# Inductive simulation of calorimeter showers with normalizing flows

Matthew R. Buckley, Ian Pang, and David Shih
*NHETC, Department of Physics and Astronomy, Rutgers University, Piscataway, New Jersey 08854, USA*

Claudius Krause*
*NHETC, Department of Physics and Astronomy, Rutgers University, Piscataway, New Jersey 08854, USA and Institut für Theoretische Physik, Universität Heidelberg, 69120 Heidelberg, Germany*

(Received 21 July 2023; accepted 9 January 2024; published 13 February 2024)

Simulating particle detector response is the single most expensive step in the Large Hadron Collider computational pipeline. Recently it was shown that normalizing flows can accelerate this process while achieving unprecedented levels of accuracy, but scaling this approach up to higher resolutions relevant for future detector upgrades leads to prohibitive memory constraints. To overcome this problem, we introduce *Inductive CaloFlow* (iCaloFlow), a framework for fast detector simulation based on an inductive series of normalizing flows trained on the pattern of energy depositions in pairs of consecutive calorimeter layers. We further use a teacher-student distillation to increase sampling speed without loss of expressivity. As we demonstrate with datasets 2 and 3 of the *CaloChallenge2022*, iCaloFlow can realize the potential of normalizing flows in performing fast, high-fidelity simulation on detector geometries that are ∼10–100 times higher granularity than previously considered.

DOI: 10.1103/PhysRevD.109.033006

## I. INTRODUCTION

The computational resources required by the physics program of the Large Hadron Collider (LHC) are immense. In addition to the formidable demands set by the acquisition, reconstruction, and analysis of the physics events themselves, the physics goals of the LHC call for detailed and accurate simulations of these events. Such simulation requires even greater computer resources than the data acquisition and analysis itself (see [1–5] for recent reviews of the current status and future plans for LHC computing).

The most significant bottleneck (see e.g., Fig. 1 of [2]) in simulating LHC events is modeling the response of the detector—and in particular that of the calorimeter—to incident particles using Geant4 [6–8]. As the high-luminosity runs of the LHC progress, the demand for efficient simulation of collider events will only grow more pressing.

In recent years, a wide variety of deep generative models—including generative adversarial networks (GANs), Variational AutoEncoder (VAE)-based models, normalizing flows, and diffusion models [9–30]—have demonstrated their potential for fast and accurate surrogate modeling of Geant4. The probability distribution of energy depositions within a calorimeter can be learned by neural networks trained on collections of Geant4 events, and then new simulated events can be produced from these networks much faster than running Geant4 itself. This approach has gone beyond proof-of-concept, with AtlFast3 [22] (the current official fast-simulation framework of the ATLAS collaboration) adopting a GAN for a portion of its calorimeter simulation.

To spur new solutions [23,26,27] to the problem of fast calorimeter simulation, the Fast Calorimeter Simulation Challenge 2022 (*CaloChallenge2022*) [31] presents three datasets [32–34], each with increasing numbers of detector segments. Previous work [26] showed that the CaloFlow method of [17,18], based on normalizing flows [35–37] (see also [38,39] for reviews), could be quite successful at fast and accurate emulation of dataset 1 calorimeter showers. Generalizing the CaloFlow method to datasets 2 [33] and 3 [34], which are a factor of ∼10 and ∼100 larger in dimensionality compared to dataset 1, is the focus on this work.

The primary obstacle to scaling the CaloFlow approach up to datasets 2 and 3 is memory consumption. While normalizing flows are a powerful method for density estimation and generative modeling in high dimensional spaces, they can be very memory intensive, since they attempt to parametrize a bijective transformation between the data space and a latent space of the same dimension. Scaling the same architecture used in CaloFlow up to datasets 2 and 3 would easily outstrip the locally available GPU memory, since the total number of model parameters scales as $\mathcal{O}(d^2)$, where $d$ is the data dimension. Only diffusion models have been applied to these high-dimensional datasets so far [23].

Recently, L2LFlows [28] was proposed in order to overcome this obstacle. L2LFlows is based on the physical intuition that—since particles propagate through the

*Corresponding author: Claudius.Krause@oeaw.ac.at

 

calorimeter primarily in one direction—the pattern of energy deposition in one layer should depend in large part on the pattern in the previous layers. Trained on a toy dataset (derived from the one used here [15,19]) for the International Large Detector (ILD) [40,41] of comparable size to dataset 2, L2LFlows introduced one flow per calorimeter layer, conditioned on up to five previous layers through an embedding network. By not training a single flow to learn the voxel energies of the entire calorimeter, as was the case in [17,18,26], but instead breaking up the model into separate flows (each no larger than a single layer of the calorimeter), L2LFlows was able to keep the memory footprint of the model manageable while still scaling it up to a higher granularity calorimeter.

In this paper, we take the approach of L2LFlows one step further and replace the separate flows for each calorimeter layer with a *single* flow that generates the voxel energies in each layer, conditioned on the previous layer. This allows for the entire calorimeter shower to be learned inductively: training not on entire events over all layers, but rather on pairs of layers. Like in a mathematical induction proof, the initial layer is learned separately. Then, new events are simulated layer by layer, with the results for the $i$th layer serving as conditional input for the same normalizing flow now generating the $(i+1)$th.

In more detail, our new framework, which we dub *Inductive CaloFlow* (iCaloFlow), uses three normalizing flows to learn and generate calorimeter events. First, Flow-1 learns the pattern of total energy deposition in each layer of the calorimeter, conditioned on the incident energy entering the detector. This is a relatively low-dimension dataset, with the dimension being the number of layers (45 for the examples used in this paper). Next, Flow-2 learns the pattern of normalized energy deposition within the first layer of the calorimeter, conditioned on the total energy deposited in the layer. Finally, Flow-3 learns the pattern of normalized energy deposition in the $i$th layer, conditioned on both the total energy deposited in the layer and the pattern of energy deposition in the previous $(i-1)$th layer. This last flow is trained simultaneously over the deposition pattern of every layer beyond the first.

To achieve both high fidelity and ultra-fast generation speed, we follow the teacher-student method of CaloFlow. First we train masked autoregressive flow (MAF) [42] versions of Flow-1, Flow-2, and Flow-3 on the Geant4 data. MAFs are fast in density estimation but slow [by a factor of $\mathcal{O}(d)$] in generation. Then we fit "student" versions of Flow-2 and Flow-3 to their MAF teacher counterparts, using the technique of probability density distillation (PDD) [43].[1] The student models are inverse autoregressive flows (IAFs) [44], which are fast in generation but slow in density estimation.

The result is a new deep learning architecture, capable of learning and quickly generating calorimeter events even for the largest detector layout of *CaloChallenge2022*. We quantify the fidelity of the events generated by both the iCaloFlow teacher and student flows, using a combination of histograms of physical features, classifier-based metrics (as in Refs. [17,18,26,28]), and other metrics. We also measure and report on the generation speed of iCaloFlow for different batch sizes and hardware.

We reserve a detailed comparison with L2LFlows for future work: while such a comparison would be very interesting, it is nontrivial to perform. L2LFlows was trained on a completely different dataset, comparable in size to dataset 2, so direct comparisons are not possible without significant additional computational effort.[2] Here we just note that the main differences with L2LFlows—using Flow-3 instead of separate flows for each calorimeter layer, and conditioning on the previous layer instead of up to five previous layers—make iCaloFlow significantly more memory efficient, but likely at the cost of being less expressive (and hence a worse fit to the data). The iCaloFlow IAF student model also results in a speed advantage over L2LFlows's teacher-only MAF, again probably at the expense of fidelity.

In Sec. II, we describe the datasets we train our algorithm on and generate new data to compare against. In Sec. III, we describe the multiple flows that make up the iCaloFlow algorithm in detail, along with our training procedure for both the teacher and student. In Sec. IV, we show the results of our event generation, and use a classifier to quantitatively compare with the original dataset. We conclude in Sec. V.

## II. CALOCHALLENGE DATA

Datasets 2 and 3 [33,34] of the *CaloChallenge2022* [31] consist of 100,000 Geant4-simulated electron showers each, with incident energies sampled uniformly in log-space from 1 GeV to 1 TeV. The simulated detector volume has a cylindrical geometry of radius 80 cm, with 45 layers of active silicon detector (thickness 0.3 mm), alternating with inactive tungsten absorber layers (thickness 1.4 mm). The length of the voxel along the $z$-axis is 3.4 mm, which corresponds to two physical layers (tungsten-silicon-tungsten-silicon). Taking into account only the absorber value of radiation length ($X_0 = 3.504$ mm) it makes the $z$-cell size equal $0.8\ X_0$.

Each detector layer is segmented in read-out voxel cells in the azimuthal angle $\alpha$ and radial distance $r$ from the center of the cylindrical detector volume. The two datasets only differ in their voxelization. Dataset 2 has each layer

[1] Flow-1 is not paired with an IAF as the output dimension is relatively small, and so generation time for the MAF is comparatively short.

[2] L2LFlows would have to be retrained on dataset 2 and also generalized to dataset 3 which is a factor of ∼10 larger.

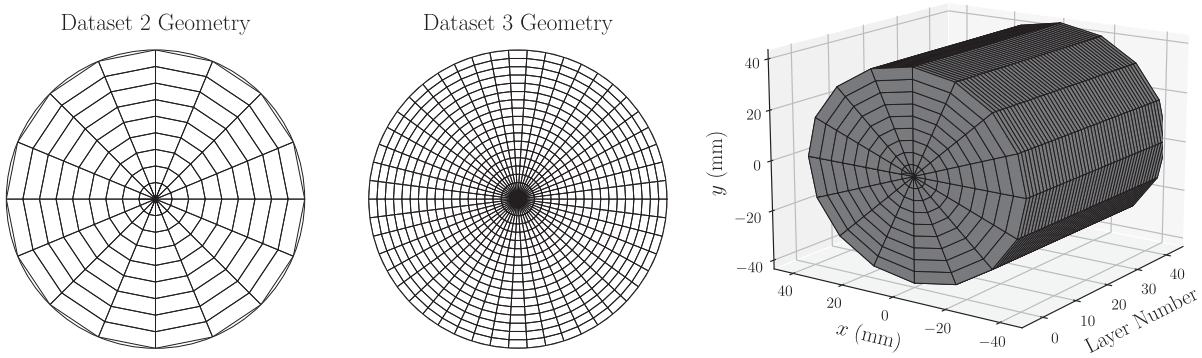


FIG. 1. Geometry of the detector voxels in each layer for dataset 2 (left) and dataset 3 (center) and the three-dimensional geometry of dataset 2 (right). Dataset 2 has 9 concentric rings, each divided into 16 voxels in $\alpha$, while dataset 3 has 18 rings, each divided into 50 segments. Both dataset 2 and 3 contain 45 layers in depth (as shown for dataset 2).

divided into 9 concentric rings of voxels in the radial direction. Each ring is then divided into 16 voxels in $\alpha$, for 144 voxels in each of the 45 layers (6480 for the entire detector). Dataset 3 has 18 radial segments and 50 azimuthal, for 900 voxels in the 45 layers (40500 total). Figure 1 shows the geometry of the voxels within a layer for both datasets, as well as the positioning of layers along the calorimeter axis.

Each event record consists of the total energy of the incident electron $E_{\text{inc}}$, together with the energy depositions recorded in each voxel $\mathcal{I}_{ia}$, where $i$ is the layer index and $a$ is the voxel index within the layer. The minimum energy deposition in each voxel is 15 keV. The time required to generate a Geant4 shower depends strongly on the shower incident energy. It is approximately $\mathcal{O}(100\ \text{s})$ when averaged over the incident energies of these datasets [45], but the time required per shower is much longer for the showers with higher incident energies. Since the underlying detector geometry of datasets 2 and 3 is the same and only the voxelization is different, the generation times with Geant4 are the same for both datasets.

## III. iCaloFlow

### A. Overview

The purpose of iCaloFlow (and of the *CaloChallenge2022* generative modeling problem more generally) is to learn and sample from the conditional probability density $p(\mathcal{I}_{ia}|E_{\text{inc}})$ that describes the Geant4 reference data. In Fig. 2 we show a schematic of iCaloFlow, which consists of three flows:

(i) Flow-1 learns the joint probability distribution of total energy deposited in each layer $E_i$, conditioned on the incident energy of the event $E_{\text{inc}}$: $p_1(E_i|E_{\text{inc}})$. It is necessary to learn this probability distribution as $E_i$ is a conditional input for Flow-2 and Flow-3 in the

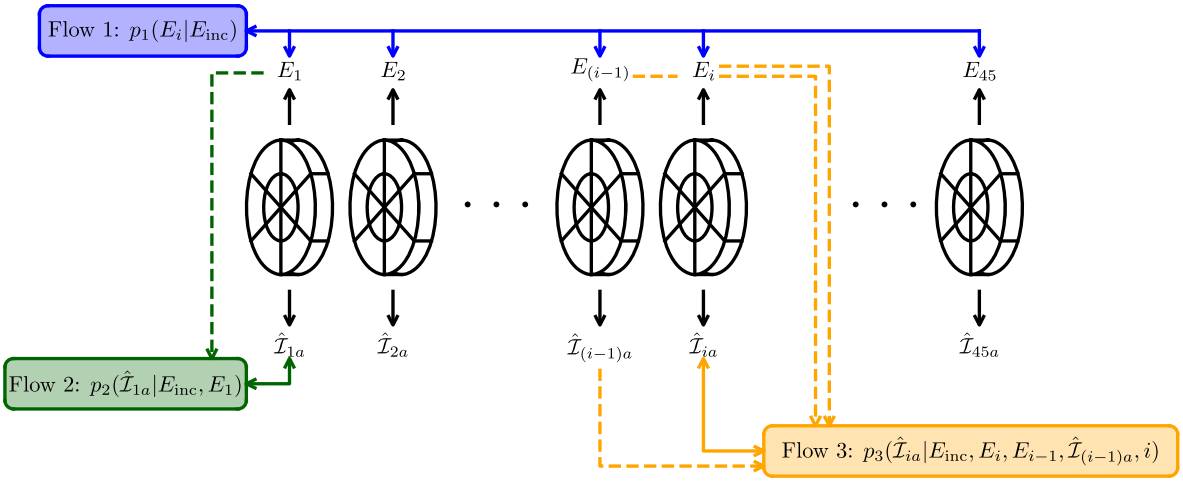


FIG. 2. Schematic of the three iCaloFlow flows. Solid lines are bidirectional—the direction into each flow denotes the density estimation step and the direction out of the flow denotes the sample generation step. Note that there are postprocessing steps (see main text) after each generation step, which are omitted in the schematic. Dashed lines indicate the conditional input to the respective flows. Flow-3 is used iteratively on subsequent layers.

generation step. We found that removing $E_i$ as a conditional input in Flow-2 and Flow-3 decreased the quality of our generated samples.

(ii) Flow-2 learns the probability distribution of the unit-normalized voxel energies in the first layer of the calorimeter, $\hat{\mathcal{I}}_{1a} \equiv \mathcal{I}_{1a}/\sum_b \mathcal{I}_{1b}$, conditioned on $E_{\text{inc}}$ and the energy deposited in the first layer, $E_1$: $p_2(\hat{\mathcal{I}}_{1a}|E_{\text{inc}}, E_1)$. Here $a$ is the voxel index.

(iii) Finally, Flow-3 learns the probability distribution of unit-normalized voxel energies in every layer after the first, $\hat{\mathcal{I}}_{ia} \equiv \mathcal{I}_{ia}/\sum_b \mathcal{I}_{ib}$ for $i \in [2, 45]$, where the $i$th layer is conditioned on the energy deposited in the layers $i$ and $i-1$ ($E_i$ and $E_{i-1}$), $E_{\text{inc}}$, the unit-normalized voxel energies in the $(i-1)$th layer $\hat{\mathcal{I}}_{(i-1)a}$, and the one-hot[3] encoded layer number $i$: $p_3(\hat{\mathcal{I}}_{ia}|E_{\text{inc}}, E_i, E_{i-1}, \hat{\mathcal{I}}_{(i-1)a}, i)$.

The conditional inputs, dimension of conditionals, and the outputs for each of the three flows are summarized in Table I. When generating new showers with iCaloFlow, first the nominal energies per layer $\tilde{E}_i$ are generated for a given incident $E_{\text{inc}}$ with Flow-1.[4] Then the unit-normalized voxel energies $\hat{\mathcal{I}}_{1a}$ of layer 1 is generated using Flow-2, conditioned on $\tilde{E}_1$ and $E_{\text{inc}}$. Finally, the normalized voxel energies $\hat{\mathcal{I}}_{ia}$ of layers $i = 2, \ldots, 45$ are generated sequentially (inductively), using Flow-3 with the conditional inputs of the previous layer's $\hat{\mathcal{I}}$ distribution provided by Flow-2 (for generating Layer 2) or Flow-3 (for all subsequent layers). By generating all layers beyond the first from a single normalizing flow rather than each layer separately as in L2LFlows [28], our model is far more efficient in memory usage. The potential downsides are that training cannot be trivially parallelized and our model may be less expressive.

One important subtlety with the construction of iCaloFlow is that Flow-2 and Flow-3 cannot learn the unit-normalization constraint of the $\hat{\mathcal{I}}_{ia}$ training data perfectly. In practice, the sum of the generated showers will be distributed around unity, and there will always be some mismatch between the nominal layer energy generated by Flow-1 and the actual layer energy produced by the combination of all three flows. We will address this mismatch here[5] by taking the latter as the actual layer energy: after multiplying $\hat{\mathcal{I}}_{ia}$ with the output of Flow-1 and applying a minimum energy threshold of $E_{\text{min}} = 15$ keV to obtain $\mathcal{I}_{ia}$, we take the sum over voxels $a$ as the total energy in the layer:

TABLE I. The conditional inputs for each flow, and the features whose probability distributions are the output of each flow (for both teachers and their paired student). $E_{\text{inc}}$ is the incident energy of the particle, $E_i$ ($i = 1, \ldots, 45$) is the total energy deposited in layer $i$, $\tilde{E}_i$ is a proxy for $E_i$ (see text), and $\hat{\mathcal{I}}_{ia}$ is the pattern of normalized energy deposition in the voxels $a$ in layer $i$. For Flow-3, dataset 2 has smaller conditional dimension (191) compared to dataset 3 (947).

| | Conditionals | Dimension of conditional | Output |
|---|---|---|---|
| Flow-1 | $E_{\text{inc}}$ | 1 | $\tilde{E}_i$ |
| Flow-2 | $E_{\text{inc}}, E_1$ | 2 | $\hat{\mathcal{I}}_{1a}$ |
| Flow-3 | $E_{\text{inc}}, E_i, E_{i-1}, \hat{\mathcal{I}}_{(i-1)a}, i$ | 191 or 947 | $\hat{\mathcal{I}}_{ia}$ |

$$E_i \equiv \sum_a \tilde{E}_i \hat{\mathcal{I}}_{ia} \Theta(\tilde{E}_i \hat{\mathcal{I}}_{ia} - E_{\text{min}}). \tag{1}$$

We will think of the output of Flow-1 as just an intermediate step needed for the subsequent conditioning, and refer to output of Flow-1 when generating new events as the *proxy* $\tilde{E}_i$. We note that referring to the output of a flow as a proxy for the distributions it was trained on is nonstandard, but necessary in this case due to the differences in normalization and the multiple ways of defining the energy deposited in a layer. In a sense, also the normalized shower shapes $\hat{\mathcal{I}}_{ia}$ that Flow-2 and Flow-3 learn can be thought of as proxies, since the resulting samples need to be unnormalized and thresholded as a postprocessing step,

$$\mathcal{I}_{ia} = \tilde{E}_i \hat{\mathcal{I}}_{ia} \Theta(\tilde{E}_i \hat{\mathcal{I}}_{ia} - E_{\text{min}}). \tag{2}$$

## B. Architectures

### 1. Teacher MAFs

The iCaloFlow teacher flows (including Flow-1) are MAFs [42], which learn a bijective transformation $f$ between a latent space $z$, with a simple $N$-dimensional probability distribution,[6] and the target space $x$. For consistency with the "forward" function of the code, we define $z = f(x)$.

Following [17,18,26], all three teacher flows use compositions of rational quadratic spline (RQS) transformations [46] as their transformation function $f$. The neural networks defining the parameters $\vec{\kappa}$ of the RQS consist of MADE blocks [47]. The MADE blocks allow for tractable training given the large dimensionality of the training data for Flow-2 and Flow-3, at the cost of longer evaluation time for generating new events. For details of the architectures used for the teacher flows, see Table II.

[3]One-hot encoding is used for layer numbers instead of ordinal encoding using the layer number directly, because other than the location in the detector, there is no information in the layer number, i.e., layer 30 is not 15 times more important than layer 2.

[4]This does not directly correspond to the energy deposited in the layer, $E_i$, see the discussion on postprocessing below.

[5]This problem is also present in the previous iterations of CaloFlow [17,18,26] and in L2LFlows [28], but differences in the reference datasets led to different treatments of the problem. For more details, see Appendix A.

[6]In this work, the latent space follows an $N$-dimensional Gaussian distribution.

TABLE II. Summary of architecture of the various MAF teacher and IAF student models used in iCaloFlow. For the hidden layer sizes, the first number is the number of hidden layers in each MADE block and the second number is the number of nodes in each hidden layer (e.g., $2 \times 256$ refers to 2 hidden layers per MADE block with 256 nodes per hidden layer).

| | | | | Layer sizes | | | |
|---|---|---|---|---|---|---|---|
| | | Dimension of base distribution | Number of MADE blocks | Input | Hidden | Output | Number of RQS bins |
| DS2 | Flow-1 | 45 | 8 | 256 | $1 \times 256$ | 1035 | 8 |
| | Flow-2 teacher | 144 | 8 | 256 | $2 \times 256$ | 3312 | 8 |
| | Flow-2 student | 144 | 8 | 256 | $2 \times 256$ | 3312 | 8 |
| | Flow-3 teacher | 144 | 8 | 256 | $2 \times 256$ | 3312 | 8 |
| | Flow-3 student | 144 | 8 | 384 | $2 \times 384$ | 3312 | 8 |
| DS3 | Flow-1 | 45 | 8 | 256 | $1 \times 256$ | 1035 | 8 |
| | Flow-2 teacher | 900 | 8 | 256 | $2 \times 256$ | 20700 | 8 |
| | Flow-2 student | 900 | 8 | 256 | $2 \times 256$ | 20700 | 8 |
| | Flow-3 teacher | 900 | 8 | 256 | $1 \times 256$ | 20700 | 8 |
| | Flow-3 student | 900 | 8 | 256 | $1 \times 256$ | 20700 | 8 |

#### 2. Student IAFs

Though the three MAFs are capable of generating complete events simulating the Geant4 output, the sampling time is quite slow. While this is not a particular concern for Flow-1, the sampling time is an issue especially for Flow-3, which has a large output dimension and must be sampled 44 times sequentially (i.e., cannot be parallelized) in order to generate a single event.

The solution proposed in [18], which we carry over here to datasets 2 and 3 of *CaloChallenge2022*, is to pair every slow-sampling MAF (i.e., FLOW-2 and Flow-3) with a fast-sampling inverse autoregressive flow (IAF). Since training the IAF with a negative log-likelihood of the probability of the data is prohibitively slow, the IAF is trained by fitting it to a pretrained MAF using the PDD method developed in [18]. The goal of this training is for the IAF to learn $f_{\mathrm{IAF}} = f_{\mathrm{MAF}}$, or equivalently—since only the fast passes through the flows can be used meaningfully for optimization—to have $f_{\mathrm{MAF}}$ and $f_{\mathrm{IAF}}^{-1}$ be each other's inverse. For more details of the training and loss terms, see Sec. III C 2 and [18].

In order to carry out the PDD method, we require $f_{\mathrm{IAF}} = f_{\mathrm{MAF}}$ at every individual step of the normalizing flow. Hence, the IAF must be composed of the same number of MADE-RQS blocks as its corresponding pretrained MAF. However, the hidden layer sizes in the blocks can be different. We use a larger hidden layer, with 384 nodes, for Flow-3 student which led to better performance in dataset 2. For dataset 3, Flow-3 student has the same hidden layer size (256 nodes) as the teacher due to memory constraints. For a detailed listing of the architecture hyperparameters for the student IAFs, see Table II.

### C. Training

Prior to training the teachers and students, we must standardize and preprocess the datasets. New events generated from the trained flows will be in the standardized space, and thus the transformations are inverted to produce events in the physical units. We detail this process for all three flows in Appendix A. The same preprocessing was used when training both the teachers and students.

#### 1. Teachers

The teacher MAFs are trained using the mean negative log-likelihood of the data evaluated on the output of the MAF as the loss function,

$$L_{\mathrm{teacher},1} = -\langle \log p_1(E_i|E_{\mathrm{inc}})\rangle, \tag{3}$$

$$L_{\mathrm{teacher},2} = -\langle \log p_2(\hat{\mathcal{I}}_{1a}|E_{\mathrm{inc}}, E_1)\rangle, \tag{4}$$

$$L_{\mathrm{teacher},3} = -\langle \log p_3(\hat{\mathcal{I}}_{ia}|E_{\mathrm{inc}}, E_i, E_{i-1}, \hat{\mathcal{I}}_{(i-1)a}, i)\rangle. \tag{5}$$

All teacher MAFs in this work are trained with independent ADAM optimizers [48].

Given the different sizes of datasets 2 and 3, we use a slightly different training strategy for the two datasets. We use 70,000 samples of dataset 2 for training the flows and the remaining 30,000 samples for model selection. The OneCycle learning rate (LR) schedule [49] was implemented for all three flows. With this LR schedule (see

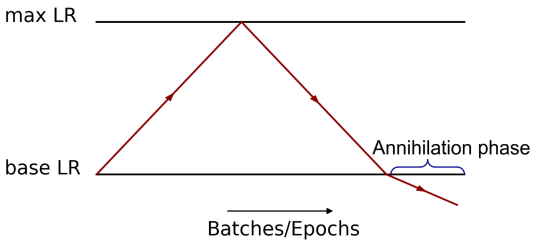


FIG. 3. Illustration of OneCycle LR schedule with annihilation phase [26].

TABLE III. Hyperparameters used when training dataset 2 and 3 iCaloFlow models. For dataset 3 Flow-3, the number of epochs is written as $N + N$ to indicate that we trained the flow for $N$ epochs on samples of File 1, followed by another $N$ epochs on samples of File 2.

| Trained with multistep LR | | Initial LR | Epochs | Batch size |
|---|---|---|---|---|
| DS3 teacher | Flow-1 | $1 \times 10^{-4}$ | 750 | 1000 |
| | Flow-2 | $1 \times 10^{-4}$ | 750 | 1000 |
| | Flow-3 | $1 \times 10^{-4}$ | $20 + 20$ | 500 |

| Trained with OneCycle LR | | Base LR | Max LR | Epochs | Batch size |
|---|---|---|---|---|---|
| DS2 teacher | Flow-1 | $1 \times 10^{-5}$ | $1 \times 10^{-4}$ | 500 | 1000 |
| | Flow-2 | $2 \times 10^{-5}$ | $1 \times 10^{-3}$ | 200 | 1000 |
| | Flow-3 | $2 \times 10^{-5}$ | $1 \times 10^{-3}$ | 60 | 1000 |
| DS2 student | Flow-2 | $2 \times 10^{-5}$ | $1 \times 10^{-3}$ | 400 | 1000 |
| | Flow-3 | $2 \times 10^{-5}$ | $5 \times 10^{-4}$ | 100 | 1000 |
| DS3 student | Flow-2 | $2 \times 10^{-5}$ | $1 \times 10^{-3}$ | 400 | 100 |
| | Flow-3 | $4 \times 10^{-6}$ | $1 \times 10^{-4}$ | 15+15 | 100 |

Fig. 3), the LR begins at a chosen base LR and is updated after each batch such that it increases up to a maximum LR. After this, the LR is made to decrease from the maximum LR to the base LR. The schedule finishes with an annihilation phase where the base LR is further decreased up to a factor of 10. As in [26], we find that using the OneCycle LR schedule for dataset 2 enabled us to obtain a lower training loss within a shorter number of epochs. We show a summary of the training hyperparameters in Table III. The reason for the smaller number of epochs in Flow-3 compared to the other ones is that the dataset is now 44 times larger (consisting of data from layer $i$ and $i-1$ for $i \in [2, 45]$).

Training Flow-1 and Flow-2 for dataset 3 uses a similar strategy as used for dataset 2. However, a different learning rate schedule was used when training the flows for dataset 3. Note that the *CaloChallenge2022* provides dataset 3 as two files of 50,000 events each (Files 1 and 2). For the first stages of training, we combine the showers of these two files into one dataset with 100,000 showers—70,000 of these are used for training and the remaining 30,000 are used for model selection.

We train Flow-3 for dataset 3 in two stages. First, we train it for 20 epochs using 40,000 samples from File 1 for training and 10,000 for model selection. Then, using the same optimizer, we train Flow-3 for another 20 epochs using 40,000 samples of File 2. We again use the remaining 10,000 samples for model selection. Simultaneous handling of all 100,000 showers was not possible due to prohibitive memory requirements. A multistep LR schedule was used when training the dataset 3 teacher flows, where we halve a chosen initial LR after epochs 400 and 500 (see Table III for summary of training hyperparameters). We found that training the flows with OneCycle LR schedule resulted in a slightly poorer performance for dataset 3.

For Flow-1 and Flow-2 of both datasets, the epoch with the lowest test loss is selected for subsequent sample generation. For Flow-3 of both datasets, due to the large training data, the test loss is evaluated after every 250 batches and also at the end of each epoch. The model checkpoint with the lowest test loss is selected for subsequent sample generation.

### 2. Students

The main idea behind teacher-student training of the IAF via PDD is to enforce that they are each other's inverses using only their fast passes:

$$x \equiv f_{\mathrm{IAF}}^{-1}(f_{\mathrm{MAF}}(x)), \tag{6}$$

and

$$z \equiv f_{\mathrm{MAF}}(f_{\mathrm{IAF}}^{-1}(z)). \tag{7}$$

We refer to these conditions as $x$-pass and $z$-pass, respectively. For each of the passes, we can construct a set of mean-squared error (MSE) losses that force the IAF to converge to the MAF:

$$L_x = \mathrm{MSE}(x, f_{\mathrm{IAF}}^{-1}(f_{\mathrm{MAF}}(x))) \tag{8}$$

$$L_z = \mathrm{MSE}(z, f_{\mathrm{MAF}}(f_{\mathrm{IAF}}^{-1}(z))). \tag{9}$$

In addition, [18] proposed two more MSE loss terms (which we will refer to as $L_{x-\mathrm{MADE}}$ and $L_{z-\mathrm{MADE}}$ here) that enforce the agreement between MAF and IAF at the level of the outputs and parameters of each individual MADE block (see [18] for details). These were found to improve the performance of the teacher-student training and we also include them here.

When training the Flow-2 students, we were able to achieve good agreement between the teachers and students by using the same loss function as in [18]:

$$L = (L_x + L_{x-\mathrm{MADE}}) + (L_z + L_{z-\mathrm{MADE}}). \tag{10}$$

However, for Flow-3 students, using the same loss function resulted in a significant disagreement between the teachers and students for some distributions. Instead, we found that training with $x$-loss only,

$$\tilde{L} = L_x + L_{x-\mathrm{MADE}} \tag{11}$$

resulted in better agreement. We note that a similar behavior was found in [50] where the loss constructed only using the $x$-pass was more robust to the large dimensionality of the training data.

When training each student using PDD, the ratio of the data used for training and model selection was chosen to match that of the corresponding teacher. Like the teacher MAFs, all the student IAFs are trained with independent

Adam optimizers [48]. The OneCycle LR schedule was implemented for all the student IAFs and the details of the training hyperparameters are shown in Table III. Note that a smaller batch size was used when training dataset 3 student flows due to memory constraints.

For Flow-1 and Flow-2 of both datasets, the epoch with the lowest mean Kullback–Leibler (KL) divergence[7] is selected for subsequent sample generation. For Flow-3 of both datasets, due to the large training data, the intermediate mean KL divergence of each epoch is evaluated after every 250 batches, while the final mean KL divergence is evaluated at the end of each epoch. The model checkpoint with the lowest evaluated mean KL divergence is selected for subsequent sample generation.

## IV. RESULTS

After training teacher and student flows on datasets 2 and 3, respectively, we generate 100,000 calorimeter showers from each model for each dataset. Incident energies are uniformly sampled from log-space between 1 GeV and 1 TeV—the same range and distribution of energies as the training data.

### A. Shower images

In Fig. 4, we show the pattern of energy deposition in all layers for two example events from dataset 2 (one with $E_{\text{inc}} = 693$ GeV and one with 86 GeV), compared with two events generated by iCaloFlow with equal incident energies. (Similar plots for dataset 3 events are difficult to visualize clearly due to the density of voxels.) In Fig. 5, we show the pattern of energy deposition in layers 1, 10, 20, and 45, averaged over all the events in the dataset for both dataset 3 and the sampled events from iCaloFlow.

### B. Distributions

We next consider more detailed diagnostic plots, comparing the distribution of various high-level features between the Geant4, teacher, and student events. First, we examine the energy deposition in each layer (again noting this is obtained by the sum of the voxel energies output from Flow-2 and Flow-3). In Fig. 6, we show the energies deposited in each layer, averaged over all the generated showers, which we denote by $\langle E_i \rangle$. As expected, the generated distributions are similar for both datasets 2 and 3, with small variations due to the different training regimes and normalization differences in the output of the flows. The output of the student networks largely follows that of the teachers, with the most significant deviations at both low and high layer numbers.

[7]The KL divergence between the student $s(x)$ and teacher $t(x)$ distributions is defined as $KL(s,t) = \int dx\, s(x) \log \frac{s(x)}{t(x)} \sim \sum_{x\sim s} \log \frac{s(x)}{t(x)}$.

In Fig. 7, we show the total energy deposition $E_i$ within a layer for our selected set of layers ($i = 1, 10, 20, 45$). Here we see good overall agreement between Geant4 and iCaloFlow distribution, with the exception of Layer 1 due to our choice of postprocessing. In particular, our decision not to normalize the outputs of Flow-2 and Flow-3 to unity results in a difference between the energy deposited in a layer $E_i$ and the proxy output of Flow-1, $\tilde{E}_i$. On the other hand, we find that the teacher voxel energy distributions in Figs. 8 and 9 are generally in good agreement with the Geant4 distributions. (As discussed further in Appendix A, enforcing $E_i = \tilde{E}_i$ would "fix" the distribution in Fig. 7, but at the cost of creating an excess of low-energy voxels in Figs. 8 and 9.) However, the student distributions suffer from an excess at low voxel energies. We found that this discrepant behavior is largely due to our choice of noise that is added to voxel energies during preprocessing. However, the addition of noise is necessary in our setup to ensure that the flow does not only learn zero energy voxels [17]. Note that many voxels have zero energy deposition, which are not captured in this logarithmic plot.

We show the ratio of energy deposited in a layer to the incident beam energy in Fig. 10, and the ratio of all deposited energy to the incident beam energy in Fig. 11. There is good overall agreement between iCaloFlow and Geant4 events despite some excess at low ratios in the earlier layers. We observed that the showers in the low ratio excess in Layer 1 in these figures are the same showers found in the low energy excess in Layer 1 shown in Fig. 7. These showers are characterized by having mostly zero energy voxels with a few bright voxels, this makes it difficult for the flow to learn the underlying distribution.

We turn now to other aspects of the pattern of energy deposition within each layer. First, we consider the sparsity $f_0$ for each layer, defined as the fraction of voxels that have nonzero energy deposition and shown in Fig. 12. We note that the lowest and highest layer numbers have larger fraction of zero energy voxels. It appears to be more difficult for iCaloFlow to learn distributions in these layers with small $f_0$ as evident from the deviations found in distributions discussed in this section. In Figs. 13 (dataset 2) and 14 (dataset 3), we show box-and-whisker plots of the distribution of energy deposited within all voxels in each radial ring within the layer (9 such rings in the geometry of dataset 2, each with 16 voxels. Dataset 3 has 18 rings in increasing radius, each with 50 voxels). In the main plots of Figs. 13 and 14, we show the average distribution of all voxels with nonzero energy deposition. Beneath, we show the averaged sparsity of the voxels within the ring. These distributions have good agreement between the Geant4 and iCaloFlow events.

As a final comparison of the pattern of energy distribution, in Fig. 15 we show the distribution of the centers of energy along the $x$ axis of the detector for our representative layer. The center of energy $\mathcal{C}_j$ along a

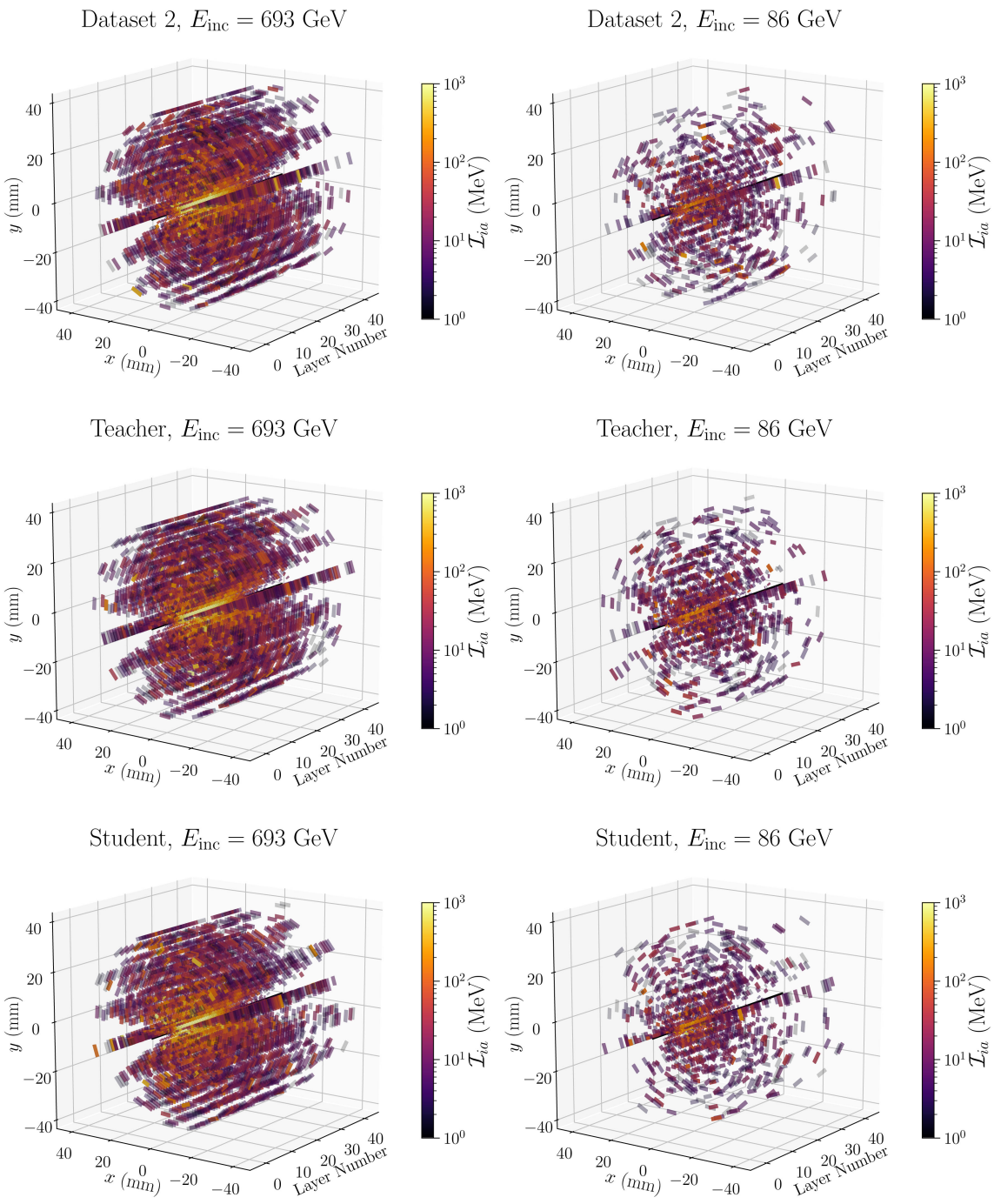


FIG. 4. Pattern of energy deposition from two example events generated by Geant4 in dataset 2 (top row), iCaloFlow teacher (middle row), and iCaloFlow student (bottom row). Events have $E_{\text{inc}} = 693$ GeV (left column) and $E_{\text{inc}} = 86$ GeV (right column). For visual clarity, voxels with less than 1 MeV of energy have been suppressed. The beam axis is shown with a black line. For display purposes, the separation between layers and voxels within a layer have both been artificially increased from the real detector geometry.

coordinate ($j = x$ or $y$) is defined as the sum of the energy deposited in each voxel times the voxel's coordinate distance from the origin, normalized by the total energy deposited. We show only $\mathcal{C}_x$, as the distribution of $\mathcal{C}_y$ is statistically identical due to the symmetry of the detector around the incident beam. Again, we see the largest deviations in the tails of the centers of energy for the early and late layers of dataset 3.

While these plots capture only a limited set of diagnostic criteria for the generated events as compared to the Geant4 data, it appears that many of the distributions match well. The most glaring and important exceptions are at low

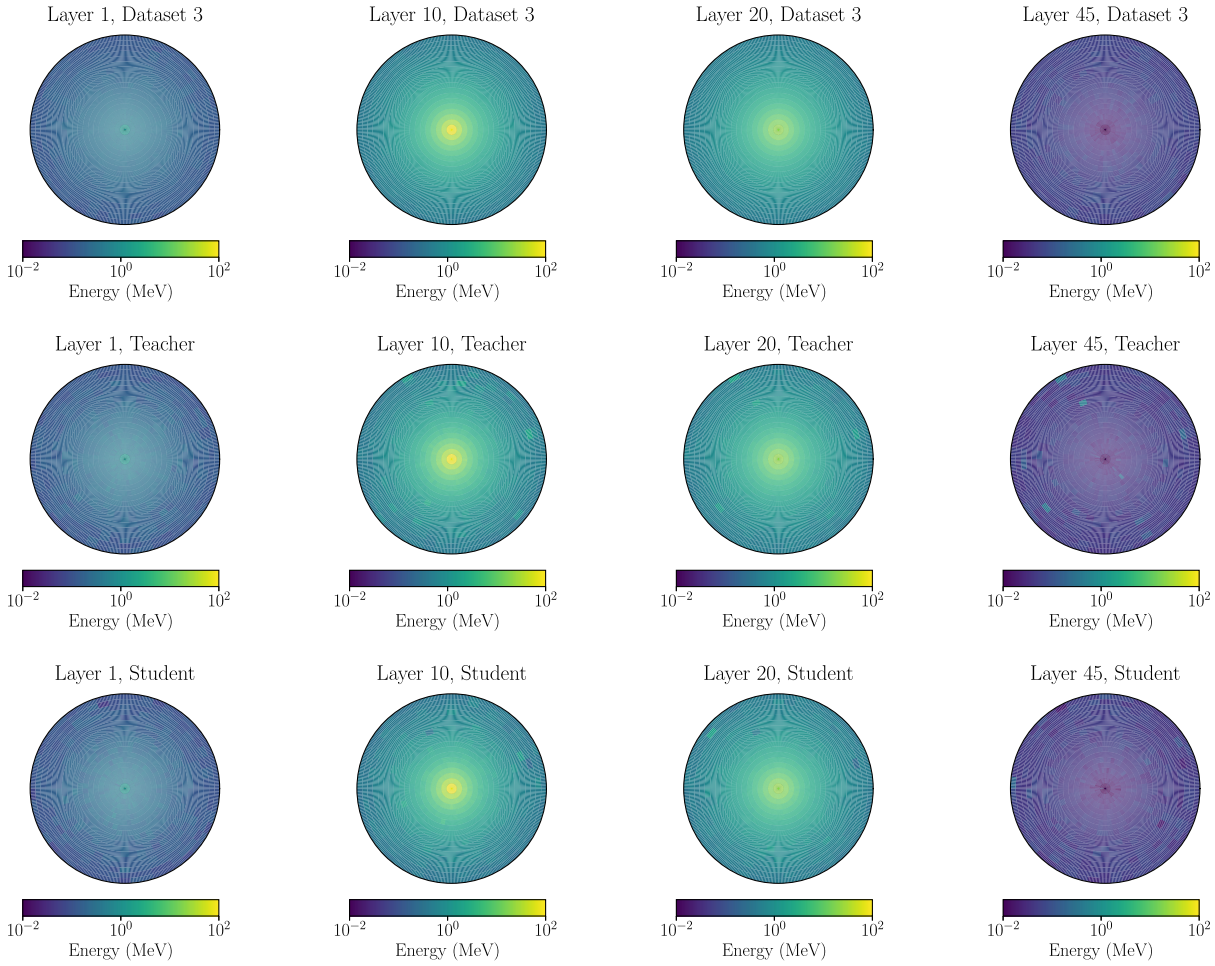


FIG. 5. Averaged energy deposition pattern of events in layers 1, 10, 20, and 45 (from left to right) for the Geant4 data in dataset 3 (top row), events sampled from iCaloFlow teacher trained on dataset 3 (middle row), and events sampled from iCaloFlow student trained on dataset 3 (bottom row).

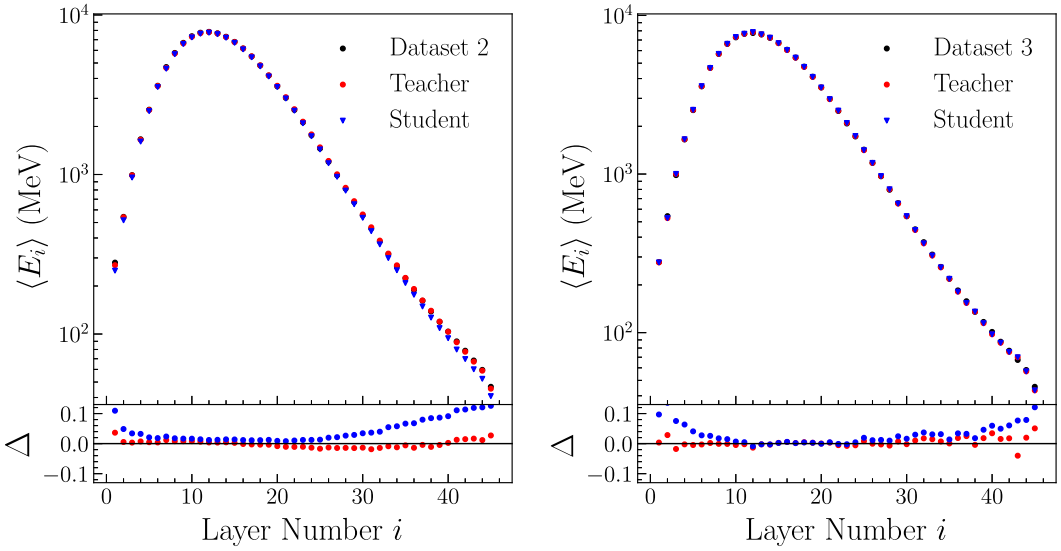


FIG. 6. Averaged total energy deposition $\langle E_i \rangle$ for each layer $i$ for datasets 2 (left) and 3 (right). In each plot the averaged energy of the Geant4 data is shown in black, and the distribution generated by iCaloFlow teacher (student) in red (blue). The fractional difference $\Delta$ between the truth-level and generated distributions is shown below the main figures.

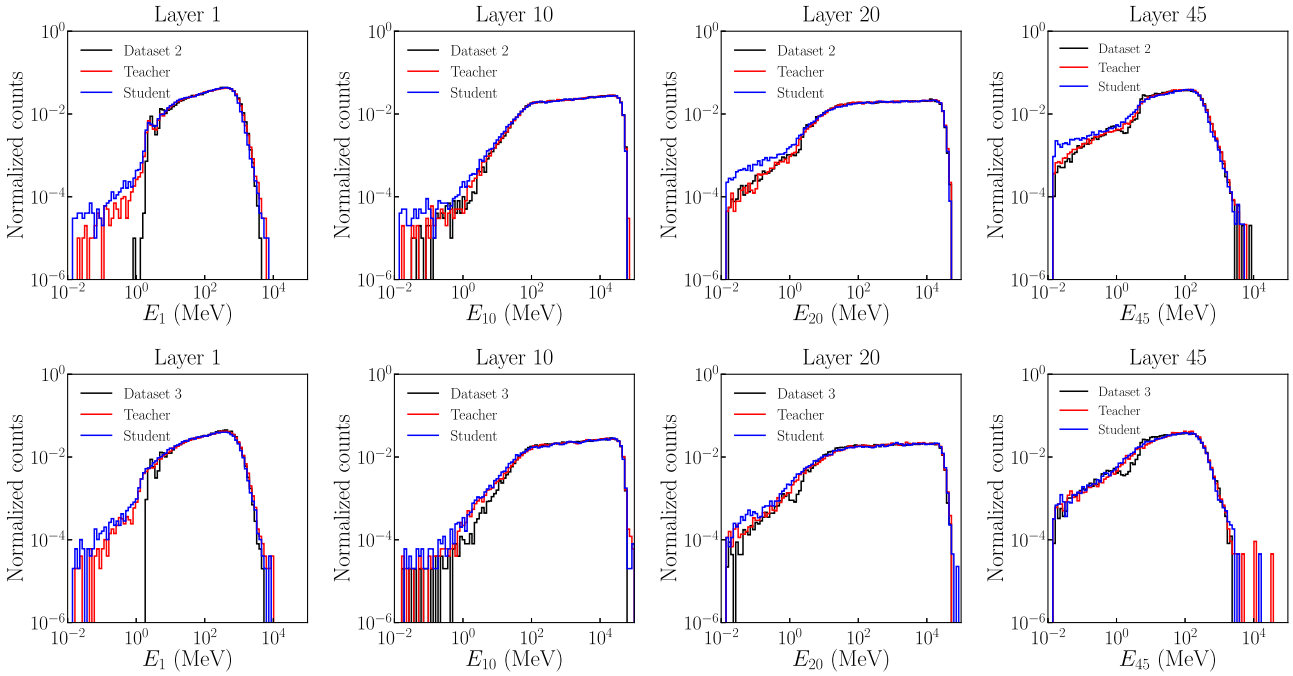


FIG. 7. Histograms of total energy deposited in a layer $i$ ($E_i$), for $i = 1$, 10, 20, and 45 (from left to right), for dataset 2 (top row) and dataset 3 (bottom row). Distribution of Geant4 data is shown as black lines, and that of the iCaloFlow teacher (student) trained on dataset 2 or 3 (as appropriate) in red (blue).

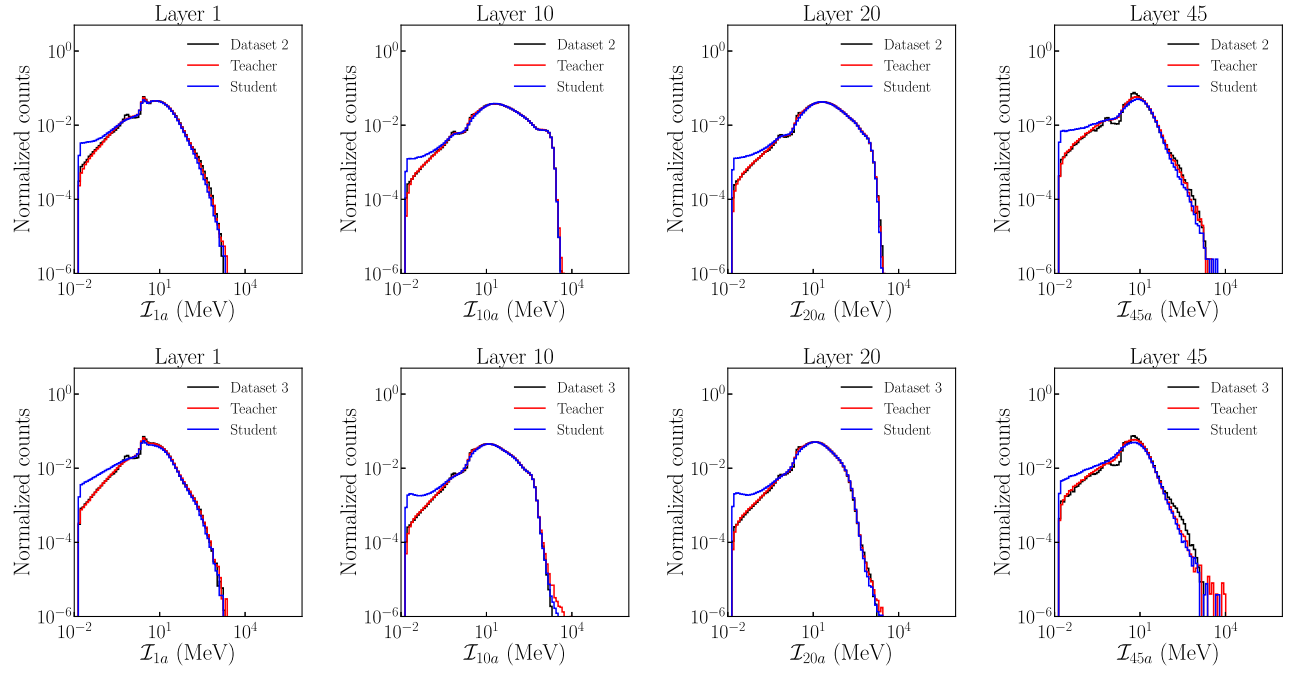


FIG. 8. Histograms of energy deposition per voxel in the layers 1, 10, 20, and 45 (from left to right) for dataset 2 (upper row) and dataset 3 (lower row). Distributions of Geant4 data are shown as black lines, and those of iCaloFlow teacher (student) trained on dataset 2 or 3 (as appropriate) in red (blue).

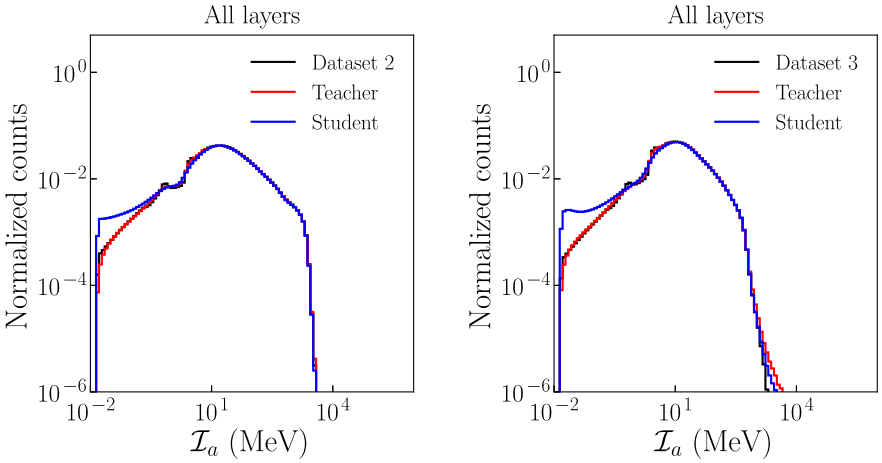


FIG. 9. Histograms of energy deposition per voxel in all layers for dataset 2 (left) and dataset 3 (right). Distributions of Geant4 data are shown as black lines, and those of iCaloFlow teacher (student) trained on dataset 2 or 3 (as appropriate) in red (blue).

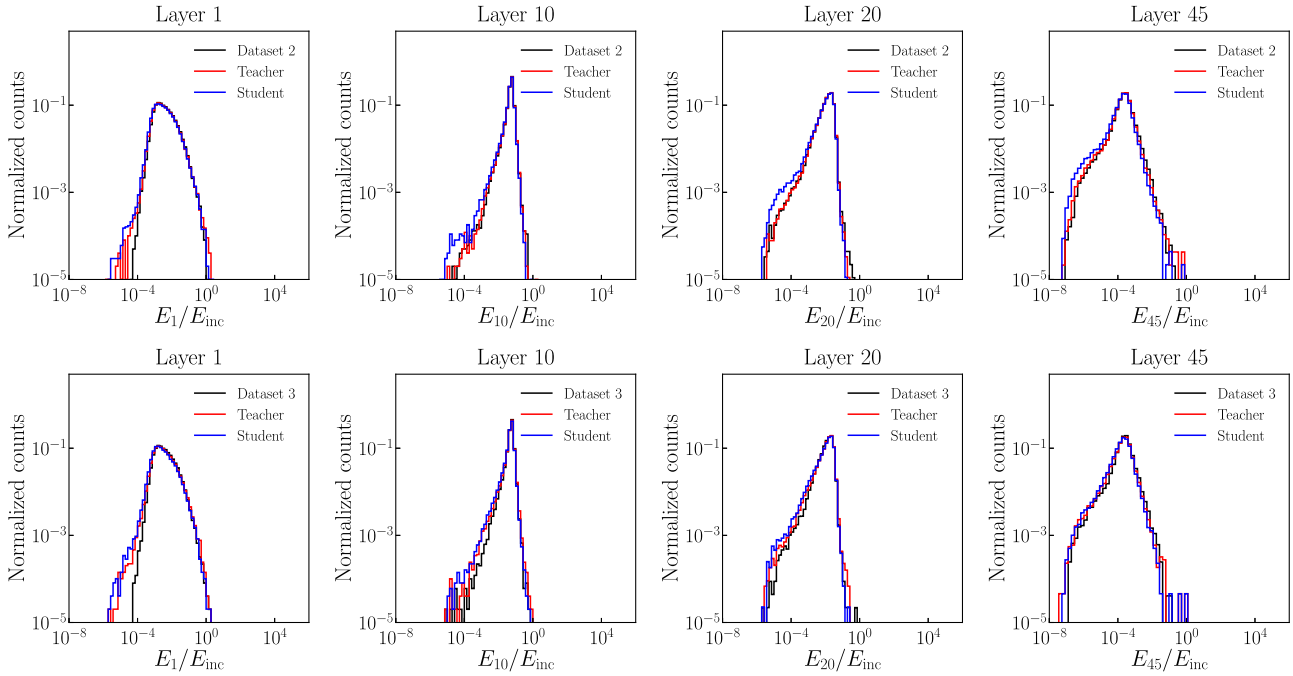


FIG. 10. Histograms of the ratio of total energy deposited and incident energy in a layer $i$ ($E_i$), for $i = 1$, 10, 20, and 45 (from left to right), for dataset 2 (top row) and dataset 3 (bottom row). Distribution of Geant4 data is shown as black lines, and that of the iCaloFlow teacher (student) trained on dataset 2 or 3 (as appropriate) in red (blue).

deposited energies, most notably in early layers (see Fig. 7 for example).

### C. Classifier metrics

The histograms and averaged deposition patterns of the generated events as compared to the Geant4 data suggest that iCaloFlow can match many of the properties of the data. Given the complexity of the events however, these simple distributions may not capture correlations within events, thereby giving a misleading impression of the accuracy of the sampler. We wish to more quantitatively determine if the generated probability distribution $p_{\text{generated}}$ is identical to that of the data $p_{\text{data}}$.

To answer this question, we follow the conventions of [17,18], and use a binary classifier trained to distinguish the Geant4 and generated events [51]. (Such binary classifiers can also be used to reweight generative models to improve their fidelity, see [52].) Table IV shows the results of 10 independent classifier runs on generated events from

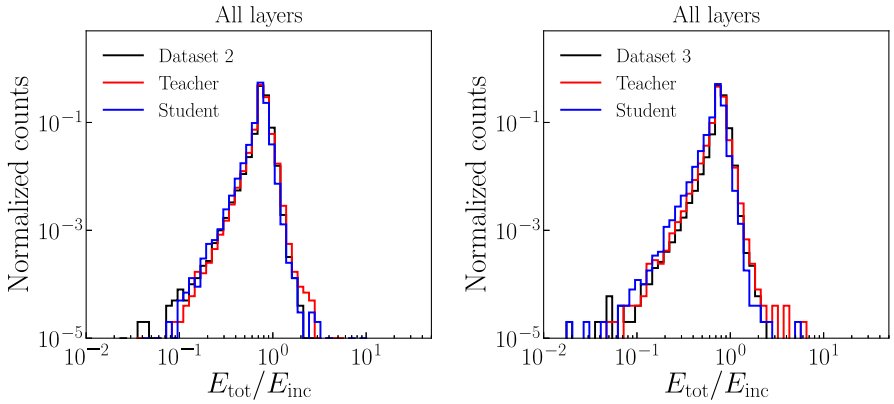


FIG. 11. Histograms of the ratio of total energy deposition (all layers) and incident energy for dataset 2 (left) and dataset 3 (right). Distributions of Geant4 data are shown as black lines, and those of iCaloFlow teacher (student) trained on dataset 2 or 3 (as appropriate) in red (blue).

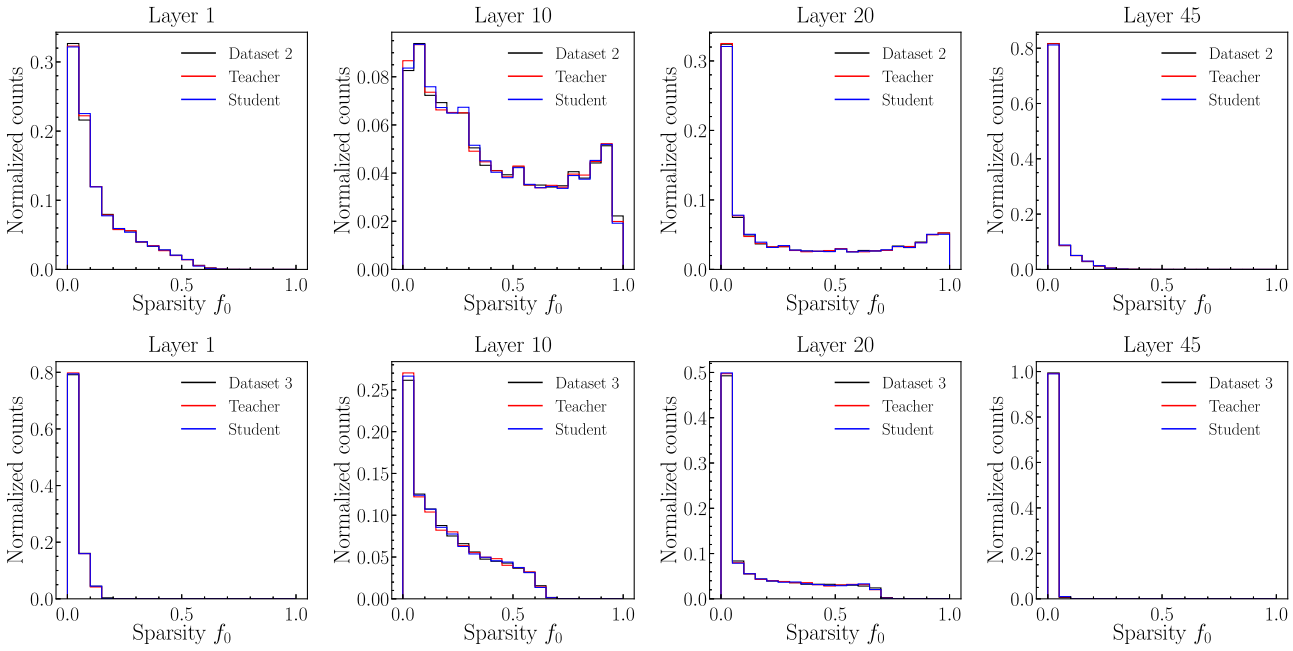


FIG. 12. Histograms of fraction of voxels in layers 1, 10, 20, and 45 (from left to right) which have nonzero energy deposition $f_0$ for dataset 2 (upper row) and dataset 3 (lower row). Distributions of Geant4 data are shown as black lines, and those of the iCaloFlow teacher (student) trained on dataset 2 or 3 (as appropriate) in red (blue).

both datasets. "Low-level features" refers to all energy depositions per voxel (normalized with $E_{\text{inc}}$ and multiplied by a factor 100) and $E_{\text{inc}}$ itself (preprocessed as $\log_{10} E_{\text{inc}}$). "High-level features" are the incident energy (preprocessed as $\log_{10} E_{\text{inc}}$), the energy deposited in each of the layers [preprocessed as $\log_{10}(E_i + 10^{-8})$], the center of energy in the $x$ and $y$ directions (normalized with a factor 100), and the width of the $x$ and $y$ distributions (normalized with a factor 100). More details on the architecture and trainings procedure can be found in Appendix B.

In Table IV, the results of the classifier runs are presented as AUC and JSD scores. According to the Neyman-Pearson lemma, we expect the AUC to be 0.5 if the true and generated probability densities are equal. The AUC is 1 if the classifier is able to perfectly distinguish between generated and true samples. The second metric, JSD $\in [0, 1]$, is the Jensen-Shannon divergence which also measures the similarity between the two probability distributions. The JSD is 0 if the two distributions are identical and 1 if they are disjoint.

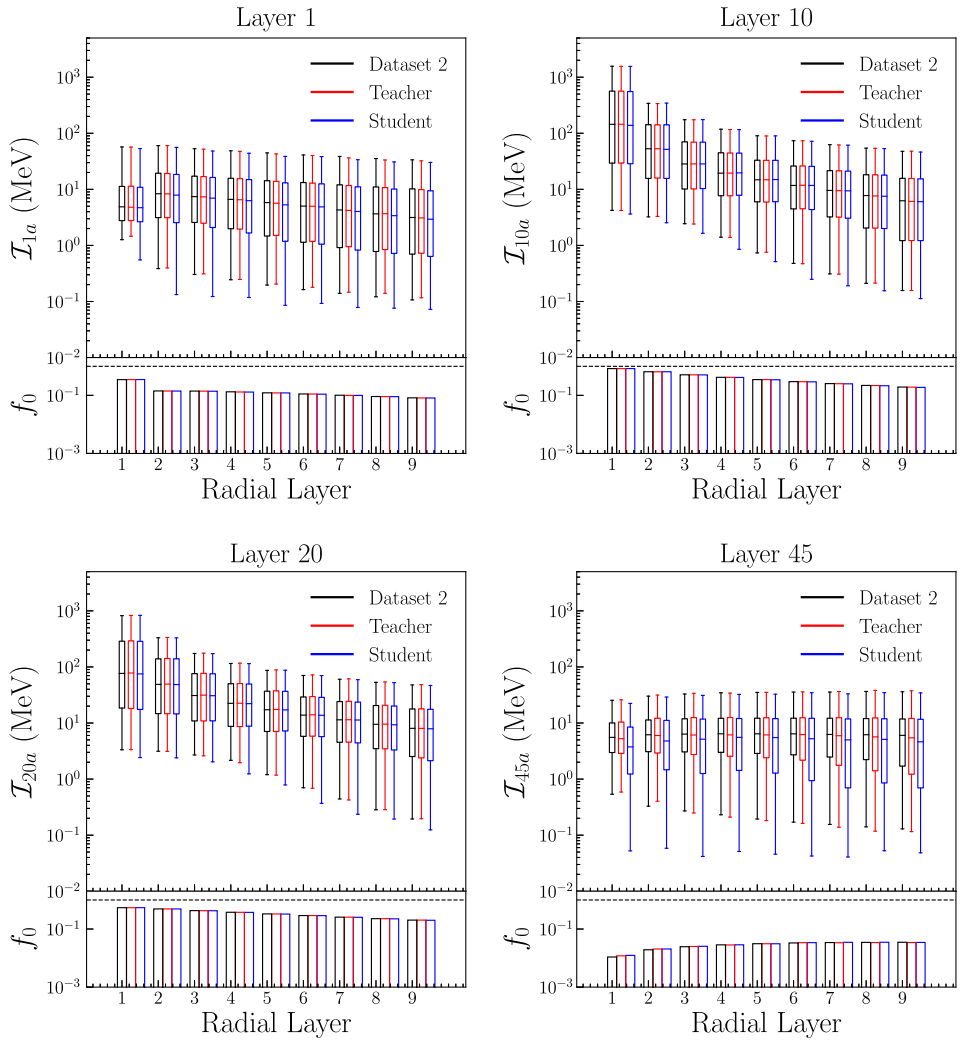


FIG. 13. Box and whisker plots showing the distribution of energy deposited in each ring of voxels at fixed radial distance from the beam line (9 such rings for dataset 2) in layers 1, 10, 20, and 45 of dataset 2. Geant4 data are shown in black, iCaloFlow teacher (student) events in red (blue). Each box extends from the first quartile of energies greater than zero to the third quartile. The whiskers extend from the 5th to 95th percentile of the nonzero energy deposition. Lower subplots show $f_0$, the average fraction of voxels in each radial ring with zero energy deposition.

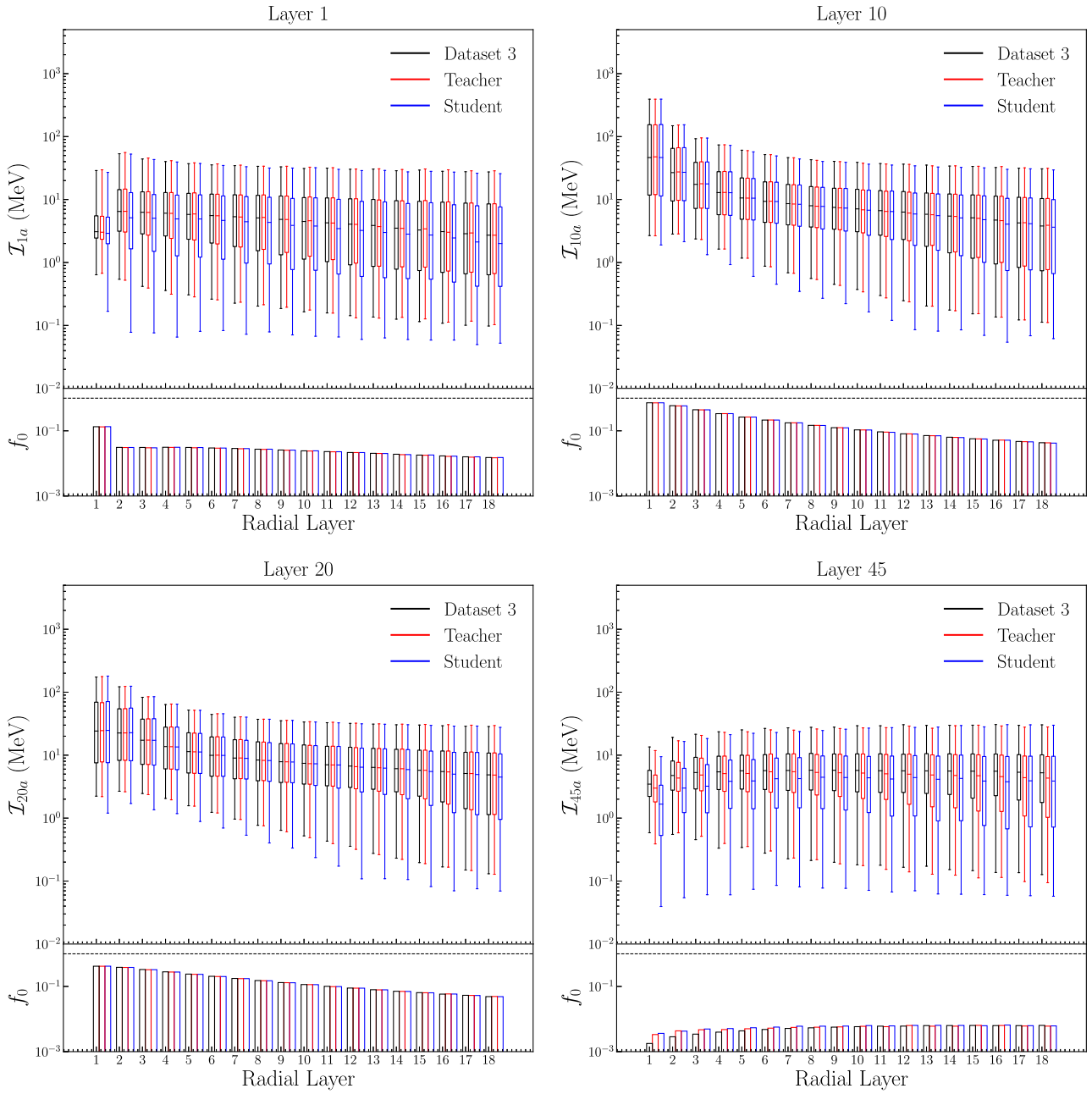


FIG. 14. Box and whisker plots showing the distribution of energy deposited in each ring of voxels at fixed radial distance from the beam line (18 such rings for dataset 3) in Layers 1, 10, 20, and 45 of dataset 3. Geant4 data are shown in blue, iCaloFlow events in red. Each box extends from the first quartile of energies greater than zero to the third quartile. The whiskers extend from the 5th to 95th percentile of the nonzero energy deposition. Lower subplots show $f_0$, the average fraction of voxels in each radial ring with zero energy deposition.

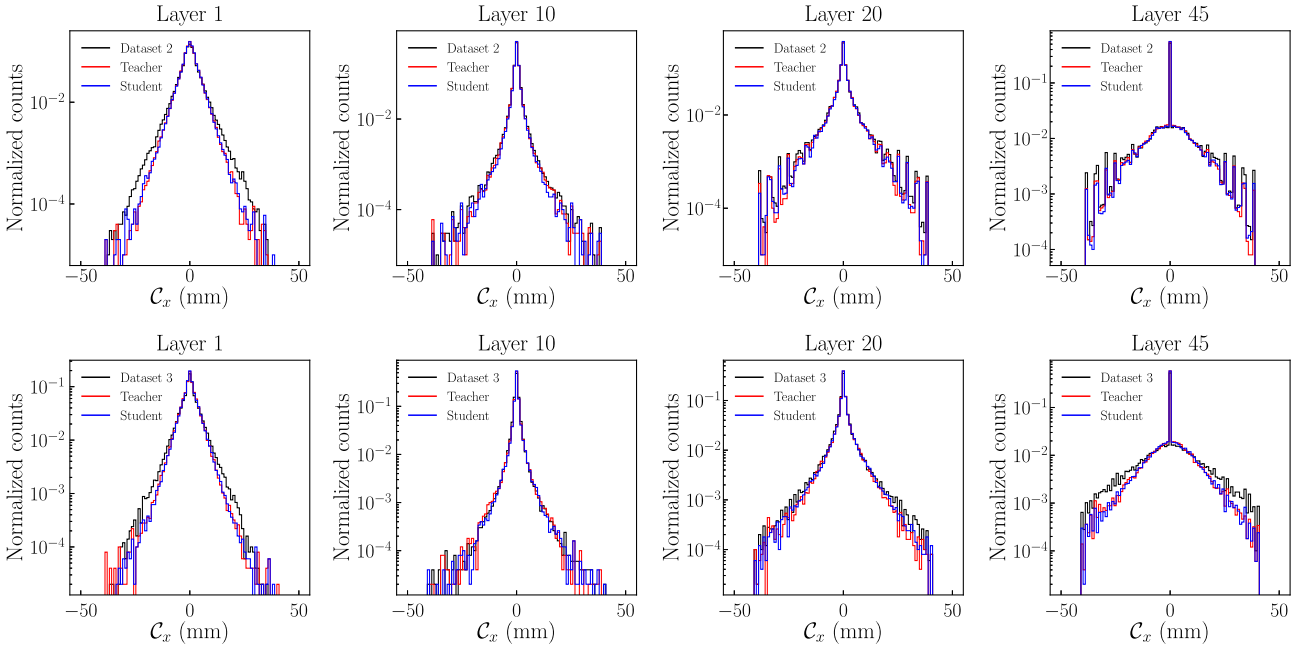


FIG. 15. Histograms of the centers of energy along the $x$-axis, $\mathcal{C}_x$, for dataset 2 (left) and dataset 3 (right). Distributions of Geant4 data are shown as black lines, and those of the iCaloFlow teacher (student) trained on dataset 2 or 3 (as appropriate) in red (blue). Due to the symmetry of the detector and incident beam, the distributions for the centers of energy in the $y$ direction ($\mathcal{C}_y$) are statistically identical to $\mathcal{C}_y$.

In our study, we find that iCaloFlow is able to produce generated samples of sufficiently high fidelity for dataset 2 to fool the classifier. This is evident by the dataset 2 teacher and student classifier scores being clearly less than unity in Table IV.

In contrast, the AUC scores for our dataset 3 teacher and student models are all >0.9, which indicates lower fidelity. Also, the high-level scores are greater than the low-level scores for our dataset 3 models. This is likely due to the low-level classifier having insufficient capacity to separate the iCaloFlow and Geant4 dataset 3 samples. It would be interesting to explore more sophisticated architectures to improve the classification performance of the low-level classifier, but we save this for future work.

In general, the teacher models performed better than the student models at the classifier tests. This is to be expected since the students require a second distillation step, and it is not surprising that some fidelity is lost in the process.

TABLE IV. Mean and standard deviation of 10 independent classifier runs.

| | Low-level features | | High-level features | |
|---|---|---|---|---|
| | AUC | JSD | AUC | JSD |
| DS2 teacher | 0.797(5) | 0.210(7) | 0.798(3) | 0.214(5) |
| DS2 student | 0.840(3) | 0.286(5) | 0.838(2) | 0.283(4) |
| DS3 teacher | 0.911(3) | 0.465(6) | 0.941(1) | 0.561(3) |
| DS3 student | 0.910(8) | 0.462(18) | 0.951(1) | 0.601(5) |

TABLE V. Average time taken to generate a single shower event by iCaloFlow teacher and student models for datasets 2 and 3. The timing was computed for different generation batch sizes on our Intel Core i9-7900X CPU and our TITAN V GPU. We were not able to generate shower events on the CPU for large batch sizes due to memory constraints.

| Model | Batch size | GPU times (ms) | CPU times (ms) |
|---|---|---|---|
| DS2 teacher | 1 | $7.12 \times 10^4$ | $8.84 \times 10^4$ |
| | 10 | $7.10 \times 10^3$ | $1.21 \times 10^4$ |
| | 100 | $7.75 \times 10^2$ | $3.47 \times 10^3$ |
| | 1000 | $1.65 \times 10^2$ | ⋯ |
| | 10000 | $1.21 \times 10^2$ | ⋯ |
| DS2 student | 1 | $1.06 \times 10^3$ | $4.08 \times 10^3$ |
| | 10 | $2.79 \times 10^2$ | $4.08 \times 10^2$ |
| | 100 | $2.99 \times 10^1$ | $4.93 \times 10^1$ |
| | 1000 | 2.18 | ⋯ |
| | 10000 | **1.34** | ⋯ |
| DS3 teacher | 1 | $4.76 \times 10^5$ | $3.86 \times 10^6$ |
| | 10 | $5.28 \times 10^4$ | $3.31 \times 10^5$ |
| | 100 | $9.17 \times 10^3$ | ⋯ |
| | 1000 | $5.17 \times 10^3$ | ⋯ |
| | 5000 | $4.57 \times 10^3$ | ⋯ |
| DS3 student | 1 | $1.29 \times 10^3$ | $3.36 \times 10^3$ |
| | 10 | $1.33 \times 10^2$ | $6.22 \times 10^2$ |
| | 100 | $3.58 \times 10^1$ | ⋯ |
| | 1000 | 8.84 | ⋯ |
| | 5000 | **5.87** | ⋯ |

Finally, we note that the authors of CaloScore [23] have produced results for datasets 2 and 3 using a score-based diffusion model. They performed a similar classifier test on their samples and obtained an AUC of 0.98 for both datasets.

### D. Timing

Shower generation timings for different generation batch sizes are shown in Table V. The speedups going from teacher to student are the expected $\mathcal{O}(d)$, where $d$ is the dimensionality of a single layer (144 for DS2 and 900 for DS3). On the GPU, iCaloFlow is generally faster than Geant4, which takes approximately 100 s per event, except for dataset 3 teacher with a generation batch size of 1. On the CPU, iCaloFlow is still faster than Geant4 for both dataset 2 models and also dataset 3 student. With the student models, we were able to generate a single shower on the GPU as quickly as 1.34 ms (5.87 ms) for dataset 2 (3)—$\mathcal{O}(10^5)$ faster than Geant4.

## V. CONCLUSIONS

In this paper, we have introduced a new approach to generating highly-granular calorimeter showers with normalizing flows. This is the first flow-based approach that has been applied to calorimeter shower data with dimensionality as large as dataset 3 of *CaloChallenge2022*. Instead of learning the joint distribution of all voxels, we focus on the distribution per calorimeter layer and use an inductive algorithm to generate the sample. This approach is inspired by the physical nature of the shower, which starts to develop close to the entry point of the incoming particle and evolves deeper into the detector.

Previous approaches used separately trained flows for each layer [28], while we use a single flow for the step of generating layer $i$ based on the shower in layer $i-1$ (treating $i=1$ separately), in the spirit of a mathematical proof by induction. The advantage of this approach is in its much more efficient setup. Instead of $45+1$ normalizing flows, we need only three. This efficiency (along with the inductive approach) would be preserved even if we increased the size of Flow-3 to be much larger than Flow-2 and the flows of [28]. However, the downside of our inductive approach is that training Flow-3 cannot be trivially parallelized across multiple GPUs (as the equivalents were in [28]). Also, sharing the same flow across all layers is presumably less expressive than using one flow for each layer.

In both L2LFlows and iCaloFlow, event generation must occur sequentially through each layer. On top of this, the slow sampling from the MAF strongly motivates the added step of distilling them into student IAF networks. Compared to the teachers, the students speed up event production by a factor of approximately the number of voxels in a layer—though it is still necessary to iterate through all the layers. iCaloFlow is the first work to include a teacher-student pairing in any CaloFlow-like setup for higher dimensional calorimeter shower data such as datasets 2 and 3.

At this time, no exhaustive comparison with other generative neural networks is possible. The most natural comparison point, the algorithm of [28], is applied to a different data set. During the completion of *CaloChallenge2022*, iCaloFlow will be compared against other codes, and a more complete understanding of its strengths and weaknesses will be available.

A possible limitation to this inductive algorithm is generalizing it to nonregular calorimeter geometries. In iCaloFlow, Flow-3 requires each layer to be identical in structure, allowing the efficient weight-sharing and training that are the major advantages of the algorithm. Significant modifications would be required to adapt to layer-dependent voxelization. While this might not be possible for arbitrary geometries, some variations (for example, breaking a large voxel in one layer into groups of small voxels in other layers) could admit an inductive flow approach.

Even with these caveats, the efficiency, event generation speed (when using the IAF students), and relatively high fidelity of iCaloFlow make it a competitive architecture for fast calorimeter event generation. One promising future direction is to realize a similar inductive setup based on coupling-layer flows, which are equally fast in density estimation and generation. This would presumably reduce training time as we would not have to train both teacher and student flows.

## ACKNOWLEDGMENTS

We thank Gopolang Mohlabeng for discussion in the early stages of this project. This work was supported by DOE grant DOE-SC0010008. C. K. would like to thank the Baden-Württemberg-Stiftung for support through the program *Internationale Spitzenforschung*, project *Uncertainties—Teaching AI its Limits* (BWST_IF2020-010). In this work, we used the NumPy 1.16.4 [53], Matplotlib 3.1.0 [54], SCIKIT-LEARN 0.21.2 [55], H5PY 2.9.0 [56], PyTorch 1.11.1 [57], and nFlows 0.14 [58] software packages.

## APPENDIX A: PRE- AND POSTPROCESSING

The incident energy is a conditional for all three flows. In all cases, it is transformed from the physical range (1 GeV to 1 TeV) to normalized log-space:

$$E_{\text{inc}} \to \log_{10} \frac{E_{\text{inc}}}{10^{4.5}\ \text{MeV}} \in [-1.5, 1.5]. \tag{A1}$$

For Flow-1, the total energy deposited in layer $i$ ($E_i$) is obtained from summing all voxels of the layer. As the flow has difficulty learning a distribution where many elements are exactly zero, uniform noise in the range [0, 5] keV is added to $E_i$ [17]. We then divided by 65 GeV (this is

slightly larger than the maximum energy deposition in any layer in any of the events of the dataset):

$$E_i \to x_i \equiv (E_i + \text{rand}[0, 5\ \text{keV}])/65\ \text{GeV}. \quad \text{(A2)}$$

As a final step, we apply a logit transformation:

$$y_i = \log \frac{u_i}{1-u_i}, \qquad u_i \equiv \alpha + (1-2\alpha)x_i, \quad \text{(A3)}$$

where the offset $\alpha \equiv 10^{-6}$ ensures that the boundaries $x_i = 0$ and 1 map to finite numbers.

For both Flow-2 and Flow-3, we apply the same processing steps for the energy depositions per voxel, $\mathcal{I}_{ia}$. First, we add uniform noise in the range [0, 5] keV to every voxel. Then, we normalize the voxel by the sum of all voxels of the given layer and then apply the logit transformation of Eq. (A3) to it:

$$\begin{aligned} \mathcal{I}_{ia} &\to \mathcal{I}_{ia} + \text{rand}[0, 5\ \text{keV}] \\ \mathcal{I}_{ia} &\to \mathcal{I}_{ia} / \sum_b \mathcal{I}_{ib} \equiv \hat{\mathcal{I}}_{ia} \\ u_{ia} &= \alpha + (1-2\alpha)\hat{\mathcal{I}}_{ia} \\ y_{ia} &= \log \frac{u_{ia}}{1-u_{ia}}. \end{aligned} \quad \text{(A4)}$$

When generating new events from our trained flows, the preprocessing steps are inverted, and normalized showers are transformed back to physical space using the proxy output of Flow-1. Voxel energies below the detector threshold of 15 keV are set to zero.

Flow-2 is conditioned on the energy deposited in the first layer, $E_1$. This is preprocessed as in Eqs. (A2) and (A3), with the additional step of dividing $y_1$ by 4:

$$\begin{aligned} E_1 &\to x_i \equiv (E_i + \text{rand}[0, 5\ \text{keV}])/65\ \text{GeV} \\ u_1 &\equiv \alpha + (1-2\alpha)x_1 \\ y_1 &= \frac{1}{4}\log \frac{u_1}{1-u_1}. \end{aligned} \quad \text{(A5)}$$

This final division by 4 is to bring the range of $y_1$ down to $\mathcal{O}(1)$.

The conditional inputs for Flow-3 are preprocessed as follows: $\mathcal{I}_{(i-1)a}$ follows the steps of Eq. (A4). $E_i$ and $E_{(i-1)}$ follow Eqs. (A2) and (A3) (without the factor 4 of Flow-2). The layer number $i$ is one-hot encoded in a vector of length 44.

When sampling new events from iCaloFlow, these transformations are inverted. As a final step, the detector energy threshold is applied on the generated voxel energies by setting all voxels with $\mathcal{I}_{ia} < 15$ keV to zero. As noted previously, when generating new patterns of energy deposition from Flow-2 and Flow-3, the generated $\hat{\mathcal{I}}_{ia}$ are not enforced to sum to one. Forcing them to sum to one, as was the choice made in the previous iterations of CaloFlow [17,18,26], here causes problems in the voxel energy histograms in Figs. 8 and 9. It enhances low-energy voxels above the cut-off threshold, leading to a large excess at the lower end of the voxel energy histograms. This was less of a problem previously, for dataset 1 and for the CaloGAN dataset, because the normalization was learned better, perhaps because these datasets were lower dimensional. Meanwhile, in L2LFlows [28], which was trained on a dataset comparable in dimensionality to dataset 2, the voxel energies were normalized by maximum voxel energy in the dataset, so no renormalization with $E_i$ was needed and this issue was avoided. However, we found that normalizing by maximum voxel energy did not improve our results here.

## APPENDIX B: CLASSIFIER

The performance metric based on the classifier test uses the neural network architecture that was provided by the *CaloChallenge2022* [31,59] evaluation script. It is based on the classifiers that were used in [17,18] and was also used (with slightly different hyperparameters) in [26]. To be precise, the classifier is a deep, fully-connected neural network with an input and two hidden layers with 2048 nodes each. All activation functions are leaky ReLUs, with default negative slope of 0.01, except the output layer, which uses a sigmoid activation function for its single output number. We do not use any regulators such as batch normalization or dropout.

The input for low- and high-level feature classification is preprocessed as described in the main text. For dataset 2, we work with the second provided file of 100,000 showers [33]. We split it in train/test/validation sets of the ratio (60:20:20). For dataset 3, we use the third file provided at [34] split in the ratio (40:10) for training and model selection and the full fourth file of [34] to obtain the final scores.

All networks are optimized by training 50 epochs with an Adam [48] optimizer with initial learning rate of $2 \times 10^{-4}$ and a batch size of 1000 (250) for datasets 2 (3), minimizing the binary cross entropy. We use the model state with the highest accuracy on the validation set for the final evaluation and we subsequently calibrate the classifier using isotonic regression [60] of `sklearn` [55] based on the validation dataset before evaluating the test set.

[1] *The High Luminosity Large Hadron Collider: The New Machine for Illuminating the Mysteries of Universe*, edited by O. Brüning and L. Rossi (World Scientific, Singapore, 2015), 10.1142/9581.

[2] P. Calafiura, J. Catmore, D. Costanzo, and A. Di Girolamo, ATLAS HL-LHC computing conceptual design report, Technical Reports No. CERN-LHCC-2020-015, No. LHCC-G-178, CERN, Geneva, 2020.

[3] CMS Collaboration, CMS phase-2 computing model: Update document, Technical Reports No. CMS-NOTE-2022-008, No. CERN-CMS-NOTE-2022-008, CERN, Geneva, 2022, https://cds.cern.ch/record/2815292.

[4] ATLAS Collaboration, ATLAS software and computing HL-LHC roadmap, Technical Reports No. CERN-LHCC-2022-005, No. LHCC-G-182, CERN, Geneva, 2022, https://cds.cern.ch/record/2802918.

[5] HEP Software Foundation Collaboration, HL-LHC computing review: Common tools and community software, in *2022 Snowmass Summer Study*, edited by P. Canal *et al.* (2020), arXiv:2008.13636.

[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] 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).

[12] 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).

[13] 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).

[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] ATLAS Collaboration, Fast simulation of the ATLAS calorimeter system with generative adversarial networks, Technical Report No. ATL-SOFT-PUB-2020-006, CERN, Geneva, 2020, http://cds.cern.ch/record/2746032.

[17] C. Krause and D. Shih, CaloFlow: Fast and accurate generation of calorimeter showers with normalizing flows, Phys. Rev. D **107**, 113003 (2023).

[18] C. Krause and D. Shih, CaloFlow II: Even faster and still accurate generation of calorimeter showers with normalizing flows, Phys. Rev. D **107**, 113004 (2023).

[19] 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).

[20] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, D. Hundhausen, G. Kasieczka *et al.*, Fast and accurate electromagnetic and hadronic showers from generative models, EPJ Web Conf. **251**, 03049 (2021).

[21] E. Buhmann, S. Diefenbacher, D. Hundhausen, G. Kasieczka, W. Korcari, E. Eren *et al.*, Hadrons, better, faster, stronger, Mach. Learn. Sci. Tech. **3**, 025014 (2022).

[22] ATLAS Collaboration, AtlFast3: The next generation of fast simulation in ATLAS, Comput. Software Big Sci. **6**, 7 (2022).

[23] V. Mikuni and B. Nachman, Score-based generative models for calorimeter shower simulation, Phys. Rev. D **106**, 092009 (2022).

[24] ATLAS Collaboration, Deep generative models for fast photon shower simulation in ATLAS, arXiv:2210.06204.

[25] A. Adelmann *et al.*, New directions for surrogate models and differentiable programming for high energy physics detector simulation, in *2022 Snowmass Summer Study* (2022), arXiv:2203.08806.

[26] C. Krause, I. Pang, and D. Shih, CaloFlow for CaloChallenge dataset 1, arXiv:2210.14245.

[27] J. C. Cresswell, B. L. Ross, G. Loaiza-Ganem, H. Reyes-Gonzalez, M. Letizia, and A. L. Caterini, CaloMan: Fast generation of calorimeter showers with density estimation on learned manifolds, in *Proceedings of the 36th Conference on Neural Information Processing Systems* (2022), https://ml4physicalsciences.github.io/2022/.

[28] S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, C. Krause, I. Shekhzadeh, and D. Shih, L2LFlows: Generating high-fidelity 3D calorimeter images, J. Instrum. **18**, P10017 (2023).

[29] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, W. Korcari, K. Krüger, and P. McKeown, CaloClouds: Fast geometry-independent highly-granular calorimeter simulation, J. Instrum. **18**, P11025 (2023).

[30] H. Hashemi, N. Hartmann, S. Sharifzadeh, J. Kahn, and T. Kuhr, Ultra-high-resolution detector simulation with intra-event aware GAN and self-supervised relational reasoning, arXiv:2303.08046.

[31] M. Faucci Giannelli, G. Kasieczka, B. Nachman, D. Salamani, D. Shih, and A. Zaborowska, Fast calorimeter simulation challenge 2022, https://calochallenge.github.io/homepage/ (2022).

[32] M. F. Giannelli, G. Kasieczka, C. Krause, B. Nachman, D. Salamani, D. Shih *et al.*, Fast calorimeter simulation challenge 2022—dataset 1, 10.5281/zenodo.6234054 (2022).

[33] M. F. Giannelli, G. Kasieczka, C. Krause, B. Nachman, D. Salamani, D. Shih *et al.*, Fast calorimeter simulation challenge 2022—dataset 2, 10.5281/zenodo.6366271 (2022).

[34] M. F. Giannelli, G. Kasieczka, C. Krause, B. Nachman, D. Salamani, D. Shih *et al.*, Fast calorimeter simulation challenge 2022—dataset 3, 10.5281/zenodo.6366324 (2022).
[35] L. Dinh, D. Krueger, and Y. Bengio, NICE: Non-linear independent components estimation, arXiv:1410.8516.
[36] L. Dinh, J. Sohl-Dickstein, and S. Bengio, Density estimation using real NVP, arXiv:1605.08803.
[37] D. Rezende and S. Mohamed, Variational inference with normalizing flows, in *International Conference on Machine Learning* (PMLR, 2015), pp. 1530–1538, https://proceedings.mlr.press/v37/rezende15.html.
[38] I. Kobyzev, S. J. Prince, and M. A. Brubaker, Normalizing flows: An introduction and review of current methods, IEEE Trans. Pattern Anal. Mach. Intell. **43**, 3964 (2020).
[39] G. Papamakarios, E. T. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, Normalizing flows for probabilistic modeling and inference, J. Mach. Learn. Res. **22**, 1 (2021), https://www.jmlr.org/papers/v22/19-1028.html.
[40] H. Abramowicz *et al.*, The international linear collider technical design report—Volume 4: Detectors, arXiv:1306.6329.
[41] The ILD Concept Group, International large detector: Interim design report, arXiv:2003.01116.
[42] G. Papamakarios, T. Pavlakou, and I. Murray, Masked autoregressive flow for density estimation, in *Advances in Neural Information Processing Systems* (2017), Vol. 30, https://papers.nips.cc/paper_files/paper/2017.
[43] A. van den Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu *et al.*, Parallel WaveNet: Fast high-fidelity speech synthesis, in *Proceedings of the 35th International Conference on Machine Learning*, edited by J. Dy and A. Krause, Vol. 80 of Proceedings of Machine Learning Research (PMLR, 2018), pp. 3918–3926, https://proceedings.mlr.press/v80/oord18a.html.
[44] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, Improved variational inference with inverse autoregressive flow, in *Advances in Neural Information Processing Systems* (2016), Vol. 29, https://papers.nips.cc/paper_files/paper/2016.
[45] A. Zaborowska (private communication).
[46] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, Neural spline flows, in *Advances in Neural Information Processing Systems* (2019), Vol. 32, p. 7511, https://papers.nips.cc/paper_files/paper/2019.
[47] M. Germain, K. Gregor, I. Murray, and H. Larochelle, Made: Masked autoencoder for distribution estimation, in *International Conference on Machine Learning* (PMLR, 2015), pp. 881–889, https://proceedings.mlr.press/v37/germain15.html.
[48] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980.
[49] L. N. Smith and N. Topin, Super-convergence: Very fast training of neural networks using large learning rates, in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications* (SPIE, Baltimore, MD, 2019), Vol. 11006, pp. 369–386, https://www.spiedigitallibrary.org/conference-proceedings-of-spie/11006/1100612/Super-convergence--very-fast-training-of-neural-networks-using/10.1117/12.2520589.short.
[50] C.-W. Huang, F. Ahmed, K. Kumar, A. Lacoste, and A. Courville, Probability distillation: A caveat and alternatives, in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, edited by R. P. Adams and V. Gogate, Vol. 115 of Proceedings of Machine Learning Research (PMLR, 2020), pp. 1212–1221, https://proceedings.mlr.press/v115/huang20c.html.
[51] D. Lopez-Paz and M. Oquab, Revisiting classifier two-sample tests, arXiv:1610.06545.
[52] 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).
[53] 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).
[54] J. D. Hunter, Matplotlib: A 2d graphics environment, Comput. Sci. Eng. **9**, 90 (2007).
[55] 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://www.jmlr.org/papers/v12/pedregosa11a.html.
[56] A. Collette, *Python and HDF5* (O'Reilly, 2013), ISBN 9781449367831.
[57] 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.
[58] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, nFlows: Normalizing flows in PyTorch, 10.5281/zenodo.4296287 (2020).
[59] M. Faucci Giannelli, G. Kasieczka, B. Nachman, D. Salamani, D. Shih, and A. Zaborowska, Fast calorimeter simulation challenge 2022 github page, https://github.com/CaloChallenge/homepage (2022).
[60] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, On calibration of modern neural networks, arXiv:1706.04599.