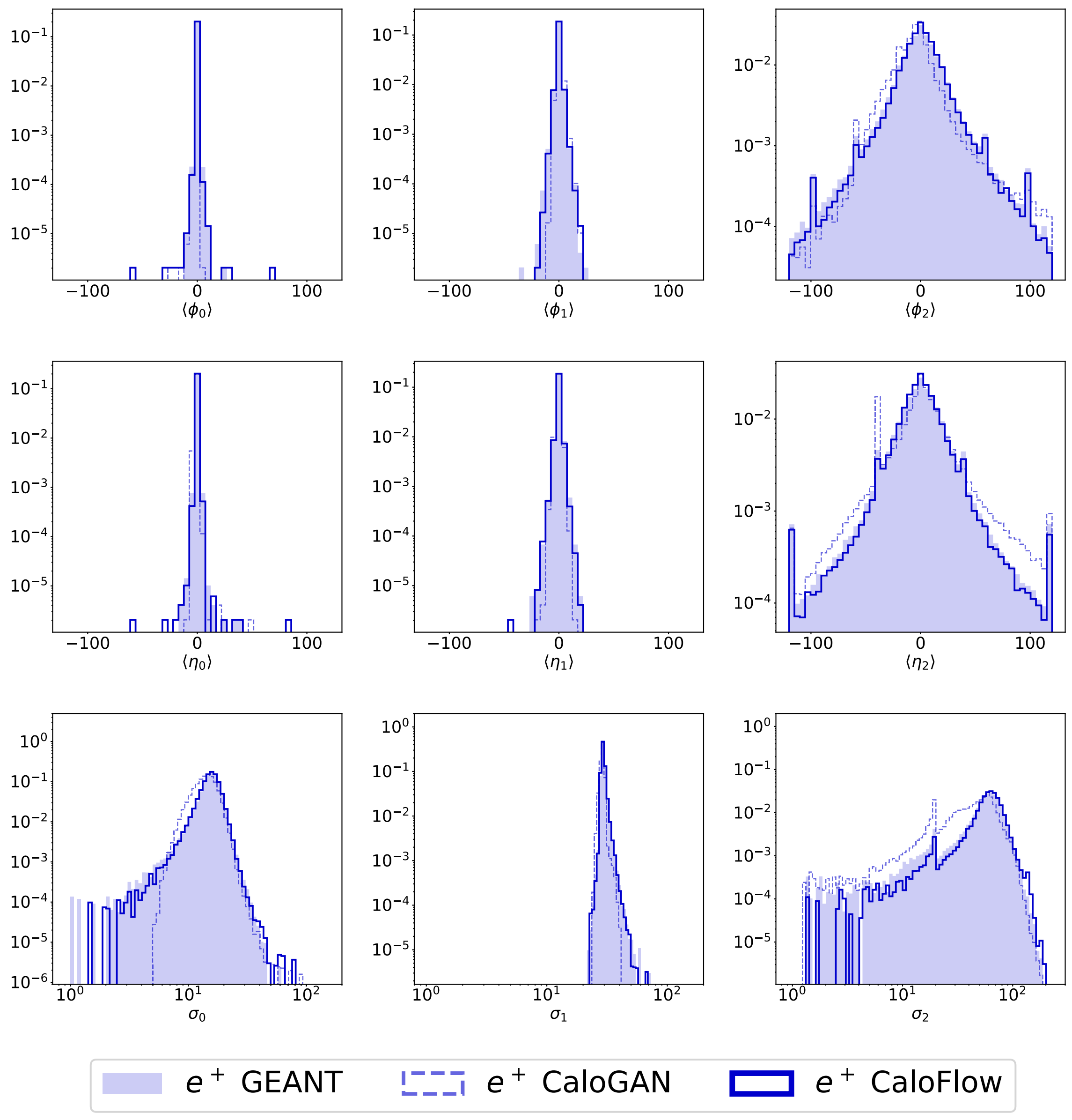

FIG. 17. Further distributions that are sensitive to flow II for e^+, as learned by flow II. Top and center row show the location of the deposition centroid in ϕ and η direction; the bottom row shows the standard deviation of the η centroid.

VI. CONCLUSIONS

In this work, we have demonstrated, for the first time, that generative modeling with normalizing flows is capable of reproducing the sparse and very high-dimensional dataset that corresponds to GEANT4 calorimeter showers. With both qualitative comparisons of individual and average images, as well as more quantitative comparisons of distributions, we see that CaloFlow reproduces GEANT4 with extremely high fidelity. Even more impressively, we demonstrated that a binary classifier trained on CaloFlow vs GEANT4 fails to achieve anything close to 100% accuracy, indicating that the CaloFlow images approximate the distribution of GEANT4 images to a very high precision. This is the first time any generative model has passed this stringent test.

For our proof of concept, we have relied heavily on the setup of the CaloGan. This is a simplified three-layer calorimeter, but still much higher dimension that previous applications of normalizing flows in HEP. Further work is needed for realistic setups, such as the actual ATLAS/CMS detector or future high-granularity detectors for HL-LHC and ILD, as well as taking into account the incoming angle of the particle and other real-world complications.

For our normalizing flow, we combined the RQS transformation with the MADE block to maximize the expressivity of the density estimator. It would be interesting to

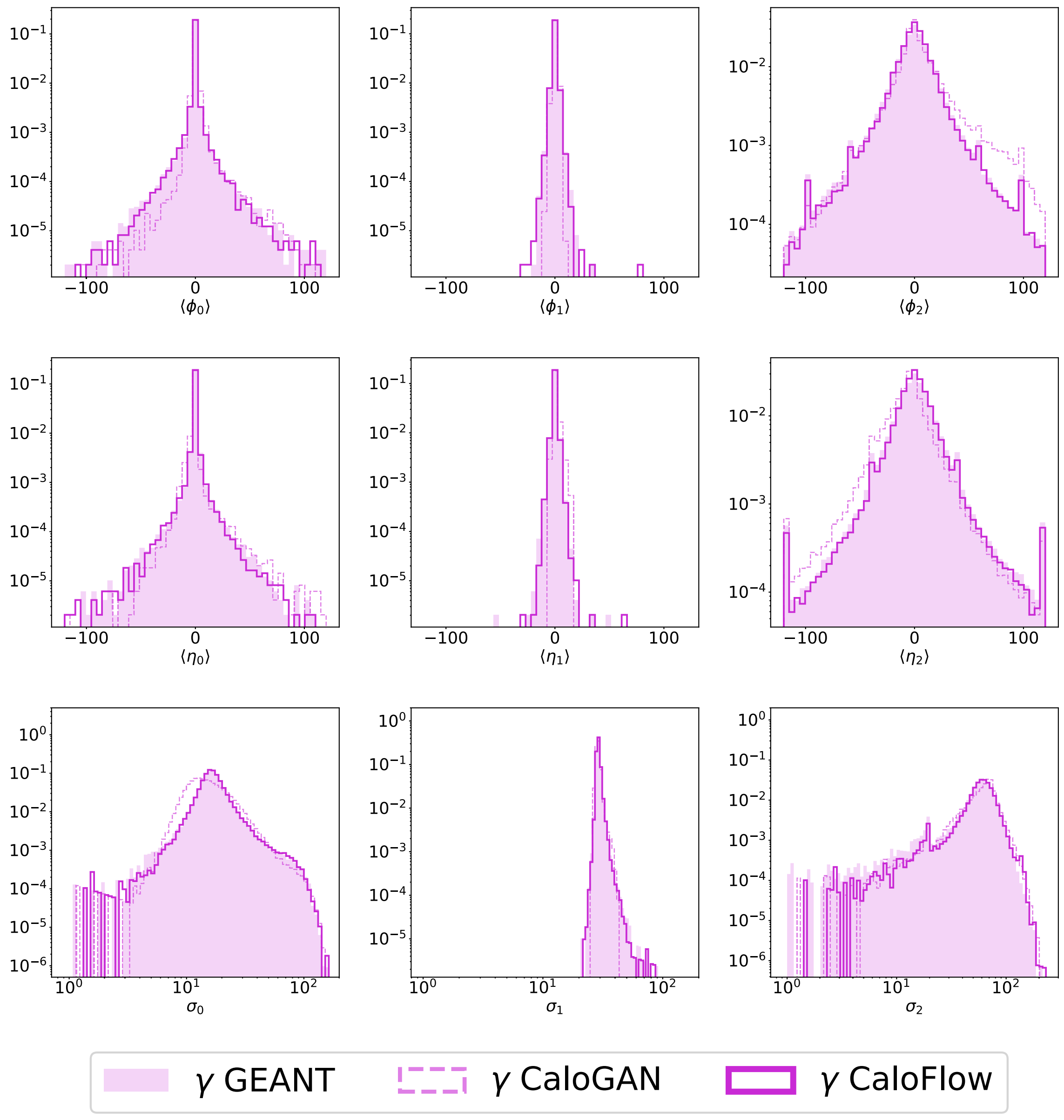

FIG. 18. Further distributions that are sensitive to flow II for γ, as learned by flow II. Top and center row show the location of the deposition centroid in ϕ and η direction; the bottom row shows the standard deviation of the η centroid.

explore other density estimators, which could have their individual pros and cons. For instance, the MAF, while more expressive than coupling layers, is also considerably slower at sampling. The MAF does have a cousin called the IAF [75] for which the opposite is true (fast to sample, slow to density estimate), but the implementation in NFLOWS was not memory efficient enough to be able to train on the full 504-dimensional dataset. We are currently investigating whether alternative implementations of the IAF could overcome these memory limitations, or whether it is possible to jointly train a MAF-IAF pair as in Parallel WaveNet [77]. This could result in another considerable speedup to CaloFlow; we expect roughly scaling like the dimensionality of the dataset, so a factor of $\sim$500 in this case. Interestingly, according to the timing benchmarks in Sec. V E, such a speedup would lead to CaloFlow being as fast as the GAN.

To better and more explicitly enforce energy conversation, we invented a two-step flow, where in the first step we generate the energies deposited in each layer, and in the second step we learn the shower shapes of each layer. This improved the performance of CaloFlow greatly and should also have applications to other generative modeling approaches, e.g., GANs. Another interesting future direction would be to train flows that explicitly conserve energy by learning the energy-conserving manifold directly [86].

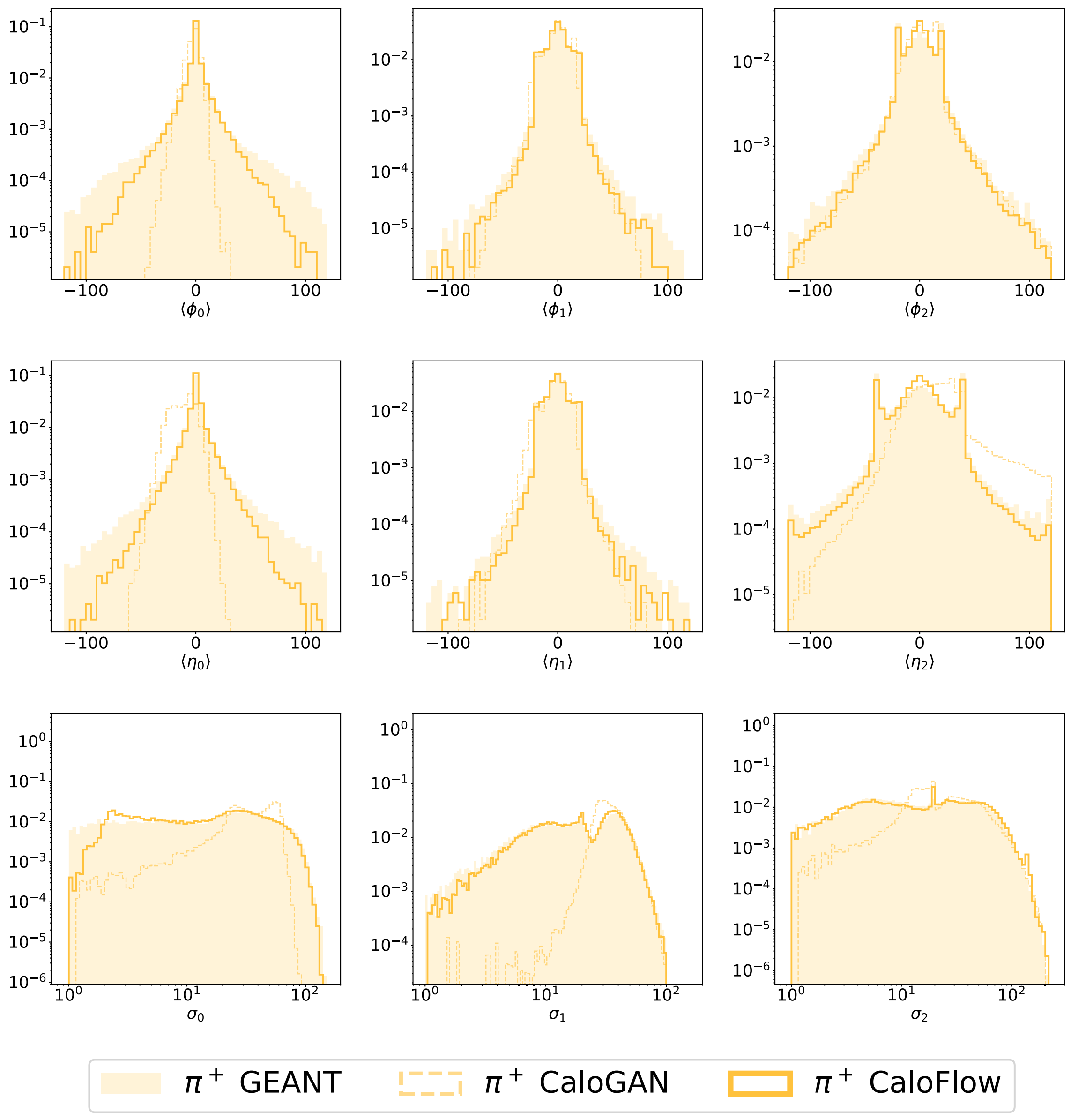

FIG. 19. Further distributions that are sensitive to flow II for π^+, as learned by flow II. Top and center row show the location of the deposition centroid in ϕ and η direction; the bottom row shows the standard deviation of the η centroid.

This would be an alternative to the two-step flow configuration considered in this work. Additional refinements of the latent space, as they were discussed recently in [92], provide an additional avenue for improvements. Finally, given the success in the ultimate classifier test demonstrated here, it is likely that the DCTRGAN method of [41]—which essentially takes the output of said classifier and uses it to reweight the generated events—could be fruitfully applied to further refine CaloFlow starting from the low-level calorimeter images.

We believe normalizing flows offer a powerful and fresh approach to generative modeling at the LHC. Compared to GANs, they have many advantages, including explicit, tractable likelihoods, more stable and convergent training, and principled model selection (based on the negative log-likelihood). While more work comparing CaloFlow to more state-of-the-art GANs (e.g., WGAN-GP) needs to be done, we believe this proof of concept (especially the ultimate classifier test) is a very promising start.

In this work, we used the NumPy 1.16.4 [93], MATPLOTLIB 3.1.0 [94], PANDAS 0.24.2 [95], SKLEARN 0.21.2 [96], H5PY 2.9.0 [97], PyTorch 1.7.1 [91], and NFLOWS 0.14 [73] software packages.

Our code is available at [98].

TABLE III. AUC and JSD metrics for the classification of GEANT4 vs CaloGan and CaloFlow showers. Classifiers were trained on each particle type (e^+, γ, π^+) separately. The results of two classifiers based on DNN and CNN architectures are shown; for details on the classifier architectures and training, see Appendix B. All entries show mean and standard deviation of ten runs and are rounded to three digits. We see that the classifiers can distinguish GEANT4 from CaloGan showers with nearly perfect accuracy (AUC = 1.0) in all cases, whereas GEANT4 vs CaloFlow showers are much more difficult for the classifiers to tell apart.

		DNN		CNN	
AUC/JSD		vs CaloGan	vs CaloFlow	vs CaloGan	vs CaloFlow
e^+	Unnormalized	1.000(0)/0.995(1)	0.859(10)/0.365(14)	0.982(8)/0.770(54)	0.510(5)/0.002(1)
	Normalized	1.000(0)/0.997(0)	0.870(2)/0.378(5)	1.000(0)/0.982(2)	0.806(77)/0.269(128)
	High level	1.000(0)/0.987(1)	0.795(1)/0.229(3)		
γ	Unnormalized	1.000(0)/0.998(0)	0.756(48)/0.174(68)	0.990(3)/0.820(30)	0.511(6)/0.001(1)
	Normalized	1.000(0)/0.994(1)	0.796(2)/0.216(4)	1.000(0)/0.991(1)	0.724(52)/0.130(69)
	High level	1.000(0)/0.994(1)	0.727(2)/0.131(3)		
π^+	Unnormalized	1.000(0)/0.993(0)	0.649(3)/0.060(2)	0.984(15)/0.813(119)	0.519(6)/0.001(1)
	Normalized	1.000(0)/0.997(1)	0.755(3)/0.153(3)	1.000(0)/0.998(1)	0.867(6)/0.344(15)
	High level	1.000(0)/0.997(0)	0.888(1)/0.401(4)		

TABLE IV. Generation time of a single calorimeter shower in milliseconds. Times were obtained on a TITAN V GPU. GEANT4 needs 1772 ms per shower [10]. Note that CaloGan is based on KERAS-TensorFlow and CaloFlow is based on PYTORCH. All times are in ms.

	CaloGan		
Batch size	Batch size requested	100,000 requested	CaloFlow
10	455	2.2	835
100	45.5	0.3	96.1
1000	4.6	0.08	41.4
5000	1.0	0.07	36.2
10000	0.5	0.07	36.2

ACKNOWLEDGMENTS

We are grateful to G. Kasieczka, B. Nachman, and T. Plehn for helpful discussions and comments on the draft. We also thank B. Nachman for his assistance with reproducing the CaloGan results using their GitHub. This work was supported by DOE Award No. DOE-SC0010008.

APPENDIX A: MORE ON NORMALIZING FLOWS

The essential trick that NFs use to parametrize invertible mappings with tractable Jacobians is the autoregressive transformation.

We start with a 1D invertible transformation $z = f(x; \kappa_a, \ldots)$, where $\kappa_a = \kappa_0, \kappa_1, \ldots$ are the parameters of the transformation. Popular examples of such transformations include the affine transformation $f(x; \vec{\kappa}) = \kappa_0 x + \kappa_1$ and the rational quadratic spline family of transformations (as described in the main text). How to generalize this to multiple dimensions while maintaining invertibility? Making the coefficients functions of x would enable the forward mapping, but not the inverse mapping in general. What simultaneously enables both directions and makes the Jacobian tractable is to assume the κ_a are functions of only the previous coordinates, i.e.,

$$z_i = f_i(x_i; \kappa_{ia}(x_1, \ldots, x_{i-1})), \tag{A1}$$

where $i = 1, \ldots, d$ runs over the dimensions of the feature space. Now $x_1 \to z_1$ can be trivially inverted (κ_{1a} are just constants), and the remainder of the inverse transformations can be built up recursively. Furthermore, the Jacobian of the transformation is lower triangular, so the determinant can be calculated as a simple product of the diagonals (d operations), instead of the $\mathcal{O}(d^3)$ operations required for a general $d \times d$ matrix.

The transformation parameters can be parametrized with neural networks, and the most general autoregressive structure can be enforced by multiplying all weights with appropriate binary masks. The result is called a MADE block [69]. To ensure that all correlations between the variables are learned properly, the order of the variables is permuted in between different blocks.

A special case of the general autoregressive transformation is the so-called "coupling layer," first introduced as part of the "real nonvolume preserving" flow in [53]. In the coupling layer approach, one first splits the entire set of coordinates $\{x_1, \ldots, x_d\}$ into two subsets $A = \{x_1, \ldots, x_{d'}\}$ and $B = \{x_{d'+1}, \ldots, x_d\}$ for some $1 \leq d' < d$. Transformations of coordinates of set B are then given by parameters that depend on the coordinates of set A, while the elements of set A are not transformed.

Conditional labels can be incorporated easily in both the MADE and the coupling layer architectures by feeding this information into the networks that predict the transformation parameters. In case of a MADE architecture, the connections from the conditional information are not masked.

APPENDIX B: CLASSIFIER ARCHITECTURES

To quantitatively asses the quality of our generated samples, we train a set of classifiers to distinguish CaloFlow samples from the GEANT4 training set and compare the results to the same set of classifiers trained to distinguish CaloGan from GEANT4 data. We use the following three classifier architectures:

The first classifier architecture is a simple DNN, which takes the 504 voxel values (normalized or not), the incident energy E_{inc} encoded via Eq. (6), and the three-layer energies E_i encoded via Eq. (7) as input. It has three hidden layers with 512 neurons each that use a leaky rectified linear unit (LeakyReLU) (with negative slope 0.01) activation functions, and the last layer has a single neuron with sigmoid activation. This setup has 786,433 trainable parameters.

The second classifier uses the same DNN architecture as the first, only the input layer is modified to now take 41 high-level features as input, leading to 547,329 trainable parameters. The input features are the ones we defined in Secs. V B and V C and showed in Figs. 11–19. We extended the list by including the voxel energy distributions of the third to fifth brightest voxel of each calorimeter layer. Based on the preprocessing, we can group the features in two groups. The first group consists of E_0, E_1, E_2, $\hat{E}_{\mathrm{tot}}$, $E_0/\hat{E}_{\mathrm{tot}}$, $E_1/\hat{E}_{\mathrm{tot}}$, $E_2/\hat{E}_{\mathrm{tot}}$, l_d, σ_0, σ_1, and σ_2. All of them are transformed by $\log_{10}$ before giving them to the DNN, together with the incident energy E_{inc} encoded via Eq. (6). The second group consists of s_d, σ_{s_d}, the energy of the brightest five voxels in each layer, $E_{\mathrm{ratio},0}$, $E_{\mathrm{ratio},1}$, $E_{\mathrm{ratio},2}$, the sparsities of each layer, and the centroids in ϕ and η direction of each layer. We multiply the energies of the brightest voxels by 10 and divide the centroid locations by 100 to have all numbers of $\mathcal{O}(1)$ before we give them directly to the DNN.

The third classifier is a CNN-based architecture derived from the top-tagging study of [99]. It consists of a series of two convolutional (with 128 and 64 channels, respectively), one max-pooling, two convolutional (with 64 channels each), and one max-pooling layer per calorimeter layer. Kernel sizes and padding of the first two convolutional layers are chosen such that the images have shape (n_{batch}, 64, 6, 6) after the first max-pooling layer. Kernel sizes and padding of the second two convolutional layers are chosen to keep the shape the same. The last max-pooling layer transforms the images to (n_{batch}, 64, 4, 4). The output of the three calorimeter layers is flattened and concatenated to a single vector, together with the incident energy E_{inc} encoded via Eq. (6), and the three-layer energies E_i encoded via Eq. (7) to a total size of 772. This vector is fed into a dense NN with two hidden layers of 512 neurons each and a one-dimensional output layer. Activations throughout the whole network are LeakyReLU functions (with negative slope 0.01), except for the very last layer, where we use a sigmoid. This architecture has a total of 1.262913×10^6 trainable parameters. To save training time, the CNN is trained with single precision.

The dataset containing 100,000 “real” and 100,000 “fake” showers was split into the training set of 120,000 samples, a test set of 40,000 samples, and a validation set of 40,000 samples. The classifiers are trained for 150 epochs with a minibatch size of 1000 samples using the Adam [83] optimizer and an initial learning rate of 10^{-3}. We use the epoch with the highest accuracy of the validation set for the subsequent evaluation of the AUC and the JSD on the test set. Before evaluating the classifier scores, we calibrate the classifier using isotonic regression [100] of SKLEARN [96] based on the validation dataset.

APPENDIX C: CLASSIFIERS IN LOGIT SPACE

Table V shows the results of the classifier runs when the data are preprocessed to logit space. The results are similar than in the case without this preprocessing step: The CaloGan sample can always be distinguished from the GEANT4 sample, but the CaloFlow sample is much harder to separate from GEANT4 data.

TABLE V. AUCs/JSD for the classification of e^+, γ, and π^+ showers from GEANT vs CaloGan and CaloFlow. All entries show mean and standard deviation of ten runs and are rounded to three digits. Data are preprocessed to logit space.

		DNN		CNN	
AUC/JSD		vs CaloGan	vs CaloFlow	vs CaloGan	vs CaloFlow
e^+	Unnormalized	1.000(0)/0.999(1)	0.668(3)/0.064(2)	1.000(0)/0.999(1)	0.737(150)/0.207(186)
	Normalized	1.000(0)/0.999(0)	0.675(3)/0.072(2)	1.000(0)/0.999(1)	0.879(74)/0.405(155)
γ	Unnormalized	1.000(0)/0.997(0)	0.674(5)/0.070(4)	1.000(0)/0.999(1)	0.742(110)/0.174(122)
	Normalized	1.000(0)/0.999(0)	0.678(6)/0.072(5)	1.000(0)/0.998(1)	0.885(21)/0.389(51)
π^+	Unnormalized	1.000(0)/0.999(0)	0.669(3)/0.072(3)	1.000(0)/1.000(0)	0.891(19)/0.409(48)
	Normalized	1.000(0)/0.999(0)	0.755(10)/0.165(12)	1.000(0)/1.000(0)	0.958(4)/0.625(18)

[1] HEP Software Foundation Collaboration, HEP software foundation community white paper working group—detector simulation, arXiv:1803.04165.

[2] HEP Software Foundation Collaboration, HL-LHC computing review: Common tools and community software, in 2022 Snowmass Summer Study, edited by P. Canal *et al.*, 10.5281/zenodo.4009114 (2020).

[3] P. Calafiura, J. Catmore, D. Costanzo, and A. Di Girolamo, Atlas HL-LHC computing conceptual design report, Technical Report No. CERN-LHCC-2020-015, CERN, Geneva, 2020.

[4] CMS Collaboration, CMS offline and computing public results, Technical Report approved HL-LHC resource projections, CERN, Geneva, 2020, https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults.

[5] ATLAS Collaboration, The new fast calorimeter simulation in ATLAS, Technical Report No. ATL-SOFT-PUB-2018-002, CERN, Geneva, 2018.

[6] GEANT4 Collaboration, GEANT4–a simulation toolkit, Nucl. Instrum. Methods Phys. Res., Sect. A **506**, 250 (2003).

[7] J. Allison, K. Amako, J. Apostolakis, H. Araujo, P. Arce Dubois, M. Asai *et al.*, Geant4 developments and applications, IEEE Trans. Nucl. Sci. **53**, 270 (2006).

[8] J. Allison, K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso *et al.*, Recent developments in Geant4, Nucl. Instrum. Methods Phys. Res., Sect. A **835**, 186 (2016).

[9] M. Paganini, L. de Oliveira, and B. Nachman, Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters, Phys. Rev. Lett. **120**, 042003 (2018).

[10] M. Paganini, L. de Oliveira, and B. Nachman, CaloGan: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks, Phys. Rev. D **97**, 014021 (2018).

[11] M. Erdmann, L. Geiger, J. Glombitza, and D. Schmidt, Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks, Comput. Software Big Sci. **2**, 4 (2018).

[12] M. Erdmann, J. Glombitza, and T. Quast, Precise simulation of electromagnetic calorimeter showers using a Wasserstein generative adversarial network, Comput. Software Big Sci. **3**, 4 (2019).

[13] ATLAS Collaboration, Deep generative models for fast shower simulation in ATLAS, Technical Report No. ATL-SOFT-PUB-2018-001, CERN, Geneva, 2018.

[14] D. Belayneh *et al.*, Calorimetry with deep learning: Particle simulation and reconstruction for collider physics, Eur. Phys. J. C **80**, 688 (2020).

[15] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol *et al.*, Getting high: High fidelity simulation of high granularity calorimeters with high speed, Comput. Software Big Sci. **5**, 13 (2021).

[16] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol *et al.*, Decoding photons: Physics in the latent space of a BIB-AE generative network, EPJ Web Conf. **251**, 03003 (2021).

[17] ATLAS Collaboration, Fast simulation of the ATLAS calorimeter system with generative adversarial networks, Technical Report No. ATL-SOFT-PUB-2020-006, CERN, Geneva, 2020.

[18] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair *et al.*, Generative adversarial networks, arXiv:1406.2661.

[19] S. Voloshynovskiy, M. Kondah, S. Rezaeifar, O. Taran, T. Holotyak, and D. J. Rezende, Information bottleneck through variational glasses, arXiv:1912.00830.

[20] HEP ML Community, A living review of machine learning for particle physics, https://iml-wg.github.io/HEPML-LivingReview/.

[21] L. de Oliveira, M. Paganini, and B. Nachman, Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis, Comput. Software Big Sci. **1**, 4 (2017).

[22] S. Alonso-Monsalve and L. H. Whitehead, Image-based model parameter optimization using model-assisted generative adversarial networks, IEEE Trans. Neural Networks **31**, 5645 (2020).

[23] A. Butter, T. Plehn, and R. Winterhalder, How to GAN event subtraction, SciPost Phys. Core **3**, 009 (2020).

[24] J. Arjona Martinez, T. Q. Nguyen, M. Pierini, M. Spiropulu, and J.-R. Vlimant, Particle generative adversarial networks for full-event simulation at the LHC and their application to pileup description, J. Phys. Conf. Ser. **1525**, 012081 (2020).

[25] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, and R. Winterhalder, How to GAN away detector effects, SciPost Phys. **8**, 070 (2020).

[26] SHiP Collaboration, Fast simulation of muons produced at the SHiP experiment using generative adversarial networks, J. Instrum. **14**, P11028 (2019).

[27] S. Carrazza and F. A. Dreyer, Lund jet images from generative and cycle-consistent adversarial networks, Eur. Phys. J. C **79**, 979 (2019).

[28] A. Butter, T. Plehn, and R. Winterhalder, How to GAN LHC events, SciPost Phys. **7**, 075 (2019).

[29] J. Lin, W. Bhimji, and B. Nachman, Machine learning templates for QCD factorization in the search for physics beyond the standard model, J. High Energy Phys. 05 (2019) 181.

[30] R. Di Sipio, M. Faucci Giannelli, S. Ketabchi Haghighat, and S. Palazzo, DijetGAN: A generative-adversarial network approach for the simulation of QCD dijet events at the LHC, J. High Energy Phys. 08 (2019) 110.

[31] B. Hashemi, N. Amin, K. Datta, D. Olivito, and M. Pierini, LHC analysis-specific datasets with generative adversarial networks, arXiv:1901.05282.

[32] V. Chekalina, E. Orlova, F. Ratnikov, D. Ulyanov, A. Ustyuzhanin, and E. Zakharov, Generative models for fast calorimeter simulation: The LHCb case, EPJ Web Conf. **214**, 02034 (2019).

[33] K. Zhou, G. Endrodi, L.-G. Pang, and H. Stocker, Regressive and generative neural networks for scalar field theory, Phys. Rev. D **100**, 011501 (2019).

[34] K. Datta, D. Kar, and D. Roy, Unfolding with generative adversarial networks, arXiv:1806.00433.

[35] P. Musella and F. Pandolfi, Fast and accurate simulation of particle detectors using generative adversarial networks, Comput. Software Big Sci. **2**, 8 (2018).

[36] D. Derkach, N. Kazeev, F. Ratnikov, A. Ustyuzhanin, and A. Volokhova, Cherenkov detectors fast simulation using

neural networks, Nucl. Instrum. Methods Phys. Res., Sect. A **952**, 161804 (2020).

[37] H. Erbin and S. Krippendorf, GANs for generating EFT models, Phys. Lett. B **810**, 135798 (2020).

[38] J. M. Urban and J. M. Pawlowski, Reducing autocorrelation times in lattice simulations with generative adversarial networks, Mach. Learn. Sci. Tech. **1**, 045011 (2020).

[39] L. de Oliveira, M. Paganini, and B. Nachman, Controlling physical attributes in GAN-accelerated simulation of electromagnetic calorimeters, J. Phys. Conf. Ser. **1085**, 042017 (2018).

[40] Y. Alanazi *et al.*, Machine learning-based event generator for electron-proton scattering, Phys. Rev. D **106**, 096002 (2022).

[41] S. Diefenbacher, E. Eren, G. Kasieczka, A. Korol, B. Nachman, and D. Shih, DCTRGAN: Improving the precision of generative models with reweighting, J. Instrum. **15**, P11004 (2020).

[42] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman, and T. Plehn, GANplifying event samples, SciPost Phys. **10**, 139 (2021).

[43] R. Kansal, J. Duarte, B. Orzari, T. Tomei, M. Pierini, M. Touranakou *et al.*, Graph generative adversarial networks for sparse data generation in high energy physics, *34th Conference on Neural Information Processing Systems* (2020), arXiv:2012.00173.

[44] A. Maevskiy, F. Ratnikov, A. Zinchenko, and V. Riabov, Simulating the time projection chamber responses at the MPD detector using generative adversarial networks, Eur. Phys. J. C **81**, 599 (2021).

[45] Y. S. Lai, D. Neill, M. Płoskoń, and F. Ringer, Explainable machine learning of the underlying physics of high-energy particle collisions, Phys. Lett. B **829**, 137055 (2022).

[46] S. Choi and J. H. Lim, A data-driven event generator for hadron colliders using Wasserstein generative adversarial network, J. Korean Phys. Soc. **78**, 482 (2021).

[47] F. Rehm, S. Vallecorsa, V. Saletore, H. Pabst, A. Chaibi, V. Codreanu *et al.*, Reduced precision strategies for deep learning: A high energy physics generative adversarial network use case, arXiv:2103.10142.

[48] S. Carrazza, J. Cruz-Martinez, and T. R. Rabemananjara, Compressing PDF sets using generative adversarial networks, Eur. Phys. J. C **81**, 530 (2021).

[49] T. Lebese, B. Mellado, and X. Ruan, The use of generative adversarial networks to characterise new physics in multi-lepton final states at the LHC, arXiv:2105.14933.

[50] D. Saxena and J. Cao, Generative adversarial networks (GANs): Challenges, solutions, and future directions, arXiv:2005.00065.

[51] I. Kobyzev, S. J. D. Prince, and M. A. Brubaker, Normalizing flows: An introduction and review of current methods, arXiv:1908.09257.

[52] G. Papamakarios, E. Nalisnick, D. Jimenez Rezende, S. Mohamed, and B. Lakshminarayanan, Normalizing flows for probabilistic modeling and inference, arXiv:1912.02762.

[53] L. Dinh, J. Sohl-Dickstein, and S. Bengio, Density estimation using real NVP, arXiv:1605.08803.

[54] G. Papamakarios, T. Pavlakou, and I. Murray, Masked autoregressive flow for density estimation, arXiv:1705.07057.

[55] C. Gao, J. Isaacson, and C. Krause, i-flow: High-dimensional integration and sampling with normalizing flows, Mach. Learn. Sci. Tech. **1**, 045023 (2020).

[56] C. Gao, S. Höche, J. Isaacson, C. Krause, and H. Schulz, Event generation with normalizing flows, Phys. Rev. D **101**, 076002 (2020).

[57] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale, and S. Schumann, Exploring phase space with neural importance sampling, SciPost Phys. **8**, 069 (2020).

[58] S. Pina-Otey, V. Gaitan, F. Sánchez, and T. Lux, Exhaustive neural importance sampling applied to Monte Carlo event generation, Phys. Rev. D **102**, 013003 (2020).

[59] B. Stienen and R. Verheyen, Phase space sampling and inference from weighted events with autoregressive flows, SciPost Phys. **10**, 038 (2021).

[60] M. Bellagente, M. Haußmann, M. Luchmann, and T. Plehn, Understanding event-generation networks via uncertainties, SciPost Phys. **13**, 003 (2022).

[61] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, A. Rousselot, R. Winterhalder, L. Ardizzone, and U. Köthe, Invertible networks or partons to detector and back again, SciPost Phys. **9**, 074 (2020).

[62] S. Choi, J. Lim, and H. Oh, Data-driven estimation of background distribution through neural autoregressive flows, arXiv:2008.03636.

[63] S. Bieringer, A. Butter, T. Heimel, S. Höche, U. Köthe, T. Plehn, and S. T. Radev, Measuring QCD splittings with invertible networks, SciPost Phys. **10**, 126 (2021).

[64] B. Nachman and D. Shih, Anomaly detection with density estimation, Phys. Rev. D **101**, 075042 (2020).

[65] M. Arjovsky, S. Chintala, and L. Bottou, Wasserstein GAN, arXiv:1701.07875.

[66] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, Improved training of Wasserstein GANs, arXiv:1704.00028.

[67] B. Nachman, L. de Oliveira, and M. Paganini, Electromagnetic calorimeter shower images, Mendeley Data, 10.17632/pvn3xc3wy5.1 (2017).

[68] M. Paganini, L. de Oliveira, and B. Nachman, hep-lbdl/calogan: CaloGAN generation, training, and analysis code, Zenodo, 10.5281/zenodo.584155 (2017).

[69] M. Germain, K. Gregor, I. Murray, and H. Larochelle, MADE: Masked autoencoder for distribution estimation, arXiv:1502.03509.

[70] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, Neural spline flows, in Adv. Neural Inf. Process. Syst., edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019), Vol. **32**, https://proceedings.neurips.cc/paper/2019/file/7ac71d433f282034e088473244df8c02-Paper.pdf.

[71] D. Lopez-Paz and M. Oquab, Revisiting classifier two-sample tests, arXiv:1610.06545.

[72] D. Jimenez Rezende and S. Mohamed, Variational inference with normalizing flows, arXiv:1505.05770.

[73] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, nflows: Normalizing flows in PyTorch, 10.5281/zenodo.4296287 (2020).

[74] L. Dinh, D. Krueger, and Y. Bengio, NICE: Non-linear independent components estimation, arXiv:1410.8516.

[75] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, Improving variational inference with inverse autoregressive flow, arXiv:1606.04934.

[76] C. Krause and D. Shih, preceding paper, Accelerating accurate simulations of calorimeter showers with normalizing flows and probability density distillation, Phys. Rev. D **107**, 113002 (2023).

[77] A. van den Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu *et al.*, Parallel WaveNet: Fast high-fidelity speech synthesis, arXiv:1711.10433.

[78] J. A. Gregory and R. Delbourgo, Piecewise rational quadratic interpolation to monotonic data, IMA J. Numer. Anal. **2**, 123 (1982).

[79] C. Krause and D. Shih, Electromagnetic calorimeter shower images of CaloFlow, zenodo, 10.5281/zenodo.5904188 (2021).

[80] M. Rosenblatt, Remarks on some nonparametric estimates of a density function, Ann. Math. Stat. **27**, 832 (1956).

[81] E. Parzen, On estimation of a probability density function and mode, Ann. Math. Stat. **33**, 1065 (1962).

[82] C. M. Bishop, Mixture density networks, 1994, https://research.aston.ac.uk/en/publications/mixture-density-networks.

[83] D. P. Kingma and J. Ba, ADAM: A method for stochastic optimization, arXiv:1412.6980.

[84] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, arXiv:1207.0580.

[85] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. **15**, 1929 (2014).

[86] J. Brehmer and K. Cranmer, Flows for simultaneous manifold learning and density estimation, arXiv:2003.13913.

[87] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro *et al.*, TENSORFLOW: Large-scale machine learning on heterogeneous systems, arXiv:1603.04467.

[88] F. Chollet *et al.*, KERAS, https://keras.io (2015).

[89] X. Nguyen, M. J. Wainwright, and M. I. Jordan, On surrogate loss functions and f-divergences, arXiv:math/0510521.

[90] B. Nachman and J. Thaler, Learning from many collider events at once, Phys. Rev. D **103**, 116013 (2021).

[91] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan *et al.*, PYTORCH: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett (Curran Associates, Inc., 2019), pp. 8024–8035, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[92] R. Winterhalder, M. Bellagente, and B. Nachman, Latent space refinement for deep generative models, arXiv:2106.00792.

[93] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau *et al.*, Array programming with NUMPY, Nature (London) **585**, 357 (2020).

[94] J. D. Hunter, MATPLOTLIB: A 2d graphics environment, Comput. Sci. Eng. **9**, 90 (2007).

[95] T. pandas development team, pandas-dev/pandas: PANDAS, 10.5281/zenodo.3509134 (2020).

[96] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel *et al.*, Scikit-learn: Machine learning in PYTHON, J. Mach. Learn. Res. **12**, 2825 (2011), https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.

[97] A. Collette, PYTHON and HDF5 (O'Reilly, North Sebastopol, CA, 2013).

[98] C. Krause and D. Shih, CaloFlow, https://gitlab.com/claudius-krause/caloflow.

[99] S. Macaluso and D. Shih, Pulling out all the tops with computer vision and deep learning, J. High Energy Phys. 10 (2018) 121.

[100] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, On calibration of modern neural networks, arXiv:1706.04599.