Fermilab

# CaloChallenge 2022: A Community Challenge for Fast Calorimeter Simulation

FERMILAB-PUB-24-0728-CMS

arXiv:2410.21611

DOI: 10.1088/1361-6633/ae1304

Fermilab Accepted Manuscript

This manuscript has been authored by Fermi Research Alliance, LLC
under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy,
Office of Science, Office of High Energy Physics.

arXiv:2410.21611v2 [physics.ins-det] 13 Nov 2025

Review Article

# CaloChallenge 2022: A Community Challenge for Fast Calorimeter Simulation



"Calorimeter Simulation", generated via midjourney, 2022

**Claudius Krause**[1,2] **(main editor)**,
**Michele Faucci Giannelli**[3,4] **(editor)**,
**Gregor Kasieczka**[5] **(editor)**, **Benjamin Nachman**[6] **(editor)**,
**Dalila Salamani**[7] **(editor)**, **David Shih**[8] **(editor)**,
**Anna Zaborowska**[7] **(editor)**,

**Oz Amram**[9], **Kerstin Borras**[10,11], **Matthew R. Buckley**[8],
**Erik Buhmann**[5], **Thorsten Buss**[5,10],
**Renato Paulo Da Costa Cardoso**[7], **Anthony L. Caterini**[12],
**Nadezda Chernyavskaya**[7], **Federico A.G. Corchia**[13,14],
**Jesse C. Cresswell**[12], **Sascha Diefenbacher**[6],
**Etienne Dreyer**[15], **Vijay Ekambaram**[16], **Engin Eren**[10],
**Florian Ernst**[2,7], **Luigi Favaro**[2], **Matteo Franchini**[13,14],
**Frank Gaede**[10], **Eilam Gross**[15], **Shih-Chieh Hsu**[17],
**Kristina Jaruskova**[7], **Benno Käch**[5,10], **Jayant Kalagnanam**[18],
**Raghav Kansal**[9,19], **Taewoo Kim**[12], **Dmitrii Kobylianskii**[15],
**Anatolii Korol**[10], **William Korcari**[5], **Dirk Krücker**[10],

**Katja Krüger**[10], **Marco Letizia**[20,21], **Shu Li**[22,23,24], **Qibin Liu**[22,23,24], **Xiulong Liu**[17], **Gabriel Loaiza-Ganem**[12], **Thandikire Madula**[25], **Peter McKeown**[7,10], **Isabell-A. Melzer-Pellmann**[10], **Vinicius Mikuni**[6], **Nam Nguyen**[18], **Ayodele Ore**[2], **Sofia Palacios Schweitzer**[2], **Ian Pang**[8], **Kevin Pedro**[9], **Tilman Plehn**[2], **Witold Pokorski**[7], **Huilin Qu**[7], **Piyush Raikwar**[7], **John A. Raine**[26], **Humberto Reyes-Gonzalez**[21,27,28], **Lorenzo Rinaldi**[13,14], **Brendan Leigh Ross**[12], **Moritz A.W. Scham**[10,11,29], **Simon Schnake**[10,11], **Chase Shimmin**[30], **Eli Shlizerman**[17], **Nathalie Soybelman**[15], **Mudhakar Srivatsa**[18], **Kalliopi Tsolaki**[7], **Sofia Vallecorsa**[7], **Kyongmin Yeo**[18], **Rui Zhang**[31,32]

[1] Institute of High Energy Physics (HEPHY), Austrian Academy of Sciences (OeAW), Dominikanerbastei 16, A-1010 Vienna, Austria

E-mail: Claudius.Krause@oeaw.ac.at

[2] Institut für Theoretische Physik, Universität Heidelberg, Germany

[3] Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Roma Tor Vergata, Roma, 00133, Italy

[4] Department of Microtechnology and Nanoscience, Chalmers University of Technology, 41296 Gothenburg, Sweden

[5] Institut für Experimentalphysik, Universität Hamburg, Germany

[6] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

[7] CERN, Espl. des Particules 1, 1211 Meyrin, Switzerland

[8] NHETC, Department of Physics and Astronomy, Rutgers University, Piscataway, NJ 08854, USA

[9] Fermi National Accelerator Laboratory, Batavia, IL 60510, USA

[10] Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

[11] III. Physikalisches Institut A, RWTH Aachen University, Germany

[12] Layer 6 AI, Toronto, Canada

[13] Department of Physics and Astronomy, Alma Mater Studiorum - University of Bologna, 6/2, Viale Carlo Berti Pichat, I-40127 Bologna, Italy

[14] Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Bologna, 6/2, Viale Carlo Berti Pichat, I-40127 Bologna, Italy

[15] Weizmann Institute of Science, Rehovot, Israel

[16] IBM Research, India

[17] University of Washington, Seattle, WA 98195, USA

[18] IBM T. J. Watson Research Center, Yorktown Heights, NY USA

[19] California Institute of Technology, Pasadena, CA 91125, USA

[20] MaLGa–DIBRIS, University of Genova, Genova, Italy

[21] Istituto Nazionale di Fisica Nucleare (INFN), Sezione di Genova, Genova, Italy

[22] Tsung-Dao Lee Institute (TDLI), Shanghai Jiao Tong University, Shanghai 201210, China

[23] Key Laboratory for Particle Astrophysics and Cosmology (MOE), Shanghai Key Laboratory for Particle Physics and Cosmology (SKLPPC), Shanghai Jiao Tong University, Shanghai 200240, China

[24] Institute of Nuclear and Particle Physics, School of Physics and Astronomy, Shanghai Jiao Tong University, Shanghai 200240, China
[25] University College London (UCL), London, WC1E 6BT, UK
[26] Département de Physique Nucléaire et Corpusculaire, University of Geneva, 1211 Geneva, Switzerland
[27] Department of Physics, University of Genova, Genova, Italy
[28] Institut für Theoretische Teilchenphysik und Kosmologie, RWTH Aachen University, 52074 Aachen, Germany
[29] Institute for Advanced Simulation, Forschungszentrum Jülich, Jülich, Germany
[30] Yale University, New Haven, CT 06520, USA
[31] Department of Physics, Nanjing University, Nanjing 210093, China
[32] Department of Physics, University of Wisconsin Madison, Wisconsin 53706, USA

**Abstract.** We present the results of the "Fast Calorimeter Simulation Challenge 2022" — the CaloChallenge. We study state-of-the-art generative models on four calorimeter shower datasets of increasing dimensionality, ranging from a few hundred voxels to a few tens of thousand voxels. The 31 individual submissions span a wide range of current popular generative architectures, including Variational AutoEncoders (VAEs), Generative Adversarial Networks (GANs), Normalizing Flows, Diffusion models, and models based on Conditional Flow Matching. We compare all submissions in terms of quality of generated calorimeter showers, as well as shower generation time and model size. To assess the quality we use a broad range of different metrics including differences in 1-dimensional histograms of observables, KPD/FPD scores, AUCs of binary classifiers, and the log-posterior of a multiclass classifier. The results of the CaloChallenge provide the most complete and comprehensive survey of cutting-edge approaches to calorimeter fast simulation to date. In addition, our work provides a uniquely detailed perspective on the important problem of how to evaluate generative models. As such, the results presented here should be applicable for other domains that use generative AI and require fast and faithful generation of samples in a large phase space.

*Report Numbers*: HEPHY-ML-24-05, FERMILAB-PUB-24-0728-CMS, TTK-24-43

Submitted to: *Rep. Prog. Phys.*

**Contents**

**1 Introduction** **6**

**2 Datasets** **9**
2.1 Dataset 1 Photons and Pions . . . . . 10
2.2 Datasets 2 and 3 . . . . . 11

**3 GAN-based Submissions** **13**
3.1 CaloShowerGAN . . . . . 13
3.2 Matching Deep Mean-field Attentive (MDMA) GAN . . . . . 16
3.3 BoloGAN . . . . . 18
3.4 DeepTree . . . . . 20

**4 Normalizing Flow-based Submissions** **24**
4.1 L2LFlows . . . . . 25
4.2 (inductive) CaloFlow . . . . . 28
4.3 CaloINN . . . . . 31
4.4 SuperCalo . . . . . 33
4.5 CaloPointFlow . . . . . 34

**5 Diffusion-based Submissions** **37**
5.1 CaloDiffusion with GLaM . . . . . 38
5.2 CaloClouds: Fast Geometry-Independent Highly-Granular Calorimeter Simulation . . . . . 39
5.3 Score-based Generative Models for Calorimeter Shower Simulation . . . . 41
5.4 CaloGraph . . . . . 42
5.5 Diffusion Transformer . . . . . 44

**6 VAE-based Submissions** **46**
6.1 Latent Generative Models for Calo Simulation with VQ-VAE . . . . . 47
6.2 CaloMan: Fast Generation of Calorimeter Showers with Density Estimation on Learned Manifolds . . . . . 50
6.3 DNN CaloSim . . . . . 55
6.4 Geant4 Transformer . . . . . 56
6.5 CaloVAE+INN . . . . . 59
6.6 CaloLatent: Score-based Generative Modeling in the Latent Space for Calorimeter Shower Generation . . . . . 62

**7 Conditional Flow Matching-based Submissions** **64**
7.1 CaloDREAM: Vision Transformer CFM . . . . . 65
7.2 CaloForest . . . . . 67

**8 Introduction to metrics** **71**
8.1 High-level Features (Histograms) . . . 71
8.2 Correlations . . . 72
8.3 Classifier-based Metrics. . . . 72
8.4 Computer Science-inspired Metrics . . . 75
8.5 Manifold-based Metrics . . . 76
8.6 Generation Timings . . . 77
8.7 Memory Requirements . . . 78

**9 Results: Individual Metrics** **78**
9.1 Preprocessing . . . 79
9.2 Dataset 1, Photons (ds 1 – $\gamma$) . . . 79
9.3 Dataset 1, Pions (ds 1 – $\pi^+$) . . . 88
9.4 Dataset 2, Electrons (ds 2) . . . 96
9.5 Dataset 3, Electrons (ds 3) . . . 106

**10 Results: Correlations Between Metrics** **114**
10.1 Metric Comparison . . . 115
10.2 Pareto Fronts . . . 124

**11 Conclusions and Outlook** **128**
11.1 Overall Physics Results . . . 129
11.2 Take-aways of the CaloChallenge beyond Detector Simulation . . . 131
11.3 Outlook to the Future . . . 132

**A Histograms of high-level features** **135**
A.1 Dataset 1, Photons (ds 1 – $\gamma$) . . . 135
A.2 Dataset 1, Pions (ds 1 – $\pi^+$) . . . 139
A.3 Dataset 2, Electrons (ds 2) . . . 143
A.4 Dataset 3, Electrons (ds 3) . . . 152

**B Consistency check of the multiclass classifier** **161**

**C Numerical Results in Tables** **168**
C.1 Dataset 1, Photons (ds 1 – $\gamma$) . . . 168
C.2 Dataset 1, Pions (ds 1 – $\pi^+$) . . . 174
C.3 Dataset 2, Electrons (ds 2) . . . 177
C.4 Dataset 3, Electrons (ds 3) . . . 186

**D Generation time vs. number of parameters** **192**

**References** **194**

## 1. Introduction

At the Large Hadron Collider (LHC) and countless other particle or nuclear physics facilities, we aim to study Nature at the most fundamental level, searching for answers to questions such as the nature of dark matter and dark energy, the baryon-anti-baryon asymmetry in the universe, and the mass and hierarchy of neutrinos, which are all not explained in the Standard Model. Simulations based on first principles provide a crucial bridge between theory and experiment and are at the core of the successful physics program of these facilities. With the increasing amount of data that the LHC will generate in the upcoming runs, the amount of simulated events required for accurate and sensitive analyses will grow steadily, and with it the computational resources needed to generate them. In figure 1, we see the projected CPU needs of the two general purpose experiments, ATLAS [1] and CMS [2], with similar challenges standing in front of other experiments, e.g. LHCb [3]. The largest fraction of the CPU consumption goes into simulation and within that, into the simulation of the detector responses and especially the calorimeters. These detectors are particularly challenging due to the need to track many secondary particles produced in extensive showers that result from particles stopping inside dense materials. State-of-the-art physics-based simulations use GEANT4 [4, 5, 6] and are a major computational bottleneck, forecast to overwhelm the computing budget of existing and future experiments.

Without significant research and development of new simulation techniques and algorithms, the data collection will significantly outpace the Monte Carlo production capabilities of the experiments which in turn will limit the precision of many measurements as they will be limited by the statistics of the Monte Carlo simulation. Maintaining the current MC-to-data ratio is therefore a high priority for the LHC experiments. A possible mitigation can be achieved by replacing the expensive calorimeter simulations with faster alternatives. Such faster calorimeter simulation techniques [7, 8, 9, 10, 11], which are usually called "fast simulation", typically rely on parametrized responses of the calorimeter, tailored to specific types of incoming particles. By employing these parametrizations, effectively bypassing the intricate

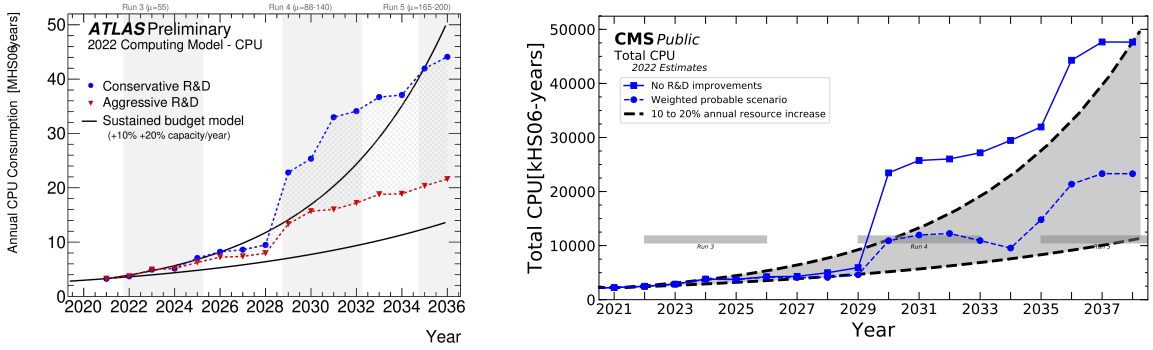



Figure 1: Projected CPU requirements. **Left:** For ATLAS [1]. **Right:** For CMS [2].

shower development process carried out by GEANT4, the simulation of an event is significantly sped up. However, these models usually lack the high fidelity that is required by the precision measurements carried out by the LHC experiments.

A possible alternative solution is provided by the immense progress in computer science, machine learning, and especially generative AI in the past two decades. Deep generative models (DGMs) learn, implicitly or explicitly, the distribution of (simulated) data from a given sample and then generate new data according to this distribution. Continuous research with impressive progress over nearly a decade [12] has shown that these models have the potential to become fast and faithful alternatives for detector simulation, as was summarized in a recent review on DGMs for calorimeter simulation [13]. For that reason, such models also started to be included in the fast simulation packages of the experiments [9, 14] in recent years.

Motivated by the aim of spurring the further development and benchmarking of fast and high-fidelity calorimeter shower generation using deep learning methods, the Fast Calorimeter Simulation Challenge ("CaloChallenge") was initiated in early 2022. It is modeled after two previous, highly successful data challenges in HEP — the top tagging community challenge [15] and the LHC Olympics 2020 anomaly detection challenge [16].

In the CaloChallenge, participants were tasked with training their favorite generative model on the provided calorimeter shower datasets, learning to sample from the conditional probability distribution $p(\mathcal{I}|E_{\text{inc}})$, where $\mathcal{I}$ are the voxel energy deposits and $E_{\text{inc}}$ is the incident energy. The particle and nuclear physics communities have been developing fast simulation methods for some time, and the goal of this challenge was to accelerate and expand on these efforts, while offering common benchmarks with which to assess the strengths and weaknesses of the new approaches, and a common evaluation pipeline for fair comparison.

This is the community paper summarizing the outcome of the CaloChallenge. Over 60 participants contributed to the development of 31 different DGMs (some close variants or distillations of each other, 23 of them completely distinct) for fast calorimeter simulation, making use of cutting-edge techniques in generative modeling with deep learning, including GANs, VAEs, normalizing flows, diffusion models, and conditional flow-matching models. Table 1 gives an overview of the presented models and links the corresponding code repositories.

Many submissions were presented at ML4Jets 2022 at Rutgers [17], ML4Jets 2023 in Hamburg [18], and the CaloChallenge Workshop in Frascati [19]. They have been published in separate research articles, either in peer-reviewed journals or in machine learning conferences. A small subset of the submissions have been compared previously in [20], independent of this study here and without submitted samples, but with retrained models based on code repositories instead.

The document is structured as follows: in Section 2, we describe the calorimeter datasets that we provided. Then, we introduce the individual approaches, grouped by their main generative architecture: Generative Adversarial Networks (GANs) in Section 3, Normalizing Flows (NFs) in Section 4, Diffusion Models in Section 5,

Table 1: Models submitted to the CaloChallange.

| Approach | Model | Code | Dataset | | | | Section |
|---|---|---|---|---|---|---|---|
| | | | $1-\gamma$ | $1-\pi$ | 2 | 3 | |
| GAN | `CaloShowerGAN` [21] | [22] | ✓ | ✓ | | | 3.1 |
| | `MDMA` [23, 24] | [25] | | | ✓ | ✓ | 3.2 |
| | `BoloGAN` [26] | [27] | ✓ | ✓ | | | 3.3 |
| | `DeepTree` [28, 29] | [30] | | | ✓ | | 3.4 |
| NF | `L2LFlows` [31, 32] | [33] | | | ✓ | ✓ | 4.1 |
| | `CaloFlow` [34, 35] | [36, 37] | ✓ | ✓ | ✓ | ✓ | 4.2 |
| | `CaloINN` [38] | [39] | ✓ | ✓ | ✓ | | 4.3 |
| | `SuperCalo` [40] | [41] | | | ✓ | | 4.4 |
| | `CaloPointFlow` [42] | [43] | | | ✓ | ✓ | 4.5 |
| Diffusion | `CaloDiffusion` [44] | [45] | ✓ | ✓ | ✓ | ✓ | 5.1 |
| | `CaloClouds` [46, 47] | [48, 49] | | | | ✓ | 5.2 |
| | `CaloScore` [50, 51] | [52, 53] | ✓ | | ✓ | ✓ | 5.3 |
| | `CaloGraph` [54] | [55] | ✓ | ✓ | | | 5.4 |
| | `CaloDiT` [56] | [57] | | | ✓ | | 5.5 |
| VAE | `Calo-VQ` [58] | [59] | ✓ | ✓ | ✓ | ✓ | 6.1 |
| | `CaloMan` [60] | [61] | ✓ | ✓ | | | 6.2 |
| | `DNNCaloSim` [62, 63] | [64] | | ✓ | | | 6.3 |
| | `Geant4-Transformer` [65] | [66] | | | | ✓ | 6.4 |
| | `CaloVAE+INN` [38] | [39] | ✓ | ✓ | ✓ | ✓ | 6.5 |
| | `CaloLatent` [67] | [68] | | | ✓ | | 6.6 |
| CFM | `CaloDREAM` [69] | [70] | | | ✓ | ✓ | 7.1 |
| | `CaloForest` [71] | [72] | ✓ | ✓ | | | 7.2 |

Variational Autoencoders (VAEs) in Section 6, and Conditional Flow Matching Models (CFMs) in Section 7. Section 8 introduces the metrics which we employ to compare the submissions. We then show our results, where we first focus on the scores of the individual metrics in Section 9 and then look at the correlations and Pareto fronts in Section 10. On the one hand, this sheds light on interesting trade-offs, while on the other hand, it tells us about the metrics themselves. We summarize and present an outlook in Section 11. In the appendices, we collect additional reference plots as well as tables with the detailed numbers that are presented in the figures of Sections 9 and 10.

## 2. Datasets

The challenge offers three datasets, ranging in difficulty from easy through medium to hard. The difficulty is set by the dimensionality of the calorimeter showers, *i.e.* the number layers and the number of voxels in each layer.

Each dataset has the same general format. The detector geometry consists of concentric cylinders with particles propagating along the $z$-axis. The detector is segmented along the $z$-axis into $N_z$ discrete layers. Each layer has $N_r$ bins along the radial direction and $N_\alpha$ bins in the angle $\alpha$. The number of layers and bins in $r$ and $\alpha$ is summarized in table 2. The coordinates $\Delta\phi$ and $\Delta\eta$ correspond to the $x$ and $y$ axis of the cylindrical coordinates. Figure 2 shows a 3-dimensional view of a geometry with 3 layers, with each layer having 3 bins in the radial direction and 6 bins in the angular direction. The right image shows the front view of the geometry, as seen along the $z$ axis.

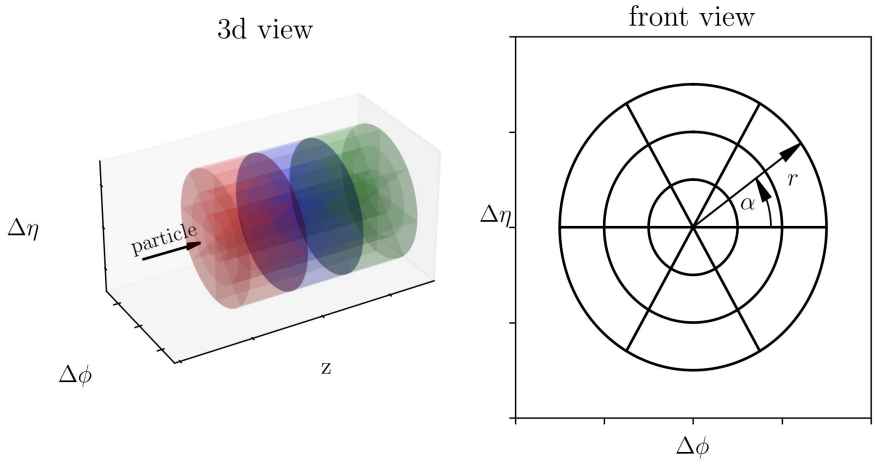



Figure 2: Schematic view of the voxelization in all datasets. Along the direction of the incoming particle ($z$), the volume is segmented in $N_z$ layers. Each layer has $N_r$ radial and $N_\alpha$ angular bins. **Left:** 3-dimensional view. **Right:** Front view. Images taken from [73], separately published in [34].

Each CaloChallenge dataset comes as one or more `.hdf5` files that were written with python's `h5py` package [74] using `gzip` compression. Within each file, there are two hdf5-datasets: `incident_energies` has the shape `(num_events, 1)` and contains the energy of the incoming particle in MeV; `showers` has the shape `(num_events, num_voxels)` and stores the energy deposited in showers, where the energy depositions of each voxel (in MeV) are flattened. The mapping of array index to voxel location is done in the order (radial bins, angular bins, layer), so the first entries correspond to the radial bins of the first angular slice in the first layer. Then, the radial bins of the next angular slice of the first layer follow, and so on.

For every dataset in the CaloChallenge, there is one dataset file to be used for

Table 2: Voxelization of layers in each dataset. We show $N_r \times N_\alpha$ and the total number of voxels, $N_i$, per layer. For datasets 1: a "–" indicates that this layer is not in the dataset, as the numbering is based on the ATLAS detector definitions [9].

| Layer Number | 0 | 1 | 2 | 3 | ... | 12 | 13 | 14 | ... | 44 | total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ds 1 – $\gamma$ | $8 \times 1 = 8$ | $16 \times 10 = 160$ | $19 \times 10 = 190$ | $5 \times 1 = 5$ | – | $5 \times 1 = 5$ | – | – | – | – | 368 |
| ds 1 – $\pi^+$ | $8 \times 1 = 8$ | $10 \times 10 = 100$ | $10 \times 10 = 100$ | $5 \times 1 = 5$ | – | $15 \times 10 = 150$ | $16 \times 10 = 160$ | $10 \times 1 = 10$ | – | – | 533 |
| ds 2 | $9 \times 16 = 144$ | | | | | | | | | | 6480 |
| ds 3 | $18 \times 50 = 900$ | | | | | | | | | | 40 500 |

Table 3: Number of samples available per incident energy for each of the training and evaluation datasets for dataset 1 – $\gamma$ and dataset 1 – $\pi^+$.

| $E_{\text{inc}}$ | 256 MeV – 131 GeV | 262 GeV | 524 GeV | 1.04 TeV | 2.1 TeV | 4.2 TeV | total |
|---|---|---|---|---|---|---|---|
| ds 1 – $\gamma$ | 10 000 per energy | 10 000 | 5000 | 3000 | 2000 | 1000 | 121 000 |
| ds 1 – $\pi^+$ | 10 000 per energy | 9800 | 5000 | 3000 | 2000 | 1000 | 120 800 |

training the generative models and a second one for the evaluation (both by the individual collaborations and by us). For dataset 3, we split the training and evaluation data each into two separate files to have more manageable files sizes.

### *2.1. Dataset 1 Photons and Pions*

Dataset 1 can be downloaded from [75, 76]. It is based on the ATLAS open datasets [77] and contains the simulation of single photons and single charged pions generated at the surface of the ATLAS calorimeter system and pointing back to the center of the detector. The interaction of the particles in the calorimeters was simulated with the official ATLAS software, which is based on GEANT4, using a special configuration in which detailed hits were produced and noise from electronics and cross-talk was not included; this allows modeling perfect showers that can be injected in the simulation chain before these effects occur, making it more realistic. These samples were used to train the GANs presented in the AtlFast3 paper [9] and the FastCaloGAN note [26].

Initially, only one dataset for the pion sample was available. Later, a second, independent dataset was provided by ATLAS, so we updated the Zenodo and all trainings and evaluations were done with two independent training and evaluation datasets [76]. There are four datasets, two for photons and two for charged pions. Each dataset contains the voxelized shower information obtained from single particles in the range $0.2 < |\eta| < 0.25$; therefore the particles impact the detector with an angle.

For each particle, there are 15 incident energies from 256 MeV up to 4 TeV produced in powers of two. 10 000 events are available in each sample except for those at higher energies having lower statistics, see table 3. The number of radial and angular bins varies from layer to layer and is also different for photons and pions, resulting in 368 voxels for photons (called "ds $1-\gamma$" throughout) and 533 for pions (called "ds $1-\pi^+$" throughout), see table 2.

In the results section, a 1 MeV threshold is applied to all voxels to eliminate the low energy tail that affects some of the models but has no physics impact. This assessment is based on how the energy deposited in the calorimeter is transformed and calibrated into reconstructed objects (i.e. photons or jets) using clusters built from the calorimeters' cells [78]. ATLAS calorimeters are segmented in cells to increase the granularity and improve the spatial reconstruction of showers, and this segmentation is reproduced in the simulation. The cells have rectangular shapes that are easier to construct, hence they do not match the cylindrical voxel geometry described above. This required an additional step in the AtlFast3 simulation in which the energy from the voxels was reassigned to the actual calorimeter cells. In the offline reconstruction, ATLAS uses topological clusters that are started (seeded) from cells having at least 4 times the noise; they are subsequently grown to include neighboring cells with energy twice the noise level, and then they are finalized with any cell adjacent to the cluster that is above the noise threshold. The lowest cell noise in the layers considered in the datasets is about 10 MeV for layer 1 with other layers having up to 50 MeV. Therefore, a chosen 1 MeV threshold in the voxels' energy is reasonable even when taking into account the fact that multiple voxels could map to the same cell; this only occurs in the core of the shower where most of the energy is deposited and therefore the threshold cut will not take place, *i.e.* all masked voxels are peripheral voxels that actually map to multiple cells, further diluting the energy associated to each cell.

### *2.2. Datasets 2 and 3*

Datasets 2 and 3 have been simulated with the Par04 [79] example of GEANT4. The geometry used in the Par04 example is an idealised calorimeter, with concentric cylinders of alternating absorber and active materials. A draft of its layout is presented in figure 3. Both datasets were simulated for the same detector which consists of 90 physical layers, with each layer composed of 1.4 mm of tungsten (W) as an absorber and 0.3 mm of silicon (Si) as active material. The inner radius of this calorimeter is 800 mm and its depth is 153 mm.

Particle showers are generated by electrons that enter the detector perpendicularly to the detector's cylinders' axis, as depicted in figure 3's upper electron. Datasets with different, varying incident angles, like the second electron in figure 3 pointing to the lower right, were also published but go beyond the scope of this challenge [80].

The Par04 example of GEANT4 writes out only energy deposited in the active material, so it must be corrected for the deposits in the absorber. A simple scaling

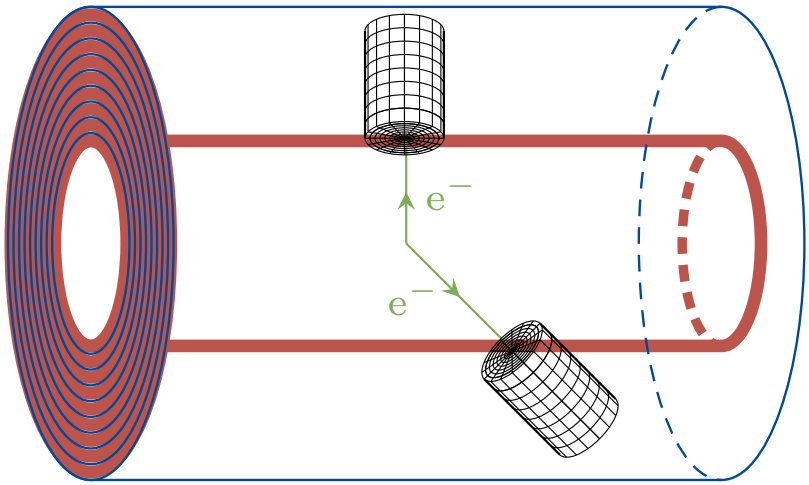



Figure 3: The Par04 detector [79] consists of concentric cylinders of absorber (red) and active material (blue). The energy deposited by incident particles is recorded in a cylindrical readout (black).

factor has been derived from the simulation, $f = 1/0.033$, uniform for all energies and cells of the detector. It means that on average 3.3% of particle's energy is registered in the detector. The Par04 simulation has an energy threshold below which cell energy is not stored (to reduce the file size). It is chosen to a very low value of 0.5 keV, which translates to 15.15 keV after the energy scaling. We also apply this cutoff to all submissions before the final evaluation.

The particle entrance position and direction determine the position (0,0,0) and orientation ($z$-axis) of the cylindrical readout, like the one shown in figure 2. The size of each readout voxel is $\Delta r \times \Delta\varphi \times \Delta z$ and unlike for dataset 1, both datasets 2 and 3 have the same number of voxels in each of the $N_z$ layers. Also the number of voxels along $z$-axis is the same in datasets 2 and 3, but they differ in terms of segmentation in radius ($r$) and angle ($\alpha$). The size along $z$-axis is equal to $\Delta z = 3.4\,\text{mm}$ which corresponds to two physical layers (W-Si-W-Si). Taking into account only the absorber's value of radiation length ($X_0(\text{W}) = 3.504\,\text{mm}$ [81]) it makes the size along $z$-axis approximately $\Delta z = 2 \cdot 1.4\,\text{mm}/3.504\,\text{mm} = 0.8X_0$. In radius, the size of the voxels is 4.65 mm for dataset 2 and 2.325 mm for dataset 3, which in approximation, taking only the Moliére radius of the absorber, is $\Delta r = 4.65\,\text{mm}/9.327\,\text{mm} = 0.5R_M$ for dataset 2 and $\Delta r = 2.325\,\text{mm}/9.327\,\text{mm} = 0.25R_M$ for dataset 3. The angular segmentation consists of 16 voxels for dataset 2 ($\Delta\varphi = 2\pi/16 \approx 0.393\,\text{rad}$) and 50 voxels for dataset 3 ($\Delta\varphi = 2\pi/50 \approx 0.126\,\text{rad}$).

The total number of voxels for dataset 2 is $N_z \times N_r \times N_\alpha = 45 \times 9 \times 16 = 6480$ and for dataset 3 it is $N_z \times N_r \times N_\alpha = 45 \times 18 \times 50 = 40500$, see table 2.

Files can be downloaded from [82] and [83] for dataset 2 and 3 respectively. Dataset 2 consists of two files (one for training and one for evaluation) with 100 000 showers of electrons each with energies sampled from a log-uniform distribution ranging from 1 GeV to 1 TeV. Dataset 3 contains showers of electrons sampled from the same incident energy distribution. Due to the size, there are 4 files with 50 000 showers each. One half of the available sample should be used in training, with the remaining half used as a reference file in evaluation.

## 3. GAN-based Submissions

Generative Adversarial Networks (GANs) [84] are one of the earliest types of deep generative models and reached fame by being able to produce photorealistic images [85]. A GAN consists of two networks, a generator and a critic‡. They are trained adversarially in a game where the generator produces fake samples that the discriminator tries to distinguish from real samples. On the upside, GANs are very flexible, as their only hard requirement is finding two networks that map to the correct space. Furthermore, GANs are typically very fast compared to other generative models and can produce samples with high fidelity. On the downside, their training is unstable, and they are difficult to optimize. For this reason, several improvements were proposed, *e.g.* the Wasserstein GAN [86, 87]. CaloGAN [12, 88] was the first tool that demonstrated the feasibility of using a deep generative model to perform a fast calorimeter simulation. GANs are also the first model to be used in production, as FastCaloGAN [89, 26] was deployed as part of AtlFast3 [9] and used by the ATLAS experiment to produce several billion events.

### *3.1. CaloShowerGAN*

By Michele Faucci Giannelli and Rui Zhang, with figures and tables referring to these approaches as `CaloShowerGAN` [21], `CaloShower2GAN` [21], `CaloShower3GAN` [21] and code being available at [22].

---

*Introduction* Building on the success of FastCaloGAN, `CaloShowerGAN` [21] is designed to have a similar interface so that the ATLAS collaboration could easily integrate it. However, `CaloShowerGAN` significantly diverges from FastCaloGAN in the internal structure of the tool and achieves a significant improvement in reproducing both photons and pions. This is realized through a new pre-processing of the training data by further optimizing the model architecture and hyperparameters.

‡ Also called discriminator, if the cross entropy loss is used.

*Architecture* For example, `CaloShower3GAN` employs three GANs for the parametrization of the photons in different energy ranges; this is motivated by how the energy is deposited in the different layers of the calorimeter as a function of the primary particle energy. The energy thresholds to define low, medium and high energy ranges are 4 GeV and 262 GeV, whereas `CaloShower2GAN` merges the medium and high energy ranges. Only one GAN is used for the pions in all three versions as the nature of the hadronic interaction allows even low-energy pions to interact in the deeper layers of the calorimeter.

The GAN architecture (see figure 4) was significantly optimized for this challenge, details on the optimization process are described in `CaloShowerGAN` [21]. The optimal hyperparameters used in the photon and pion GANs are shown in table 4.

*Preprocessing* Several normalization steps are used to simplify the training of the GANs. The first normalization is based on the kinetic energy of the particles as done in FastCaloGAN and other tools. This normalization procedure allows standardizing all values within the input vector to a similar order of magnitude for all input momenta, eliminating the significant difference between the momenta of the samples. In this way, the GAN can focus on reproducing the shape of the showers rather than its absolute value.

`CaloShowerGAN` employs additional normalization for layer-specific energy and total energy. This information improves the training because the networks do not have to extract it from the data as it is explicitly provided; thus the GANs can focus on learning correlations and shapes improving the overall performance. Details on the implementation of this normalization can be found in [21].

The condition label is also transformed to a normalized range of $[0, 1]$ using the following equation:

$$\hat{E} = \frac{\log \frac{E_{\text{kin}}}{E_{\text{min}}}}{\log \frac{E_{\text{max}}}{E_{\text{min}}}}. \tag{1}$$

Here $E_{\text{min}}$ ($E_{\text{max}}$) is the minimum (maximum) kinetic energy of the incoming particle in the training data.

*Training* The batch size used for training the GANs is 1024 and the training runs for a total of $10^6$ iterations. Due to the adversarial nature of GAN training, the final iteration does not necessarily yield the best outcome, therefore the GANs are evaluated at intervals of $10^3$ iterations. This is a compromise between the time required for evaluation and the speed of learning of the GANs.

The evaluation is inspired by the methodology used in FastCaloGAN using the total energy distribution for all energies as a figure of merit. The $\chi^2$ value for each GAN model is computed between the binned distributions of the GEANT4 sample and generated sample by the model and then normalized by the number of degrees of freedom

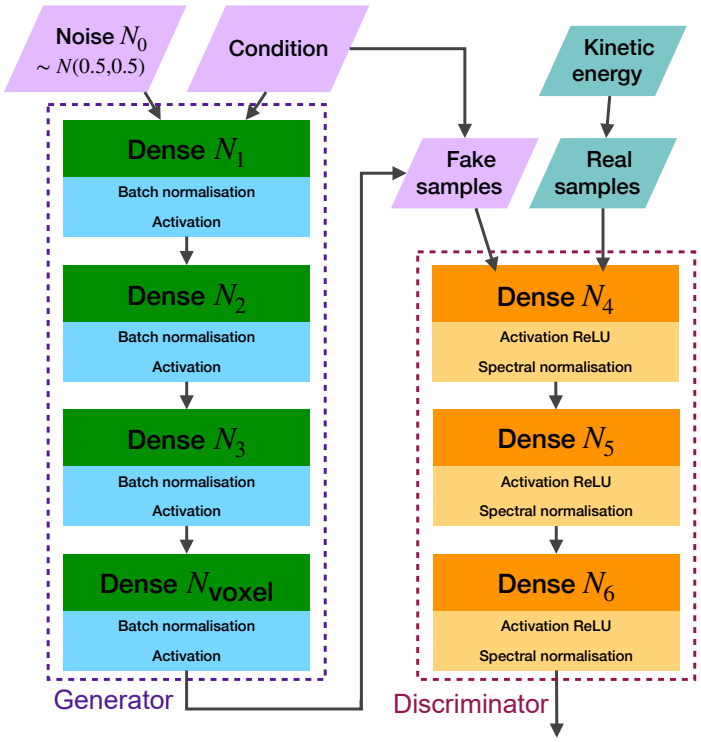



Figure 4: Architecture of `CaloShowerGAN`, taken from [21].

used in each distribution ($\chi^2$/NDF). The model that gives the lowest $\chi^2$/NDF among the saved iterations is considered the best and is used in the challenge.

Table 4: Optimal hyperparameter values for the photon and pion in `CaloShowerGAN`, taken from [21].

| Hyperparameter | Photon | Pion |
|---|---|---|
| Latent space size | 100 | 200 |
| Generator size ($N_1, N_2, N_3$) | 100, 200, 400 | 200, 400, 800 |
| Discriminator size ($N_4$, $N_5$, $N_6$) | 368, 368, 368 | 800, 400, 200 |
| Generator optimizer | Adam [90] | Adam [90] |
| Learning rate | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ |
| $\beta_1$ | 0.5 | 0.5 |
| $\beta_2$ | 0.999 | 0.999 |
| Discriminator optimizer | Adam [90] | Adam [90] |
| Learning rate | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ |
| $\beta_1$ | 0.5 | 0.5 |
| $\beta_2$ | 0.999 | 0.999 |
| Batch size | 1024 | 1024 |
| D/G ratio | 8 | 5 |
| $\lambda$ | 3 | 20 |
| Activation (generator) | Swish | ReLU |
| Activation (discriminator) | ReLU | ReLU |
| Neuron weight initialization (generator) | Glorot Normal | He Uniform |
| Neuron weight initialization (discriminator) | He Uniform | He Uniform |
| Trainable parameters (generator, discriminator) | 261k, 408k | 871k, 829k |

### *3.2. Matching Deep Mean-field Attentive (MDMA) GAN*

By Benno Käch, Dirk Krücker, Isabell Melzer-Pellmann, Moritz Scham, and Simon Schnake, with figures and tables referring to this approach as `MDMA` [23, 24] and code being available at [25].

---

*Introduction* The `MDMA` [23, 24] was first applied to the JetNet-150 [91] datasets, yielding state-of-the-art results not relying on kinematic inputs to the generative model. The model is designed to work on a point cloud (PC) representation of its input. As such, the calorimeter data is first preprocessed to convert the hits to a point cloud, where the coordinates of each point are given by $(E, z, \alpha, R)$ of every hit in the detector. This representation is especially efficient as the granularity of the detector grows and there are a large number of empty cells, thus only dataset 2 and 3 were considered for this model.

*Architecture* The generator and critic consist of the same main building blocks, which use a cross-attention-based information aggregation mechanism. As there is a large number of hits on average (*e.g.* $\sim 1600$) the quadratic computational scaling of self-attention is not feasible. Therefore, a synthetic "mean-field" $\bar{\mathbf{x}}$ is introduced, initially set as the mean of all points in a cloud, acting as an intermediary for information exchange between points. First, the mean-field is updated via cross-attention (*i.e.* the Query $Q$ in the attention aggregation is the embedded mean-field $\bar{\mathbf{x}}$, whereas the Key $K$ and Value $V$ are an embedding of the hits in the detector). Then, the mean-field is further processed with a fully-connected layer, additionally using the number of hits as an input and a gated linear unit is applied to an embedding of the incoming energy and the mean-field. Finally, the mean-field is concatenated to every hit and a point-wise layer is applied. This aggregation is permutation-equivariant since cross-attention itself is permutation-equivariant and all the other aggregations are independent of the other points in the cloud. The difference between the generator and the critic is only in the final layer. For the critic a 2-layer deep neural network (DNN) is applied to obtain a score for every shower, whereas for the generator the output is mapped down to 4 dimensions, corresponding to the energy and index of the cell. A schematic for the minimal building block is shown in figure 5. The input for the generator is noise sampled from a normal distribution with dimensions four times the number of hits per shower.

*Training* During training, showers of similar length are grouped together to form batches and padded to the same length. Padded points have no influence on the output. The model is trained as a Wasserstein GAN [87] with gradient penalty [92] to regularize. Additionally, weight normalized linear layers [93] are employed in the critic. To enhance the convergence of the generator, an L2-loss between the mean of the mean-field in the final layer of the critic for real and generated jets is calculated and minimized (hence the name mean-field matching). To enforce the conditioning with the incoming energy, the generator also minimizes an L2-loss between the detector responses for real and generated showers. During training, the condition of real showers are given to the generator allowing the matching for the L2-loss. The showers are post-processed by rotating the shower by a random angle. This alleviates the suboptimal coordinate choice, which does not respect the periodicity in the angular coordinate. Note that since point clouds are generated, not only the incoming energy of the incoming particle is supplied as a condition, but also the number of hits in the shower needs to be supplied. For this study they were taken from the validation set — in practice one would need another model to sample the probability mass function $p(n|E)$.

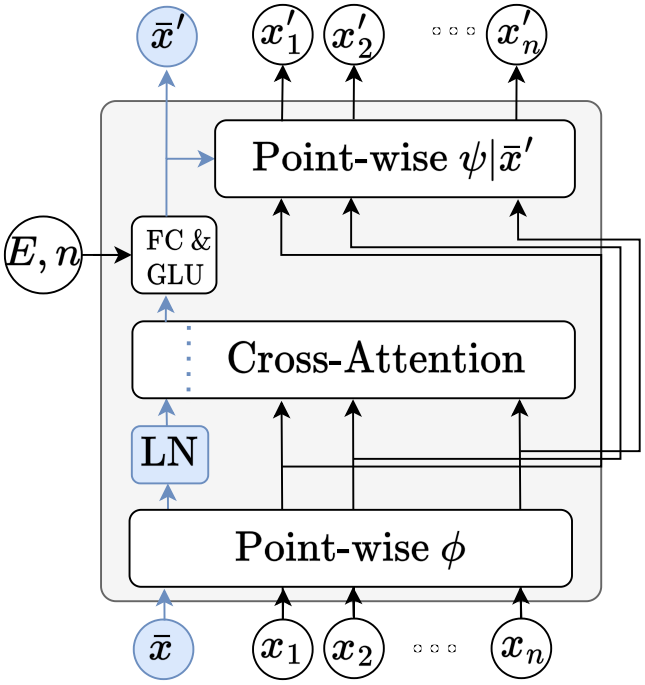



Figure 5: Main building block of the MDMA architecture. The calorimeter is represented as a point cloud, where every point $\mathbf{x_i}$ is a hit in the detector and $\bar{\mathbf{x}}$ is the artificial mean-field. First, the points are mapped to a higher dimensional latent space, where after normalization cross-attention is calculated between the mean-field and the other points. Then the conditional information for the shower (*i.e.* the incident energy $E_{\text{inc}}$ and the number of hits $n$) are introduced with a fully-connected (FC) layer and a gated linear unit (GLU). Finally, the updated mean-field is concatenated to the other points and a point-wise layer is used to update the points independently on each other. This architecture yields permutation-equivariance and scales linearly with the number of hits in the computational complexity. Figure adapted from [23, 24].

### *3.3. BoloGAN*

By Federico Andrea Guillaume Corchia, Matteo Franchini, and Lorenzo Rinaldi, with figures and tables referring to this approach as BoloGAN [26] and code being available at [27].

---

*Introduction* BoloGAN [26] is a GAN-based calorimeter simulation tool derived and evolved from FastCaloGAN, a fast simulation tool developed in the ATLAS Collaboration at CERN [26, 9].

*Architecture* The tool uses the Wasserstein GAN [87] with a gradient penalty (WGAN-GP) term [92] in the loss function of the discriminator, providing good performance and

training stability, and conditioning onto the kinetic energy of the particle (conditional WGAN-GP). The conditional WGAN-GP is implemented in TensorFlow 2.0 [94]. The generator and the discriminator both employ three hidden layers, the generator being preceded by a latent space of 100 values and having the output layer with as size the number of voxels for the specific particle type and pseudorapidity interval; the last layer of the discriminator has one single output node. The general scheme is shown in figure 6. `BoloGAN` WGAN-GP hyperparameters are set as shown in table 5, depending on particle type and on energy. These hyperparameters and the general architecture correspond to a trade-off between modeling performance and time required to train the GANs: the program is, in fact, intended to have the possibility to train multiple GANs at the same time, useful for modeling different particle types and pseudorapidity layers with accuracy. The GAN is trained first on a single energy point, then the other energy points are progressively added to training starting from the ones closest in energy to the initial sample. Conditioning is applied, as mentioned, onto the kinetic energy of the particle and the energy in each voxel is normalized by the true energy of the sample, so that all energy samples are scaled to the same values and training can focus on the shape of the total energy (which is the figure of merit, as shall be shown in the continuation). Truth energies used as labels for conditioning are also normalized to the highest energy, in this way all values are in the (0,1] range.

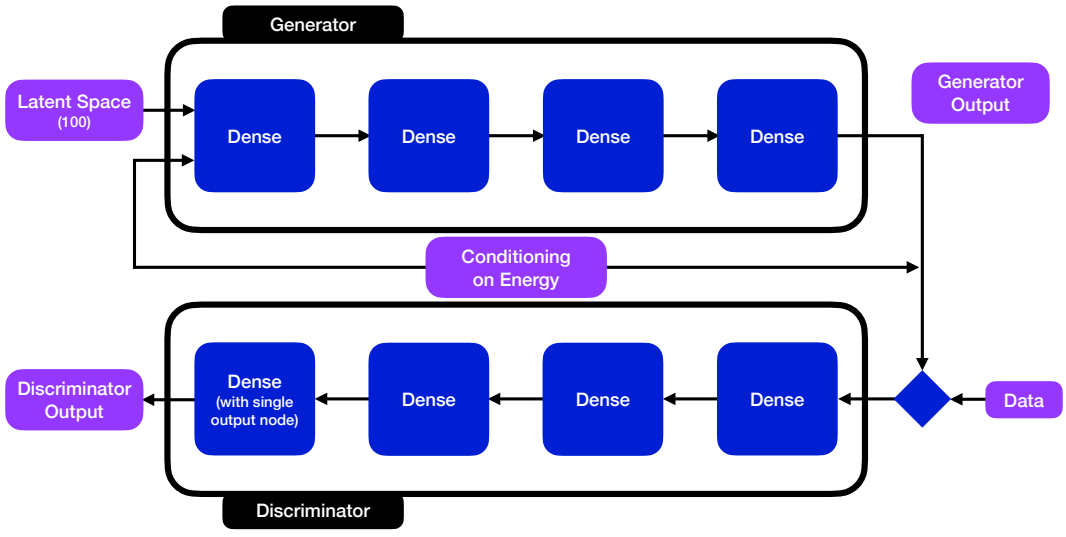



Figure 6: Architecture of `BoloGAN`, adapted from [26].

*Training* We performed training for 1 million epochs with a TensorFlow checkpoint saved every 1000 epochs. This granularity allows for monitoring improvement in training without having to save too many checkpoints, which would hamper speed and disk

| Parameter | Pions | Low En. Photons | High En. Photons |
|---|---|---|---|
| Latent Space | 100 | 100 | 100 |
| Generator Nodes Output Shape | 200, 400, 800, 533$^\dagger$ | 100, 200, 400, 368$^\dagger$ | 50, 100, 200, 368$^\dagger$ |
| Discriminator Nodes Output Shape | 800, 400, 200, 1 | 368$^\dagger$, 368$^\dagger$, 368$^\dagger$, 1 | 368$^\dagger$, 368$^\dagger$, 368$^\dagger$, 1 |
| Activation Function | ReLU | ReLU | Swish |
| Optimizer | Adam [90] | Adam [90] | Adam [90] |
| Learning Rate | $10^{-4}$ | $10^{-4}$ | $10^{-4}$ |
| Discriminator/Generator Training Ratio | 5 | 8 | 8 |
| Beta | 0.5 | 0.5 | 0.5 |
| Lambda | 10 | 3 | 3 |
| Batch Size | 512 | 1024 | 1024 |
| Used Batch Normalization Layer | No | Yes | Yes |

Table 5: `BoloGAN` WGAN-GP hyperparameters. Low (high) energy photons are those up to (above) 4.096 GeV. Values marked with $^\dagger$ are equal to the number of voxels in the corresponding case.

space. Because of the interplay between the generator and the discriminator, the final epoch is not necessarily the best one, also considering that there may be an unfavorable fluctuation in training. For this reason, a $\chi^2$ between the reference sample and the one simulated by the GAN, evaluated over the sum of the energy in all voxels (which corresponds to the total energy deposited into the calorimeter by the particle), is used to choose the best GAN iteration. The iteration with the lowest $\chi^2$ is considered the final choice to perform simulation activities. The total energy for each possible incident energy value was chosen as it is easy to define while it is difficult to reproduce. For every checkpoint, 10k events are generated per incident energy value and the $\chi^2$ between the reference sample and the GAN-simulated one is calculated; the total $\chi^2$ for a checkpoint is the sum of the $\chi^2$ for the individual incident energy values and the checkpoint with the lowest total $\chi^2$ is finally chosen as the best GAN iteration.

The program is currently able to simulate calorimeter showers for photons, electrons, and pions between 256 MeV and 4 TeV over the full detector acceptance. For the CaloChallenge, the tool was applied to Dataset 1 for both photons and pions. For pions one single GAN for all energy values has been trained, while for photons two GANs have been trained, one for low energies (*i.e.* up to 4.096 GeV) and the other one for high energies (above 4.096 GeV).

### *3.4. DeepTree*

By Moritz A.W. Scham, Benno Käch, Simon Schnake, Dirk Krücker, and Kerstin Borras, with figures and tables referring to this approach as `DeepTree` [28, 29] and code being available at [30].

*Introduction* `DeepTree` [28, 29] is a point cloud (PC)-based GAN model, that uses a tree-like structure for upscaling PCs in the generator and for downscaling them in the critic. A calorimeter shower can be converted to a PC by taking the coordinates and the energy of the hits as points in an unordered set. Representing calorimeter showers as PCs instead of voxels separates the hits from the detector geometry. This offers multiple advantages: PCs are well-suited for handling sparsity in calorimeter data and they are very efficient if only a fraction of cells contain hits. Their adaptability to irregular calorimeter geometries makes PCs a versatile choice for various detector configurations. Lastly, the generator architecture developed for one calorimeter using PCs can be easily transferred to different calorimeter types. On the downside, this independence of the detector geometry also means that the model needs to learn the geometry of the detector from the dataset. In a postprocessing step, the generated points must be assigned to the individual cells of the calorimeter. Since the PC-based model does not know the detector geometry, several points may be generated and assigned to the same calorimeter cell. To obtain a unique output for each cell, these points must be combined in some way. Designing a PC-based model that yields a varying number of points (cardinality) by itself is challenging. Here, the cardinality is sampled from the dataset and provided to the model. Because dataset 1 (3) yields a too low (high) cardinality, this model targets dataset 2.

*Generator* The generator of this GAN constructs PCs by starting with a random vector as the root of a tree and then attaching one level of leaves after the other. The output of the generator is the last level of the tree.

*Branching Layer* Starting with the root node, a *Branching Layer* (figure 7) takes the current leaves from the tree, maps each leaf to a given number of nodes $n_i$ and attaches these nodes as new leaves to the tree. Then further *Branching Layers* are applied until the desired number $\prod_i \mathrm{n_i}$ of nodes is reached. For these projections, multiple deep feed-forward neural networks (DNNs) are used. With each Layer, the number of nodes increases ($1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 5 \cdot 10 = 6000$) and the number of features decreases $(64, 25, 15, 10, 8, 6, 4)$. The cardinality $c$ is sampled from the evaluation set and only the first $c$ points produced by the generator are used.

*Ancestor MPL* In between the branching layers, a Message Passing Layer (MPL) is applied to this tree-structured graph. The edges in the graph are constructed so that each node receives messages from each of its ancestors as well as itself. As a message-passing algorithm, GINConv [95, 96] is chosen. For GINConv, the messages are the features of the source node (here: the ancestor). These messages are aggregated by summing over all messages addressed to the target node. These aggregates are added to the target node (scaled with a learnable weight) and passed through an DNN. The nodes are then updated with the output of the DNNs plus, as a residual connection, the nodes themselves.

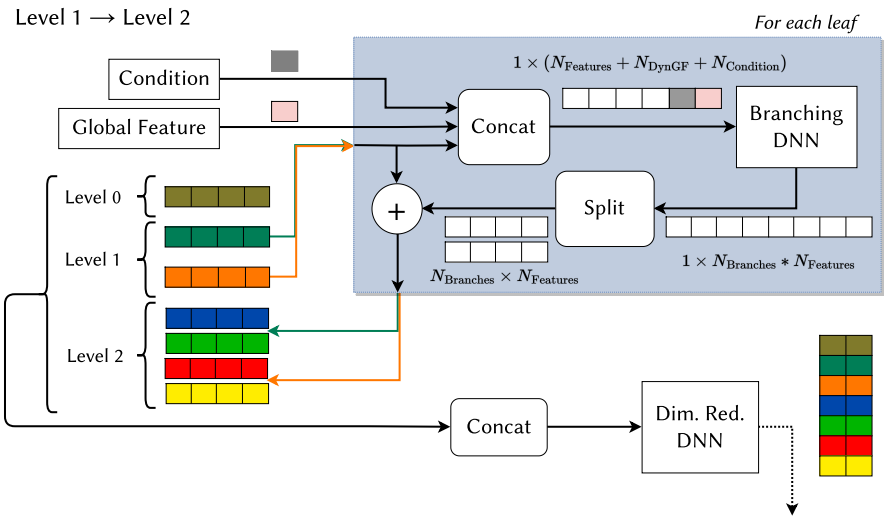



Figure 7: The Branching Layer of the `DeepTree` generator, as described in section 3.4. In this example, the nodes of the 2nd level of the tree are produced and attached as the new leaves. With the nodes from level 1 (dark green / orange) as parents, the children in level 2 (blue+green / red+yellow) are generated independently: The condition and a vector representing the state of all points and are appended to the parent. The result is mapped by a branching DNN to the size of the parent times the number of branches. After splitting up the vectors into the new children, the parent is added to each of them. With the new children added as leaves to the tree, all the levels of the tree are stacked up and passed through a dimensionality reduction DNN. Figure adapted from [28].

In addition, the generator contains layers that condition the MPLs and the branching layers with a vector representing the current state of the leaves in the tree. For this, the leaves are passed separately through a first DNN, then summed up and passed through a second DNN. The DNNs of the generator consist of 3 hidden layers of 100 nodes without bias. The first two hidden layers are followed by a batch normalization [97] layer and LeakyReLU activation with a negative slope of 0.1.

*Critic* The critic, shown in figure 8, aims to reduce the size of the PC iteratively. This is achieved by a pooling operation called Bipartite Pool. It constructs a bipartite graph that densely connects the input PC to a fixed number of trainable nodes and applies an MPL to this graph. As an MPL, Gatv2Conv [98] is used with 16 attention heads. Before each pooling, the points are processed by an embedding layer consisting of an DNN and a Central Node Update layer (CNU) with a residual connection. The CNU transforms input points separately with an DNN, aggregates them with multiple methods ("multi-aggregation"), and maps the points back to their original dimension with another DNN. This multi-aggregation is a concatenation of sum, maximum, cardinality, and width. The width is computed as mean absolute deviation from the mean $\frac{1}{n}\sum_i |x_i - \bar{x}|$. Three "subcritics" are applied to different levels of pooling and the input PC. Each subcritic

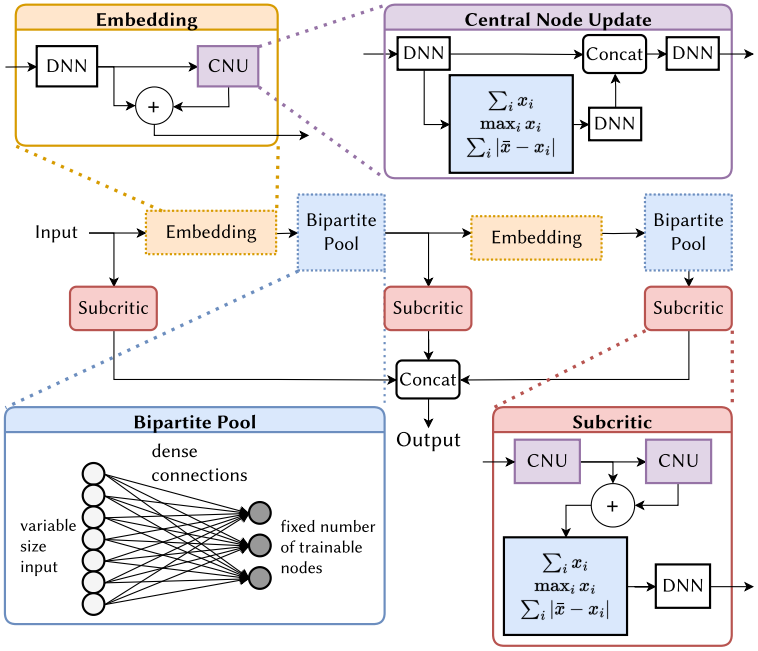



Figure 8: The `DeepTree` critic, as described in section 3.4. Figure adapted from [29].

uses two CNUs with a residual connection, followed by the multi-aggregation. The aggregated vector is passed through an DNN to produce a single output. Contrary to the generator, the DNNs of the critic use a dropout of 0.5 and spectral normalization, except for the DNN inside each embedding layer, which use batch normalization. All three subcritics are trained simultaneously, and their losses are added.

*Preprocessing and Postprocessing* The showers on the grid are converted to PCs by taking the $r, \alpha$ and $z$ indices, as well as the energy of each cell with an energy deposition. Uniform noise (0,1) is then added to these indices to make the distribution continuous (reversible by a floor operation). These values are then scaled to the interval $[0, 1]$, transformed with a *logit* function (inverse function: *expit*) and finally normalized (mean $\to 0$ and standard deviation $\to 1$). The energy of the hits is scaled with a Box-Cox transformation with normal scaling (`PowerTransformer` in [99]). For evaluation and generation, these transformations are inverted.
As conditional variables, the shower energy $E_{\text{gen}}$, the average hit energy $\bar{E}$, and the cardinality $c$ are sampled from the evaluation set and provided to both the generator and the critic. § $E_{\text{gen}}$ and $\bar{E}$ are first transformed with a Box-Cox transformation with normal scaling. The cardinality $c$ undergoes the same transformations as the cell indices,

§ This setup could introduce a bias in the evaluation. However, this does not impose a major restriction on the model, as producing two conditional variables is typically manageable and could be supported by an auxiliary NF in the future. In most practical cases, particularly when aiming to replace GEANT4, the energy would be provided as input, allowing for controlled generation of showers.

but instead of normal scaling, a quantile transformation (`QuantileTransformer` in [99]) is applied.
As the generator is not directly aware of the calorimeter cells, it may produce multiple hits for a single calorimeter cell. This is especially true for events with a very high cardinality. Simply summing the hits in each cell would lead to a low cardinality and points containing very high energies. To mitigate this effect, an algorithm‖ is employed, that moves hits from "overcrowded" cells, to empty, neighboring cells (in $r/\alpha/z$). Since hits of higher energy are more important, it tries to move the hits in the cells in order of energy, skipping the highest energy hit. In case there are not enough empty neighboring cells are available, the remaining hits are summed up. Due to this technique, the maximum cardinality that the generator produces goes from $\approx 3500$ to $\approx 4600$ (dataset: maximum $\approx 5300$ of 6000 total cells).
As an additional postprocessing step, the generated PCs are shifted by a random value in $\alpha$, resulting in a uniform $\alpha$ distribution.

## 4. Normalizing Flow-based Submissions

A normalizing flow models a complex density by applying a sequence of transformations to a simpler base distribution, thereby constructing a flexible distribution over continuous random variables. The objective of the normalizing flow is to learn a bijective transformation $T$ between two spaces. Initially, a vector $x$ is sampled from an intricate and generally unknown probability density $p_x(x)$. We define $T$ as the transformation $x = T(u)$, where $u \sim p_u(u)$ is a simple base distribution that is known and for which one can calculate the likelihood and sample from efficiently. Both $T$ and $p_u(u)$ can have parameters.

The transformation $T$ must be invertible, and both $T$ and $T^{-1}$ must be differentiable. Such transformations are categorized as diffeomorphisms. The density $p_x(x)$ is then well-defined and can be constructed by a change of variables

$$p_x(x) = p_u(T^{-1}(x)) \left|\det J_T(u)\right|, \tag{2}$$

where $J_T(u)$ is the Jacobian of $T$. Diffeomorphisms are notable for their composability, which allows us to construct $T$ from multiple smaller, invertible, and differentiable transformations $T = T_K \circ \ldots \circ T_1$, where each $T_k$ maps $z_{k-1}$ to $z_k$. Assuming $z_0 = u$ and $z_K = x$, the transformations sequentially modify the distribution, illustrated in figure 9.

Normalizing flows offer two operational pathways: the inverse path, which is utilized for density estimation and transformation optimization, and the forward path, which functions as a generative model. In the inverse direction, samples from the complex distribution $p_x(x)$ are mapped to the base distribution $p_u(u)$, optimizing the process, typically by maximizing the likelihood (or minimizing the negative log-likelihood). Conversely, the forward path initiates with sampling from the base distribution $p_u(u)$ and maps these samples to the data space represented by $p_x(x)$. The designation of

‖ Available on PyPI: `https://pypi.org/project/caloutils/`

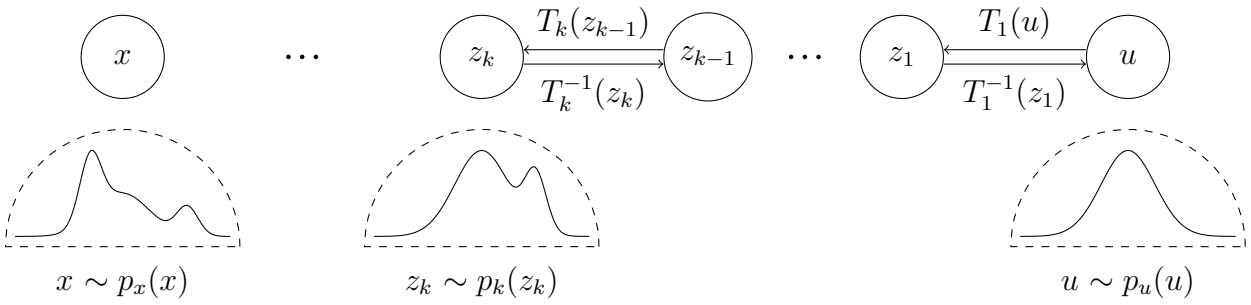



Figure 9: Visualization of a Normalizing Flow.

directions as inverse or forward is arbitrary. The flow is easily extended to conditional distributions $p_x(x \mid c)$ by including the conditional information $c$ in each transformation in $T$.

The requirement of an efficient computation of the log-likelihood is addressed by specific design choices of the network architecture which make the Jacobian determinant in (2) tractable [100, 101, 102]. Two common approaches are autoregressive flows [103, 104] and coupling-based flows [105, 106, 107]. Autoregressive flows have a fast and a slow direction of evaluation. When density estimation is fast, they are called "Masked Autoregressive Flows" (MAFs) [103], when sampling is fast, they are called "Inverse Autoregressive Flows" (IAFs) [104], with the autoregressive property being realized by masked neural networks called MADE (from "Masked Autoencoder for Distribution Estimation") blocks [108] in both cases. For further details and a review of common architectures for building these normalizing flows, please refer to the works by Kobyzev *et al.* [100] and Papamakarios *et al.* [101], from which the notation has been adapted.

### *4.1. L2LFlows*

By Thorsten Buss, Sascha Diefenbacher, Frank Gaede, Gregor Kasieczka, Claudius Krause, and David Shih, with figures and tables referring to these approaches as `L2LFlows-MAF` [31, 32] and `conv. L2LFlows` [32] and code being available at [33].

---

*Introduction* Following [109, 31, 35], we split the task of learning the distribution of showers into smaller pieces: A single Energy Distribution Flow and multiple Causal Flows. The Energy Distribution Flow (EDF) learns the distribution of layer energies (*i.e.*, the total energies deposited in a layer) conditioned on the incident energy. One of the Causal Flows (CFs) is trained for each of the 45 layers in the calorimeter, learning the shower shape conditioned on the incident energy, the layer energy in that particular layer, and the shower shapes of the previous layers. Conditioning on the output of previous flows is necessary to ensure consistency among the layers. Since we need the earlier flows' output as conditional input for the later flows, during generation,

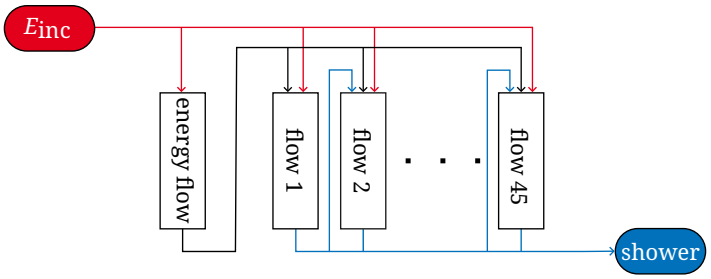



Figure 10: Diagram illustrating the overall architecture of `L2LFlows`, taken from [32]. Arrows directed at flows illustrate the conditional input of the flow. Arrows coming from flows illustrate what the flow generates.

we first draw samples from the Energy Distribution Flow and then sequentially draw samples from the causal flows. In this sense, it is an auto-regressive model. Figure 10 illustrates the generation process starting from the incident energy $E_{\text{inc}}$ and ending with a generated shower.

*Energy Distribution Flow* The task of the Energy Distribution Flow is to learn the distribution of layer energies conditioned on the incident energy

$$p(E_1, \ldots, E_N | E_{\text{inc}}) \tag{3}$$

where $E_1$ to $E_N$ denote the layer energies. The architecture used is up to hyperparameters the one published in [31]. It is a MAF [103] consisting of 6 MADE blocks [108] with rational quadratic splines (RQS) [110]. We apply fixed permutations that are randomly initialized between these MADE blocks.

Similar to `CaloFlow`, we use log and logit transformation as preprocessing. Logarithmic transformations help the network deal with inputs distributed over several orders of magnitude. Logit transformations help the network to generate only samples in an appropriate range. During inference time, the preprocessing is inverted.

Sometimes, the Energy Distribution Flow produces outliers with high energies. For that reason, we reject all sampled layer energies with an energy ratio of $E_{\text{dep}}/E_{\text{inc}} > 2.6$, where $E_{\text{dep}}$ is the total deposited energy.

*Causal Flows* Each Causal Flow learns the probability distribution of shower shapes in one particular calorimeter layer. This distribution can be denoted as

$$p(\mathcal{I}_i | \mathcal{I}_1, \ldots, \mathcal{I}_{i-1}, E_1, \ldots, E_N, E_{\text{inc}}) \tag{4}$$

where $\mathcal{I}_i \in \mathbb{R}^{n \times n}$ is the shower shape in layer $i$ given by the deposited energy in each calorimeter cell. We assume approximate locality and only pass up to five previous layers, the energy deposited in layer $i$, and the incident energy as conditional input to the flows. This helps the flow to focus on the most informative features.

We deploy two different network architectures to solve this task. The first one, called `L2LFlows-MAF`, is a MAF like the Energy Distribution Flow. This architecture is

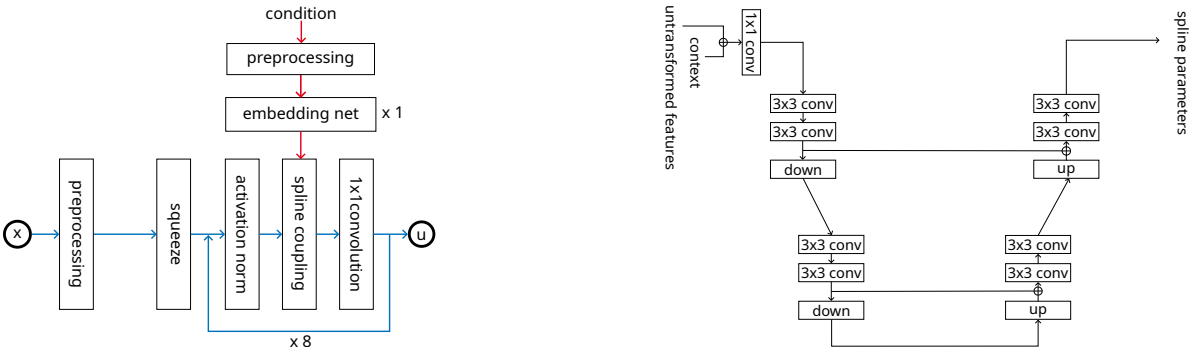



Figure 11: Diagrams illustrating the structure of our convolutional flows, taken from [32]. **Left:** The overall structure of a single Causal Flow. **Right:** A U-net as it is used in the coupling blocks.

similar to the one published in [31]. The second one, called `conv. L2LFlows`, is a flow based on coupling blocks with convolutional U-nets [111] as sub-networks.

The MAF architecture consists of four MADE blocks alternated with randomly initialized but fixed permutations. To better deal with high-dimensional conditional inputs, a summary network is applied. It receives all the conditional inputs and summarizes the information. Our summary network has 64 output knots.

In figure 11, the convolutional flow architecture is illustrated. On the right-hand side, we see how a data sample $x$ is transformed into a noise sample $u$. First, it is transformed using a preprocessing function. Next, a squeezing operation [105] stacks pixels lying in small patches into different channels. This is necessary since, in the coupling blocks, we want to split the information along the channel dimension in order to preserve the spatial structure.

The heart of this architecture consists of eight GLOW blocks [106]. They comprise an activation norm, a spline coupling block [112, 110], and an invertible $1\times1$ convolution, where the activation norm is a normalization operation, and the invertible $1 \times 1$ convolution replaces the random but fixed permutation in the MAF. The spline coupling block can learn correlations between pixels and transform inputs in a nonlinear way.

We used U-nets to learn features on different scales. This is in contrast to RealNVP [105] and GLOW[106], which used a so-called multi-scale architecture. The U-nets are employed in the coupling blocks as sub-networks. We found this setup to be more flexible and to result in higher fidelity than the multi-scale architecture from RealNVP. The U-net architecture is illustrated on the left-hand side of figure 11.

Since convolutional architectures scale much better with input dimension, the main task of the embedding network is not to reduce the number of inputs but rather to bring the input in a shape the flow can handle.

*Training* We train the `conv. L2LFlows` on datasets 2 and 3 for 800 epochs. The MAF version, `L2LFlows-MAF`, is only trained on dataset 3 for 1000 epochs. In both cases, Adam [90] is used as an optimizer. An exponential decay learning rate scheduler is

used in the MAF version, while a one-cycle learning rate scheduler [113] is used in the convolutional version. To ensure a stable training behavior, L2 regularization [114] and L2 gradient clipping are applied.

To mitigate the challenges arising from data sparsity, we fill zero voxels with log Gaussian distributed values. Since this noise is below the energy threshold, it will be cut away after generation. Furthermore, we add noise between zero and 1 keV to all voxels. We rotate showers by random angles during training as data augmentation.

The log-likelihood is additive under joining distributions. Therefore, training each flow individually is equivalent to training all flows jointly. This allows for straightforward parallelization on different compute nodes.

We implement our models using PyTorch [115] and NFlows [116]. Using single floating point precision is sufficient since we fixed numerical instabilities in NFlows.

### *4.2. (inductive) CaloFlow*

By Matthew R. Buckley, Claudius Krause, Ian Pang, and David Shih, with figures and tables referring to these approaches as `CaloFlow teacher` [34], `CaloFlow student` [34], `iCaloFlow teacher` [35], and `iCaloFlow student` [35] and code being available at [36, 37].

---

*Introduction* Following the excellent performance of `CaloFlow` [109, 117] on a simplified calorimeter setup, we adapt `CaloFlow` to the more realistic setup in dataset 1. This corresponds to `CaloFlow teacher` [34] and `CaloFlow student` [34] submissions.

*Architecture and Preprocessing* In `CaloFlow`, we implement a two-flow method that learns the normalized voxel-level shower energies $\hat{\mathcal{I}}_a$ conditioned on the corresponding incident energies of the showers $E_{\text{inc}}$ denoted by $p(\hat{\mathcal{I}}_a|E_{\text{inc}})$. Here $a$ is the voxel index and the normalization is performed for each layer such that the normalized voxel energies in each layer sum to unity. In the original `CaloFlow` studies [109, 117], it was found that training a single flow to obtain $p(\hat{\mathcal{I}}_a|E_{\text{inc}})$ resulted in problems related to energy conservation. Hence, `CaloFlow` makes use of a two-flow (flow-1 and flow-2) setup. Flow-1 is constructed to learn the probability density of calorimeter layer energies¶ conditioned on incident energy $p_1(E_i|E_{\text{inc}})$, while flow-2 is designed to learn the probability density of the voxel level shower energies conditioned on incident energy and calorimeter layer energies $p_2(\hat{\mathcal{I}}_a|E_{\text{inc}}, E_i)$. When sampling from `CaloFlow`, the layer energies are first sampled using flow-1 given an input shower incident energy $E_{\text{inc}}$. Next, the layer energies $E_i$ from flow-1 and the incident energy $E_{\text{inc}}$ are used as conditional inputs for flow-2 which outputs the shower distribution $\hat{\mathcal{I}}_a$ of the event. Both flow-1 and flow-2 are chosen to be MAFs [103]. In particular, their transformation functions are compositions of rational quadratic splines (RQS) [110]. A class of neural networks known

¶ The layer energy of a given calorimeter layer is the sum of all the voxel energies in that layer.

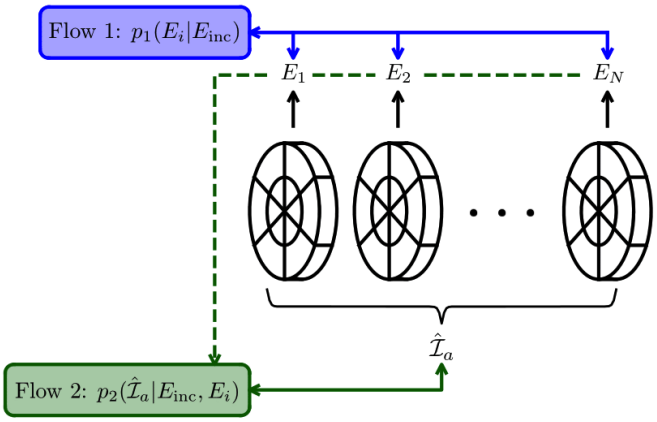



Figure 12: Schematic of the flows used in the `CaloFlow` approach. Solid lines are bidirectional — the direction into each flow denotes the density estimation step and the direction out of the flow denotes the sample generation step. Dashed lines indicate the conditional input to the respective flows.

as MADE blocks [108] are used to define the parameters $\vec{\kappa}$ of the RQS transformations. Figure 12 shows a schematic of the `CaloFlow` approach. Separate flows were trained for the photon and pion datasets.

*Training* Since MAFs are fast in performing density estimation but slow in generation, we opt to train a corresponding IAFs [104] that is fast in generation instead. We only trained an IAF for flow-2 as flow-1 has a lower dimensional output and is relatively fast to sample from. Since training the IAF with the negative log-likelihood of the data is prohibitively slow, the IAF is trained by fitting it to a pre-trained MAF using the Probability Density Distillation (PDD) method [118] that was also applied in [117]. This method is also known as teacher-student training where the MAF (IAF) is referred to as the teacher (student). The objective of this training is to enable the IAFs to learn $f_{\text{IAF}} = f_{\text{MAF}}$, or equivalently, to ensure that $f_{\text{MAF}}$ and $f_{\text{IAF}}^{-1}$ serve as inverse functions of each other. This equivalence is crucial, as only the fast passes through the flows can be used meaningfully for optimization. In practice, the fitting is implemented based on two training loss terms that we refer to as $z$ and $x$-losses. To compute the $z$-loss, we begin with a sample $z$ which is then passed through the student IAF to obtain a sample $x'$ in data space and the corresponding likelihood $s(x')$. The data sample $x'$ is then mapped via the teacher MAF back to the latent space which obtains the likelihood $t(x')$. Similarly, to compute the $x$-loss, one can start with a data sample $x$ which maps to latent space $z'$ via the teacher, and then map back to data space via the student. In the original PDD study [118] study, the KL divergence of $s(x')$ and $t(x')$ was initially used as the training loss. However, the authors noted that it does not converge well. Hence, as in [117], we used a training loss function that is based on a mean square

error that compares relevant values[+] at each equivalent stage of the teacher and student passes.

In `CaloFlow` , each flow-1 MAF consists of six MADE blocks, while each flow-2 MAF/IAF consists of eight MADE blocks. We used two hidden layers for each flow-1 MADE block and a single layer for each flow-2 MADE block. The hidden layers in flow-1 each have 64 nodes. We experimented with different flow-2 MADE block hidden layers sizes and settled for the following choice: 378 for $\gamma$ teacher; 736 for $\gamma$ student; 533 for $\pi^+$ teacher; 500 for $\pi^+$ student.

*Introduction* Applying `CaloFlow` to the higher dimensional voxelization in datasets 2 and 3 is extremely memory intensive. The number of splines and therefore RQS parameters grows linearly with data dimension $d$, making the number of parameters in the output layer grow as $\mathcal{O}(d^2)$, easily dominating over the number of parameters in the hidden layers. Hence, we proposed a new method, that we dub inductive `CaloFlow` or `iCaloFlow`, to overcome this obstacle. This method corresponds to the `iCaloFlow teacher` [35] and `iCaloFlow student` [35] submissions.

*Architecture and Preprocessing* Our `iCaloFlow` method uses three normalizing flows to learn and generate calorimeter showers. 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 generation step. 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, $E_1$: $p_2\left(\hat{\mathcal{I}}_{1a}|E_{\text{inc}}, E_1\right)$. Here $a$ is the voxel index. 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 $i^{\text{th}}$ layer is conditioned on the energy deposited in the layers $i$ and $i-1$ ($E_i$ and $E_{i-1}$), the incident energy $E_{\text{inc}}$, the unit-normalized voxel energies in the $(i-1)^{\text{th}}$ layer $\hat{\mathcal{I}}_{(i-1)a}$, and the one-hot† encoded layer number $i$: $p_3\left(\hat{\mathcal{I}}_{ia}|E_{\text{inc}}, E_i, E_{i-1}, \hat{\mathcal{I}}_{(i-1)a}, i\right)$. Figure 13 shows a schematic of the `iCaloFlow` approach. Like in `CaloFlow`, we used MAFs with RQS transformations for flow-1, and MAF-IAF pairs for flow-2 and flow-3. For dataset 2, flows-2 and -3 in `iCaloFlow` consist of eight MADE blocks with two hidden layers of size 256‡. For dataset 3, the configuration is similar, only flow-3 uses just one hidden layer for both MAF and IAF setups. Flow-1

[+] For the student model for ds 1 – $\gamma$, these consist of coordinates before and after passing them through the flows and RQS parameters from individual MADE blocks within the bijectors. For the student model for ds 1 – $\pi^+$, we did not enforce agreement with the teacher at the level of individual MADE blocks, but only at the endpoints of the flows.

† 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.

‡ With the exception of the flow-3 IAF, which has hidden layer sizes of 384.

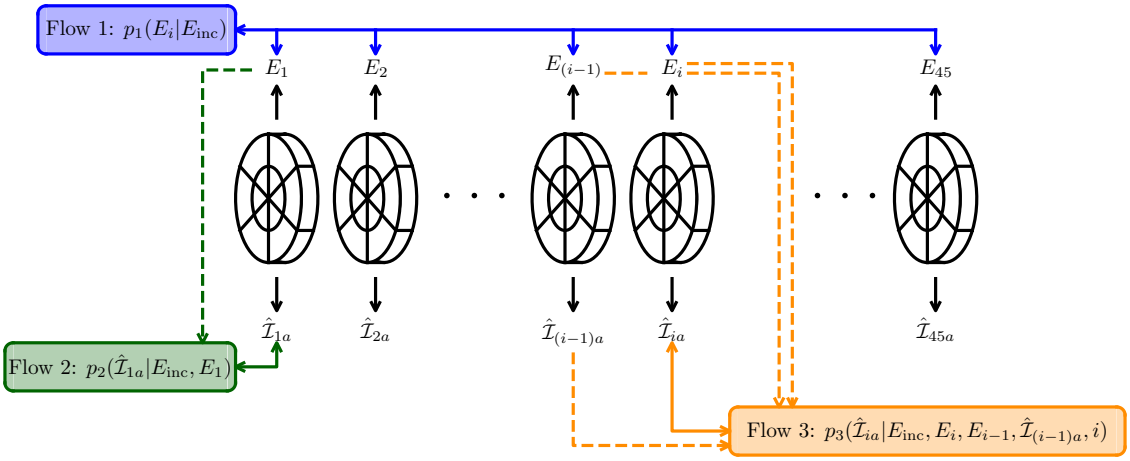



Figure 13: Schematic of the three iCaloFlow flows, taken from [35]. Solid lines are bidirectional — the direction into each flow denotes the density estimation step and the direction out of the flow denotes the sample generation step. Dashed lines indicate the conditional input to the respective flows. Flow-3 is used iteratively on subsequent layers.

always has just one hidden layer.

The number of trainable parameters for the CaloFlow models are included in Tables C5 and C12. For the teacher models, the total parameter count matches that of sample generation, which is the sum of parameters in flow-1 and flow-2 (teacher). As for the student models, the parameter count during sample generation is the sum of parameters in flow-1 and flow-2 (student). Given the necessity of a pre-trained teacher model for each student model, the total parameter count encompasses parameters from flow-1, flow-2 (teacher), and flow-2 (student).

### *4.3. CaloINN*

By Luigi Favaro, Florian Ernst, Claudius Krause, Tilman Plehn, and David Shih, with figures and tables referring to this approach as CaloINN [38] and code being available at [39].

---

*Introduction* In CaloINN [38] we train a normalizing flow for the generation of showers in dataset 1 and dataset 2. We use the INN variant of FrEIA [119] with coupling layers, which unlike autoregressive methods provides fast evaluation in the forward and backward directions. This is achieved by transforming only a subset of the input features with a reversible transformation. The parameters of the transformation are predicted by a network conditioned on the remaining features and the incident energy. The CaloINN

architecture allows for a generation step of $\mathcal{O}(1)$ ms per shower on a single GPU without the necessity of a second distillation process.

*Architecture and Preprocessing* The architecture takes voxels normalized by the layer energy as input. The information of the energy per layer is encoded in extra energy dimensions, similarly preprocessed as in `CaloFlow teacher`, as shown in (5). To explore the expressive power of a single flow network, we simply append the energy ratio variables to the feature vector. We do not explore a separate training for the energy and the voxel dimensions which would simplify and improve the learning process of the energy dimensions.

$$u_0 = \frac{\sum_i E_i}{E_{\text{inc}}} \qquad \text{and} \qquad u_i = \frac{E_i}{\sum_{j \geq i} E_j} \,, \tag{5}$$

After creating the final feature vector, we add uniform noise sampled from $\mathcal{U}(0, b)$, where $b = 5 \cdot 10^{-6}$ for dataset 1 and $b = 10^{-6}$ for dataset 2. Then, we apply a regularized log transformation with parameter $\alpha = 10^{-8}$ according to

$$x = \log(x_0 + \alpha) \,. \tag{6}$$

We show the schematic of a coupling block in figure 14. In each coupling block, the input vector is split in two halves, $x_t$ and $x_c$, of equal size. The block only transforms half of the input features, which are selected randomly during initialization, defining the transformation

$$T(x_t, x_c; E_{\text{inc}}) = \begin{cases} y_t = f(x_t; x_c, E_{\text{inc}}) \\ y_c = x_c \end{cases} \,. \tag{7}$$

The splitting between $x_t$ and $x_c$ is fundamental in coupling layer-based normalizing flows. This choice provides a triangular Jacobian matrix which is tractable and can be promptly evaluated both during training and generation. The transformation $f$ applied to the features is a RQS [110] for dataset 1 and a cubic spline for dataset 2. The prediction of the spline parameters is obtained with a sub-network consisting of a sequence of dense layers with 256 nodes for each hidden layer. The number of hidden layers is four for dataset 1 and three for dataset 2. After permuting the order of the features, we normalize the output to mean zero and unit standard deviation with an ActNorm [106] layer. This allowed us to improve the stability of the training and utilize a deeper model. In the large-scale architecture, we stack twelve blocks for dataset 1 and fourteen blocks for dataset 2 to construct the full flow.

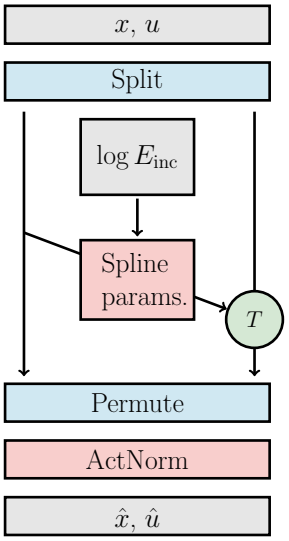



Figure 14: Schematic representation of the `CaloINN` coupling block, taken from [38].

### *4.4. SuperCalo*

By Ian Pang, John Andrew Raine, and David Shih, with figures and tables referring to this approach as `SuperCalo` [40] and code being available at [41].

---

*Introduction* Our approach, which we dub as `SuperCalo` [40], presents a way to generate high-dimensional calorimeter showers by super-resolving low-resolution calorimeter showers. The showers used in the CaloChallenge datasets are represented as 3-dimensional images that are binned into voxels in position space. We will refer to these voxels as *fine voxels*. A coarse-grained representation of each shower can be obtained by grouping together neighboring fine voxels to make *coarse voxels*. In this approach, we split the task of learning $p(\vec{E}_{\text{fine}}|E_{\text{inc}})$ into two parts. Here $\vec{E}_{\text{fine}}$ is the energy deposited in the fine voxels and $E_{\text{inc}}$ is the incident energy of the particle. First, we learn to sample from $p(\vec{E}_{\text{coarse}}|E_{\text{inc}})$, where $\vec{E}_{\text{coarse}}$ is the energy deposited in the coarse voxels. Next, we learn to super-resolve the coarse voxels to obtain the fine voxels, which is equivalent to sampling from $p(\vec{E}_{\text{fine}}|\vec{E}_{\text{coarse}})$.

*Architecture and Preprocessing* However, trying to learn $p(\vec{E}_{\text{fine}}|\vec{E}_{\text{coarse}})$ with a single model would be no better in terms of model size than the original problem of learning $p(\vec{E}_{\text{fine}}|E_{\text{inc}})$. As a result, we rewrite the distribution according to the following ansatz:

$$p(\vec{E}_{\text{fine}}|\vec{E}_{\text{coarse}}) = \prod_{i=1}^{N_{\text{coarse}}} p(\vec{e}_{\text{fine},i}|E_{\text{coarse},i}, \ldots). \tag{8}$$

In other words, each coarse voxel, with deposited energy $E_{\text{coarse},i}$, is upsampled to its fine voxels, with deposited energies $\vec{e}_{\text{fine},i}$, using a universal super-resolution model

that may be conditioned on some coarse shower information. Here is the list of the conditional inputs that we used in the super-resolution model:

- Incident energy of the incoming particle, $E_{\text{inc}}$
- Deposited energy in coarse voxel $i$, $E_{\text{coarse},i}$
- Fine layer energies of layers spanned by coarse voxel $i$
- Deposited energy in neighboring coarse voxels in $\alpha$, $r$ and $z$ directions§
- One-hot encoded coarse layer number
- One-hot encoded coarse radial bin

Since it is not obvious which choice of coarse shower representation would result in the highest fidelity high-resolution showers (fine voxels), we experimented with a few choices and picked the one that gave the best results. In particular, we grouped the fine voxels such that 1 coarse voxel = 1 $r$ bin $\times$ 2 $\alpha$ bins $\times$ 5 $z$ bins. This choice results in a 648 dimensional coarse shower.

Similar to `CaloFlow teacher` [34] and `CaloFlow student` [34], we used a flow-1 + flow-2 setup to learn the distribution of energy deposited in coarse voxels in each shower conditioned on the incident energy of the particle $p(\vec{E}_{\text{coarse}}|E_{\text{inc}})$. Next, we train our super-resolution flow to learn $p(\vec{e}_{\text{fine},i}|E_{\text{coarse},i},\ldots)$. Then, generating showers with the full model chain involves sampling sequentially from flow-1, flow-2 and the super-resolution flow. All the flows used in this work are MAFs [103] with RQS [110] transformations. Like in `CaloFlow`, the parameters of each RQS transformation are computed using a MADE block [108]. Each flow consists of eight MADE blocks. Flow-1 (2) has MADE blocks with a single hidden layer with 256 (648) nodes. The super-resolution flow has MADE blocks with two hidden layers, each with size 128.

### *4.5. CaloPointFlow*

By Simon Schnake, Benno Käch, Moritz Scham, Dirk Krücker, and Kerstin Borras, with figures and tables referring to this approach as `CaloPointFlow` [42] and code being available at [43].

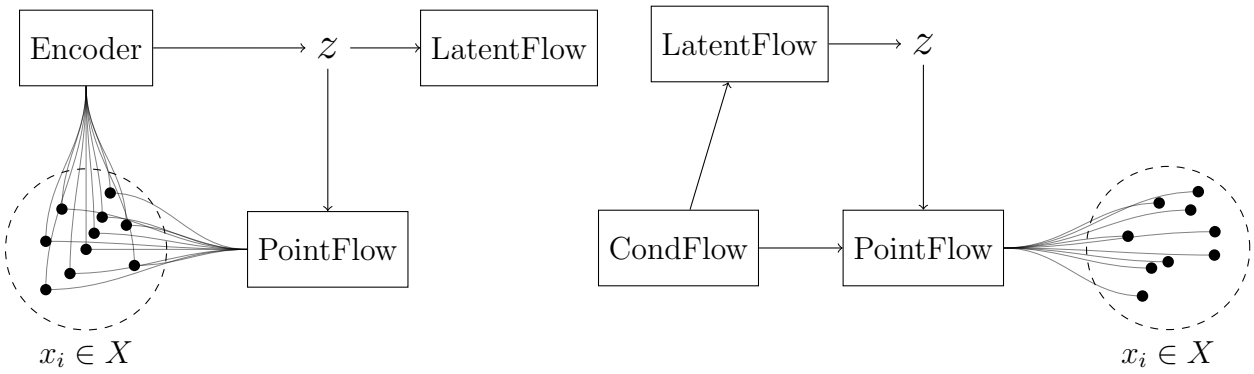
---

*Introduction* The requirement for fast simulation of calorimeter showers has led to a growing interest in using machine learning models for their efficient and high-fidelity generation. Calorimeter showers are generally sparse, with a majority of calorimeter cells being empty, necessitating a representation that is efficient and effective at capturing the essential features of the data. Point clouds offer an apt solution for representing sparse data structures due to their innate efficiency. Our modified model builds upon

§ There is maximum of 6 neighboring coarse voxels for each coarse voxel. For coarse voxels with fewer than 6 adjacent coarse voxels, the missing neighboring coarse voxel energies are padded with zeros.

the original PointFlow [120] model known for its exceptional ability to produce high-quality point clouds. The `CaloPointFlow` model leverages PointFlow's advantages while making specific adjustments to specialize in generating calorimeter data.

The model consists of four sub-models, as shown in figure 15. The initial sub-model, CondFlow, is responsible for generating the number of hits, referred to as $n_{\text{hits}}$, and the total energy, $E_{\text{sum}}$, in the calorimeter cells by a normalizing flow. The second stage comprises the permutation invariant encoder, which transforms the entire point cloud $X$ into a latent representation $z$. This transformation is based on the DeepSets [121] architecture. Subsequently, the LatentFlow, the third sub-model, produces the latent representation $z$, which is conditioned on the values of $E_{\text{sum}}$ and $n_{\text{hits}}$. The final component, the PointFlow, is a permutation equivariant normalizing flow that performs pointwise transformations. PointFlow is conditioned on $z$, as well as $E_{\text{sum}}$ and $n_{\text{hits}}$.




(a) Training (b) Sampling

Figure 15: Architecture of `CaloPointFlow`, taken from [42]. **Left:** In training. **Right:** In sampling.

*Preprocessing* Concerning the conditional variables learned through CondFlow, the number of hits, $n_{\text{hits}}$, is processed by adding uniform noise ranging from 0 to 1 and then dividing by the square root of the input energy, $E_{\text{in}}$. The total energy of the shower, $E_{\text{sum}}$, is normalized by dividing it by $E_{\text{in}}$. We then log-transform and normalize the resulting values.

Discrete distributions in continuous space resemble delta distributions and pose modeling challenges using a normalizing flow. To overcome this, dequantization techniques transform discrete distributions into continuous ones. This usually requires adding uniform noise to fill the space between 0 and 1, followed by a logit transformation [122, 105]. We introduce a new dequantization strategy, CDFDequantization, displayed in figure 16. This approach utilizes the quantile function and the cumulative distribution function (CDF), which establish invertible mappings between their distribution and the standard uniform distribution. This principle is inherent in inverse sampling, where the uniform distribution is segmented into parts

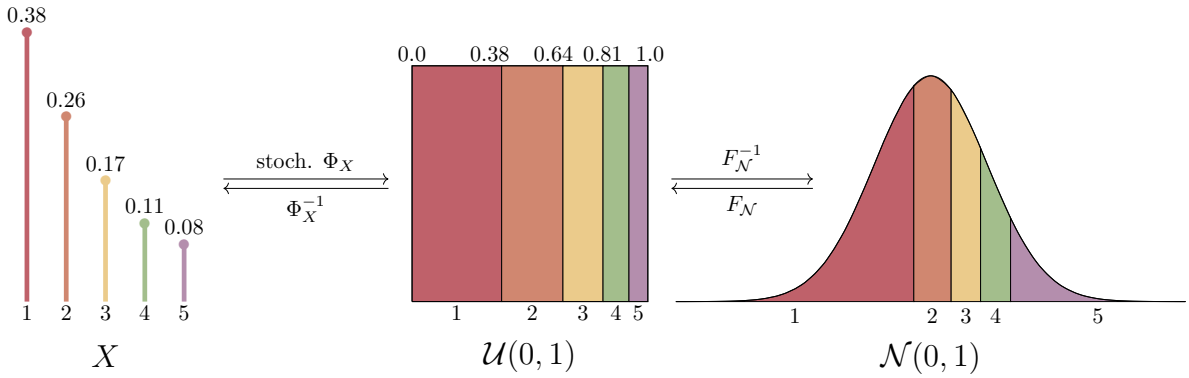



Figure 16: The CDFDequantization, adapted from [42].

corresponding to the probabilities of certain discrete values. Although the CDF mapping is not directly invertible, a stochastic inverse can be developed by transforming each discrete value $d$ into $CDF(d) + PMF(d) \cdot u$, where $u$ is randomly drawn from $\mathcal{U}(0,1)$. This process effectively raises our discrete distribution to $\mathcal{U}(0,1)$. To map the uniform distribution to the entire real space, and convert $\mathcal{U}(0,1)$ to a logistic distribution, a logit transform is typically applied in standard dequantization. However, we choose to employ the quantile function of the standard normal distribution to map our values onto the normal distribution. This design ensures the marginal distributions of our model are intrinsically normal, with correlations being the only remaining aspect to be modeled.

*Architecture* All three flows — CondFlow, LatentFlow, and PointFlow — are coupling layer-based flows [105] utilizing RQS [110]. In the previous iteration of the `CaloPointFlow` model, a coupling-layer-based normalizing flow was applied on a point-by-point basis. This involved dividing point features into two equal segments using a system of coupling blocks, with one segment being transformed based on the other. The final result is a permutation invariant system capable of managing varying numbers of points. However, a significant challenge remained in this model: it lacked a mechanism to facilitate the exchange of information between points, making it prohibitive to model inter-point correlations.

In order to solve this problem, we have implemented a modification that makes use of Deep Sets [121]. Following our coupling strategy, we opted to aggregate half of the point features objectively. We maintain transformation permutation invariance by initially mapping each point to a latent space that has high dimensionality. Afterwards, we perform a pooling operation such as max, sum, or mean to merge the information. We then transformed the aggregated latent information and incorporated it as an additional conditional variable. This process enables our point-wise normalizing flows to transfer inter-point information and therefore enhances the model's ability to capture correlations.

A major concern in point cloud-based models for simulating calorimeters is the

presence of multiple hits per cell. This occurs when points, produced in continuous space, are mapped onto the discrete space of calorimeter cells, which may result in multiple points being assigned to a single cell. This inconsistency contradicts the real data, where each cell can have only one energy value. To address this matter, the model must accurately determine the direct distances between points as well as the occupancy status of each cell, a task that is notably complex. The `CaloPointFlow` model, for instance, could not carry out this function.

Our strategy for mitigating the issue leverages the rotational invariance characteristic of the detector. The principle of rotational invariance implies that the marginal energy distribution in the angular coordinate $\alpha$ should exhibit a uniform, or flat distribution. To utilize this characteristic, we confine the generation of particle showers to the $(z, r)$ plane. Afterward, we randomly assign the angular position $\alpha$. This method effectively loosens the constraint of limiting the number of showers per cell. In dataset 2, this relaxation allows up to 16 showers per cell, and for dataset 3, the number rises to up to 50 showers per cell.

However, there may be instances where the number of hits surpasses the specified limits. In these cases, the extra hits are randomly allocated to already occupied $\alpha$ regions. Although this approach does not offer a complete resolution to the problem of multiple hits, it has been demonstrated to considerably improve the experimental outcomes in practical settings. The efficacy of the mentioned approach in augmenting the quality of data highlights its potential as a significant temporary remedy, while we attempt to devise more resilient techniques to tackle the intricacies linked with the occurrence of multiple hits in particle detectors.

## 5. Diffusion-based Submissions

Diffusion models are a class of generative models based on applying a chosen perturbation to the data and then training a model to invert that perturbation. The model is defined in terms of a forward process, in which perturbations are gradually applied to the data sample eventually reaching a known end-point distribution (such as Gaussian noise). A model is then trained to learn the reverse process, which inverts the perturbations to recover the original data sample. Once trained, new samples can be generated by sampling from the end-point distribution and iteratively applying the reverse process model. Diffusion models have been defined under two different formalisms, a score-based formulation and a denoising formalism.

In score-based models, the forward process is defined by a stochastic differential equation (SDE):

$$dx = f(x, t)dt + g(t)dW, \tag{9}$$

where $f(x, t)$ and $g(t)$ are user-specified diffusion and drift functions, respectively. $W$ is a Wiener process (Brownian motion), indexed by a time parameter $t \in [0, 1]$. The

reverse process can then be solved by the following SDE:

$$dx = [f(x,t) - g(t)^2 \nabla_x \log p_t(x)]dt + g(t)dW, \tag{10}$$

where the $\nabla_x \log p_t(x)$ is the 'score' of the data, or the gradient of the log probability. A denoising score-matching strategy is used to learn the score function [123] and then used in (10) to generate samples.

In the denoising diffusion formalism, as formulated in the original DDPM paper [124], the forward process is defined by the repeated addition of Gaussian noise to the original data in $t$ steps:

$$x_t = \sqrt{1-\beta_t} x_{t-1} + \beta_t \epsilon, \tag{11}$$

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t \mid \sqrt{1-\beta_t} x_{t-1}, \beta_t), \tag{12}$$

where $\mathcal{N}(x \mid \mu, \sigma)$ is a Gaussian likelihood and $\beta_t$ is a user-chosen 'noise schedule' that specifies how much noise is added at each step. For a sufficiently large $T$ (the total number of diffusion steps), the Gaussian noise will overwhelm the original data and $x_T$ will follow a multivariate Gaussian distribution. The reverse process is also assumed to follow a Gaussian likelihood:

$$p(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1} \mid \mu_\theta(x_t, t, z), \beta_t \mathcal{I}), \tag{13}$$

with an unknown mean $\mu_\theta$ that is learned by a neural network during training.

Though appearing conceptually different, these two formalisms have been shown to be mathematically equivalent [125] for a particular choice of the drift and diffusion functions (the 'variance-preserving' choice): the optimal model trained under one formalism is optimal for the other as well. Though in practice, because optimality is never reached, the two formalisms may offer different practical advantages and disadvantages. The diffusion literature is rapidly evolving and newer models generally alter these formalisms slightly, but the key conceptual ideas remain.

### *5.1. CaloDiffusion with GLaM*

By Oz Amram and Kevin Pedro, with figures and tables referring to this approach as `CaloDiffusion` [44] and code being available at [45].

---

*Introduction* `CaloDiffusion` [44] is based on denoising diffusion models [124], in which the perturbation applied to the image is an addition of Gaussian noise. We use the cosine noise schedule proposed in [126] with 400 diffusion steps for all datasets. Shower preprocessing is done similarly to other approaches, where the voxel energies are divided by the incident particle energy and logit transformed.

*Architecture* The denoising model follows a U-net architecture [111], with 3 sets of ResNet [127] blocks with linear attention [128]. The input is compressed by a factor of two in each dimension after each of the first two ResNet blocks. The architecture is then

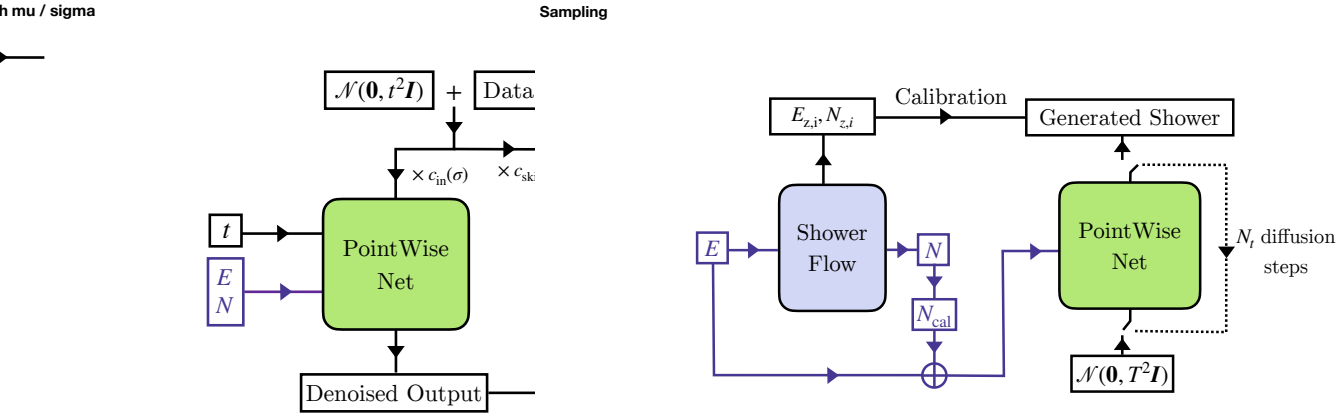
mirrored, with 3 ResNet blocks with 2 upsampling layers to return to the original data shape. Skip connections are used to ensure no information bottleneck. Conditioning variables — the diffusion noise level and the incident particle energy — are processed by a DNN and then added to the model in the middle of each ResNet block.

We make several optimizations focused on the cylindrical geometry of shower datasets. This includes cylindrical convolutions that the respect the periodic nature of the angular dimension and a novel method to condition the convolutions on the layer and radial bin values.

To handle the irregular geometry of dataset 1, we introduce a new approach: Geometry Latent Mapping (GLaM). GLaM learns an embedding from the dataset 1 geometry to a perfectly regular cylindrical geometry. Unlike an autoencoder, the embedding space is larger than the input space, so that no information is lost. The mapping is learned separately for each layer and initialized based on the geometric overlap of the input cells with a perfect cylinder. The cylindrical data is then processed using the cylindrical convolutions and then a reverse embedding is learned to restore the original shape.

### *5.2. CaloClouds: Fast Geometry-Independent Highly-Granular Calorimeter Simulation*

By Erik Buhmann, Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Anatolii Korol, William Korcari, Katja Krüger, and Peter McKeown, with figures and tables referring to this approach as `CaloClouds` [46, 47] and code being available at [48, 49].




(a) Training

(b) Sampling

Figure 17: `CaloClouds` training and sampling pipeline, taken from [47]. **Left:** Training of the PointWise Net. The training of the Shower Flow is not shown. **Right:** Sampling of `CaloClouds`.

*Introduction* The `CaloClouds` model family was introduced in [46]. The improved version, `CaloClouds II` [47] with consistency distillation [129], is here adapted to

dataset 3. As a point cloud generative model, `CaloClouds` consists of two sub-models: A normalizing flow model dubbed *Shower Flow* and a diffusion model named *PointWise Net*. An overview of the training and sampling pipeline is shown in figure 17.

*Architecture* As the names suggest, the Shower Flow generates several global calorimeter shower observables, *i.e.* the layer-wise visible energy, the layer-wise number of hits, as well as the center of gravity (center of energy) in $x$- and $y$-direction. The Shower Flow is conditioned on the incident particle's energy. We implemented it with a single bijector consisting of ten normalizing flow blocks, each containing seven coupling layers. Out of those, six are affine transformations [105] and one a RQS [110]. The generated observables are used for a post-diffusion calibration of said shower features and the total number of hits is used for the conditioning of the diffusion model.

The PointWise Net diffusion model is conditioned on the incident energy as well as the number of hits. The diffusion model is based on the implementation in [130] and the network architecture is adapted from [131]. As the name suggests, it generates each calorimeter hit i.i.d. (independent and identically distributed). From a shower physics perspective, this i.i.d. assumption is inaccurate, however it yields decent performance and allows for a fast sampling necessary for large cardinality calorimeter showers such as the ones studied in [46, 47] and in dataset 3. Layer structures taking into account inter-point correlations could be considered for smaller point clouds such as dataset 1 and are implemented, *i.e.* in `CaloGraph` [54].

*Training* Both the Shower Flow and the PointWise Net are trained separately and used sequentially during sampling of each batch. For the generation with the PointWise Net, we apply 13 diffusion steps with the Heun ODE solver, resulting in 25 function evaluations per batch. To apply the `CaloClouds` model to dataset 3, we transformed each shower into a point cloud with four features: hit energy and the Cartesian coordinates in 3-dimensional space. We normalized the 3-dimensional coordinates to the range $\{x, y, z\} \in [-1, 1]$. This results in calorimeter shower point clouds with a cardinality of up to 20 000 hits. To compare the `CaloClouds` model to other models, we projected the calorimeter point clouds back to the (voxelized) fixed grid. As this leads to a clustering of a few generated point into a single voxel, we apply a calibration step between the predicted number of hits $N$ and the (larger) calibrated generated number of hits $N_{\text{cal}}$ used for conditioning of PointWise Net.

In [46, 47], the `CaloClouds` models are used to generate point clouds with each point clouds representing clustered GEANT4 steps (simulated energy depositions) with a resolution 36 times higher than the actual resolution of the simulated electromagnetic calorimeter. A subsequent projection of there ultra-high granular calorimeter point clouds (up to 6000 points per shower) into the regular high-granularity calorimeter cells (up to 1500 calorimeter hits) results in a clustering of points leading to a precise estimation of the number of (regular cell) hits. As dataset 3 only contains regular cell hits, we expect the `CaloClouds` model performance to improve when using a ultra-high

granularity point cloud dataset. Nonetheless, even when generating showers with regular cell hits we observe a decent performance. Further speed-ups of the diffusion process could be achieved by applying consistency distillation [129], which allows for single shot generation without significant loss in fidelity.

### *5.3. Score-based Generative Models for Calorimeter Shower Simulation*

By Vinicius Mikuni and Ben Nachman, with figures and tables referring to these approaches as `CaloScore` [50, 51], `CaloScore distilled` [50, 51], and `CaloScore single-shot` [50, 51] and code being available at [52, 53].

---

*Introduction* Continuous diffusion generative models, or score-based models aim to approximate the score function of the data $\nabla \log(p(x))$ for data described by the probability density $p(x)$. The advantage of this approach is that both stochastic and deterministic solvers can be used for the generation of new observations, often leading to faster sampling times. The first diffusion generative model applied to collider physics problems was introduced in [50] and later updated to improve the generation quality and speed in [51]. In the updated version, a neural network output $v_\theta(x_t, t)$ is used to calculate the loss function by minimizing the quantity

$$\mathcal{L} = \mathbf{E}_{x_t,t} \left\| v_t - v_\theta(x_t, t) \right\|^2 . \tag{14}$$

The velocity term $v_t \equiv \alpha_t \epsilon - \sigma_t x$ is calculated based on data $x_t$ that has been perturbed by a time-dependent Gaussian perturbation $q(x_t|x) = \mathcal{N}(x_t, \alpha_t x, \sigma_t^2 I)$. The velocity parametrization is observed to lead to a lower variance loss, improving the quality of the generated samples. The approximation to the score function $s_\theta(x_t, t)$ is identified as

$$s_\theta(x_t, t) = x_t - \frac{\alpha_t}{\sigma_t} v_\theta(x_t, t). \tag{15}$$

New samples are generated from the trained model by solving the following ordinary differential equation,

$$\frac{\mathrm{d}x_t}{\mathrm{d}t} = f(x, t) - \frac{1}{2} g(t)^2 \nabla_x \log q(x_t) \,, \tag{16}$$

with the DDIM solver [132] with update rule then specified by

$$x_s = \alpha_s x_\theta(x_t, t) + \sigma_s \frac{x_t - \alpha_t x_\theta(x_t, t)}{\sigma_t}, \tag{17}$$

for time $s < t$ and position prediction $x_\theta(x_t, t) = \alpha_t x_t - \sigma_t v_\theta(x_t, t)$. While the solver still require a large number of function evaluations ($\mathcal{O}(100)$), we are able to reduce this number trough a distillation procedure [133], resulting in faster generation times requiring even a single step for the generation.

*Architecture* The neural network architecture used for the training is similar to the one used in the initial `CaloScore` paper, based on the U-net [111] architecture with additional attention layers. More specifically, datasets 2 and 3 have the number of spatial components in each dimension reduced by a factor 2 every other convolutional layer (resulting in a factor $2 \times 2 \times 2 = 8$ reduction) with fixed kernel size set to 3. This process is repeated 3 times, with lowest dimensional representation reduced by a factor 512 compared to the initial number of voxels. The 3-dimensional convolution operations used for datasets 2 and 3 use 32, 64, and 96 hidden nodes with swish [134] activation function. The attention layer is only used at the lowest dimensional representation, with data patches determined by the flattened array describing the data at the lowest dimensionality. The upsampling section of the architecture is a mirrored version, with dimensions increased by a factor 8 every other layer. Skip connections between the downsampling and upsampling sides of the architecture are combined with a concatenation operation, completing the architecture. Conditional information consisting of the time information, incident particle energy, and deposited energy per layer (in case of the diffusion model trained to generate normalized voxels), are included through an addition operation after every convolutional layer. A trainable embedding of the conditional features is created by a fully connected layer over the conditional inputs. The output size is fixed to match the expected output size of the convolutional layers. For dataset 1, the strategy is similar. The number of voxels to be simulated is reduced by a factor 2 every other layer, with this process repeated 4 times and overall reduction of factor 16 compared to the initial size. The number of hidden nodes for the 1-dimensional convolutional layers is then chosen to be 16, 32, 64, and 96 for each fixed dimensionality. Since this dataset is smaller compared to datasets 2 and 3, attention layers are used in all lower dimensional representations of the initial data. A second diffusion model is introduced to learn only the energy deposition per layer, similar to the approach used in the original `CaloFlow` paper [109]. The model used to train the diffusion model is based on the ResNet [127] architecture, consisting of multiple fully connected layers with additional skip connections. The number of ResNet layers is set to 3 in all datasets, with 128 hidden nodes in dataset 1 and 1024 in datasets 2 and 3. Additional models are trained to reduce the sampling time of the baseline `CaloScore` model. The model architecture of the distilled version is the same as the baseline model, also using the initial baseline weights as the starting point to accelerate the training procedure.

### *5.4. CaloGraph*

By Dmitrii Kobylianskii, Nathalie Soybelman, Etienne Dreyer, and Eilam Gross, with figures and tables referring to this approach as `CaloGraph` [54] and code being available at [55].

---

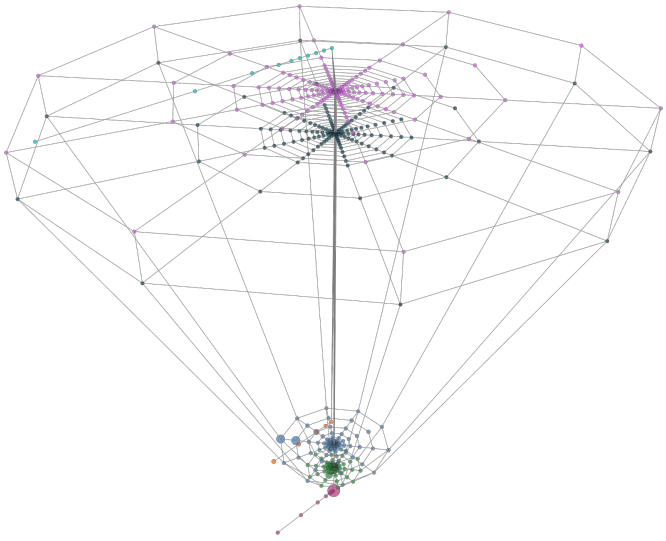


Figure 18: Event display from the pion dataset in graph form, taken from [54]. The nodes represent the center of the calorimeter cells, and their size relates to the cell energy.

*Introduction* `CaloGraph` [54] stands out as a diffusion model based on graphs specifically designed for low-granularity calorimeters with irregular geometries, such as those found in ATLAS. Unlike image-based methods that necessitate unique mappings for non-regular geometries and point cloud generation techniques that predict point positions requiring specific grid summation, a graph representation requires no pre- or postprocessing, except for the initial one-time graph construction. Calorimeter cells are nodes in the graph with fixed positions, and edges connect nearest neighbors within the given layer and the layers below and above. An example from the pion dataset 1 is shown in figure 18. However, managing a large number of edges will result in high memory

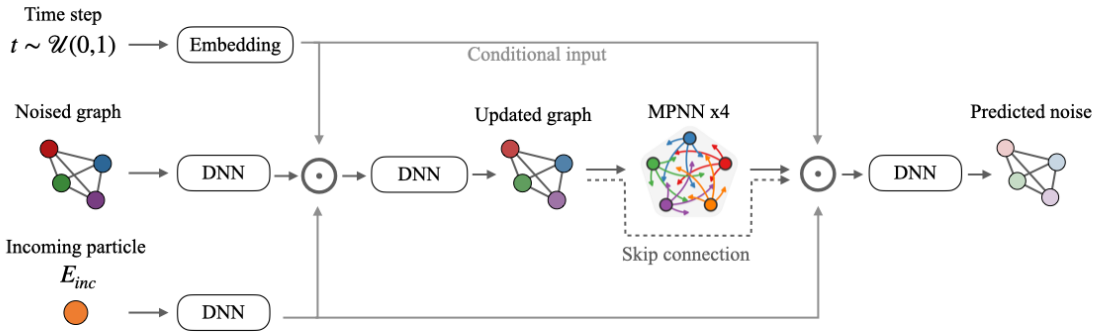



Figure 19: Architecture of `CaloGraph`, adapted from [54].

needs, making this approach mostly suited for low-granularity detectors. Therefore, we present results only for dataset 1.

*Architecture* Our approach relies on a denoising diffusion model. We use a cosine noise schedule from [126]. During the backward process, we use the "pseudo numerical methods for diffusion models" (PNDM) from [135] to solve the diffusion ODE. The architecture of the neural network is presented in figure 19.

The input to the network is the noised target graph: constructed as described above, the node features consist of the cell position ($\eta$, $\phi$, layer) and the noised cell energy. It is passed through the initial DNN and then combined with the conditional input consisting of the embedded, uniformly sampled time step as well as the incoming particle energy. The combined input is passed through another DNN, resulting in the updated graph. Subsequently, four rounds of message passing are applied before predicting the noise through a final DNN. The network has a total of 0.8 million parameters, see table C5 and table C12.

### *5.5. Diffusion Transformer*

By Renato Paulo Da Costa Cardoso, Piyush Raikwar, Anna Zaborowska, Dalila Salamani, Kristina Jaruskova, Sofia Vallecorsa, Kyongmin Yeo, Vijay Ekambaram, Nam Nguyen, Jayant Kalagnanam, and Mudhakar Srivatsa, with figures and tables referring to this approach as `CaloDiT` [56] and code being available at [57].

---

*Introduction* Currently, the state-of-the-art approach for image generation is diffusion, while the state-of-the-art architecture for almost any data modality is based on transformers [136]. We combine both methodologies for a transformer-based diffusion model.

The use of transformer models for the image generation task is not new, with approaches such as Vision transformers (ViT) [137], Swin transformers [138], etc. getting good results for image classification tasks. When paired with the diffusion process, we have impressive generative models like OpenAI Sora [139]. In our case, the model architecture is based on Diffusion with transformers (DiT) [140]. As for the diffusion process, we go with denoising diffusion probabilistic models (DDPM) [124], with the modification of using a cosine scheduler as described in [126]. We present the results of `CaloDiT` for dataset 2.

*Preprocessing* We preprocess the input data to ease the diffusion process. The preprocessing is done by scaling the shower energies in the range of -1 and 1, followed by applying a logit function and normalization of those values. The energy condition is also preprocessed to be in the 0 to 1 range via scaling it by the max energy of 1000 GeV.

*Architecture* We define the architecture of the `CaloDiT` in figure 20. We use a stack of 4 DiT blocks, which are ViT-like transformer blocks with a modified conditioning unit to accommodate diffusion timesteps. In our case, we also concatenate the energy condition along with the timestep, which is then passed to each DiT block. These conditions along with a noisy shower are passed to the model to get a denoised shower as an output. During inference, this process is repeated 400 times to generate a shower, where 400 is the number of diffusion steps our model uses.

As with any other transformer-based model, we need to represent the input as some form of sequence. While ViT and DiT are used for 2-dimensional images, the shower dataset is 3-dimensional. Thus, we split the 3-dimensional shower into multiple smaller 3-dimensional patches (patchify), 704 patches to be precise. These patches are linearly projected to a higher dimension of 144 and then passed to the model. We do the opposite for the output to combine smaller 3-dimensional patches into the 3-dimensional shower (unpatchify). Note that, we use a $2 \times 2 \times 2$ convolution layer for "patchification" to better extract the representations, but "unpatchification" is a simple reshape operation. Within each DiT block, the conditions are first passed through a DNN of 2 layers and then summed up with patch embeddings of size 144. After the final DiT block, we have layer normalization and a linear projection to match the shower dimensions, followed by unpatchification.

Since our sequence is 3-dimensional, we also adapt the sinusoidal positional embeddings [136] from 2 dimensions to 3 dimensions to represent the patches in the 3-dimensional space. This is done by allocating space for an extra dimension in the positional embedding vector. These positional embeddings are added to the patches after their linear projection before the first DiT block.

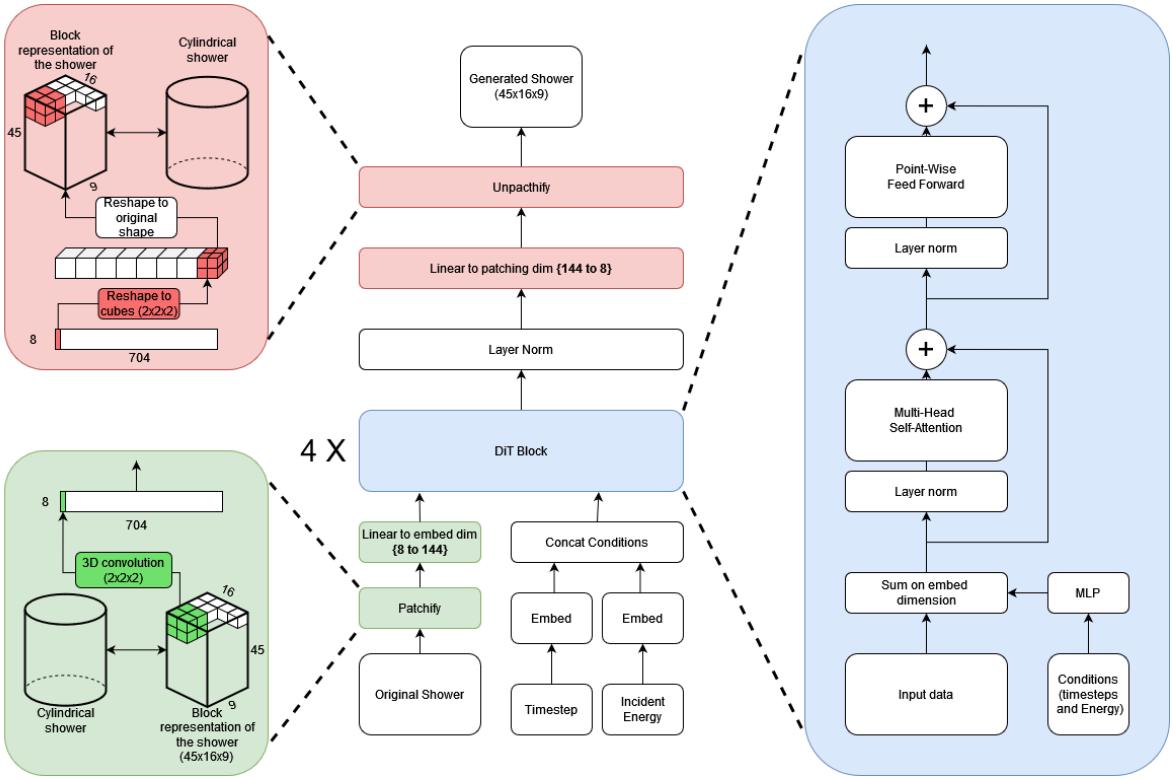



Figure 20: Architecture of `CaloDiT`.

## 6. VAE-based Submissions

Variational Autoencoders (VAEs) [141, 142] are a class of generative models which combine deep learning with probabilistic methods. A VAE is composed of two stacked neural networks acting as encoder and decoder. The encoder learns a mapping from the input space to a latent space in which a meaningful representation of the data is learned. The decoder learns the inverse mapping by reconstructing the original input from the latent representation. The VAE is designed with a prior on the representation space, hence once the model is trained to reconstruct the input, the decoder can be used independently as a generator of new data by sampling from the prior.

Moreover, improved accuracy can be achieved by performing density estimation on the latent space. The VAE thus serves as a dimensionality reduction method by learning a low-dimensional manifold of the ambient space. The approach is explored with different techniques and density estimators in this section.

The key idea behind training VAEs lies in variational inference (VI). VI approximates probability densities based on the optimization of the Kullback-Leibler (KL) divergence [143]. VI uses a family of densities and finds the closest member of that family to the target density using the KL divergence. The KL divergence is a fundamental quantity in information theory to measure the difference between two probability distributions. If a probability distribution $q$ is used to approximate $p$ then the KL divergence measures the loss in information using the approximation.

### *6.1. Latent Generative Models for Calo Simulation with VQ-VAE*

By Qibin Liu, Chase Shimmin, Xiulong Liu, Eli Shlizerman, Shu Li, and Shih-Chieh Hsu, with figures and tables referring to these approaches as `Calo-VQ` [58] and `Calo-VQ(norm)` [58] and code being available at [59].

---

*Introduction* Calorimeters with high granularity often feature a large number of voxels, reaching up to tens of thousands. Directly sampling of such high-dimensional and highly sparse data is usually challenging and inefficient. To address this, a two-stage method, as illustrated in figure 21, is proposed. This method is based on a vector-quantized variational auto-encoder (VQVAE) [144] and a transformer-based token generative model [136, 145]. In the following, we describe the implementation of this model, including processing of calorimeter data, representation, architecture and training procedure.

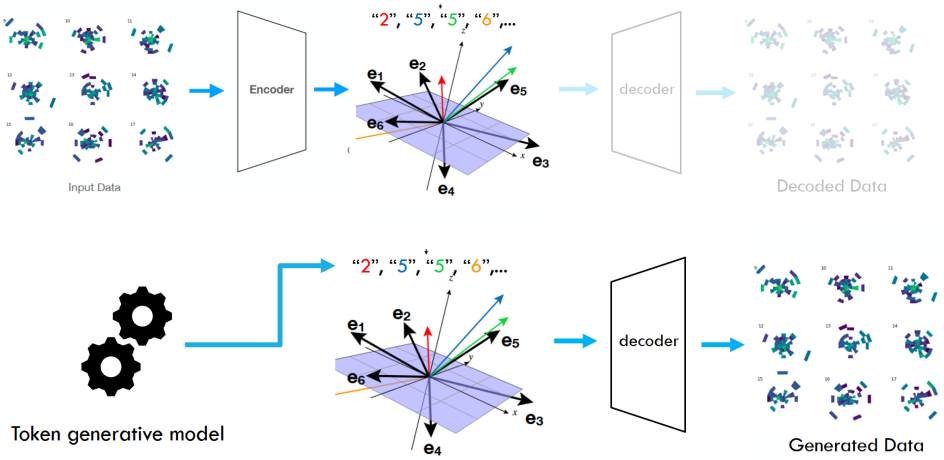



Figure 21: Demonstration of the `Calo-VQ` architecture, taken from [58]. The upper and lower parts show the two stages of the model, respectively.

*Preprocessing* To handle the large dynamic range of calorimeter energy, the data is normalized per detector layer (for ds $1-\gamma$ and ds $1-\pi^+$) or the entire detector (for ds 2 and ds 3, except the model implicitly marked with "norm") for each sample. It is then transformed using a scaled and shifted logarithm. The pre-processing is described in (18).

$$x_i = \frac{1}{c}\log\left(a + b\frac{\mathcal{I}_i}{E_{\text{sum}}}\right) \tag{18}$$

Here, $i$ is the index of the voxel, $\mathcal{I}_i$ is the original value of each voxel, and $E_{\text{sum}}$ is the sum of energies within the same layer or the entire calorimeter, depending on the dataset. The hyperparameters $a$, $b$, and $c$ are tuned according to the input range.

*Architecture* The first stage of the model aims to reduce the dimensionality of the input. The encoder transforms the input into the representation in the latent space, followed by the decoder, which reconstructs the input. To achieve high compression ratio and effective usage of latent space, a vector quantization technique [144] is implemented. This technique labels each latent vector with one index of a fixed set of representative "code vectors". The code vectors are updated during training, minimizing the quantization loss and commitment loss, as shown in the following,

$$L_{VQ} = ||\text{sg}\left[q(z|x)\right] - e_k||^2 + ||q(z|x) - \text{sg}\left[e_k\right]||^2 \tag{19}$$

Here $\text{sg}\left[X\right]$ denotes the stop-gradient operator which will not take the gradient of $X$ into calculation. The first term denotes the quantization loss which moves the codebook $(e_k)$ to better represent the latent space $(z)$. The second term is called commitment loss which limits the arbitrarily growth of embedding space and makes the encoded vector commit to one of the codes.

The L2 distance between input and decoded output, following the vanilla VAE architecture, is used as one loss term. Additionally, the discriminator loss [146] and physics-aware losses are added to improve the quality of reconstruction, particularly for the detailed feature.

The encoder and decoder consist of convolutional and dense layers. For photon and pion datasets 1, 1-dimensional convolution and fully connected layers are combined to better process the irregular geometry. For datasets 2 and 3, since the transitional symmetry only exists in the $z$ (depth) and $\alpha$ (angular) direction, 2-dimensional convolutions are used, treating the 3-dimensional data as a multi-channel 2-dimensional image defined on the $(Z, \alpha)$ coordinates. The cylindrical convolution operation is shown in figure 22. It maintains the equivariant property of the calorimeter data mapped in cylindrical coordinates. The radial direction is therefore treated as the channel of this image-like representation.

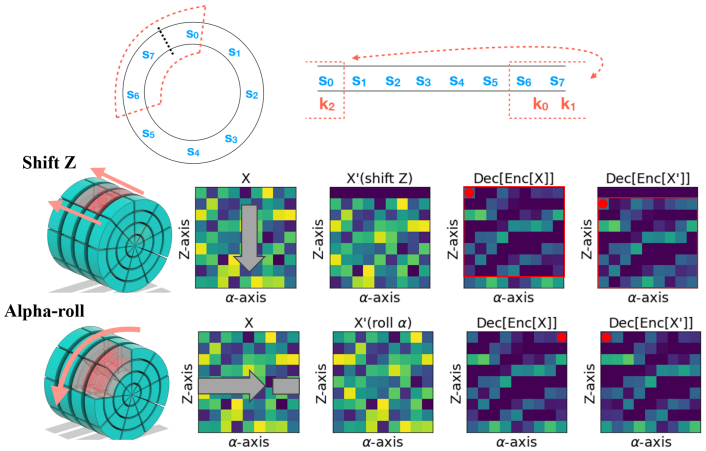



Figure 22: Cylindrical convolution operator, taken from [58]. The bottom plots show the equivariant property is well kept.

Arbitrary up-/down-sampling on the angular direction with circular (periodical) boundaries is achieved using the FFT-resampling method, as illustrated in figure 23. The data along the angular direction is first transformed with discrete FFT into frequency space, then truncated to the desired dimension and transformed back with inverse operation.

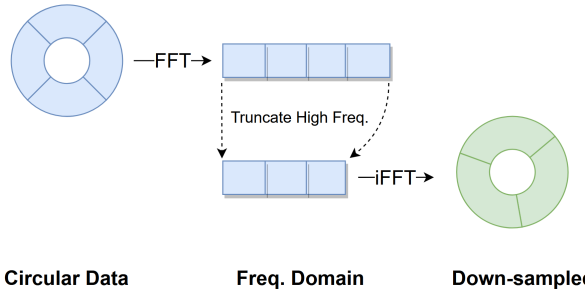



Figure 23: Illustration of the FFT down-sampling, taken from [58].

The softmax activation is used in the last layer to ensure the correct normalization as mentioned in (18). By definition, the output of softmax satisfies the required normalization in log scale with proper shift and scaling.

The second stage model focuses on learning and sampling of the probabilistic distribution, in a highly reduced and regularized latent space, characterized by a fixed-length sequence of discrete codes (tokens). The transformer-based model minGPT [145] is adapted to sample the latent codes, conditioned on the incident energy. As discussed in the previous section, the normalization factor(s) $E_{\text{sum}}$, which control the energy response of the entire calorimeter or each calorimeter layer, is digitized to discrete codes and sampled together with the latent codes of the first stage. Two codes are used to digitize one floating number with 20bit accuracy assuming 1024 choices for each code are utilized.

The final sequence to learn for the second stage is

$$\underbrace{A_1 A_2 ...}_{E_{\text{sum}}\ codes}\ \overbrace{B_1 B_2 B_3 B_4 ...}^{latent\ codes} \tag{20}$$

The parameters of the second stage model are tuned towards less transformer heads, layers and embedding while retaining the same quality of generated data (such as smaller error on $E_{\text{sum}}$).

*Training* The first stage model is trained adversarially, updating the encoder/decoder and the discriminator alternately. Learning rate is constant during the training and the model with best reconstruction loss on validation dataset is selected after training for a fixed number of epochs.

Then the quantized latent space, as discrete numbers (tokens), is used to train the second stage model. The training objective is to minimize the cross-entropy between

the predicted token based on previous ones and the truth token. A constant learning rate is used and the model with best validation loss is chosen.

The main hyper-parameters are summarized in table 6.

| | ds $1-\gamma$ | ds $1-\pi^+$ | ds 2 | ds 3 | ds 3 (norm) |
|---|---|---|---|---|---|
| Pre-processing (18): | | | | | |
| a | 1 | 1 | 1 | 1 | 1 |
| b | 8000 | 8000 | 3000 | 40 000 | 40 000 |
| c | 10 | 10 | 7 | 10 | 10 |
| Stages-1: | | | | | |
| Hidden layer | 5 | 5 | 7 | 10 | 10 |
| VQ dim | 256 | 256 | 256 | 192 | 256 |
| Condition dim | 3 | 3 | 1 | 1 | 1 |
| Codebook size | 1024 | 1024 | 1024 | 1024 | 1024 |
| R code | 10 | 14 | 2 | 2 | 90 |
| Shower code | 32 | 32 | 68 | 182 | 624 |
| #pars / M | 3.8 | 4.1 | 3.1 | 2.1 | 2.2 |
| #pars (gen) / M | 1.9 | 2.0 | 1.0 | 0.8 | 0.9 |
| Stages-2: | | | | | |
| Layer | 2 | 2 | 2 | 1 | 1 |
| Head | 2 | 2 | 2 | 1 | 1 |
| Embed | 64 | 64 | 64 | 16 | 128 |
| #pars / K | 231 | 231 | 235 | 38 | 551 |

Table 6: Setup of hyper-parameter and number of trainable parameters of preprocessing, first stage model and second stage model of `Calo-VQ`, taken from [58]. The numbers for "hidden_layers" are halved since symmetric encoder and decoder. Only the decoder and quantization module in stage 1 are used in generation mode and the number of parameters are denoted with "(gen)" in the table.

### *6.2. CaloMan: Fast Generation of Calorimeter Showers with Density Estimation on Learned Manifolds*

By Jesse C. Cresswell, Brendan Leigh Ross, Gabriel Loaiza-Ganem, Humberto Reyes-Gonzalez, Marco Letizia, and Anthony L. Caterini, with figures and tables referring to this approach as `CaloMan` [60] and code being available at [61].

---

*Introduction* As surveyed in the present work, many types of generative models have been used to model calorimeter showers, and particular emphasis has been given to

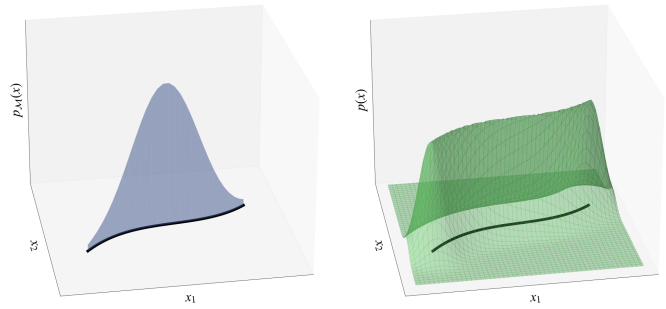


Figure 24: A low-dimensional density on a manifold (left), and a full-dimensional density model undergoing manifold overfitting (right). Although the full-dimensional model concentrates around the manifold, it distributes the density incorrectly along the manifold. Figure taken from [60].

normalizing flows [109, 117]. Despite their expressivity, NFs suffer from the fact that they model a density that has the same dimensionality as the input data. For high-dimensional data, this would mean dealing with very large NF models that compromise training and prediction speed. For calorimeter showers, size and speed can quickly become major problems as the dimensionality of raw shower representations surpasses $10^4$. However, we expect that shower generation is governed by simple underlying physical processes, and thus can be represented in a much lower dimensional space. In the context of machine learning, this is an example of the *manifold hypothesis*, which states that high-dimensional natural data actually lies on a low-dimensional embedded sub-manifold in the ambient space [147, 148, 149].

Moreover, maximum-likelihood methods, including NFs, rely on the assumption that the underlying distribution possesses a full-dimensional probability density $p(x)$ in the ambient space. This may not always be the case: if the data is confined to a low-dimensional manifold, the data manifold is a subset of measure zero, over which no continuous density can be integrated to obtain non-zero probabilities. In this situation, training a likelihood-based model typically leads to densities that spike to infinity around the manifold, but not in accordance with the data distribution. This phenomenon, illustrated in figure 24, is known as *manifold overfitting* [150, 151].

To avoid manifold overfitting, while also delivering fast, light-weight models, we propose `CaloMan`, which follows the two-step procedure outlined by [150] to build our calorimeter shower simulators. The first step of the approach is to learn a lower dimensional manifold using a *generalized autoencoder*. This can be any ML model capable of learning a latent space, and transforming it back to the ambient space. Examples include autoencoders [152], variational autoencoders [141], Wasserstein autoencoders [153], bidirectional GANs [154, 155], and adversarial variational Bayes [156]. The second step is to perform *density estimation* on the learned manifold. Any explicit likelihood estimator can be used. This includes NFs, energy based models [157],

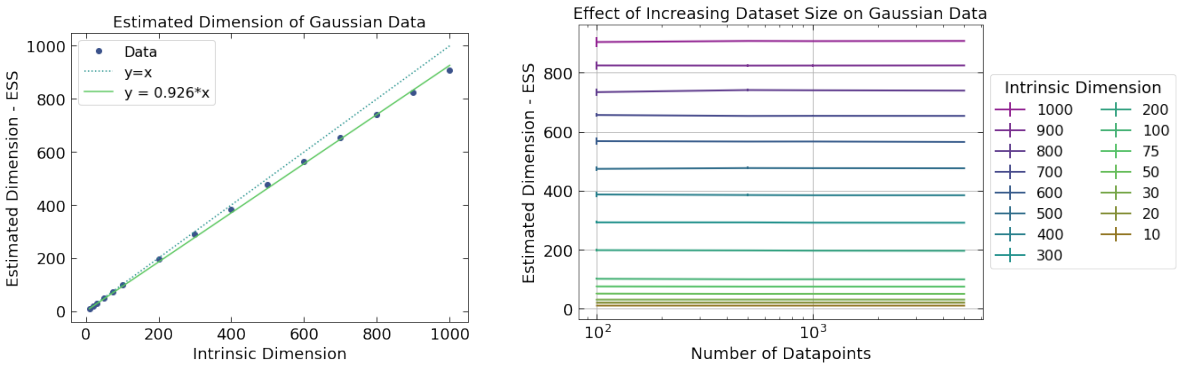



Figure 25: Benchmarking the ESS estimator. **Left:** As the dimension of random Gaussian data is increased, the ESS estimate of intrinsic dimension linearly increases ($n = 5000$). Standard deviations are smaller than the size of the dots. **Right:** ESS provides consistent estimates regardless of the number of datapoints used. Notably, accurate estimates can be obtained even when the number of datapoints is less than the dimensionality. The error bars show standard deviations over 10 seeds.

auto-regressive models [122], score-based models [158], and diffusion models [124].

*Intrinsic dimensionality of CaloChallenge datasets* Most methods for learning low-dimensional manifold structure require the dimensionality $d$ to be provided as an input. Hence, we applied methods for intrinsic dimension estimation [159], which also shed light on the fundamental nature of the calorimeter shower data.

Here we use the Expected Simplex Skewness (ESS) estimator [160] which is based on angular information between $k$-nearest neighbor points in a dataset. Most literature on estimating the intrinsic dimension of datasets focuses on relatively low dimensions (*i.e.* $d \leq 20$) and few datapoints $n$. Hence, we first benchmark ESS on synthetic datasets of known dimensionality $d$ more comparable to CaloChallenge data, using the implementation from [161] with default hyperparameters. We randomly generate $n$ datapoints from $d$-dimensional Gaussian distributions, with $n \in \{100, 500, 1000, 5000\}$, and $d$ spanning values from 10 to 1000, and apply the ESS estimator to each dataset. We repeat the test for 10 random seeds on each setting. Figure 25 shows that ESS has a consistent linear behavior as $d$ is increased, and that it is insensitive to $n$, even when $n < d$. Noticing a slight negative bias, we fit a scale factor to the data for $n = 5000$, and use its inverse to calibrate the ESS estimates on shower data. We also repeat the experiment for data drawn from a uniform distribution on a hypercube with extremely similar results. The calibration scaling factors are 1.0795 for Gaussian data, and 1.0793 for Uniform data‖.

With evidence that ESS can scale to high-dimensional datasets, we applied the

‖ We acknowledge that the synthetic data we used does not itself have low-dimensional structure, however similar benchmarking has been done for such data with consistent results [162]. More recent work shows some limitations of ESS [163].

Table 7: Estimated Dimensionality $d$ with Gaussian (G) and Uniform (U) Calibration

| Dataset | Features | ESS | ESS (G) | ESS (U) |
|---|---|---|---|---|
| Photons | 368 | 28.6 | 30.9 | 30.9 |
| Pions | 533 | 17.2 | 18.5 | 18.5 |
| Electrons 2 | 6480 | 209.0 | 225.6 | 225.5 |
| Electrons 3 | 40 500 | 749.2 | 808.7 | 808.6 |

ESS estimator to the four CaloChallenge training datasets without any preprocessing of the data, and correct the results with our calibration factors. The intrinsic dimension estimates are given in table 7, and we based our latent space dimensions on these values in the following. The results provide evidence for low-dimensional structure in calorimeter data, and emphasize the potential efficiency gains over approaches that model a full-dimensional latent space.

*Architecture and Preprocessing* Similarly to [109], for `CaloMan` we separated the training procedure into two stages. In the first stage, using a NF, we learn the distributions of energies per layer, $E_\ell$, conditioned on the incident energies $E_{\text{inc}}$. In the second stage, we model the voxel energies $\mathcal{I}$, conditioned on $E_{\text{inc}}$ and $E_\ell$ by first learning the manifold with a generalized autoencoder, then estimating a density on the manifold as described above.

*Preprocessing* Following [34], to ensure energy conservation, the $E_\ell$ are first transformed as

$$u_0 = \frac{\sum_{\ell=0}^{n} E_\ell}{E_{\text{inc}}}, \quad u_1 = \frac{E_\ell^0}{\sum_{\ell=0}^{n} E_\ell}, u_2 = \frac{E_\ell^1}{E_{\text{rem}}^2}, \ \dots, u_n = \frac{E_\ell^{n-1}}{E_{\text{rem}}^n}, \tag{21}$$

where $n$ is the total number of layers, and $E_{\text{rem}}^i = \sum_{\ell=0}^{n} E_\ell - \sum_{\ell=0}^{i} E_\ell$. The resulting variables are then transformed into logit space as

$$u_i' = \log \frac{x_i}{1 - x_i}; \tag{22}$$

where $x_i = \alpha + (1 - 2\alpha)u_i$ and $\alpha = 10^{-6}$. The incident energy $E_{\text{inc}}$ is preprocessed as:

$$E_{\text{inc}} \leftarrow \log_{10}\left(\frac{E_{\text{inc}}}{33.3\text{GeV}}\right). \tag{23}$$

For the second stage, $E_{\text{inc}}$ and $E_\ell$ are transformed as

$$E_{\text{inc}} \leftarrow \log_{10}\left(E_{\text{inc}} + 1\text{keV}\right), \quad E_\ell \leftarrow \log_{10}\left(\frac{E_\ell + 1\text{keV}}{100\text{GeV}}\right) - 1. \tag{24}$$

Finally, the voxel energy $\mathcal{I}$ is normalized so that the energies per layer sums up to one. The conditioning vector for the second stage is given by the concatenation of the incident energy $E_{\text{inc}}$ and the energies per layer $E_\ell$.

It is worth pointing out that, according to our tests, the most important modeling aspects are the separation of the pipeline in two stages and the inclusion of the energy

per layer in the conditioning vector for the second stage. For simplicity, we adopted a preprocessing strategy that closely follows [34] but we found that other reasonable preprocessing choices at each step do not significantly impact the performance of the model.

*Architecture* For all the experiments, we used a NF for the first stage and a two-step model for the second stage with a VAE as the generalized autoencoder and a NF of the same type as the density estimator.

For stage one and the photon dataset we used a 8 layer $\times$ 384 units coupling rational-quadratic neural spline flow [110] with a 6-block residual network [127] in each layer. For the pion dataset, we used the same model with 8 layer $\times$ 512 units and 3-block residual networks. The output of each residual block was combined with the conditioning input, namely the incident energy, using a gated linear unit. The NF's prior distribution was a unit variance diagonal Gaussian.

For stage two, the VAE's encoder and decoder were both DNNs with three hidden layers of 512 units each, and ReLU activations. The encoder output the means and variances for a diagonal Gaussian over the latent dimensions. The decoder output was also treated as a diagonal Gaussian with means for each data dimension but only a single variance shared across all dimensions. The prior distribution was a unit variance diagonal Gaussian over the latent space. The latent dimension was 35 for photons and 20 for pions, values which are slightly higher than the estimates reported in table 7. The NF was of the same type as in stage one with 4 layer $\times$ 128 units and 3-block residual networks. The output of each residual block was combined with the conditioning input, now the incident energy concatenated with the energies per layer, using a gated linear unit. The NF's prior distribution was a unit variance diagonal Gaussian.

*Training* All models were trained with batch sizes of 512 and the Adam optimizer [90] with a learning rate of $10^{-4}$ in stage one and $10^{-3}$ in stage two. We also applied gradient clipping with a max gradient norm of 10. The models were trained for a maximum of 200 epochs each with early stopping after 20 epochs of no validation improvement on a 20% hold-out set.

For stage two, the VAE and NF were trained sequentially. Once the VAE was trained, its parameters were frozen and the training dataset was encoded deterministically with the VAE encoder means. The encoded data were then passed as inputs to the NF. The validation and early stopping metric was the average $\chi^2$ separation power over all high level features for stage one, the reconstruction error for the VAE and the negative log-likelihood for the NF in stage two. Both the VAE and the NF were conditioned on the incident energies and the energies per layer during training. The models with the best validation metrics were used for evaluation.

### 6.3. DNN CaloSim

By Dalila Salamani, with figures and tables referring to this approach as `DNNCaloSim` [62, 63] and code being available at [64].

---

*Introduction* The VAE model explored for ds 1 – $\pi^+$ is inspired by the VAE model developed in the context of shower simulation in the ATLAS experiment [62, 63].

*Architecture* It comprises four dense layers for the encoder with 1500, 1000, 500, and 100 nodes for each layer respectively. The decoder has also four dense layers with a reversed order of the number of nodes. The latent dimension is 50. A batch normalization layer is used after each dense layer. The encoder and decoder networks are jointly trained to maximize the variational lower bound on the marginal log-likelihood for the data, approximated with the reconstruction loss (binary cross-entropy) and the KL divergence.

*Preprocessing* The model is not trained on the absolute voxel energies but rather on the voxel energy ratios, where per shower, the energies of the voxels within each layer are normalized to the total energy deposited in that layer. This reparametrization of the input allows the model to better preserve the correlations of the energies across layers. In order to re-scale back the energies after generation, the VAE model learns, in addition, the energy per layer and the total energy of the shower deposited in the calorimeter. These quantities are encoded as additional $N$+1 nodes in the input and output of the VAE model, where $N$ represents the number of calorimeter layers (in dataset 1, for pions $N$=7). As the voxel energies are encoded as ratios the additional $N$+1 nodes are also encoded as ratios, where for each layer its energy is divided by the total energy and the total energy is divided by the incident energy. In total, the number of nodes in the input and output layers is 541 = 533+7+1. The model is conditioned on the incident energy of the incoming particle.

From prior knowledge on the deposited energy, ratios of voxel values and all total energies per layer should sum up to one. This can be ensured in the output layer of the VAE by applying a softmax activation function. By using the softmax function, the values are converted into probabilities that sum to one and automatically values fall in the range of [0,1]. The softmax is applied for the voxels of each calorimeter layer and for the ratios of the energy per layer. Figure 26 shows a schematic representation of the VAE model.

*Training* One effective trick for improving the training of the VAE model is using an iterative approach, where the model is trained on varying batch sizes and learning rates. By cycling through different combinations, the model is exposed to a variety of training conditions, which helps to avoid local minima and leads easier to potentially

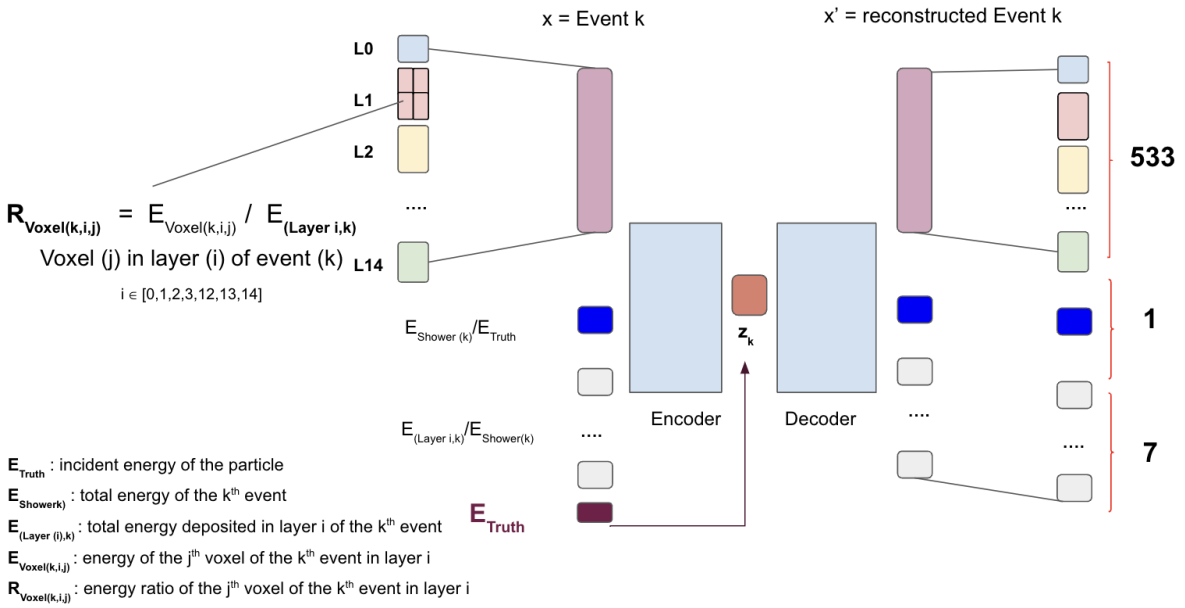



Figure 26: For the `DNNCaloSim` model, the VAE is trained to reconstruct the voxel energy ratios. The 533 inputs/outputs (the large pink boxes) represent the voxel energies for layers 0, 1, 2, 3, 12, 13, and 14. The additional input/output (dark blue) represents the normalized energy of the shower to the energy of the particle ($E_{\text{inc}}$, called $E_{\text{Truth}}$ here) and the 7 following inputs/outputs (in gray) are the ratios of the energy of the layer to the energy of the shower. The model is conditioned on the energy of the incoming particle, this is added as one additional input to the latent space (brown box).

generalization. In total 8 iterations are used with different values of batch sizes and learning rates.

### *6.4. Geant4 Transformer*

By Piyush Raikwar, Renato Paulo Da Costa Cardoso, Nadezda Chernyavskaya, Kristina Jaruskova, Witold Pokorski, Dalila Salamani, Mudhakar Srivatsa, Kalliopi Tsolaki, Sofia Vallecorsa, and Anna Zaborowska, with figures and tables referring to this approach as `Geant4-Transformer` [65] and code being available at [66].

---

*Introduction* Given the recent success of transformer-based models in various tasks, *i.e.*, from image classification (ViT) [137] to text generation (GPT-3) [164], we explore the applicability of transformers for the task of generating non-trivially structured particle showers, specifically for dataset 3. The presence of an attention mechanism and the lack of a strong inductive bias in the architecture should help in better modeling of energy distributions in a highly granular mesh, given enough data.

*Preprocessing* For the preprocessing, we divide the voxel energies (in MeV) by a scalar of 4300 to bring all entries between 0 and 1. Also, to use the incident energies of the particles as the condition, we divide the incident energies (in GeV) by 1024 so that their distribution is between 0 and 1.

Since transformers are permutation-invariant sequence-to-sequence models, a shower needs to be in the form of a sequence. A naive way would be to treat each voxel as an element of the sequence, but that would be computationally expensive. Therefore, inspired by ViT, we create non-overlapping 3-dimensional patches of the shower to feed it to our model. To be specific, the dimensions of the patch are $\Delta r \times \Delta\phi \times \Delta z = 18 \times 5 \times 3$, *i.e.*, 150 patches per shower. In addition to having a sequence, the transformer also needs to know the position of an element in the sequence, which we feed by using sinusoidal positional embeddings.

*Architecture* Our model is a two-stage model inspired by Dall-E [165], where the first stage is a Vector Quantized Variational Autoencoder (VQ-VAE) [144], followed by an Autoregressive model (AR) (see figure 27). The VQ-VAE models the low-level features by tokenizing the shower, and the AR models the high-level features by learning the distribution of these tokens. These models are trained separately.

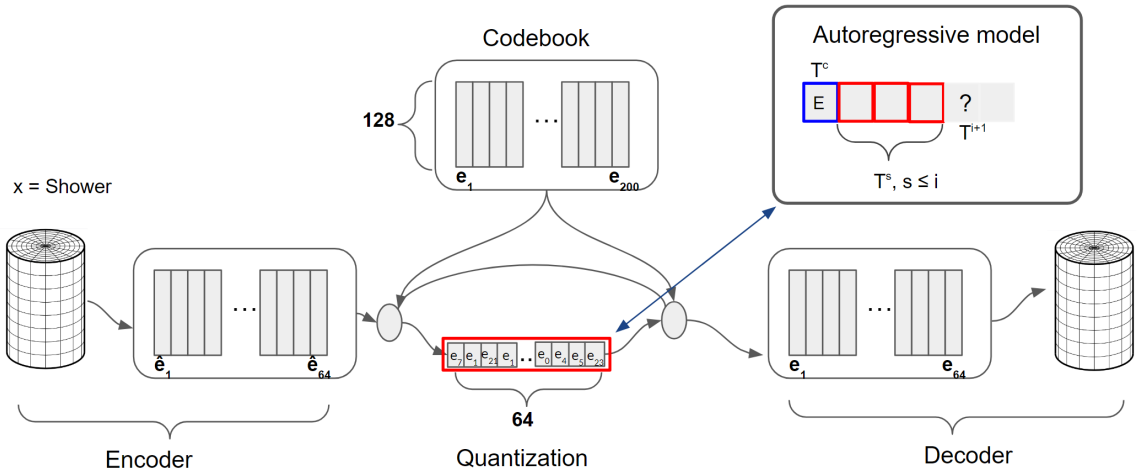



Figure 27: The figure shows the components of the VQ-VAE and the AR (top-right). The shower is tokenized using the VQ-VAE's encoder and by referring to the codebook. The tokenized representation of the shower is then fed to VQ-VAE's decoder to get back the shower. The AR models the tokenized shower's space (red tokens) separately in an autoregressive manner conditioned on $E_{\text{inc}}$ (blue token). During inference, the VQ-VAE encoder is not used. Figure taken from [65].

The VQ-VAE, as already explained for `Calo-VQ` [58] in section 6.1, is similar to a traditional VAE, except that its latent space is discrete and can be represented by a set of codebook vectors (or tokens, $T^i$) chosen from a codebook. The codebook is a learnable entity that can be trained in parallel with the encoder and decoder to represent

the latent space optimally. The output from the encoder is quantized using a nearest-neighbor search on the codebook to obtain a discrete latent space. So, the VQ-VAE is trained to reconstruct the showers where the latent space is a sequence of tokens. The distribution of these tokens is unknown, hence we cannot generate new showers. Therefore, the task of AR is to model the distribution of the tokens generated by the VQ-VAE, given the initial conditions, *i.e.*, incident energy of the particle in our case. AR learns the latent space of VQ-VAE by autoregressively predicting these tokens, *i.e.*, learning the probability of the next token given all the previous tokens. The process of sampling a new token from a multinomial distribution makes the AR a generative model. Note that VQ-VAE is not conditioned on the incident energy of the particle.

For the generation of new showers, we start by creating a condition token ($T^c$). Given the condition token, we sample the next token and this continues till we have all the required tokens. All tokens except the condition token are then passed to the VQ-VAE decoder to get the final shower.

Both VQ-VAE and AR have a ViT-like uniform architecture. Each of them is described in detail as follows.

In the case of the VQ-VAE, the patches form the sequence. These patches are linearly projected to match the projection dimension of the VQ-VAE, which is 256. To this, 3-dimensional positional embeddings of 256 dimensions are added to inject the position information. These patches are then fed to the VQ-VAE. The encoder and decoder of the VQ-VAE consist of 4 encoder-only [136] transformer blocks each. Each block consists of 16 attention heads and 512 nodes in the DNN sub-block. The patches after the last transformer block in the encoder are concatenated and projected to the desired dimensions of the latent space. That is, the number of patches is independent of the number of tokens in the latent space. Thus, a token can represent information from any of the patches. In our case, the latent space consists of 64 tokens of 128 dimensions. The codebook however consists of 200 tokens, out of which a combination of 64 tokens is used to represent a shower. The opposite is done to project the latent space back to the patches, which are then fed to the decoder. The activation function at the end of the decoder is a sigmoid, and binary cross-entropy is used as the loss function.

For the AR, the tokens form the sequence. The tokens are projected linearly to the projection dimension of the AR, which is 128. Here, we use 1-dimensional positional embeddings to denote a token's position. These tokens are then passed to the AR. AR consists of 4 decoder-only [164] transformer blocks having 8 attention heads and 256 nodes in the DNN. The condition token is created by linearly projecting our conditions to match the projection dimension of the AR. The activation function at the end of the last block is a softmax, and the model is trained with categorical cross-entropy loss, where the target tokens are obtained from the VQ-VAE encoder.

### 6.5. CaloVAE+INN

By Florian Ernst, Luigi Favaro, Claudius Krause, Tilman Plehn, and David Shih, with figures and tables referring to this approach as `CaloVAE+INN` [38] and code being available at [39].

---

*Introduction* In this section we describe our effort to improve the scaling of the `CaloINN` model. The idea is similar to the previously described VAE-Flow models. We train a VAE on the individual CaloChallenge datasets in a first step and a down-scaled `CaloINN` in the corresponding latent space. The advantages of this approach were already discussed in section 6.2 in the context of `CaloMan`. Namely, the enhanced topological properties and the smaller input dimensionality for the normalizing flow.
However, we were approaching with a different perspective. Our main goal is not to find the true manifold dimensionality, but we consider the VAE as a pure compression tool. Using this point of view it is of utmost importance to get good reconstructions first, before applying the INN in the latent space.

*Loss function* For the INN part we are employing the same loss function that was introduced in `CaloINN` (section 4.3).
For the VAE we are using the $\beta$-VAE [166, 167] ELBO loss with a Gaussian encoder and a Bernoulli decoder. For the latent space we chose a standard normal distribution resulting in the following loss function:

$$\begin{aligned}\mathcal{L}_{\text{BVAE}} = &\sum_{x\in TS}\sum_{z\sim E(z|x)} \left[x\log\left(\lambda(z)\right) + (1-x)\log\left(1-\lambda(z)\right)\right] \\ &+ \beta\cdot\sum_{x\in TS}\left[1+\log\left(\sigma_E(x)^2\right) - \mu_E(x)^2 - \sigma_E(x)^2\right].\end{aligned} \tag{25}$$

Here, $\lambda$ is the parameter of the decoder Bernoulli distribution; $\mu_E$ and $\sigma_E$ are the parameters of the Gaussian encoder, as predicted by a neural network; $E(z|x)$ is the encoder; and $TS$ indicates the training dataset. We decided to explicitly not add a physics loss term since we did not want to bias the VAE to improve its reconstruction of the relevant physics variables, as we use these variables as performance metrics.

*Preprocessing* Our preprocessing consists of four steps and is similar to the `CaloINN` preprocessing. We keep the initial calorimeter layer normalization and extract our "extra dimensions" just like before. However, we scale them with a factor of 0.9 to prevent float precision problems. We did not add noise as we found it to be not helpful during the reconstruction process. We replaced the logarithm with an $\alpha$-regularized logit as in (22) and added a final standardization layer. For the datasets 2 and 3 we used a

learnable affine transformation, for dataset 1, we simply normalized to zero mean and unit variance. The entire preprocessing is illustrated in figure 28 (left).

The biggest difference to the `CaloINN` preprocessing is the fact that the extra dimensions are not learned explicitly but used as additional conditions for the VAE. Afterwards, they are learned directly by the INN.

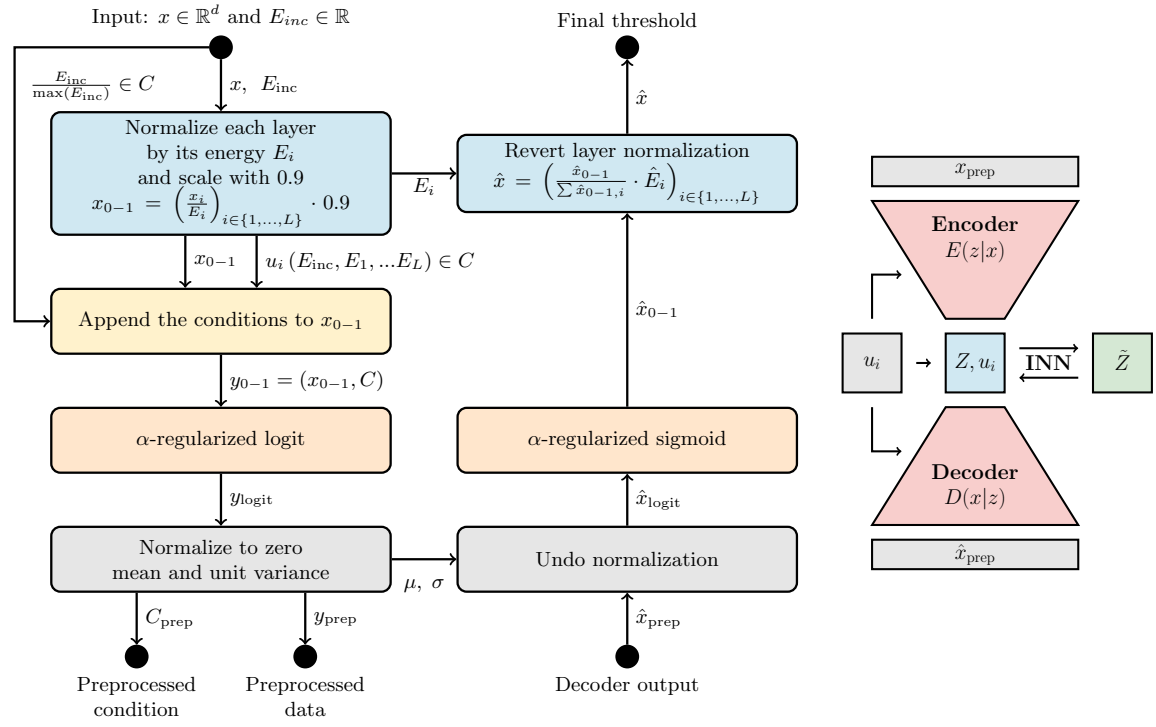



Figure 28: **Left:** Visualization of the preprocessing steps before the VAE-compression. **Right:** Schematic visualization of the VAE-INN combination of `CaloVAE+INN`, taken from [38].

*Architecture* For the practical implementation we chose a fully connected encoder and decoder for dataset 1 and a kernel VAE (KVAE) for dataset 2 and 3. The KVAE is an architecture that is a compromise between a fully connected network and and a convolutional network. It tries to find a optimum between the reconstruction quality of the DNN and the scaling properties of the convolutional architecture. The idea is to use a two-step encoding, where neighboring detector layers are jointly encoded into a sub-latent space. Afterwards these sub-latent spaces are concatenated and encoded for a second time. This architecture emphasizes the stronger correlations between neighboring layers. We call the number of jointly encoded detector layers the "kernel size" and the distance between the two first layers of neighboring encoding "blocks" the "kernel stride"

The decoder architecture is a copy of the encoder with inverted order of the size of the

hidden layers.

The INN (a NF) is trained in the latent space of the second encoding step before the sampling happens, as seen in figure 28 (right). This means, the INN is trained to predict the actual $\sigma_E$ and $\mu_E$ values, effectively doubling the size of the input space for the INN. We found that this procedure improved the sampling quality of the resulting combined architecture significantly. The reason is probably that our $\beta$-parameter in the ELBO-loss is so small that VAE learned to store non-trivial information in the $\sigma_E$-parameters. Therefore, the input space of the INN consists of the encoder means $\mu_E$, the encoder widths $\sigma_E$ and the extra dimensions $u_i$.

Our final hyperparameter configurations for the three datasets can be seen in table 8. We used $\beta = 10^{-9}$ as the KL term is just a regularization in this setup. The latent space is not required to be actually Gaussian, that is the task of the latent INN. However, the latent space must be "well enough" behaved for the INN to be trained in it.

| Parameter | VAE | |
|---|---|---|
| lr scheduler | Constant LR | Inner VAE |
| lr | $10^{-4}$ | |
| hidden dimension | 5000, 1000, 500 (ds 1) | |
| | 1500, 1000, 500 (ds 2) | |
| | 2000, 1000, 500 (ds 3) | |
| latent dimension | 50 (ds 1,2) / 300 (ds 3) | |
| # of epochs | 1000 | |
| batch size | 256 | |
| $\beta$ | $10^{-9}$ | |
| threshold $t$ [keV] | 2 (ds 1) / 15.15 (ds 2,3) | |
| hidden dimension | 1500, 800, 300 | Kernel |
| kernel size | 7 | |
| kernel stride | 3 (ds 2), 5 (ds 3) | |

| Parameter | INN (after VAE) |
|---|---|
| coupling blocks | RQS |
| # layers | 3 |
| hidden dimension | 32 |
| # of bins | 10 |
| # of blocks | 18 |
| # of epochs | 200 |
| batch size | 256 |
| lr scheduler | "one cycle" |
| max. lr | $10^{-4}$ |
| $\beta_{1,2}$ (Adam) | $(0.9, 0.999)$ |
| $\alpha$ | $10^{-6}$ |

Table 8: Network and training parameters for the `CaloVAE+INN`.

### *6.6. CaloLatent: Score-based Generative Modeling in the Latent Space for Calorimeter Shower Generation*

By Thandikire Madula and Vinicius M. Mikuni, with figures and tables referring to this approach as `CaloLatent` [67] and code being available at [68].

---

*Introduction* In our work, we introduce `CaloLatent` [67], a latent diffusion inspired surrogate model. The main idea in latent diffusion is to map the data into a compressed latent representation using a VAE. Once the latent representation has been obtained, a diffusion model can be deployed to learn the distribution of the latent space.

The motivations behind this approach are similar to those outlined by `CaloMan` and `CaloVAE+INN`.

In our approach we use the VAE backbone primarily for compression, therefore we prioritize the VAE's reconstruction ability over its generation ability. To this end, we utilize the $\beta$-VAE [166, 167] formulation where the KL divergence is weighted by a factor of $\beta$. We chose $\beta = 10^{-6}$. The diffusion model used to learn the latent distribution is a score-based diffusion model, the intricacies of which have been outlined by `CaloScore` in section 5.3.

*Preprocessing* To evaluate the performance of our model we focused on dataset 2 of the challenge. We processed the data for training using the following steps. First, we normalized the voxel energy using (26).

$$x_i' = \frac{\mathcal{I}_i}{E_i} \tag{26}$$

where $\mathcal{I}_i$ are the voxels in layer $i$ of the detector and $E_i$ is the total energy of that layer. Secondly, we apply minmax scaling to the data which is defined by (27).

$$x' = \frac{x - x_{\min}}{x_{\min} - x_{\max}} \tag{27}$$

Where $x_{\min}$ and $x_{\max}$ are the minimum and maximum voxel values respectively. The resulting data is then transformed using the logit function outlined in (22). Finally, we take the values in the logit space and apply a standardization as given by (28).

$$x' = \frac{x - \mu}{\sigma} \tag{28}$$

Where $\mu$ and $\sigma$ are the means and standard deviations, repectively.

*Architecture* `CaloLatent` is comprised of three networks. First, a scored-based diffusion model that is used to learn the distribution of the layer energy. The layer score model is a simple ResNet [127] consisting of 3 layers each with 512 nodes. Second, we have the VAE backbone. The encoder and decoder of the VAE are 3-dimensional convolutional neural networks also inspired by the ResNet architecture. Figure 29 shows a schematic diagram of the VAE encoder used for `CaloLatent`. The decoder of the VAE is a mirror image of the encoder; however, it employs up-sampling blocks in place of the encoder down-sampling blocks. The VAE reduces the data dimensionality from 6408 to 1008. The third and final network in `CaloLatent` is the score-based diffusion model used to learn the latent space, the architecture of this model is identical to that of the layer model.

*Training* All three networks are trained independently. The layer score model and the VAE are trained for 500 epochs using a cosine decaying learning rate with an initial learning rate of $4 \cdot 10^{-4}$. The latent score model is trained for 250 epochs using the same learning schedule as the other models.

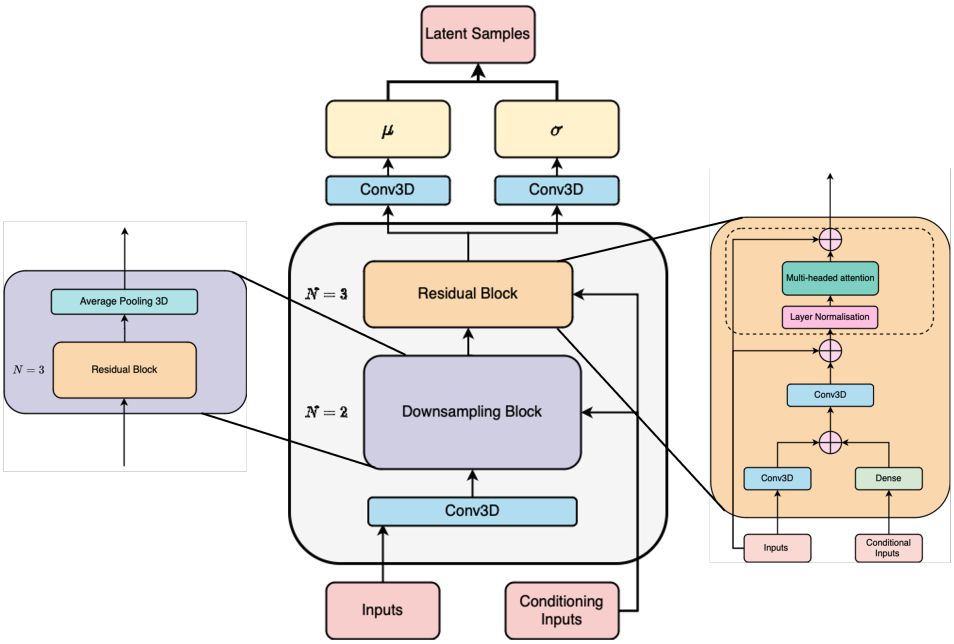



Figure 29: Schematic diagram of the `CaloLatent` VAE encoder.

## 7. Conditional Flow Matching-based Submissions

The *conditional flow matching* (CFM) algorithm was proposed in various forms simultaneously by several groups [168, 169, 170], and developed in [171] which we follow here. Like continuous normalizing flow models [172], flow matching learns to interpolate probability densities $p_t$ between the data $p_1$, and a simple prior $p_0 = \mathcal{N}(x_0 \mid 0, I)$. The interpolation is determined by a vector field at each time $\mu_t(x)$, which transports datapoints $x$ via the ODE

$$dx = \mu_t(x)dt. \tag{29}$$

When $p_t$ and $\mu_t(x)$ jointly satisfy the continuity equation for conservation of probability

$$\frac{d}{dt}p_t + \nabla_x \cdot (p_t \mu_t) = 0, \tag{30}$$

$p_t$ will be a properly normalized density at each $t$. Hypothetically, one could train a model $\nu_\theta(t, x)$ of the vector field $\mu_t(x)$ by direct regression,

$$L_{\text{FM}} = \mathbb{E}_{t\sim\mathcal{U}(0,1),\, x\sim p_t} \|\nu_\theta(t, x) - \mu_t(x)\|_2^2, \tag{31}$$

however in practice neither $p_t$ nor $\mu_t(x)$ is uniquely determined, we can only sample from $p_t$ for $t = 1$ (data) and 0 (prior), and we do not have access to $\mu_t(x)$ for evaluation. As a workaround, CFM proposes to use conditional densities $p_t(x \mid (x_1, x_0))$ and vector fields $\mu_t(x \mid (x_1, x_0))$, where $x_1 \sim p_1$ is a training datapoint and $x_0 \sim p_0$ is noise, such that both are tractable. For example, when

$$p_t(x \mid (x_1, x_0)) = \mathcal{N}(x \mid tx_1 - (1-t)x_0, \sigma^2), \tag{32}$$

$$\mu_t(x \mid (x_1, x_0)) = x_1 - x_0, \tag{33}$$

the CFM loss

$$L_{\text{CFM}} = \mathbb{E}_{t\sim\mathcal{U}(0,1),\, x_1\sim p_1,\, x_0\sim p_0,\, x\sim p_t(\cdot|(x_1,x_0)),}\|\nu_\theta(t,x) - \mu_t(x \mid (x_1,x_0))\|_2^2, \quad (34)$$

has the same gradients as (31), and therefore will lead to the same model $\nu_\theta(t,x)$, but now $p_t(x \mid (x_1,x_0))$ and $\mu_t(x \mid (x_1,x_0))$ are tractable for all $t$. Finally, new datapoints are generated by solving the ODE in (29) starting from $x_0 \sim p_0$ but using the learned vector field $\nu_\theta(t,x)$.

### *7.1. CaloDREAM: Vision Transformer CFM*

By Luigi Favaro, Ayodele Ore, Sofia Palacios Schweitzer, and Tilman Plehn, with figures and tables referring to this approach as `CaloDREAM` [69] and code being available at [70].

---

*Introduction* The `CaloDREAM` [69] architecture consists of two continuous normalizing flows trained with the CFM objective given in (34) ¶. The first is an *energy* model, which is responsible for generating the total energy deposited in each calorimeter layer. The model uses the energy ratio variables $u$, defined in (5), as a basis for the layer energies. `CaloDREAM` then employs a *shape* model to generate voxel values, given the $u$ ratios as conditions. In order to enforce energy conservation, the shape model is trained on voxels normalized by their layer energy. In the following, the unique aspects of the two models comprising `CaloDREAM` are detailed.

*Architecture — Energy model* As discussed above, the heart of a CFM model is a learnable vector field $\nu_\theta$. Although it is typical to use a single neural network as a direct parameterization, other choices are also viable. In particular, the causal nature of energy flow through a calorimeter inspires an autoregressive construction of the full vector field.

`CaloDREAM` adopts an autoregressive CFM architecture introduced in [173] to learn the distribution of energy ratios given an incident energy, $p(u|E_{\text{inc}})$. The model structure is illustrated in figure 30 (left). Instead of directly learning a 45-dimensional vector field, a single network is trained to solve 45 CFM tasks — one for each layer. In order to distinguish these tasks, the network is conditioned on previous layer energy ratios. The full vector field can be written as

$$\nu_{\text{full}}(t, u \,|\, E_{\text{inc}}) = \left(\nu_\theta(t, u_0 \,|\, c_0),\ \ldots\ , \nu_\theta(t, u_{44} \,|\, c_{44})\right), \quad (35)$$

where $\nu_\theta$ is a neural network and each $c_i$ is a condition that encodes the incident energy and the sequence of previous $u$'s

$$c_i = \begin{cases} c_i(u_0, \ldots, u_{i-1}, E_{\text{inc}}) & i > 0 \\ c_i(E_{\text{inc}}) & i = 0 \end{cases} . \quad (36)$$

¶ Here the resolution parameter $\sigma$ from (32) is taken to be zero.

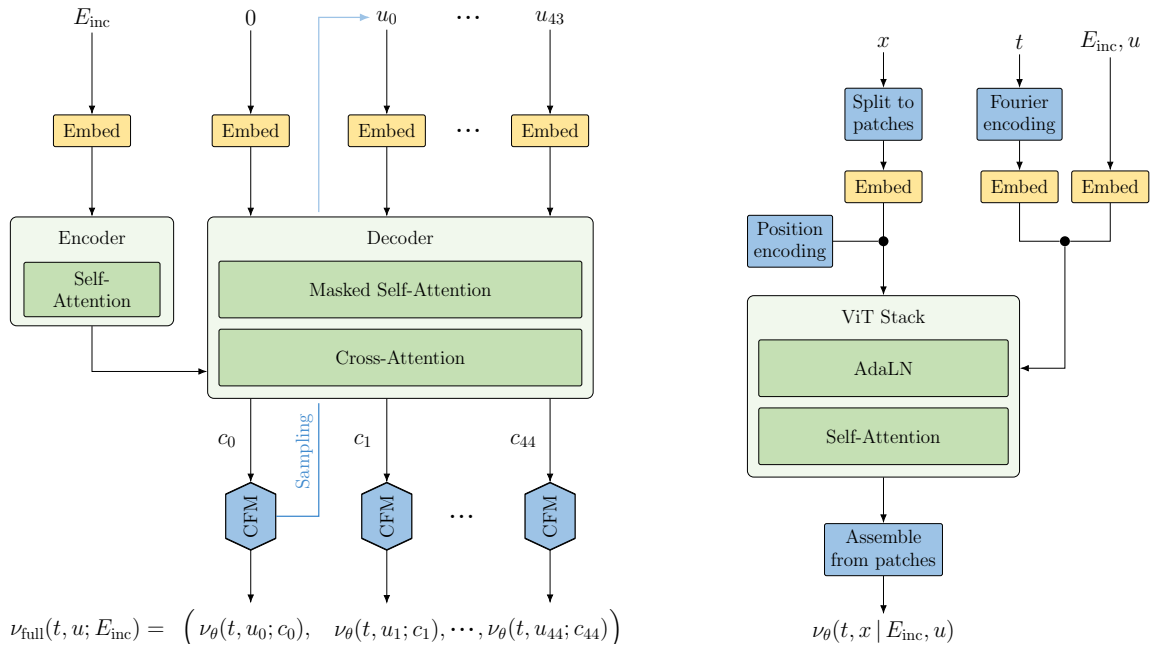



Figure 30: **Left:** Schematic diagram of the `CaloDREAM` energy network. **Right:** Schematic diagram of the shape network. Figure adapted from [69].

In practice, an encoder-decoder transformer is used to learn the conditions. Both the encoder and decoder contain four attention blocks with four heads and an embedding dimension of 64. By applying a triangular mask to the relevant attention matrices, the structure prescribed in (36) is respected. For $\nu_\theta$, a DNN with eight layers and width of 512 is used.

When training the energy model, all components of the full vector field can be evaluated in parallel. During inference, on the other hand, the CFM network must be sampled for each component in sequence. Specifically, $u_0$ is first sampled by integrating $\nu_\theta(t, u_0 \mid c_0)$ and can then be used to compute $c_1$. This condition is in turn required to generate $u_1$ and so forth.

*Architecture — Shape model* `CaloDREAM` also uses CFM for the shape model, which is responsible for learning to sample showers $\mathcal{I}$ from $p(\mathcal{I} \mid E_{\text{inc}}, u)$. Unlike the energy model, here the vector field is parameterized directly with a neural network $\nu_\psi(t, \mathcal{I} \mid E_{\text{inc}}, u)$. In order to obey energy conservation, the shape network is trained on layer-normalized voxels. The network is a vision transformer (ViT) similar to [140] and illustrated in figure 30 (right). As a first step, the network divides the shower into non-overlapping rectangular blocks in the three-dimensional calorimeter space $(z, \alpha, r)$. Each of these regions defines a *patch*, which is embedded with a shared linear layer. The embeddings are supplemented with learnable position encodings which break the permutation invariance among patches. The network uses a joint embedding for the conditional inputs, $t, E_{\text{inc}}$ and $u$. The time embedding is first transformed to Fourier space and

embedded with a two-layer dense network. The energy conditions are instead directly embedded with a separate dense network using the same architecture. The final operation sums the two embedded conditions into a single vector.

After applying these initial transformations, the patches and the conditions are passed to a stack of ViT blocks. Each block contains a self-attention over patches followed by a dense network transformation. The conditional information is introduced via affine transformations with shift and scale $a, b \in \mathbb{R}$ and an additional rescaling factor $\gamma \in \mathbb{R}$ learned by a dense network, referred to as AdaLN [140, 174] in figure 30. Concretely, the operation inside the ViT block is summarized by

$$x_{\mathrm{h}} = x + \gamma_{\mathrm{h}} g_{\mathrm{h}}(a_{\mathrm{h}} x + b_{\mathrm{h}}), \tag{37}$$

$$x_{\mathrm{l}} = x_{\mathrm{h}} + \gamma_{\mathrm{l}} g_{\mathrm{l}}(a_{\mathrm{l}} x_{\mathrm{h}} + b_{\mathrm{l}}), \tag{38}$$

where $g_{\mathrm{h}}$ is the multi-head self-attention step and $g_{\mathrm{l}}$ is the fully connected transformation. After the stack of ViT blocks, the same modulation is applied to the final patch features before projecting back to the original size. The patches can then be assembled into the full calorimeter shape.

*Training* The training is carried out using the AdamW optimizer with an initial learning rate of $10^{-3}$ and a cosine learning rate scheduler. We train the model for 800 and 600 epochs for dataset 2 and dataset 3 respectively. The patch sizes used in each dataset are $(3, 16, 1)$ and $(3, 5, 2)$. In both cases there are six self-attention blocks with six heads each. The `CaloDREAM` samples evaluated in the results section are obtained by solving the energy and shape model ODEs with the Runge-Kutta 4 solver with step size 0.02.

### *7.2. CaloForest*

By Jesse C. Cresswell and Taewoo Kim, with figures and tables referring to this approach as `CaloForest` [71] and code being available at [72].

---

*Introduction* The methods in the above sections differ in their learning tasks and architectures, but all use neural networks as function approximators. Neural network architectures are often carefully designed to have inductive biases that are beneficial for a specific data modality. A case in point are datasets 2 and 3 of section 2.2 which have an image-like structure; each layer of the calorimeter corresponds to a channel, and the voxels of each layer are arrayed in a consistent manner giving a familiar $c \times h \times w$ format. Convolutional neural networks are well-adapted for this structure, achieving efficiency through parameter sharing by applying the same kernel across the image.

However, tabular datasets (like dataset 1 in section 2.1) have minimal structure that neural networks can take advantage of. Researchers still often resort to basic DNN architectures with little-to-no useful inductive bias. Historically, tree-based algorithms

have outperformed on *discriminative* tasks for tabular data at scale [175, 176], with only very recent DNN-based foundation models starting to consistently improve upon tree-based methods [177]. There are additional advantages of tree-based models: they usually do not require any data pre-processing (whereas neural networks are highly sensitive to input data scale and distribution); they can operate on data that contains null values (whereas neural networks require null values to be dropped or imputed); they can be trained efficiently on GPU or CPU (whereas large neural networks usually necessitate GPU training, but most high performance computing clusters are still CPU based); and they have improved explainability (for example, Shapley values [178] are generally intractable to compute for large neural networks, but the TreeSHAP algorithm makes them workable for trees [179, 180]). Yet, tree-based learning is not common for *generative* tasks, even on tabular data.

We propose training generative models for the tabular dataset 1 using a tree-based function approximator, namely XGBoost [181]. Just like a DNN, XGBoost is a universal function approximator, meaning that with large enough number of parameters (*i.e.* tree depth) and training datapoints, it can in principle fit any function [182]. This begs the question of why tree-based models are still rarely used for generative tasks [183, 184]. In principle, XGBoost could be used as a replacement for the neural network function approximator in any generative modeling algorithm, such as the ones used throughout Sections 3–6. In practice, the mechanics of training trees deviates significantly from how neural networks are trained, requiring non-trivial reengineering of algorithms.

*Generative Modeling with Trees* The difficulties of replacing neural network function approximators with XGBoost (in this case for $\nu_\theta(t, x)$) are well-illustrated by the work [184] which provides an implementation of CFM backed by XGBoost. First, notice that when using neural networks, (34) would ordinarily be optimized by sampling a minibatch of data $x_1$ from $p_1$, sampling a $t$, sampling $x_0$ from the prior $p_0$ and then generating $x$ from $p_t(x \mid (x_1, x_0))$ in (32). In particular, the noise vector $x_0$ would be sampled anew every batch, eventually leading to good coverage of the distribution in the expectation. XGBoost is not trained with minibatches; for regression tasks it requires an entire dataset to be fed in and then minimizes the squared error loss overall. Therefore, the noise $x_0$ associated to each training point $x_1$ would only be sampled once. For better coverage of the noise distribution, [184] proposes to duplicate each of the $n$ training datapoints $K$ times, and for each copy of $x_1$ generate different noise $x_0$.

Second, whereas with a neural network the time step $t$ could be fed in as an additional input to the network during training and generation, simply adding $t$ as a feature to XGBoost is unlikely to give sufficient emphasis to it, because only a single feature is used in each split of each tree in the ensemble. Instead, [184] proposes to discretize $t$ into $n_t$ uniform steps and train a different XGBoost ensemble to represent $\nu_\theta(t_j, x)$ for each timestep $t_j$. The expectation over $t$ is removed in the loss function (34), and it is instead treated as a constant for each of $n_t$ separate loss functions.

Third, whereas a neural network can easily be designed with a number of outputs

equal to the number of features $p$ in $x$ (the same size as the regression target $\mu_t(x \mid (x_1, x_0))$ for a given $t$), XGBoost only outputs a scalar. A brute-force workaround is to simply train a different ensemble to predict each element of the vector field $\mu_t(x \mid (x_1, x_0))$.

Fourth, when conditional generation on a class label $y$ is required, a neural network can accept $y$ as an input during training and generation to adapt its behavior while sharing parameters. Like conditioning on $t$, conditioning on $y$ is better done by training a separate XGBoost ensemble for each of the $n_y$ classes. This is only suited to conditioning on categorical labels, not continuous ones, so it does not allow for sampling at arbitrary incident energies.

Despite these challenges, the promises of tree-based generative modeling are seductive: better performance on tabular regression tasks may translate to better tabular generation; lack of need for preprocessing; native handling of missing values; efficient training on CPU; and improved explainability. As a proof of concept for tree-based generative modeling of calorimeter showers, we applied CFM backed by XGBoost to the tabular dataset 1.

*Modeling Dataset 1* In total, for a tabular dataset of size $[n,\ p]$, the method described by [184] requires training $n_t \times p \times n_y$ XGBoost ensembles, each on a dataset of size $[n_i \times K,\ p]$, where $n_i$ is the number of points belonging to class $i$ (with $\sum_i^{n_y} n_i = n$). This poses a practical challenge. The largest training datasets benchmarked in [184] had sizes $[16\,512,\ 9]$ (largest $n$), $[288,\ 90]$ (largest $p$), and $[10\,888,\ 16]$ (largest product $np$), while dataset $1 - \pi^+$ is $370\times$ larger at $[120\,800,\ 533]$.

Unfortunately, the implementation of CFM with XGBoost published by [184] does not scale to problems of this magnitude.[+] Noting that [184] recommend $n_t = 50$ and $K = 100$, dataset $1 - \pi^+$ (with $n_y = 15$) would necessitate training 399 750 XGBoost ensembles, most of which would use a dataset of size $[1\,000\,000,\ 533]$. From the original $[n,\ p]$ dataset, the implementation attempts to create a duplicated version of size $[n_t, K \times n, p]$ as a `numpy` array in memory all at once, which for the pions training dataset requires 2.34 TiB of CPU memory. Training thousands of XGBoost ensembles on slices of the data in parallel further exacerbates the memory burden, since many copies of the data array are created and persisted in RAM or RAM disk. We estimate that a full training run using the default hyperparameters on dataset $1 - \pi^+$ would require more than 1.2 PiB of CPU memory.

However, this memory burden is not a fundamental limitation of the proposed method, but rather a lack of optimization of the original implementation. For our proof-of-concept, we reimplemented CFM with XGBoost solving many engineering challenges around memory efficiency and parallelization in python. Our implementation runs with a peak CPU memory burden of 78 GiB on dataset $1 - \pi^+$, or roughly 16 000 times less.

[+] We accessed this code repository `https://github.com/SamsungSAILMontreal/ForestDiffusion`, commit hash `855281b` dated Nov. 2, 2023.

In addition to improving the memory efficiency and runtime, we also mention methods to increase model performance. From hyperparameter ablation we found that $n_t$ has the largest effect on model fit and should be increased as high as feasible, noting that this comes with a linear increase in the training time and number of model parameters. We found that for datasets with larger $n$ a more conservative value of $K$ was sufficient compared to the recommendation in [184]. Larger tabular datasets tend to have more redundant rows built in, and different noise is added to these rows giving sufficient coverage with lower $K$. We observe that the model consistently underfits the dataset for all sizes we tested, in agreement with [184], but underfitting can be mitigated by increasing the learning rate substantially. Although XGBoost typically does not require data preprocessing, the CFM algorithm adds noise which must be commensurate to the data's typical scale. Hence, it is important to at least scale the data to a finite domain similar to the standard deviation of the added noise. The original implementation uses MinMax scaling over the entire dataset. However, when the data has distinct classes with different properties, as is the case for the incident energy levels of Dataset 1, we find that it is more beneficial to scale each class separately since XGBoost models are trained separately for each class.

In summary, we trained models for the photons and pions datasets using a single desktop workstation with 250 GiB RAM and 40 CPUs (Intel Xeon Silver 4114T). We discretized time into $n_t = 100$ steps, and duplicated each datapoint $K = 20$ times. Each XGBoost ensemble had 20 trees of maximum depth 7, a learning rate of 1.5, and all other hyperparameters left as defaults.* We trained up to 20 XGBoost ensembles in parallel at a time, each with 2 CPUs. In total, for the photons model 552 000 XGBoost ensembles were trained in 135 hours with a peak memory burden of 54 GiB, while the pions model used 799 500 ensembles, completed in 281 hours, and required 78 GiB of memory.

This proof of concept shows that tabular generative modeling with tree-based function approximators trained on CPU is feasible for calorimeter shower simulation. We have worked through an example of how to convert from neural networks to XGBoost using a modern generative framework. However, our trained models have clear room for improvement on several fronts. First, performance is not yet competitive with highly-tuned neural network approaches. Second, our models are massively overparameterized (although we do not observe overfitting), with the number of XGBoost ensembles several times larger than the number of datapoints they were trained on (each ensemble having thousands of parameters). Third, the sheer number of trees trained contributes to slow training and large model size on disk. We believe these are solvable issues. Performance could be improved by replacing the simple Euler ODE solver with more advanced methods, though we point out that the learned vector field $\nu_\theta(t, x)$ only allows sampling at discrete values of $t$. We anticipate model size could still be reduced with additional hyperparameter optimization, or by moving to multi-output trees. Training

* We used XGBoost version 2.0.0.

time could be slashed by parallelizing training steps across a cluster of CPUs, which is straightforward for this method.

## 8. Introduction to metrics

The evaluation of DGMs is a challenging task that has seen significant research in itself in the past years [185, 186, 20]. For the application of DGMs as part of the detector simulation, we are interested in surrogate models that are faithful (*i.e.* reproduce the showers of GEANT4 as close as possible), light-weight (*i.e.* do not require much space to store and are fast to load), and fast in generation. Each of these aspects by itself is hard to capture with a single number, so we will report a set of different metrics to give a more complete picture. It is expected that there will not necessarily be a single clear winner, and different submissions will have their advantages and disadvantages.

### *8.1. High-level Features (Histograms)*

We begin the evaluation by looking at high-level features, *i.e.* physical observables that are derived from the energy depositions in the calorimeter. We focus on the following set:

- The energy deposition in each voxel: $\mathcal{I}_{ia}$.
- The energy depositions in each layer of the calorimeter, as the sum over all voxels in that layer: $E_i = \sum_a \mathcal{I}_{ia}$.
- The total energy deposition in the shower, as sum over all voxels, normalized to the incident energy: $E_{dep}/E_{inc} = \sum_{a,i} \mathcal{I}_{ia}/E_{inc}$.
- The centers of energy in $\eta$, $\phi$, and $r$ direction, defined via $\sum_a l_a \mathcal{I}_{ia} / \sum_a \mathcal{I}_{ia}$. The locations $l_a$ are either $\phi_a = r_a \sin\alpha_a$, $\eta_a = r_a \cos\alpha_a$ or $r_a$, where $r_a$ and $\alpha_a$ are the centers of the voxels in $\alpha$ and $r$. These are taken as the mean of the voxel boundary values defined in the `binning.xml` files. The sum goes over all voxels $a$ in a given layer.
- The width of the center of energy distributions in $\eta$, $\phi$, $r$ direction: $\sqrt{\frac{\sum_a l_a^2 \mathcal{I}_{ia}}{\sum_a \mathcal{I}_{ia}} - \left(\frac{\sum_a l_a \mathcal{I}_{ia}}{\sum_a \mathcal{I}_{ia}}\right)^2}$
- The sparsity, defined as 1 minus the activity, with the activity being the fraction of voxels per layer with an energy deposition above threshold (threshold is defined per dataset in section 2).

For each of these observables, we compute the *separation power* between the submissions and the held-out test set. We use the same binning as shown in appendix A in the reference histograms for the two GEANT4 datasets. The separation power between two histograms is defined as [187]

$$S(h_1, h_2) = \frac{1}{2} \sum_i \frac{(h_{1,i} - h_{2,i})^2}{h_{1,i} + h_{2,i}}, \tag{39}$$

where $h_{j,i}$ is count of the $i$th bin of histogram $j$. The histogram counts are expected to be normalized: $\sum_i h_{j,i} = 1$. With these definitions, we have $S = 0$ if and only if $h_1 = h_2$ and $S = 1$ if the two distributions have no overlap.

The separation power is closely related to the $\chi^2$ homogeneity test [188, 189, 190]. The difference is that the $\chi^2$ test statistic does not include normalization of the histogram counts.

To get a better feeling for the natural statistical spread of the separation power between different GEANT4 datasets, we show a gray band in all figures of separation powers, indicating the minimal and maximal separation power we found comparing ten different pairs of GEANT4 datasets. For dataset 1, we constructed these pairs by joining, shuffling, and then splitting the events from the two given datasets (*i.e.* drawing without replacement from the joined dataset), ensuring that the $E_{inc}$ distribution is always the same. For datasets 2 and 3, we generated 9 additional datasets with GEANT4, 100 000 showers each, such that we get ten sets of pairs.

### *8.2. Correlations*

The energies deposited in subsequent layers are correlated with each other due to the size of the particle shower in $z$ direction. One measure to study if these correlations are learned correctly is given by Pearson correlation coefficient (PCC) between the layer-wise energy depositions [20]. For two sets of layer energies $\{E_i\}$ and $\{E_j\}$ of the same size, the PCC is given by

$$\mathrm{PCC}(E_i, E_j) = \frac{\sum_k \left(E_{i,k} - \mathrm{mean}(E_i)\right)\left(E_{j,k} - \mathrm{mean}(E_j)\right)}{\sqrt{\sum_k \left(E_{i,k} - \mathrm{mean}(E_i)\right)^2}\sqrt{\sum_k \left(E_{j,k} - \mathrm{mean}(E_j)\right)^2}}, \quad (40)$$

where $k$ runs over all samples in the set, and $i$ and $j$ are layer numbers.

### *8.3. Classifier-based Metrics.*

Classifiers offer a way to perform a two-sample test [191] that is sensitive to the full distribution, including correlations between features. In the context of generative models for calorimeter simulation, they have been proposed as metric in [109] and were further discussed in [186], where it was also shown that they can give valuable insights to what failure mode the generative model has.

Here we focus on two different classifier tests. The first one, a binary classification task, compares each submission with the GEANT4 test set. The second one, a multiclass classification task, compares all submissions with each other. For each, we consider two different neural network architectures.

*Binary classification* The binary classification test evaluates how well the underlying distribution was learned and therefore how close the generated distribution is to the reference. It relies on the Neyman-Pearson Lemma [192], stating that the most powerful classifier to distinguish two samples is their likelihood ratio. If a well-trained classifier

Table 9: Number of samples in training, testing and evaluation datasets in the binary classification setup.

| dataset | training | testing | evaluation |
|---|---|---|---|
| 1 - photon | 80 000 | 20 000 | 21 000 |
| 1 - pion | 80 000 | 20 000 | 20 800 |
| 2 | 60 000 | 20 000 | 20 000 |
| 3 | 60 000 | 20 000 | 20 000 |

is unable to distinguish submitted samples from the GEANT4 test set, we conclude that the submission replicates the GEANT4 distribution well [109, 186]. The result of this test, however, depends on the preprocessing that was applied to the data. Using the calorimeter showers in the physical space lets the classifier focus on the brightest voxels only, since energy depositions in them are orders of magnitude above the low-energy depositions. Applying a logarithm or logit transformation, enhances the sensitivity to mismodeling in them. While this gives a better understanding on whether or not the entire distribution was learned well, it might be that the difference is only in features and correlations that are irrelevant for the down-stream physics analysis. For that reason, we consider two different sets of input features. The first one are the energy depositions in the voxels (called "low-level" observables), the second one are the observables we introduced in Section 8.1 (called "high-level" observables).

The figure of merit in this setup is the AUC, the area under the receiver operating characteristic (ROC) curve. The ROC curve shows the true positive rate (TPR) as a function of the false positive rate (FPR). In a random classifier, the TPR will grow linearly with the FPR giving a AUC of 0.5. In a classifier that can separate the two datasets perfectly well, the ROC curve will become a step function, so the AUC becomes 1. We train ten classifiers with different random initialization and average the AUCs when reporting the results.

In training, we split the submission and GEANT4 dataset each into training, testing, and evaluation sets first, before merging them with the corresponding labels. This ensures having always a balanced setup. The number of events in each set is shown in table 9. We select the model state with the highest accuracy on the test set for the final evaluation. Before evaluating the AUC on the evaluation dataset, we calibrate the classifier with isotonic regression [193] on the test set.

However, since a different neural classifier is trained for each submission, a comparison between submissions on equal conditions is harder to make. Therefore, we consider a second classifier test based on a multiclass classification setup below.

*Multiclass classification* With the multiclass classification setup, we try to assess which of the submissions is closest to GEANT4. The method was introduced in [194] in the context of comparing hydrodynamical galaxy simulations, and subsequently applied to

high-energy physics scenarios in [31, 195]. It relies on training a single classifier with cross entropy loss on the task "submission 1 vs. submission 2 vs. ... vs. submission $n$". When evaluating the trained classifier on a GEANT4-based test set, we can read off which submission the GEANT4 sample is closest to.

As figure of merit, we consider the average of the log posterior [194]. It is defined as

$$LP(\text{model } i|\text{samples } j) = \frac{1}{N} \sum_{x_k \in j} \log p_{\text{model } i}(x_k), \tag{41}$$

the average logarithm of the probability that samples $j$ come from the model (submission) $i$. Here, the index $k$ goes over all $N$ samples in the set $j$. As a cross check of the quality of the trained multiclass classifier, we look at its performance in identifying the held-out test sets of each submission. A well-trained classifier will be able to distinguish the individual submissions from each other, so

$$LP(\text{model } i|\text{samples } j = i) > LP(\text{model } i|\text{samples } j \neq i). \tag{42}$$

We check that this holds for each trained multiclass classifier before using it for final evaluation. We train ten classifiers with different random initialization and average the mean log posteriors of the ten runs. The results of the cross check can be found in appendix B.

The submissions are split in training, testing, and evaluation sets as shown in table 9, before they are merged and shuffled into single training, testing and evaluation sets. In training (both the DNN and the CNN ResNet architecture), the best model state based on the validation loss is used for the final evaluations.

*DNN* We consider a regular DNN for the binary classification on low- and high-level observables, and the multiclass classification setup. The DNN of the binary classification consists of an input layer, two hidden layers of 2048 neurons each, and an output layer. We use leaky ReLU activations (with negative slope 0.01) in all layers except the last one, where we use a sigmoid activation. We do not use dropout or batch normalization. The network is optimized with the Adam optimizer [90], a learning rate of $2 \cdot 10^{-4}$, and in batches of 1000 samples for 50 epochs.

The DNN of the multiclass classification test consists of an input layer, one hidden layer with 4096 neurons, and an output layer. We use leaky ReLU activations (with negative slope 0.01) in all layers except the last one, where we use a softmax activation. No dropout or batch normalization is used. We optimize the network with a schedule-free AdamW optimizer [196] and an initial learning rate of $1 \cdot 10^{-3}$ in batches of 2000 samples for 25 epochs (or fewer, if the validation loss already increases).

When classifying "low-level" observables, we use the voxel energies normalized to the incident energy and the decadic logarithm of the incident energy as input features. "High-level" observables are given by the observables we introduced in section 8.1.

*CNN ResNet* An alternative architecture based on 3D CNNs is considered for the binary and multiclass classification on low-level features. Compared to a fully-connected DNN, a CNN is more capable of exploiting the spatial structures of particle showers, therefore allowing it to provide stronger separation between different models. We adapt a 3D CNN implementation [197] based on the ResNet architecture [127] to process the particle showers. Each shower is treated as a 3-dimensional image where the intensity of each pixel is the energy deposition in the corresponding voxel. This leads to images of a shape $(45, 16, 9)$ for dataset 2, and $(45, 50, 18)$ for dataset 3. The shower image is first processed by a 3D convolution with a kernel size of 7 and a stride of 2, followed by a max pooling layer with a kernel size of 3 and a stride of 2, for downsampling. Then, an 18-layer ResNet is applied to the downsampled image. A kernel size of 3 is used in all the convolutions in the ResNet, and the number of output channels in each convolution ranges between 32 to 128. A global average pooling is used to aggregate the output to a 1D feature vector summarizing the full image. This feature vector is then concatenated with the incidence energy, normalized with a batch normalization layer [97], before being processed by a final fully-connected layer for the classification.

For dataset 2, we optimize the network for 48 epochs using the AdamW optimizer [114] with learning rate $2.5 \cdot 10^{-5}$ and otherwise default settings. For dataset 3, it was sufficient to use the same optimizer setup, but with learning rate $5 \cdot 10^{-5}$ for 12 epochs.

### *8.4. Computer Science-inspired Metrics*

A standard quantitative benchmark for state-of-the-art generative models in computer vision is the Fréchet Inception distance (FID) [198]. The idea behind FID is to extract salient high-level features of real and generated images via the activations of the penultimate layer of a high-performing inception classifier, and then compare them using the Fréchet, or 2-Wasserstein, distance between Gaussian fits to the two sets of features. This metric has been shown to be highly sensitive to the quality and diversity of generated images and has been extended as well to evaluate jet simulations using the ParticleNet classifier [199]. Recently, however, [185] studied a physics-informed alternative to this method, referred to as the Fréchet physics distance (FPD) based on high-level *physical* features of the samples, rather than DNN classifier activations, which proved to be highly performant. The complementary kernel physics distance (KPD) metric was proposed as well, similarly inspired by the popular kernel Inception distance (KID) [200], which calculates a kernel-based estimate of the maximum mean discrepancy between the two sets of features. In this work, we apply FPD and KPD to evaluate the various surrogate models by using the meaningful high-level features of calorimeter simulations outlined in section 8.1. We also importantly derive baseline scores and errors with which to compare the submissions for the different datasets using the procedures described in [185]. We use the implementation of [201] with a minimum sample size of 10 000 in the computation of FPD and a batch size of 10 000 for the KPD.

*8.5. Manifold-based Metrics*

Manifold-based metrics construct a proxy for the generated and reference data manifold and provide a computationally straightforward way to asses the diversity of the submitted samples. The diversity measures how well the generated samples populate the entire data manifold. There is a trade-off between realism and diversity [202] observed for natural images, which immediately triggers the question if such a trade-off also exists for calorimeter showers. Here, we study four different metrics: Precision, Recall, Density, and Coverage, which are defined in the following [203, 204, 205]. While Precision and Density are more a measure of shower quality, Recall and Coverage measure diversity. We report results on the former two as well because all four metrics are closely related to each other and correlations between them provide additional insights.

Precision and Recall first construct a manifold of "real", *i.e.* reference, and "fake", *i.e.* generated samples. These are defined as the union of all $d$-dimensional spheres around the points $x_i$, with the radii chosen such that the $k$ nearest samples are inside the sphere,

$$\text{manifold}(x_1, \ldots, x_n) = \bigcup_{i=1}^{n} \text{B}(x_i, \text{NND}_k(x_i)). \tag{43}$$

Here, $\text{B}(x, r)$ defines a sphere around $x$ with radius $r$ and $\text{NND}_k(x)$ denotes the distance of $x$ to its $k$th nearest neighbor.

- *Precision.* Following the definition of the improved precision of [204], it counts the binary decision of whether the generated data $y_j$ is contained in any neighborhood sphere of reference samples $x_i$. It is bounded by 1.

$$\text{precision} = \frac{1}{m} \sum_{j=1}^{m} \mathbf{1}_{y_j \in \text{manifold}(x_1,\ldots,x_n)} \tag{44}$$

  Here, $\mathbf{1}$ is the indicator function and $n(m)$ is the number of reference (generated) samples.

- *Density* improves the Precision metric by taking into account that the manifold around reference outliers is overestimating the manifold [205]. It counts how many reference-sample neighborhood spheres contain $y_j$. The manifold is now defined as the superposition of spheres instead of the union, and models that place samples in regions where the reference samples are densely packed are getting a higher score. However, it is not bounded by 1 anymore.

$$\text{density} = \frac{1}{km} \sum_{j=1}^{m} \sum_{i=1}^{n} \mathbf{1}_{y_j \in \text{B}(x_i, \text{NND}_k(x_i))} \tag{45}$$

- *Recall.* Following the definition of the improved recall of [204], it is symmetrically with respect to precision. It counts the binary decision of whether the reference

data $x_i$ is contained in any neighborhood sphere of generated samples $y_j$. It is also bounded by 1.

$$\text{recall} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{x_i \in \text{manifold}(y_1,\ldots,y_m)} \tag{46}$$

- *Coverage* measures the fraction of reference samples whose neighborhoods contain at least one generated sample. It is bounded between 0 and 1.

$$\text{coverage} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\exists j \text{ s.t. } y_j \in \text{B}(x_i, \text{NND}_k(x_i))} \tag{47}$$

In our analysis, we chose $k = 5$ [205] and preprocess all voxels by $\log_{10}\mathcal{I}$. Voxels without energy deposition, *i.e.* below threshold, are set to $\log_{10} 0.1 \text{ MeV} = -1$.

### *8.6. Generation Timings*

To properly compare the generation times of all submissions, each submitting group created a singularity container [206] of the necessary software environment. We transferred them to the `clip` cluster [207] and measured the time it takes to load the container, load the model (weights and biases), if applicable move it on the GPU, generate the samples, and save them as `.hdf5` [74] to disk. While this contains an additional overhead, we think that it is more realistic, closer to the real-life application. There is some scatter from run to run, but that comes from the execution on a cluster and most likely is also present in a full simulation chain. We therefore repeat these steps ten times and show the mean and standard deviation of the run times.

Current fast simulation frameworks, with or without deep generative models, simulate with batch size of 1, since this is how simulation is handled in GEANT4, with the parallelization applied commonly on the event, and not particle, level (different events are simulated simultaneously in different threads). Most of the DGM architectures, however, benefit from larger batch sizes. We therefore study batch sizes of 1, 100, and 10 000 to show how the models behave under different use cases.

Unless explicitly noted otherwise, we generate as many samples as were in the training set. However, for some models and smaller batch sizes we had to reduce the overall number of generated events. In this cases, the overhead of loading the model will have a higher share in the overall generation time compared to the sample generation.

DGMs usually run a lot faster on graphics processing units (GPU), since these are optimized for matrix-vector multiplications. Yet, these are not as widely available on HPC clusters, where the majority of nodes have only CPUs. We therefore run the timing evaluations on both types of hardware and report the results. We run the CPU timings on an Intel® Xeon® Gold 6138 CPU @2.00GHz with 170 GB RAM. While this is more on the slow end, we used this node because of the larger RAM requirements of some models. The GPU timings were done with a NVIDIA® A100-SXM4 with 40GB Graphics RAM, 360GB RAM, and Intel® Xeon® Gold 6226R CPU @2.90GHz. These are the C2 and G4 partitions of the `clip` cluster [207], respectively.

### *8.7. Memory Requirements*

As a proxy of the memory requirements to store each model on disk, we report the number of trainable parameters that each model requires. In particular, we report two numbers. One refers to how many trainable parameters are involved in training the generative model. The other one refers to how many trainable parameters are required for generation, *i.e.* how many need to be loaded in production. These numbers can differ for example in GANs, where only the generator network is needed in production, or in cases where the generative network is a distilled version of another model. We know that the actual memory requirements depend on the floating point representation used for the parameters and on the number of additional, non-trainable parameters that are required to load and run the model. Techniques like node pruning and weight quantization can reduce the number of parameters and the memory footprint significantly, sometimes without loss in sample quality. Nevertheless, we decided to not focus on these aspects and just work with the number of trainable parameters.

## 9. Results: Individual Metrics

In total, we present here the results of 59 submissions. These are evenly spread across the different datasets and generative model architectures as can be seen in figure 31 and table 1.

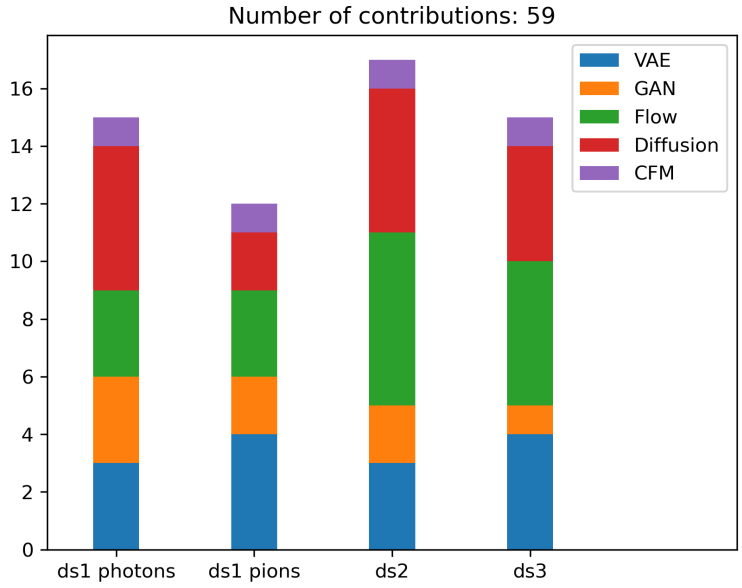



Figure 31: Number of submissions per dataset and DGM architecture.

### 9.1. Preprocessing

All submitted files were checked for `NaN` entries, if the $E_{\text{inc}}$ distribution matches the expectation, and if the correct number of samples were submitted. Then they were saved as `np.float32` numbers in a `hdf5` file [74] with gzip compression. A threshold cut was applied to all voxels before they were used in evaluation. The GEANT4 reference was treated the same way, and all results below use the second GEANT4 dataset that was provided at [76, 82, 83].

### 9.2. Dataset 1, Photons (ds 1 – $\gamma$)

We begin the evaluation with the high-level features, and especially the energy depositions in figure 32. The separation power of the submissions vary roughly within 2 orders of magnitude and they stay almost everywhere just at the upper limit of the GEANT4 reference. It is interesting to note that almost all submissions show a better performance, *i.e.* a smaller separation power, in layers with an angular segmentation (1 and 2, see table 2). Having more voxels per layer seems therefore beneficial for modeling the layer energies. The best performance is given by normalizing flow (`CaloFlow`) and diffusion model (`CaloScore`) based submissions. We should note that the otherwise well-performing `CaloDiffusion` has a bad separation power in the total energy deposition, one of the crucial observables.

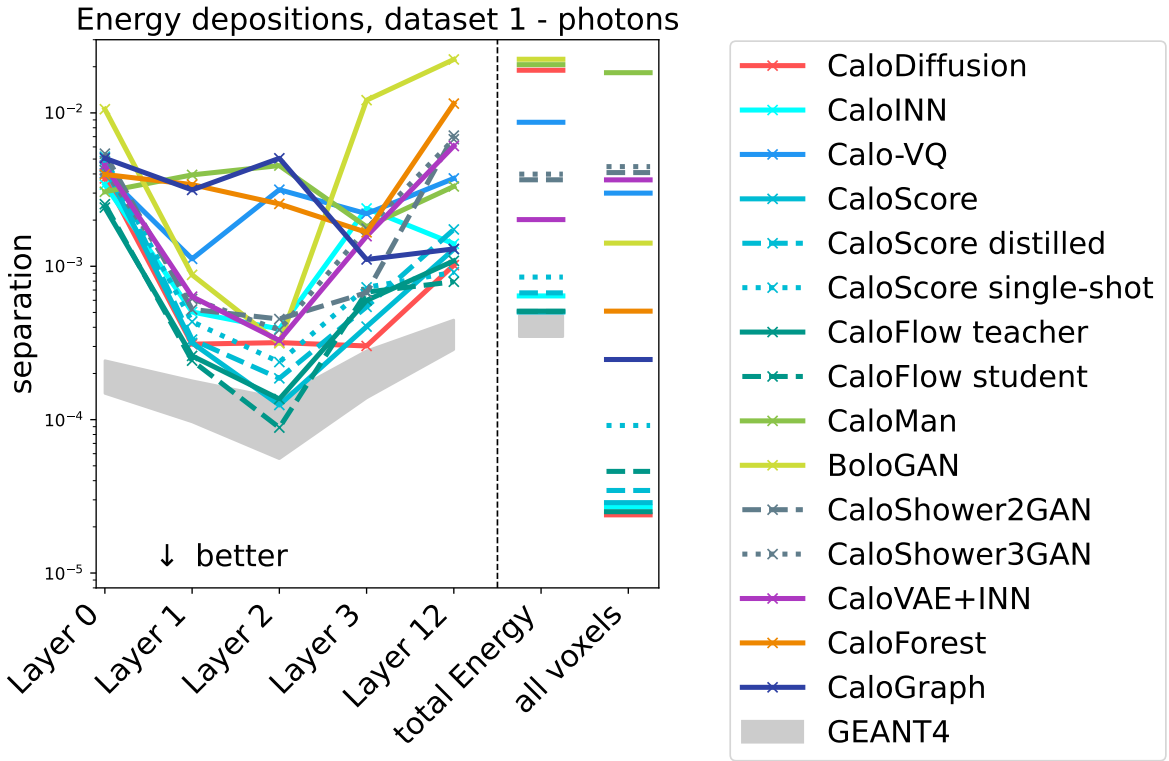



Figure 32: Separation power of energy depositions with threshold at 1 MeV.

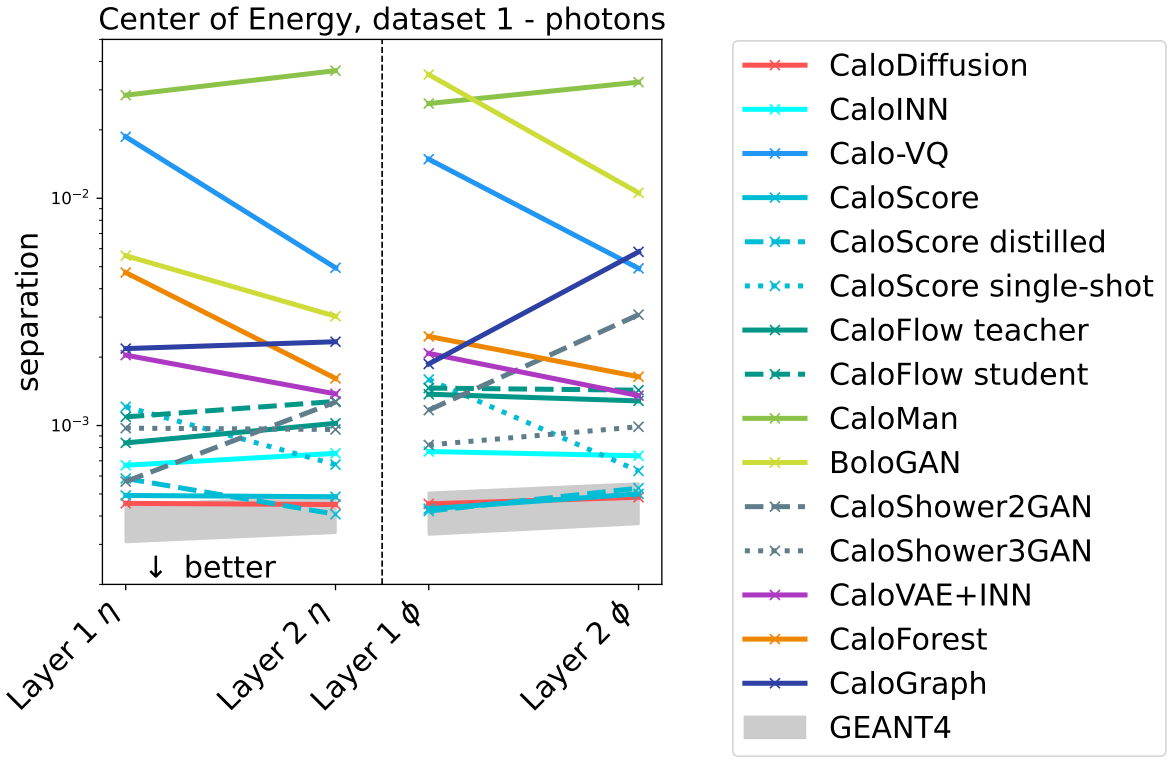



Figure 33: Separation power of centers of energy with threshold at 1 MeV.

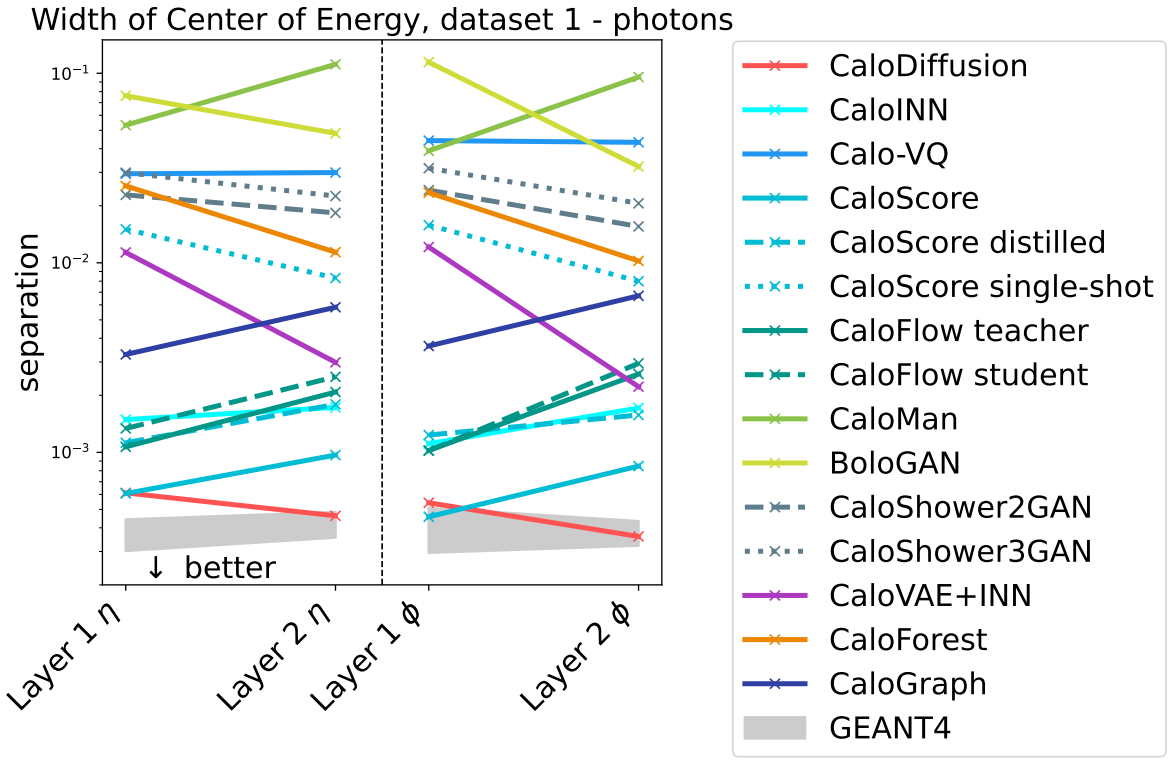



Figure 34: Separation power of widths of centers of energy with threshold at 1 MeV.

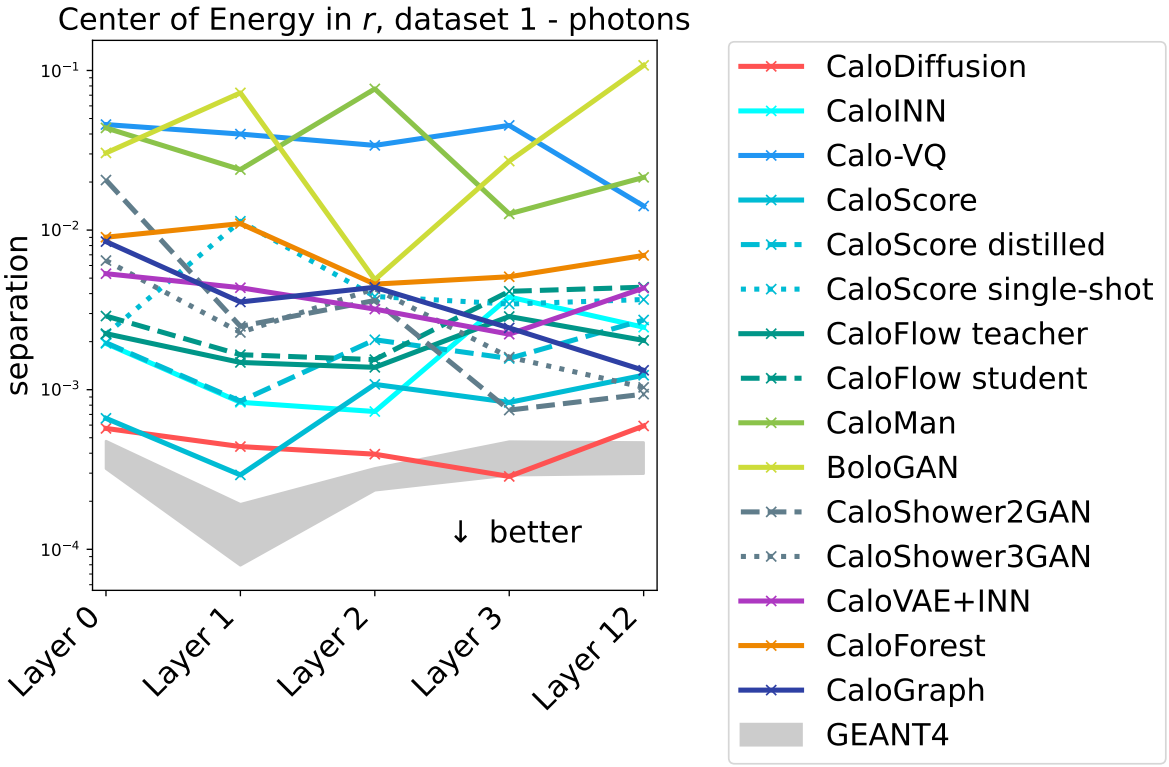



Figure 35: Separation power of centers of energy with threshold at 1 MeV.

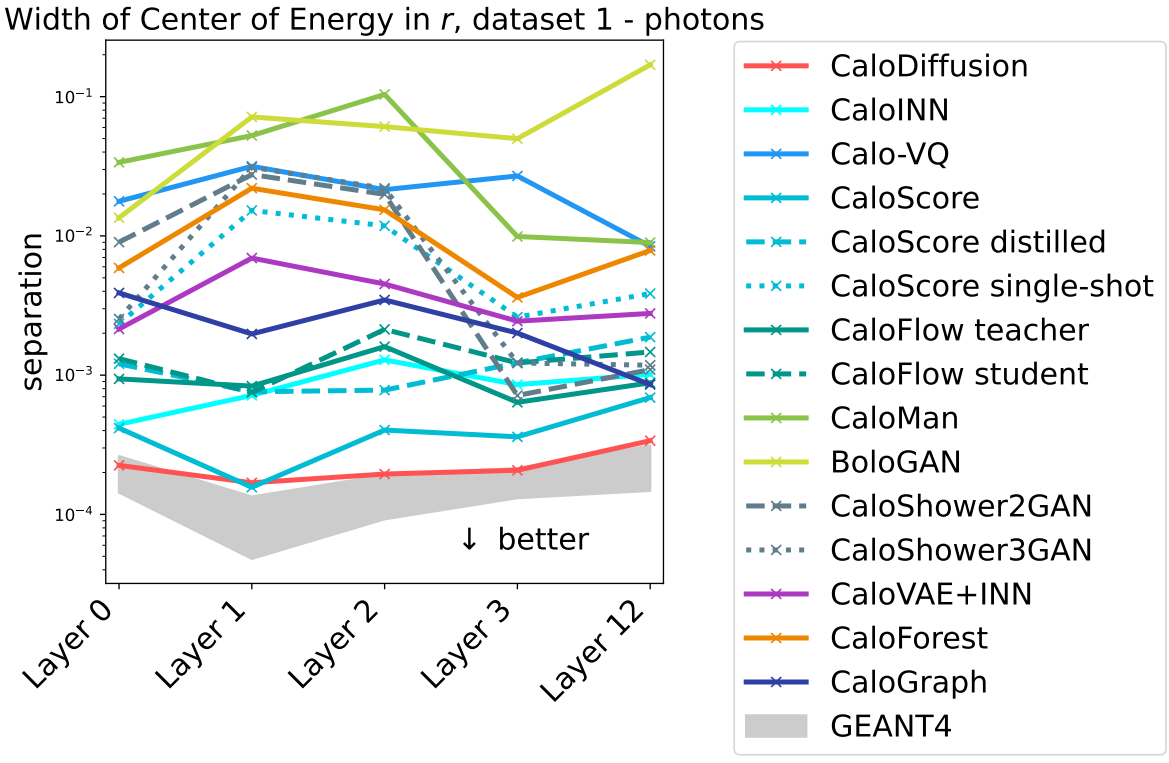



Figure 36: Separation power of widths of centers of energy with threshold at 1 MeV.

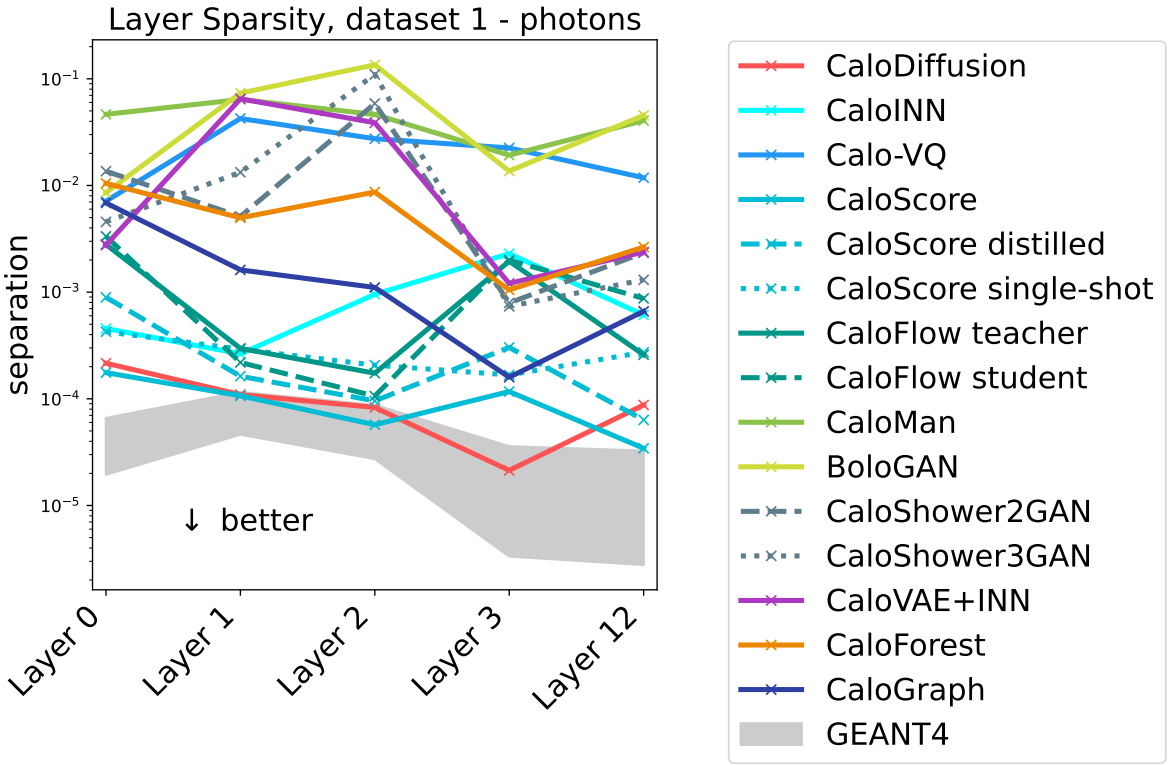



Figure 37: Separation power of the sparsity with threshold at 1 MeV.

The centers of energy in $\eta$ and $\phi$ direction are summarized in figure 33. Here we see the diffusion model based submissions `CaloDiffusion` and `CaloScore` with the best performance, at the level of the GEANT4 reference. In general, we observe all models performing equally well in $\eta$ and $\phi$ direction.

The widths of centers of energy in $\eta$ and $\phi$ in figure 34 tell a similar story. Again, the model performance in $\eta$ and $\phi$ directions is about the same, and the diffusion models `CaloDiffusion` and `CaloScore` have the best separation power, just slightly worse than the GEANT4 reference. However, the distilled versions `CaloScore distilled` and `CaloScore single-shot` are worse now, having a larger separation power than the first normalizing flow models of `CaloFlow`.

The centers of energy in $r$ in figure 35 and its width in figure 36 show separation powers that are more or less constant from layer to layer, stemming from the fact that $N_r$ roughly stays within one order of magnitude. A few submissions show worse performance in the width for layers 1 and 2, where the angular segmentation is present. The ordering of the different DGMs is about the same as for the $\eta$ and $\phi$ directions, as these are correlated. `CaloDiffusion` is at the upper level of the GEANT4 reference and `CaloScore` is next, with the distilled versions a little worse and at the level of the normalizing flow submissions `CaloINN` and `CaloFlow`.

The last observables we compare with separation powers are the sparsities shown in figure 37. These show the largest spread among the considered observables, spanning

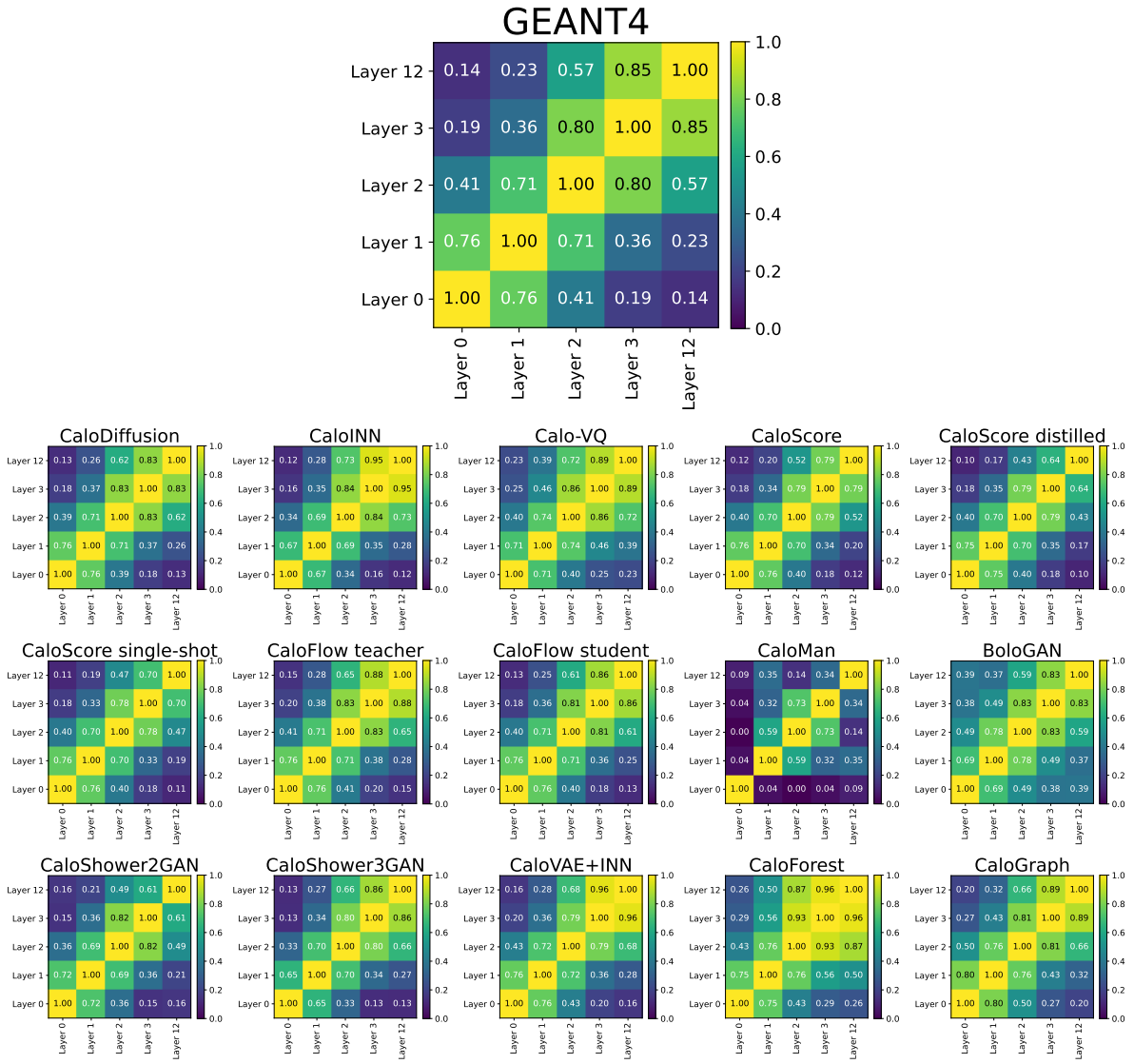



Figure 38: Pearson correlation coefficients of layer energies in ds $1-\gamma$, with threshold at 1 MeV.

four orders of magnitude between the GEANT4 reference and the worst performing submission. The ordering of the models, however, is similar to all the other considered observables. The diffusion models are at the upper end of the reference, with distilled versions in between normalizing flow based models. `CaloGraph`, which was just slightly worse than these in all other observables too, is now at the same level.

We now move on to investigate the correlations between the energies deposited in the layers in figure 38. Overall, most of the submissions reproduce the pattern induced by GEANT4 well, but there is a noticeable tendency of models to overestimate the correlation between layers 3 and 12, as seen in the top right corners. Some models based on GANs and VAEs, which had higher separation powers, also seem to have a harder time reproducing these correlations.

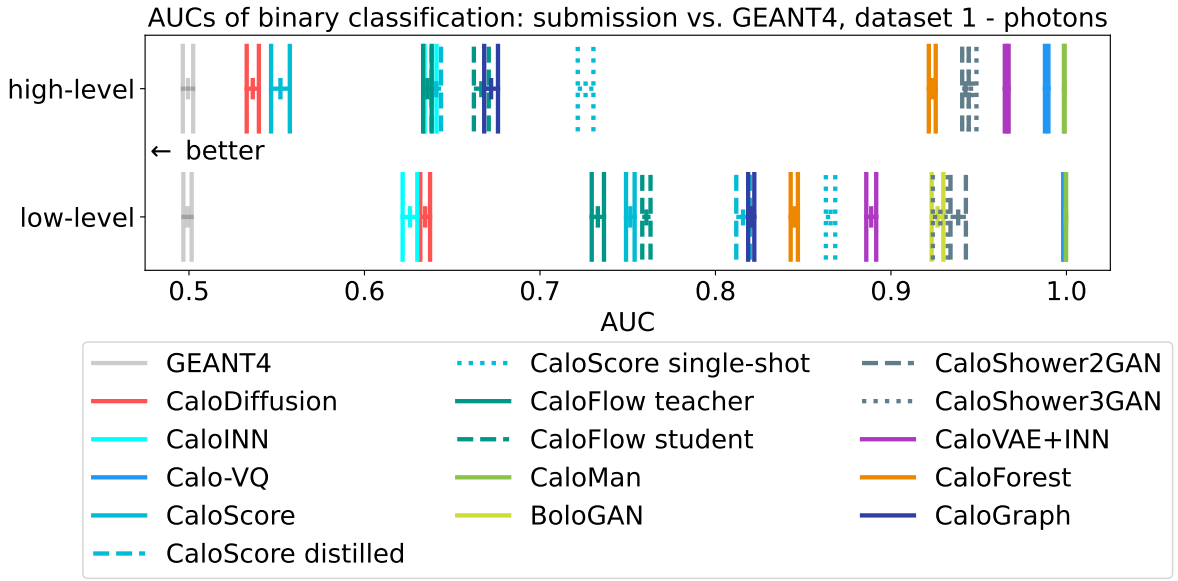



Figure 39: Low-level and high-level AUCs for evaluating GEANT4 vs. submission of ds $1-\gamma$, averaged over 10 independent evaluation runs. For the precise numbers, see Table C1.

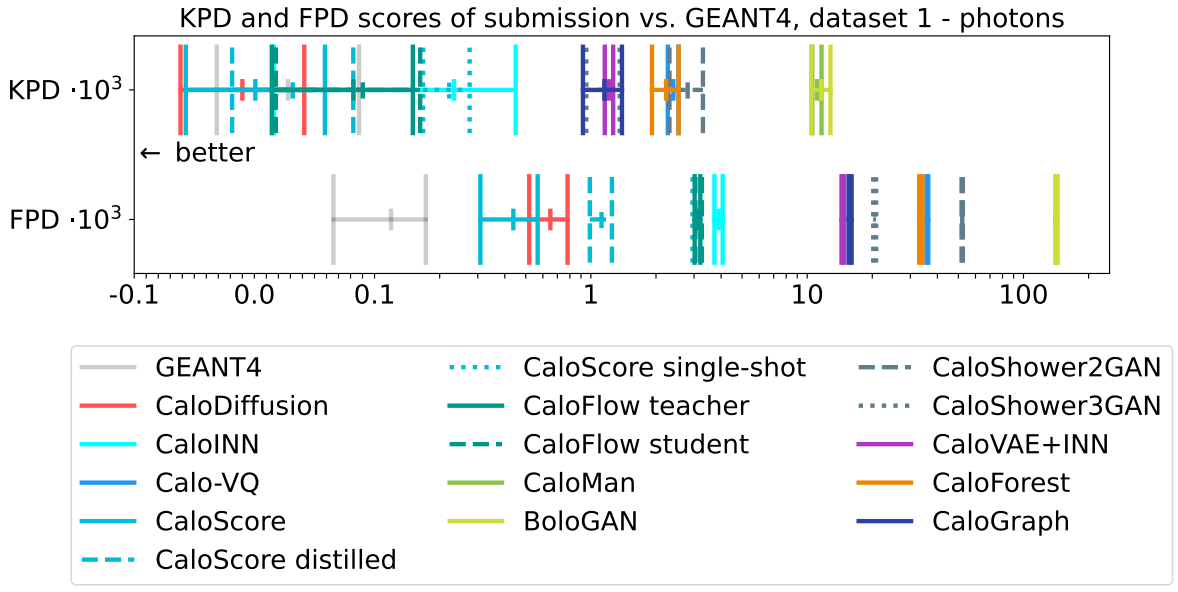



Figure 40: KPD and FPD for evaluating GEANT4 vs. submission of ds $1-\gamma$. For the precise numbers, see Table C2.

Another way to look at the correlations between all observables is given by the classifier metric. Figure 39 (and table C1) shows the AUCs of classifying low-level and high-level observables of the submission against the GEANT4 reference. In general, we observe a good consistency between the two sets of observables and a small spread of AUCs for reruns with different initialization. Submissions that have a high (low) score in the low-level observables also have a high (low) score when high-level observables are used as an input. The difference between the AUCs of the same submission is below 0.2. These results are also consistent to what we have seen in the separation power. DGMs based on diffusion models or normalizing flows achieve the best results, with AUCs of $\mathcal{O}(0.6)$. We also observe that distilled versions tend to perform worse compared to their base model. This is more prominent for `CaloScore` distilled to `CaloScore distilled` and `CaloScore single-shot` than for `CaloFlow teacher` distilled to `CaloFlow student`.

When judged by KPD and FPD in figure 40 (see also table C2), the relative performance of the submissions is confirmed by this metric, too. We do see, however, that these metrics (especially the KPD) scatter a bit more, so the flow-based and diffusion model-based submission's scores now almost all agree with each other within the uncertainties. This larger scatter of the KPD would also result in concluding that some submissions are indistinguishable from the reference data, since the KPD is consistent with 0. This, however, cannot be confirmed by the FPD and the AUCs of figure 39, which see the best scores of `CaloDiffusion` still significantly away from the baseline scores obtained with the GEANT4 reference.

Now we move on to the multiclass classifier. The crosscheck of the well-trained classifier can be found in figure B1. Figure 41 shows the main results, the mean log-posterior for the GEANT4 test set (and the same results are reported in table C3). These results are consistent with the other classifier test, with `CaloDiffusion` and `CaloINN` in the lead. Interestingly, `CaloScore`, which was having good results in terms of the separation power of the high-level observables, was overtaken by the classifier metrics by normalizing flow-based submissions like `CaloFlow` and `CaloINN`.

Overall, in terms of shower quality of ds $1-\gamma$, we observe that some models approach the GEANT4 reference, telling us that the comparatively easy and low-dimensional distribution of photon showers can indeed be learned by DGMs. In particular, we see that diffusion model and normalizing flow-based submissions get consistently better scores than GAN and VAE-based submissions.

In figure 42 and table C4 we show precision, density, recall, and coverage of the ds $1-\gamma$ submissions. We observe different classes of results. The first one shows values for all 4 metrics that are of the same order as the scores for the GEANT4 reference, indicating a diverse and realistic dataset. `CaloDiffusion`, `CaloINN`, `CaloScore`, and `CaloGraph` fall in this class.

The second prominent pattern we observe shows values of precision and coverage that are of the same size as for GEANT4, but a much larger density and a much smaller recall. Most of the GAN and VAE-based models like `Calo-VQ`, `CaloMan`, `CaloShowerGAN`,

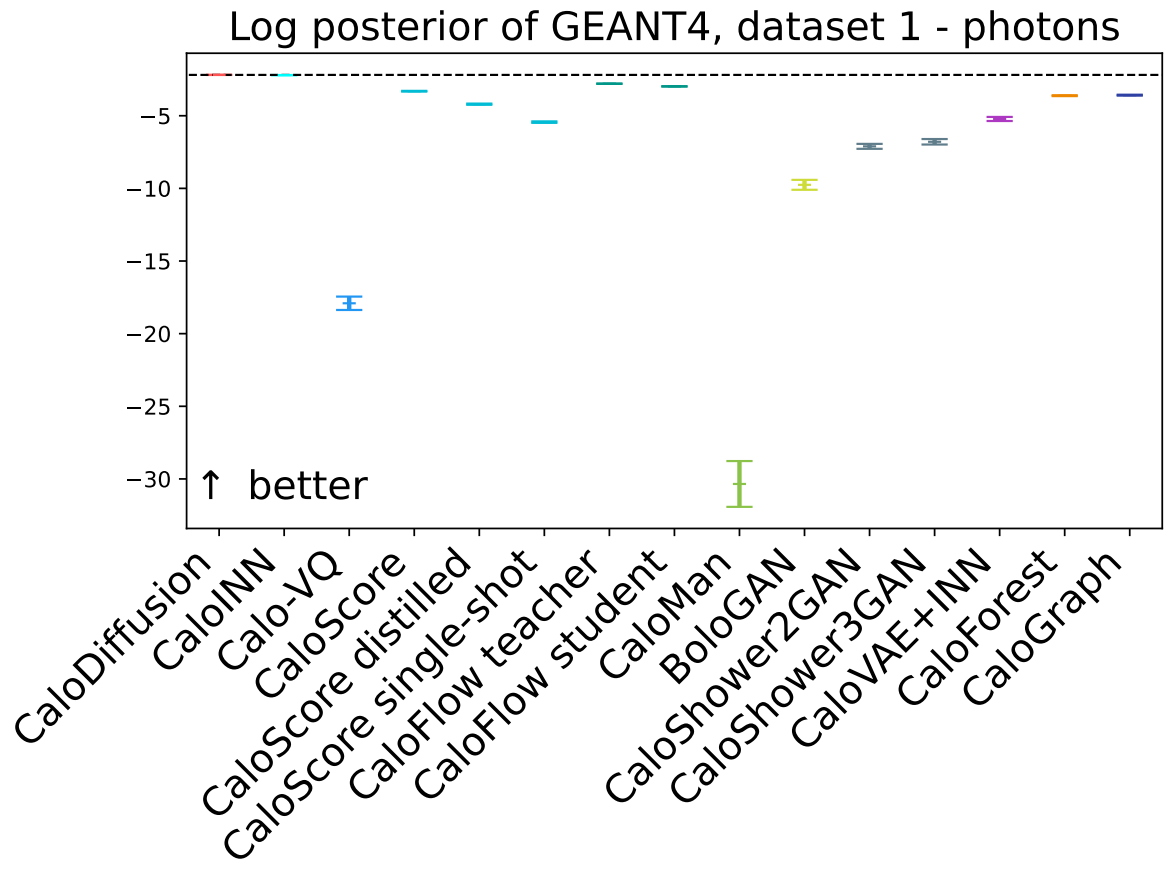



Figure 41: Log-posterior scores for ds 1 – $\gamma$ GEANT4 test data, averaged over 10 independent classifier trainings. For the precise numbers, see Table C3.

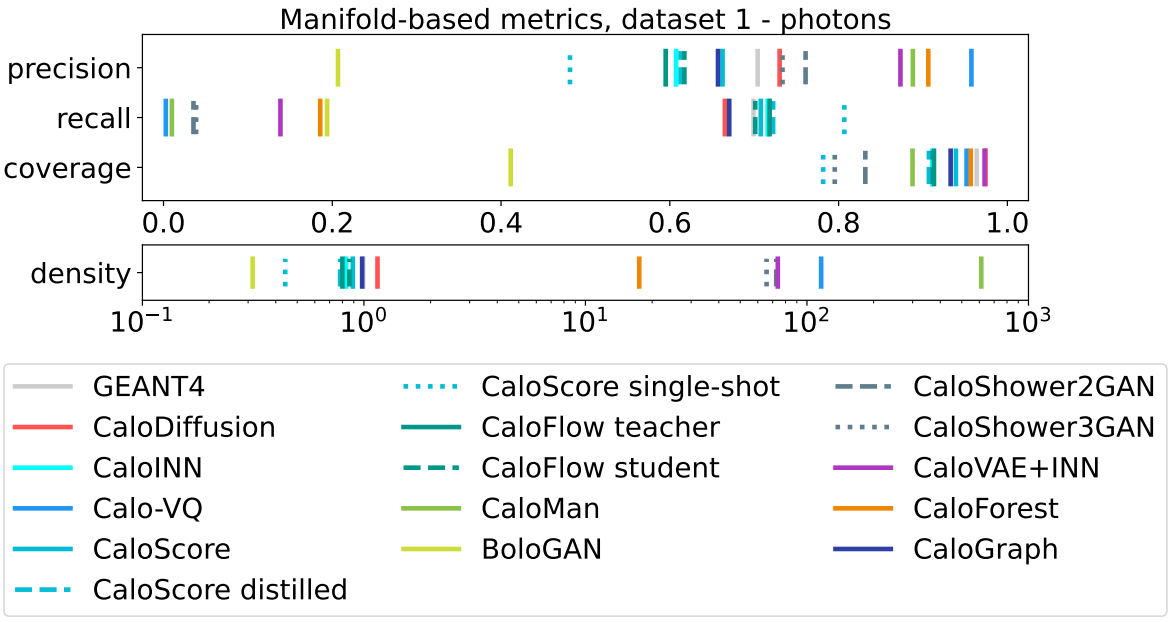



Figure 42: Precision, density, recall, and coverage for ds 1 – $\gamma$ submissions. For the precise numbers, see Table C4.

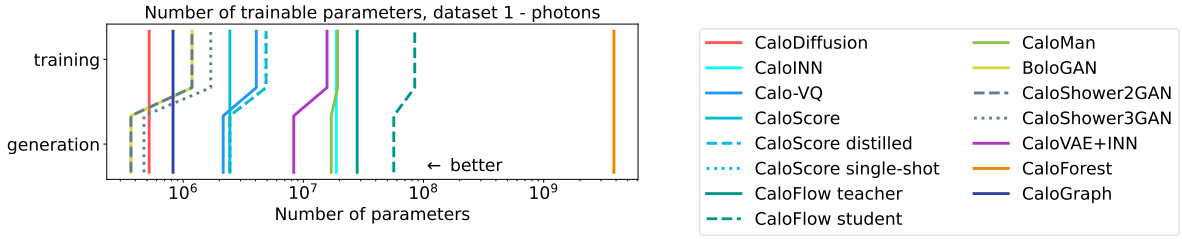



Figure 43: Number of trainable parameters for training and generation of ds $1 - \gamma$ submissions. For the precise numbers, see Table C5.

and `CaloVAE+INN`, but also `CaloForest` fall in this class. The high density suggests that the generated samples all fall close to the bulk of the reference data, but the low recall indicates that the relative distance between the generated samples is fairly small, so not many of the reference samples lie on the generated manifold. Overall, these generative models seem to focus on generating samples in the bulk that are similar to each other.

The third class have good scores for recall and coverage, but a small precision, with the density being at the order of GEANT4 or smaller. In table C4, we see `CaloScore single-shot` and `CaloFlow` in this category. They have a good distribution of samples close to the reference manifold, but a noticeable subset of them falls outside the manifold. When the density is low, it also indicates that the bulk is not as densely populated.

The last pattern we observe has all four metrics below the GEANT4 reference, as seen for `BoloGAN`.

In terms of the the requirements of resources, the situation is different. Figure 43 shows the number of trainable parameters of each submission, with the precise numbers in table C5. Normalizing Flow-based models are now at the back of the list, as they usually require larger models. GANs and VAEs are much more lightweight, as can be seen by `CaloShowerGAN` and `BoloGAN`, which need the fewest parameters. Given the rather small dimensionality of ds $1 - \gamma$, the diffusion model of `CaloDiffusion` also only needs a comparatively small number of parameters.

Which model is the fastest really depends on specific setup of the evaluation. We see the generation times per shower of the submissions in figure 44 (with details in table C6 and table C7). On the CPU in figure 44 (and table C6), we observe a reduction in generation time when moving from batch size 1 to batch size 100 for all submissions. Further increasing the batch size to 10000 does not decrease the generation time further, indicating that now the algorithms are not dominated by the `for` loop over all batches anymore. The fastest models, `BoloGAN` and `CaloVAE+INN` reach generation times of about one millisecond per shower for batch size 100, and even below for larger batch size. On the GPU in figure 44 (and table C7), generation times are usually smaller than on the CPU, but different models gained differently under the changing hardware. For batch size 100, we now have five submissions at or below one millisecond generation time. For batch size 10000, only `CaloDiffusion`, `CaloScore`, and `CaloFlow teacher` are well above the one millisecond mark. The fastest models are now GAN-

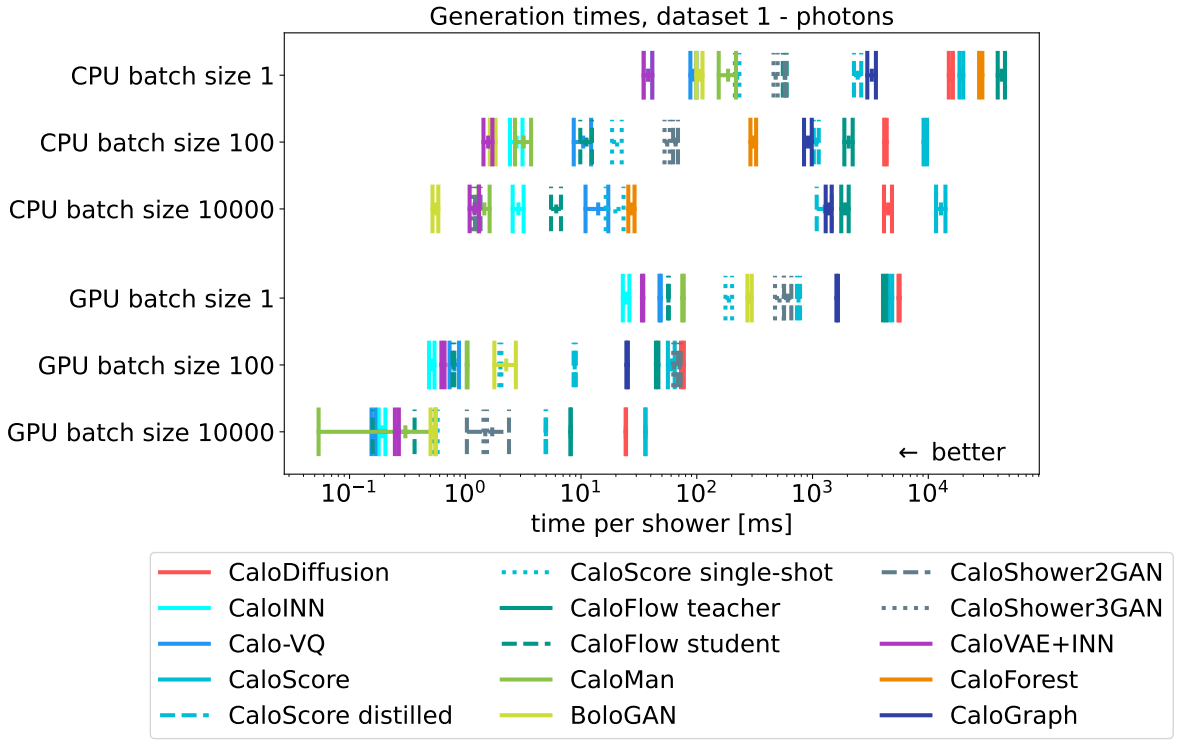



Figure 44: Timing of ds 1 – $\gamma$ submissions on CPU and GPU architectures. Not all submissions are shown everywhere due to memory and other constraints. More details are in table C6 and table C7.

based (like `BoloGAN`) or VAE-based (like `Calo-VQ`, `CaloMan`, and `CaloVAE+INN`). We now also observe improvements when increasing the batch size to 10000, even though the advantage in going from 100 to 10000 is not as big as the one going from 1 to 100. Rather surprisingly, we observe a larger generation time for the GAN-based models `CaloShower2GAN` and `CaloShower3GAN`. We suspect that this is a remnant of these being part of the larger ATLAS software pipeline that was not fully optimized for the challenge submission.

### *9.3. Dataset 1, Pions (ds 1 – $\pi^+$)*

Starting again with high-level features, we first look at the energy depositions in figure 45. The separation power of the submissions vary roughly within 2 orders of magnitude and they stay about one order of magnitude worse than the GEANT4 reference. `CaloFlow` shows the best performance overall, but occasionally another model is better in modeling a single layer. Diffusion models are not as good as for ds 1 – $\gamma$, now VAE-based models like `DNNCaloSim`, `Calo-VQ`, or `CaloMan` are better, especially for earlier layers. Again, many models show better performance in layers 12 and 13, which have a higher segmentation in angular direction.

The centers of energy, shown in figure 46, show a consistent picture in both

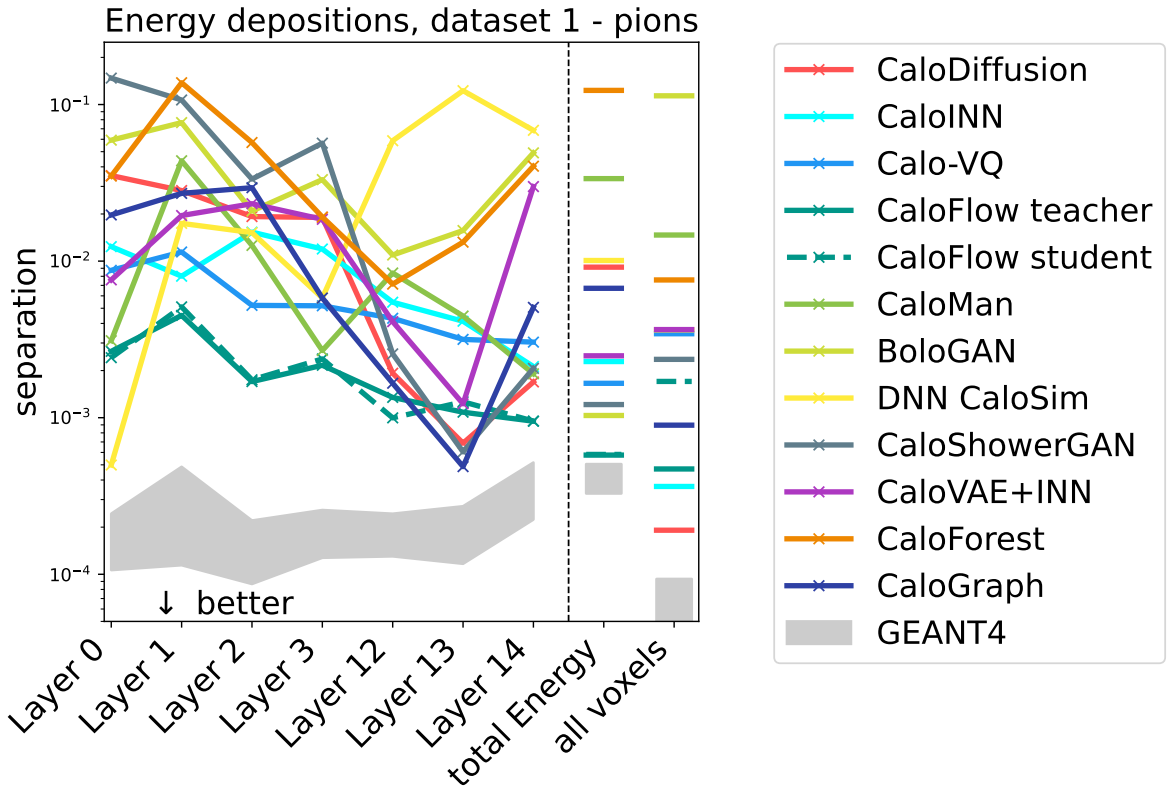



Figure 45: Separation power of energy depositions with threshold at 1 MeV.

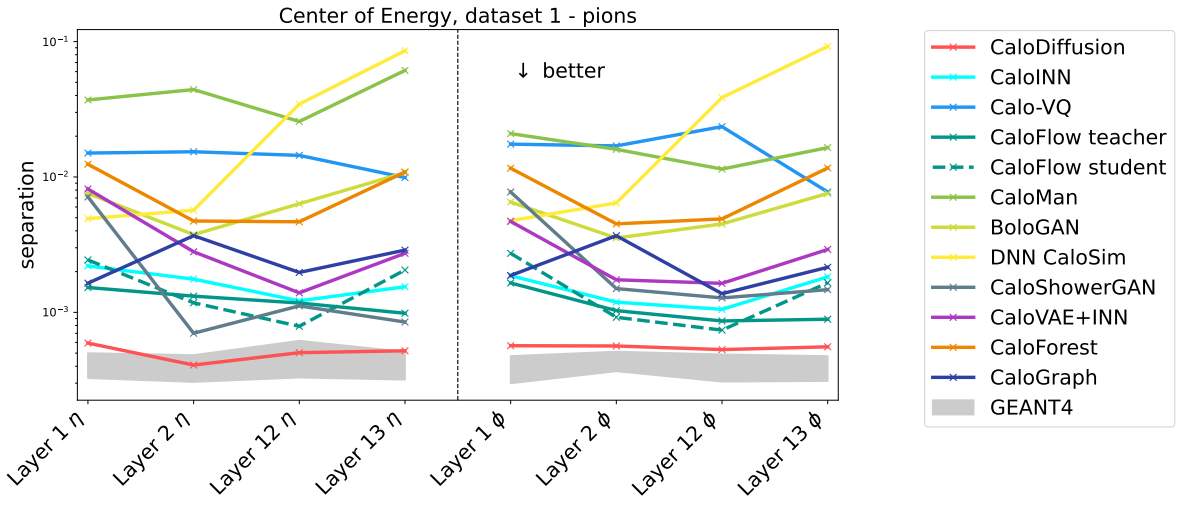



Figure 46: Separation power of centers of energy with threshold at 1 MeV.

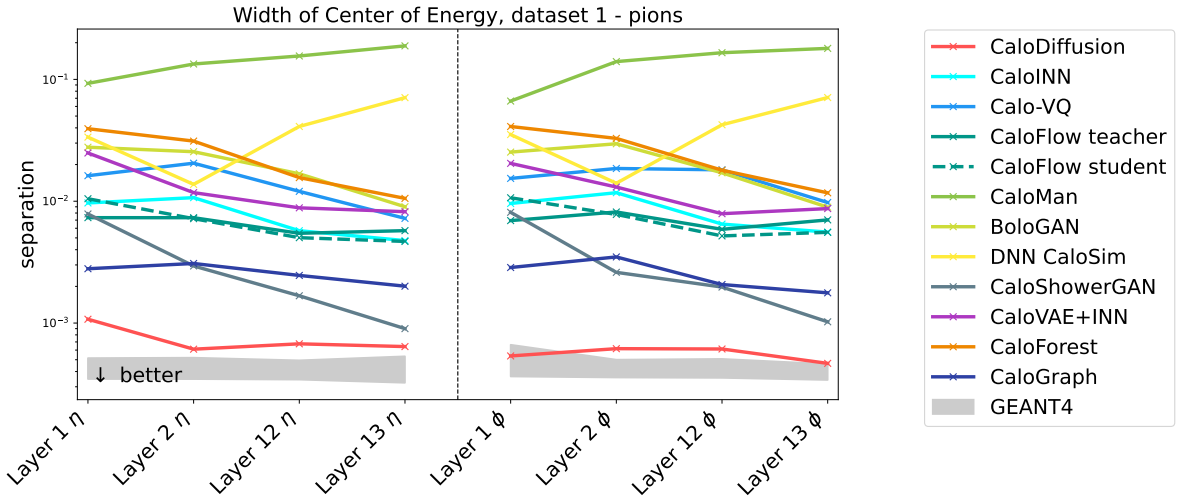



Figure 47: Separation power of widths of centers of energy with threshold at 1 MeV.

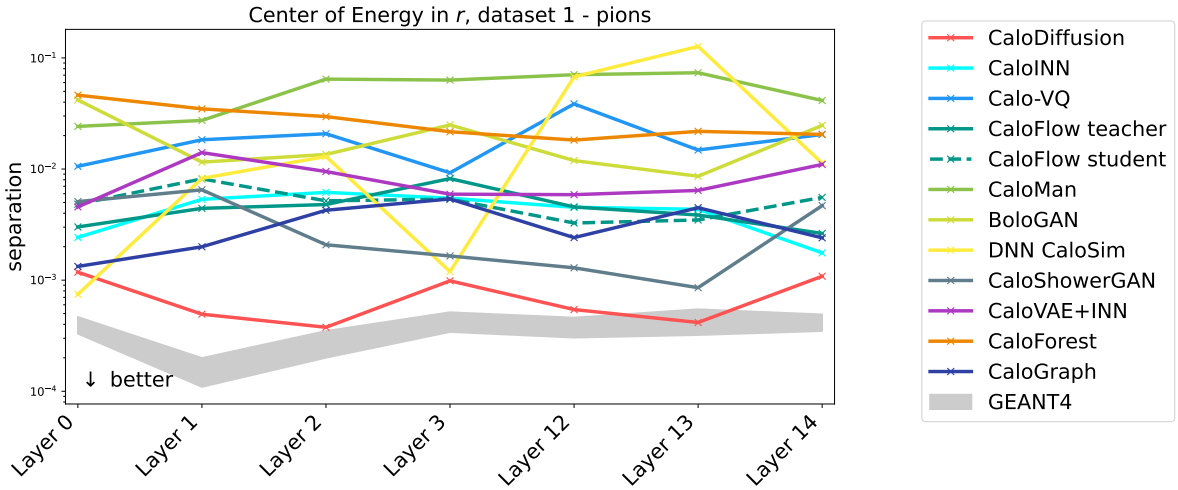



Figure 48: Separation power of centers of energy with threshold at 1 MeV.

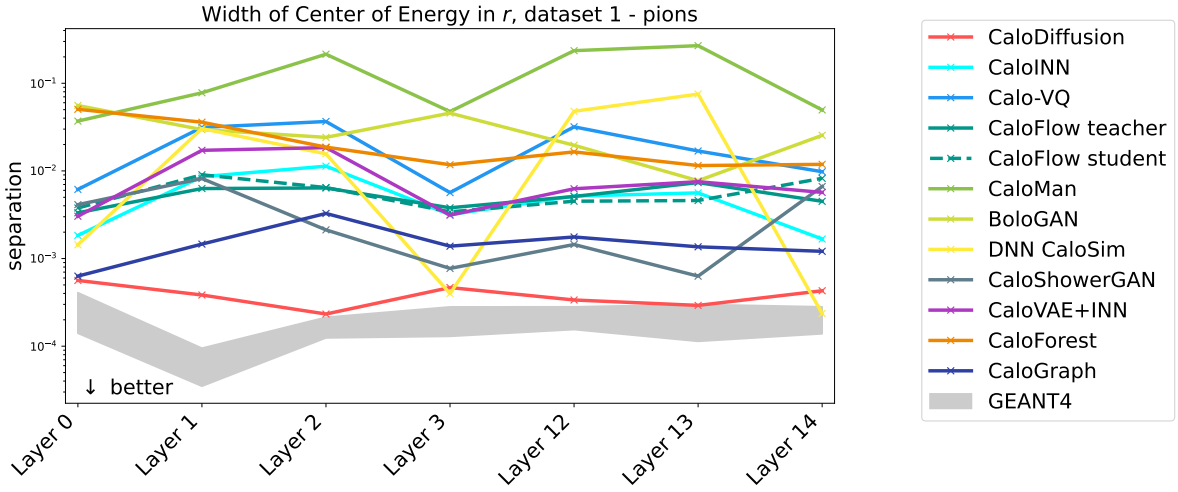



Figure 49: Separation power of widths of centers of energy with threshold at 1 MeV.

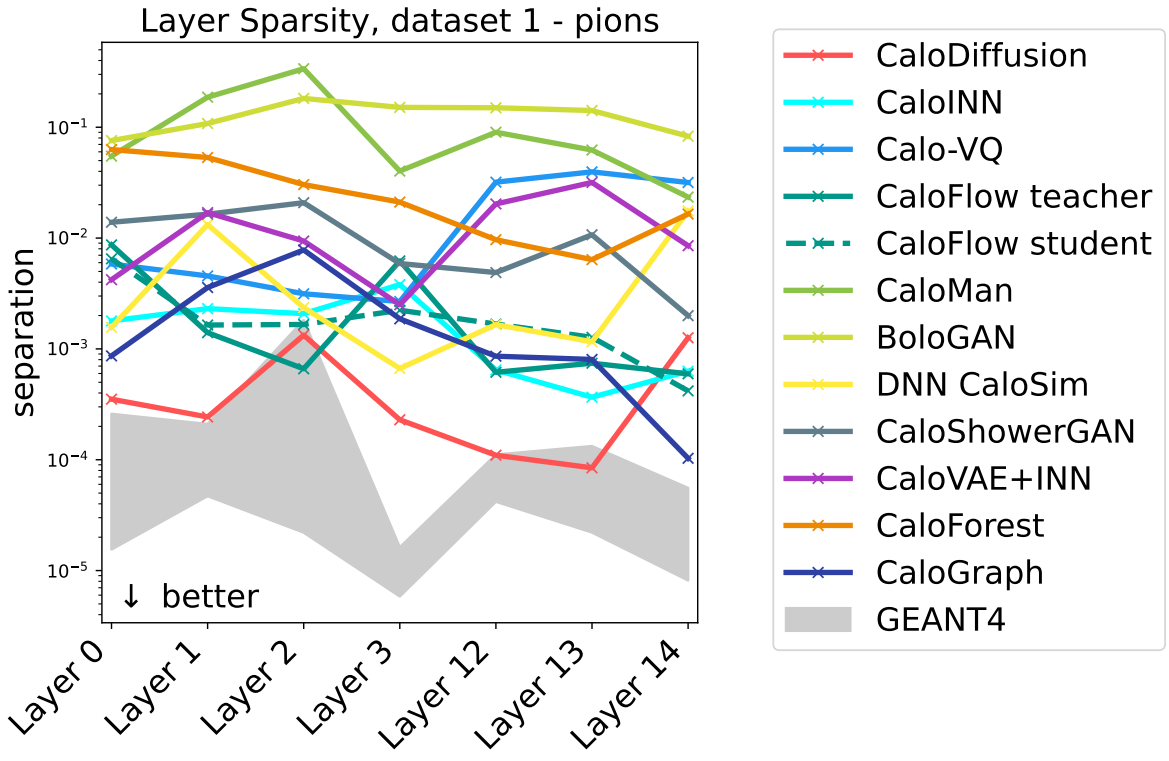



Figure 50: Separation power of the sparsity with threshold at 1 MeV.

directions $\eta$ and $\phi$. The separation power again spans about two orders of magnitude with `CaloDiffusion` just at the GEANT4 reference, followed by `CaloFlow`, `CaloINN`, and `CaloShowerGAN`. Interestingly, `DNNCaloSim` shows larger separation powers even though its performance in other metrics indicates otherwise, as we will see below.

The widths of these center of energy distributions are compared to each other in figure 47. We again observe a very good performance of `CaloDiffusion`, but now `CaloGraph` and `CaloShowerGAN` come in second before the flow-based models.

When turning to the radial direction, the centers of energy in figure 48 and its width in figure 49 show again results consistent with the evaluation along $\eta$ and $\phi$: `CaloDiffusion` with the smallest separation powers, followed by `CaloGraph` and `CaloShowerGAN`. While most submissions show separation powers of the same size for each layer, `DNNCaloSim` does a lot better in layers 0, 3, and 14 than in layers 1, 2, 12, and 13.

For the sparsities in figure 50, we see a lot more variation from layer to layer in each of the submission. Even the separation power of the GEANT4 reference varies almost two orders of magnitude between layers 2 and 3. The best performing submission is still `CaloDiffusion`, but the gap to the other submissions is smaller.

Figure 51 shows the correlation in layer energies for the submissions. The submissions `CaloDiffusion`, `CaloFlow`, `CaloMan`, and `DNNCaloSim` reproduce the pattern of GEANT4 well. Other submissions, such as `CaloINN`, `Calo-VQ`, `BoloGAN`,

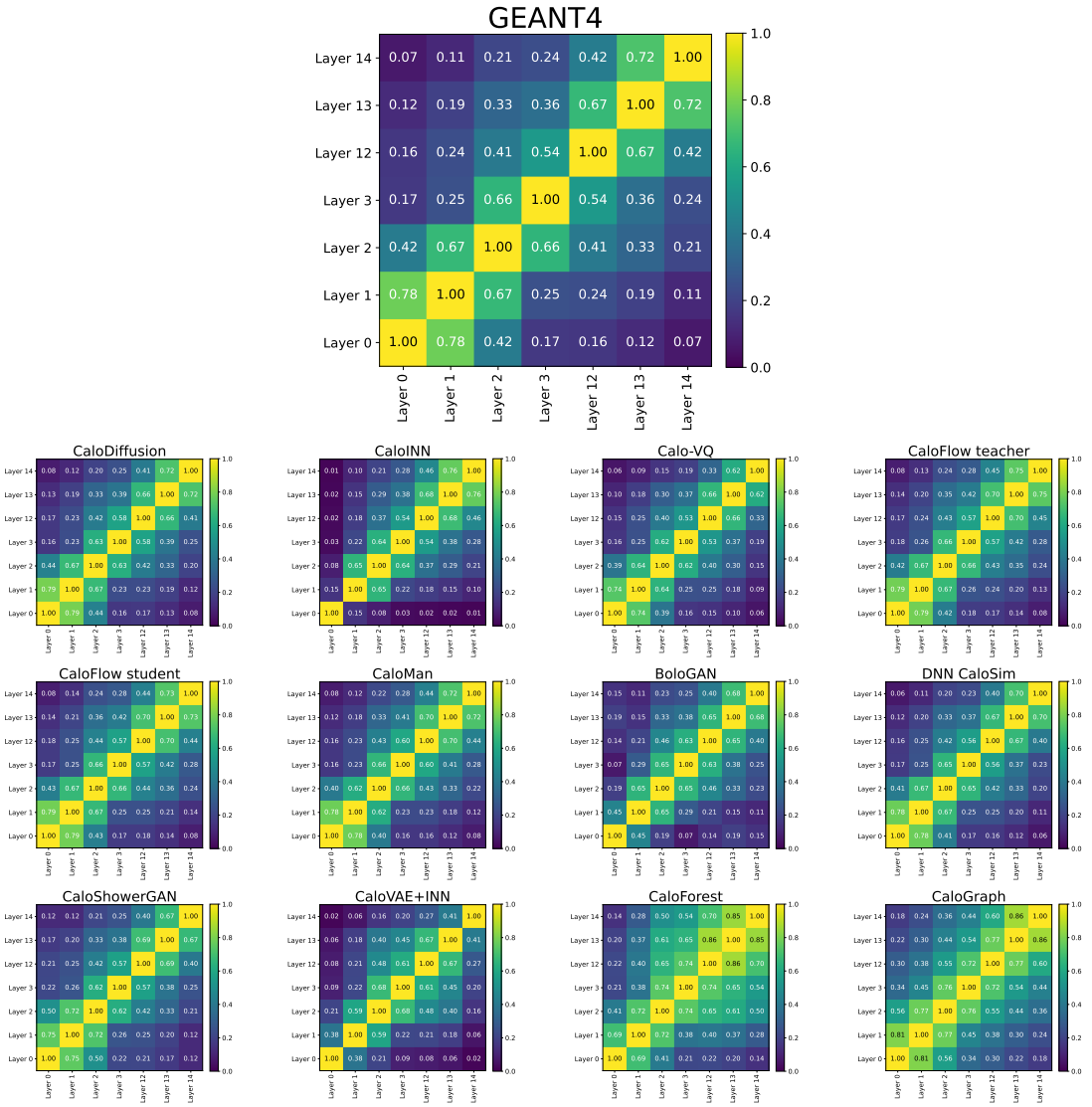



Figure 51: Pearson correlation coefficients of layer energies in ds $1 - \pi^+$, with threshold at 1 MeV.

`CaloVAE+INN`, and `CaloShowerGAN`, have some problems with correlations of layers 0 and/or 14, which are the first and last layers. `CaloForest` finds a smaller correlation between the first layers and a too large correlation for the rest while `CaloGraph` has too large correlations everywhere.

Moving on to classifier-based metrics, we find the AUCs of high- and low-level observables in figure 52 (and table C8). Here we observe several things. First, the AUC for separating the training and test GEANT4 samples is larger than the expected value of 0.5. This is due to the fact that two slightly different versions of the ATLAS software were used due to technical problems in generating high statistics with the old version used for the ATLAS internal training. The differences were expected and deemed small enough to be irrelevant for physics applications. Detailed comparison between the two

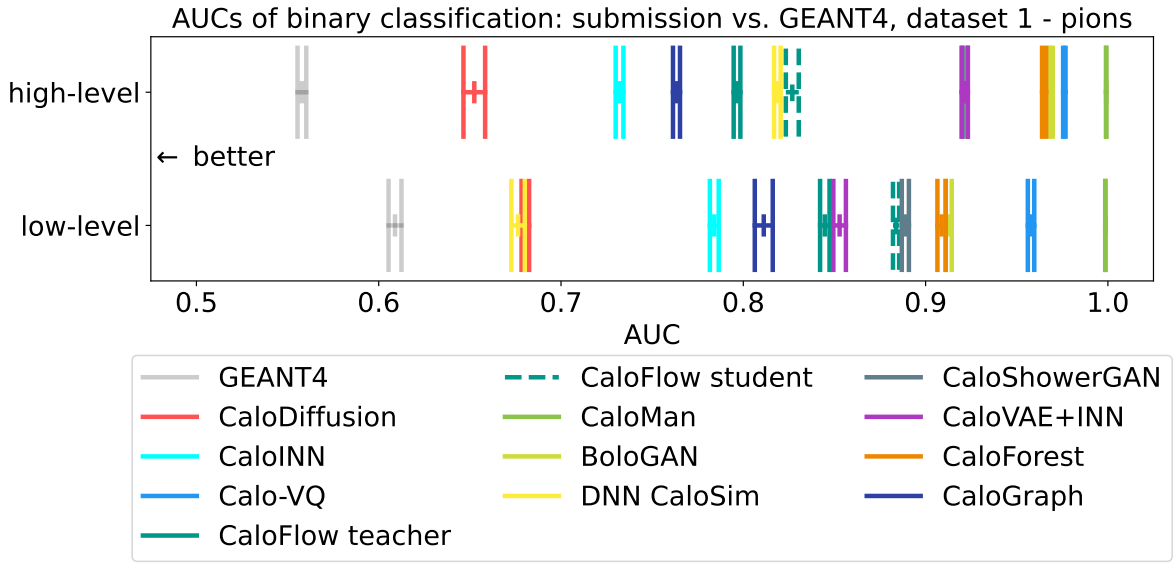



Figure 52: Low-level and high-level AUCs for evaluating GEANT4 vs. submission of ds $1 - \pi^+$, averaged over 10 independent evaluation runs. For the precise numbers, see Table C8.

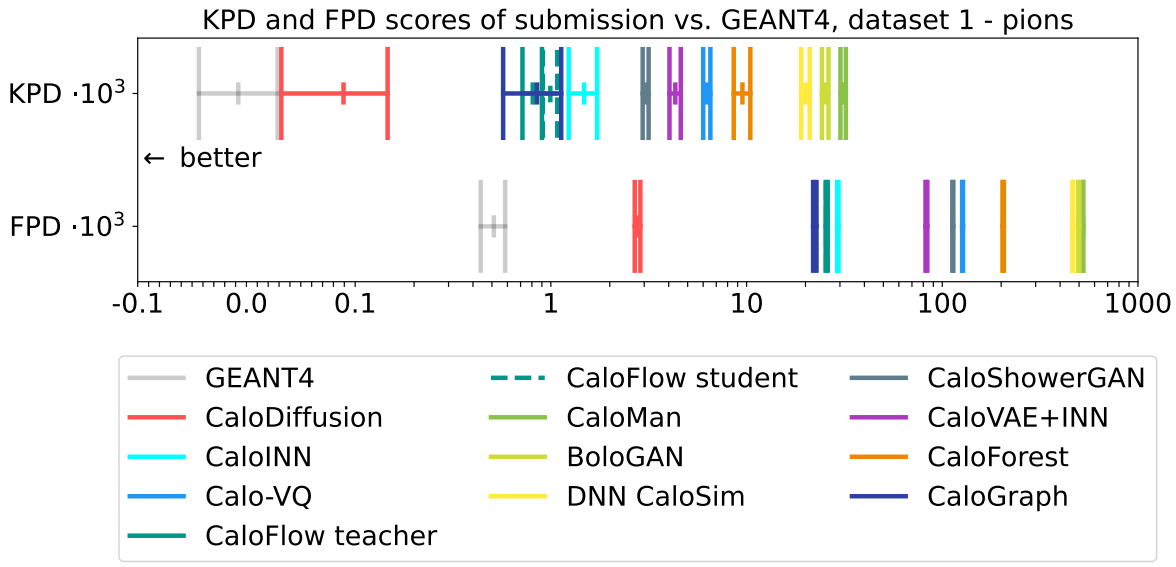



Figure 53: KPD and FPD for evaluating GEANT4 vs. submission of ds $1 - \pi^+$. For the precise numbers, see Table C9.

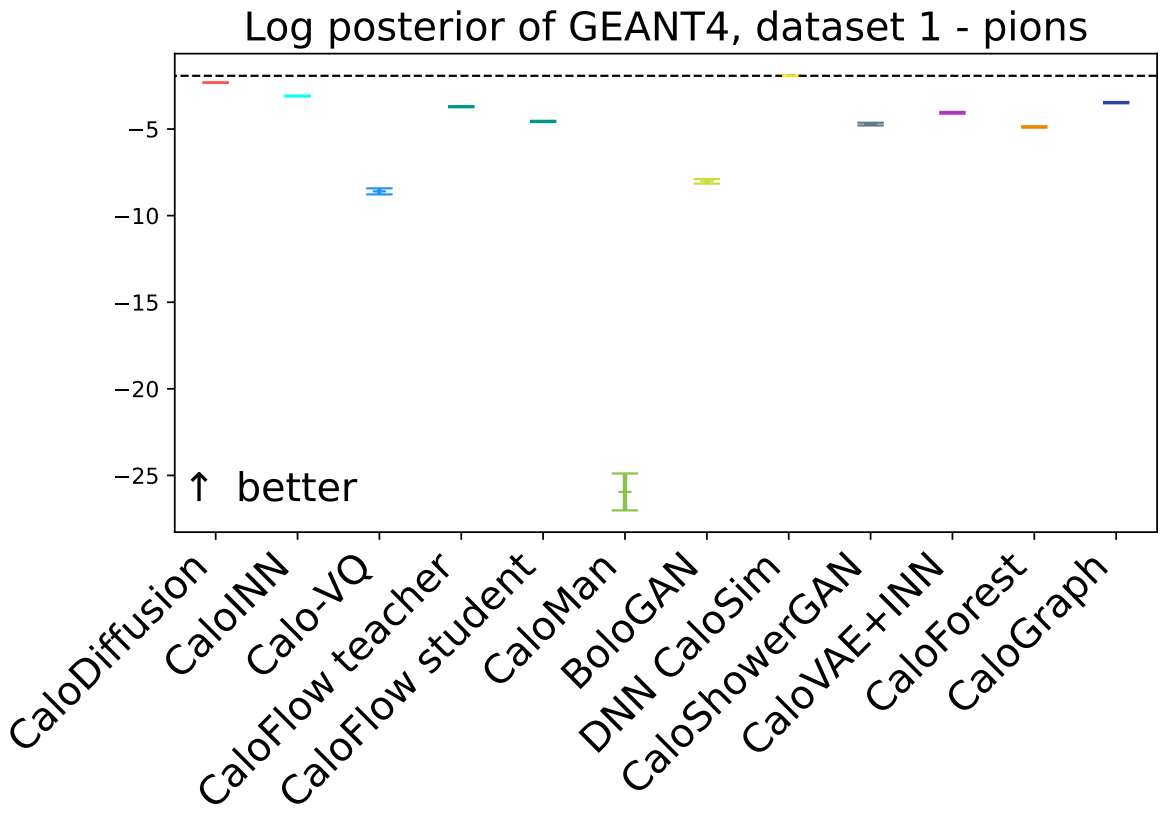



Figure 54: Log-posterior scores for ds $1 - \pi^+$ GEANT4 test data, averaged over 10 independent classifier trainings. For the precise numbers, see Table C10.

samples that justify this statement are provided in A.2. The AUC from the generative models will have this value as the maximum achievable separation instead of the usual 0.5. Second, we see very low AUCs for `CaloDiffusion`, which was already indicated by the separation powers of the obervables before. Third, we see a low AUC for `DNNCaloSim` in the low-level observables which is, however, not reflected in the AUC of the high-level observables. This latter fact also correlates with the separation powers seen before. Other than that, we see overall good scores from diffusion and normalizing flow-based models, whereas GAN and VAE-based models show AUCs worse than 0.9.

The same is true for KPD and FPD metrics shown in figure 53 (and also in table C9). The best scores are attained for `CaloDiffusion`, followed by `CaloGraph` and `CaloFlow`. The submission of `DNNCaloSim` is not among the top contestants here.

When looking at the results of the multiclass classification in figure 54, the situation is slightly different. `CaloDiffusion`, `CaloINN`, and `CaloGraph` show again good scores, but `DNNCaloSim` is outperforming them. Since the multiclass classification is also based on low-level observables, this observation confirms the low-level AUC of table C8. The consistency check of the multiclass classifier can be seen at figure B2.

In figure 55 and table C11 we show precision, density, recall, and coverage of the ds $1 - \pi^+$ submissions. We again observe similar patterns as in the ds $1 - \gamma$

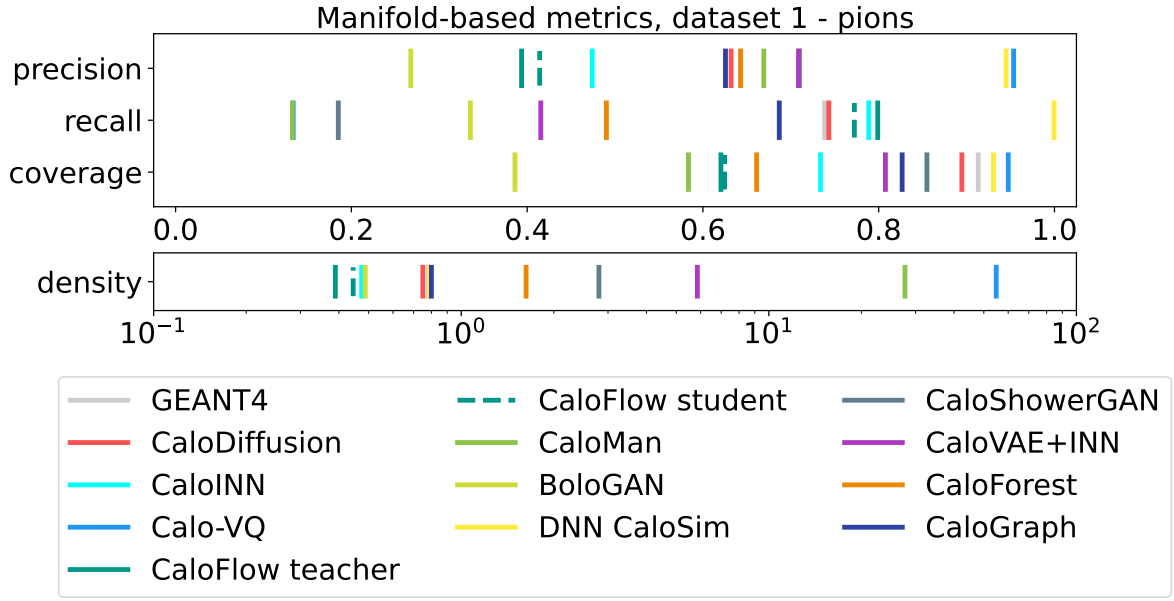



Figure 55: Precision, density, recall, and coverage for ds $1 - \pi^+$ submissions. For the precise numbers, see Table C11.

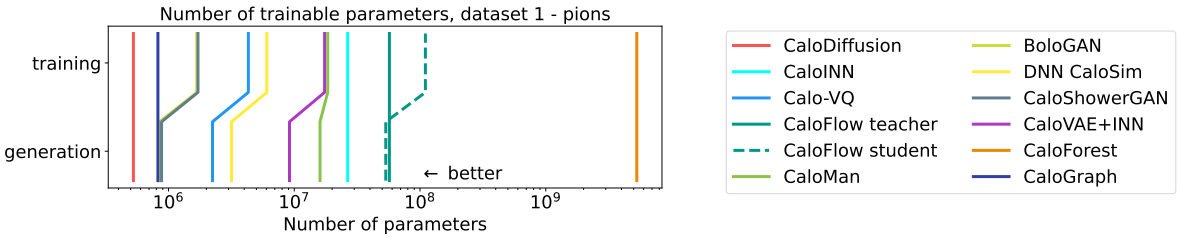



Figure 56: Number of trainable parameters for training and generation of ds $1 - \pi^+$ submissions. For the precise numbers, see Table C12.

case. The submissions `CaloDiffusion`, `DNNCaloSim`, and `CaloGraph` have their scores around the scores of the GEANT4 reference, indicating a good fit to the underlying distribution. The normalizing flow-based submissions `CaloINN` and `CaloFlow` have good recall and coverage, but a relatively small precision and density, indicating that a large enough subset of samples were generated away from the reference manifold. VAE-based submissions `Calo-VQ`, `CaloMan`, `CaloVAE+INN`, and to some extend also `CaloShowerGAN` show a large density paired with a very small recall. As for ds $1 - \gamma$, we interpret this as generative models that seem to focus on generating samples in the bulk of the data, with all samples being fairly similar to each other.

Figure 56 compares the sizes of the submissions, with table C12 giving the precise numbers. Most models require (at least in training) more than $10^6$ trainable parameters, only `CaloDiffusion` and `CaloGraph` stay below that. Overall, we observe normalizing-flow based models to be much larger than diffusion and GAN-based models. The BDT-based `CaloForest` stands out because of the many parameters that are required to

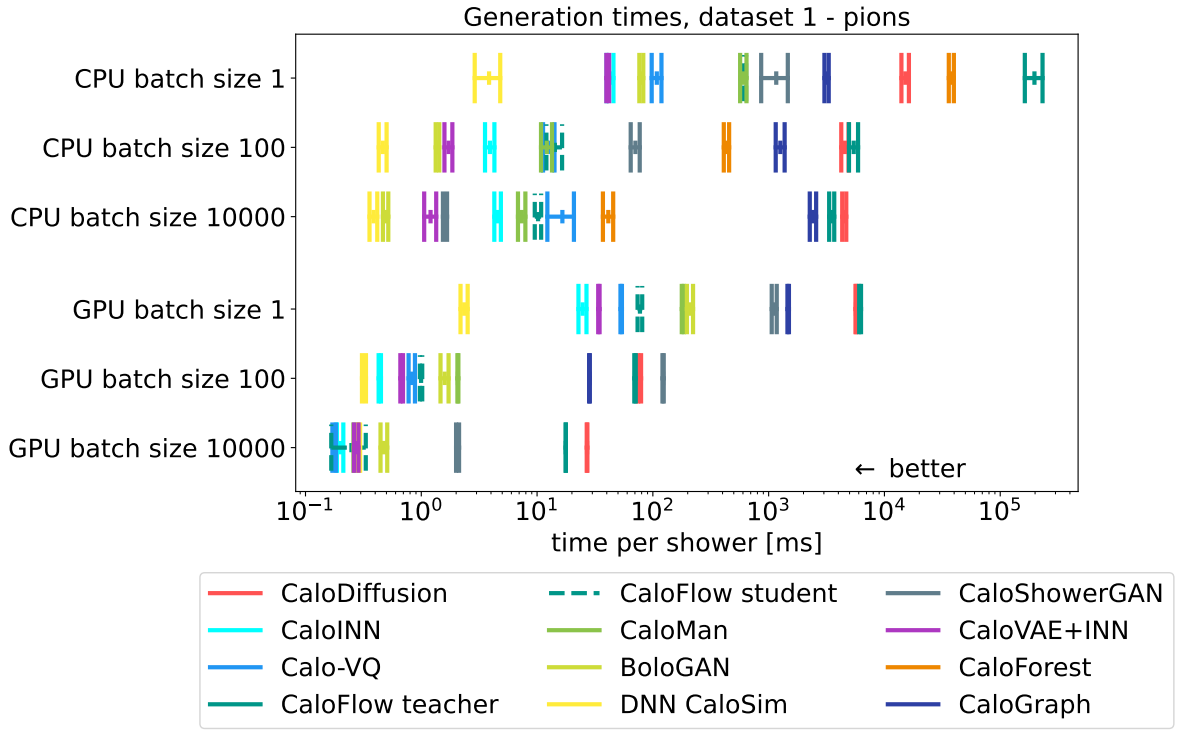



Figure 57: Timing of ds 1 – $\pi^+$ submissions on CPU and GPU architectures. Not all submissions are shown everywhere due to memory and other constraints. More details are in table C13 and table C14.

define all trees.

Figure 57 (with details in table C13 and table C14) show the generation times per particle shower of the submissions. Across all batch sizes and architectures, we see `DNNCaloSim` as being the fastest model, only beaten for very large batch sizes on a GPU but not by a large margin. This model only needs a few milliseconds (for batch size 1) to a fraction of a millisecond (for lager batch sizes) to generate a single shower. Other GAN-based and VAE-based models like `BoloGAN` and `CaloVAE+INN` also show fast shower generation times. Normalizing-flow-based submissions, however, show a strong dependence on the implemented algorithm. The coupling-layer based implementation of `CaloINN` is much faster than the MAF/IAF-based implementations of `CaloFlow`, with the MAF being much slower than the IAF, as expected [117]. `CaloForest` does not have timings on a GPU, as the tree-based algorithm only runs on a CPU. Also here, we observe a larger generation time for the GAN-based model `CaloShowerGAN`. Again, we suspect that this is a remnant of `CaloShowerGAN` being part of the larger ATLAS software pipeline that was not fully optimized for the challenge submission.

### *9.4. Dataset 2, Electrons (ds 2)*

As explained in section 2.2, the minimal energy that can be read out is given by 15.15 keV and we apply a threshold cut to all submissions before evaluation.

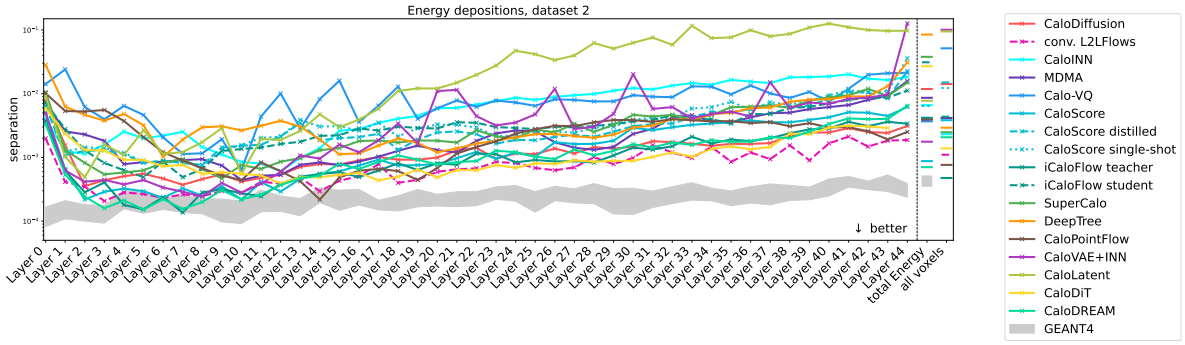



Figure 58: Separation power of energy depositions.

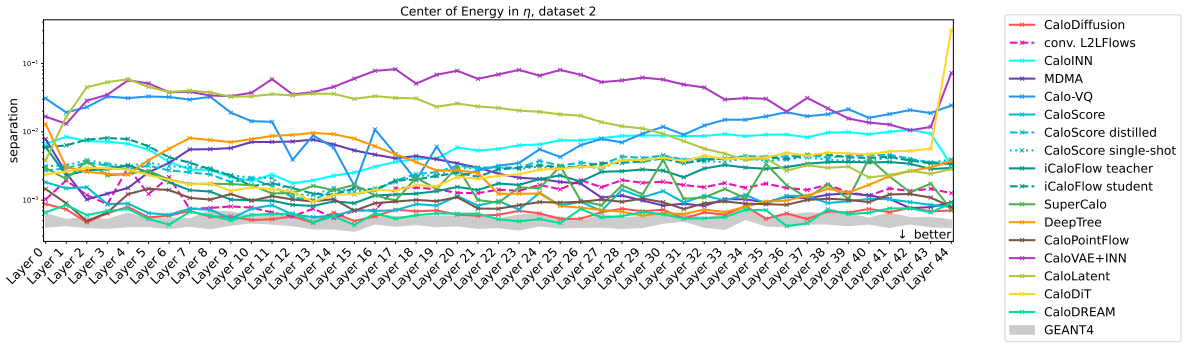



Figure 59: Separation power of centers of energy in $\eta$ direction.

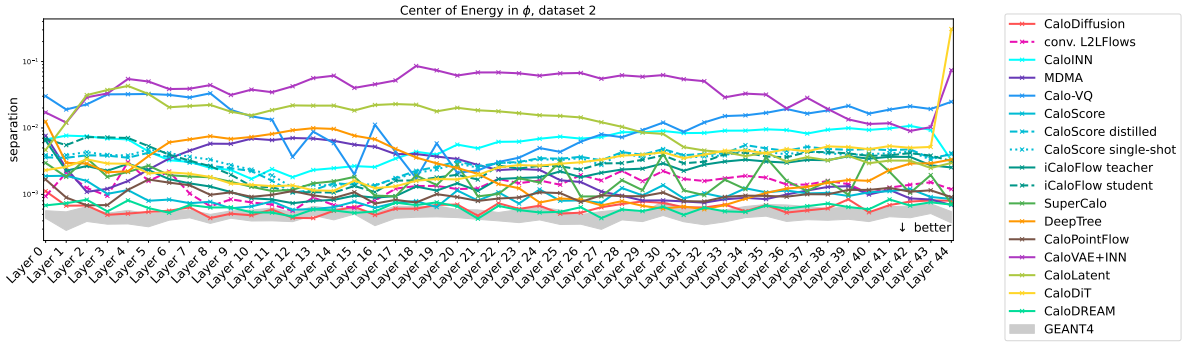



Figure 60: Separation power of centers of energy in $\phi$ direction.

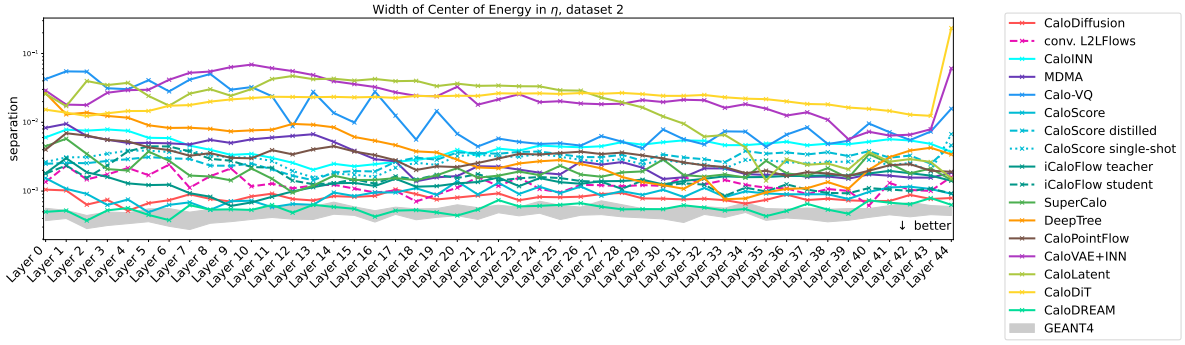



Figure 61: Separation power of widths of centers of energy in $\eta$ direction.

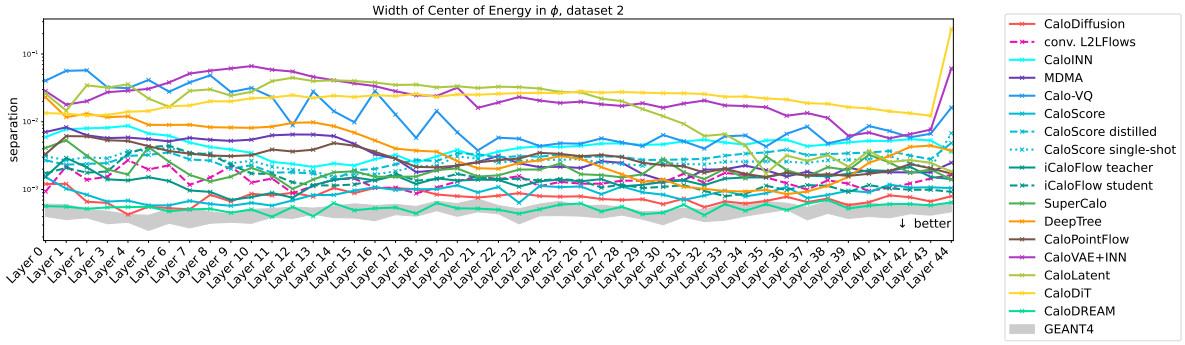



Figure 62: Separation power of widths of centers of energy in $\phi$ direction.

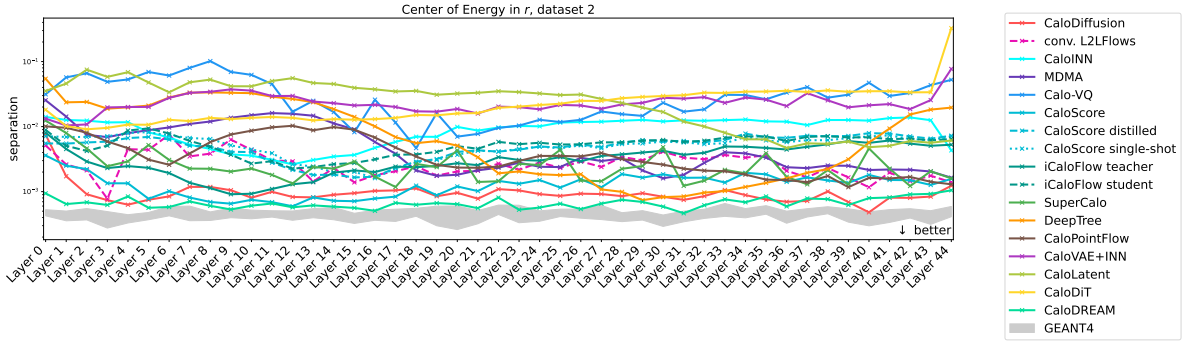



Figure 63: Separation power of centers of energy in radial direction.

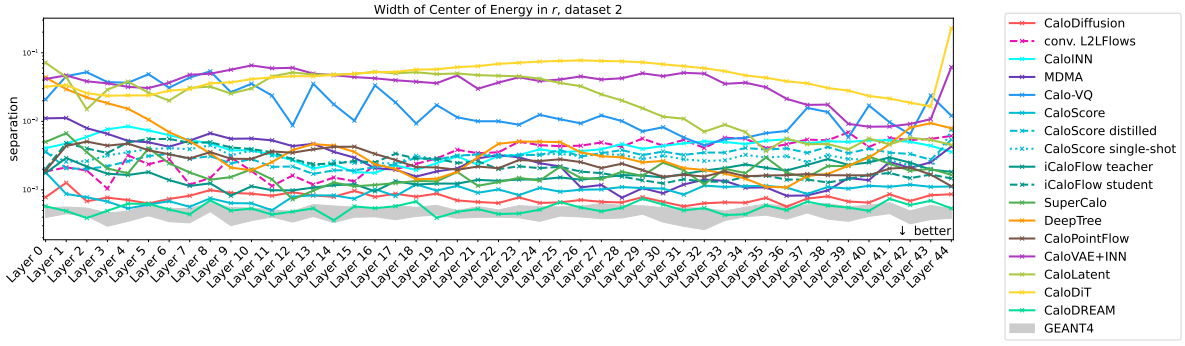



Figure 64: Separation power of widths of centers of energy in radial direction.

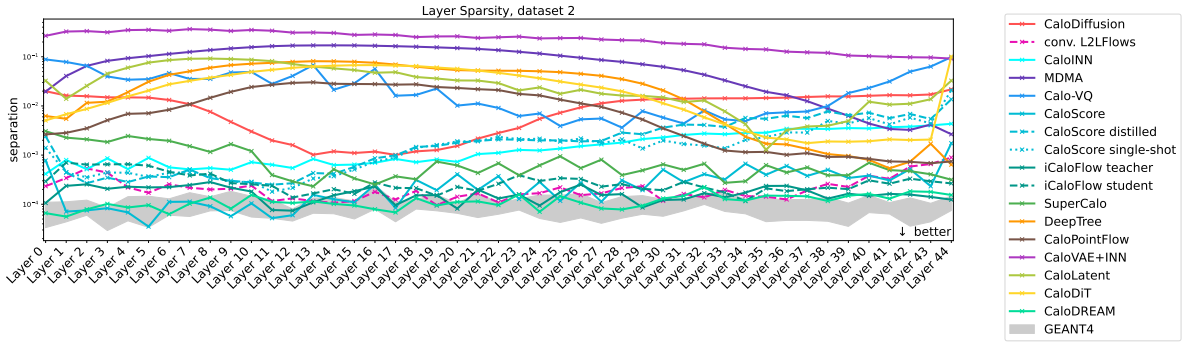



Figure 65: Separation power of the sparsity.

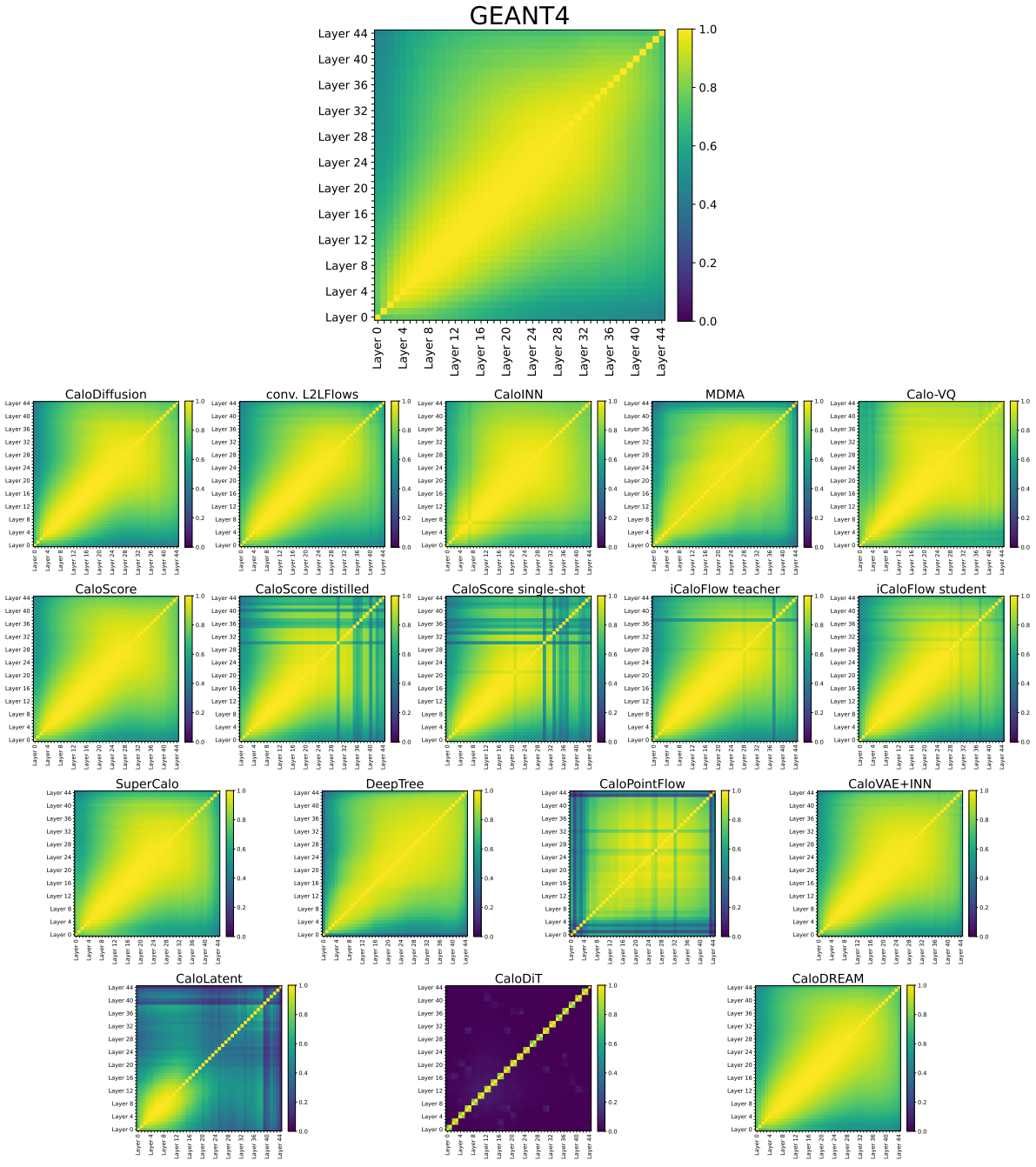



Figure 66: Pearson correlation coefficients of layer energies in ds 2.

We again start the evaluation with the separation power of the energy depositions in all layers, all voxels, and the total deposited energy in figure 58. The values for the submissions span roughly 2 orders of magnitude and only for early layers they reach down to the reference values given by GEANT4. In general, we observe all submissions getting worse towards the end of the detector, *i.e.* for a larger layer number. While some submissions show a smooth change of separation powers from layer to layer, some others oscillate with a period of a few layers.

The centers of energy in $\eta$ and $\phi$ are shown in figure 59 and figure 60. There is a rotational symmetry in the data, so the distributions in $\eta$ and $\phi$ look very similar to each other (see figure A10 and figure A12). Judging by the separation powers, the models learn the distributions in these two variables equally well, reflecting this symmetry. In detail, we see `CaloDiffusion` and `CaloDREAM` having the smallest separation powers, just at the upper bound of the GEANT4 reference band. VAE-based submissions like `CaloVAE+INN`, `CaloMan`, or `Calo-VQ` again have the the largest separation powers. Looking at the change from layer to layer, we now see a different pattern compared to the energy distributions in figure 58. Now, only some submissions show an increasing separation power for an increasing layer number. Others are either rather constant or have a large separation power for small layer numbers, show better results in the central part of the detector and then increase again towards the end. We also notice some models having a rather steep increase only in the last layer.

The separation powers of the widths of centers of energy in $\eta$ (figure 61) and $\phi$ (figure 62) are very similar to the separation powers of the centers of energies themselves. Both directions, $\eta$ and $\phi$, show almost identical results. Now `CaloDREAM` is having the best score, at the level of the GEANT4 reference band. Other submissions show again their best performance in the central region of the calorimeter segment, before the separation power rises again at larger layer numbers.

Given the rotational symmetry in $\eta$ and $\phi$, the separation powers in centers of energy and its width in radial direction resemble the ones in $\eta$ and $\phi$ strongly, as can be seen in figure 63 and figure 64.

The last set of separation powers we look at are from the sparsities in figure 65. Here, the spread between different models is larger, ranging more than three orders of magnitude. Interestingly, `CaloDREAM` still shows very good results, at the level of the GEANT4 reference band. `CaloDiffusion` on the other hand does not reproduce the sparsities well, with `CaloScore`, `SuperCalo`, `CaloINN`, and `iCaloFlow` outperforming it in all layers.

We show the Pearson correlation coefficients in layer energies in figure 66. Interestingly, we do not reproduce all findings of [20], which trained a few models from scratch, indicating that some of the observed patterns fluctuate from training to training. In general, we observe two different failure modes in these figures: One group (most prominently `CaloLatent` and `CaloDiT`) do not reproduce the correlations in a large region. A second group (consisting of `CaloScore distilled`, `iCaloFlow`, and `CaloPointFlow`) show problems in single layers, indicated by streaks in figure 66. Also, the distillation procedure worsened the pattern for `CaloScore single-shot`, but slightly improved it for `iCaloFlow student`. `CaloDiT` shows only correlation to one of the nearest neighbor layers, nothing beyond that, which is consistent with the larger separation powers we saw before.

We now turn to classifier-based metrics and start with the AUC of the binary classifiers in figure 67 (and table C15). In addition to the DNN architecture that we also used for dataset 1, we now have an additional, CNN-ResNet-based architecture

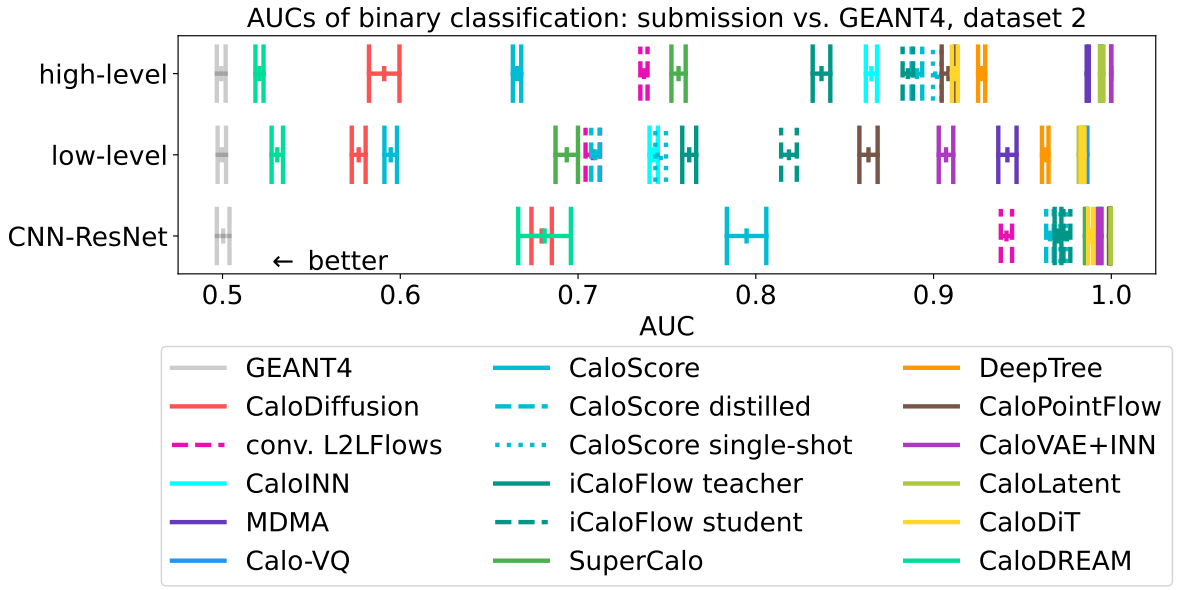



Figure 67: Low-level and high-level AUCs for evaluating GEANT4 vs. submission of ds 2, averaged over 10 independent evaluation runs. For the precise numbers, see Table C15.

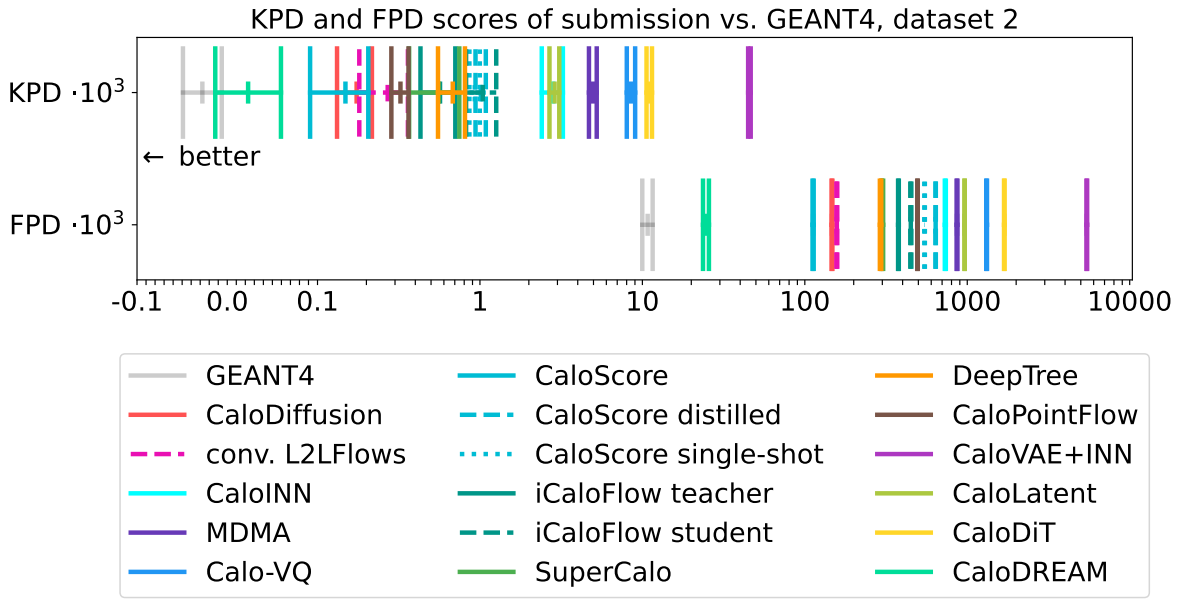



Figure 68: KPD and FPD for evaluating GEANT4 vs. submission of ds 2. For the precise numbers, see Table C16.

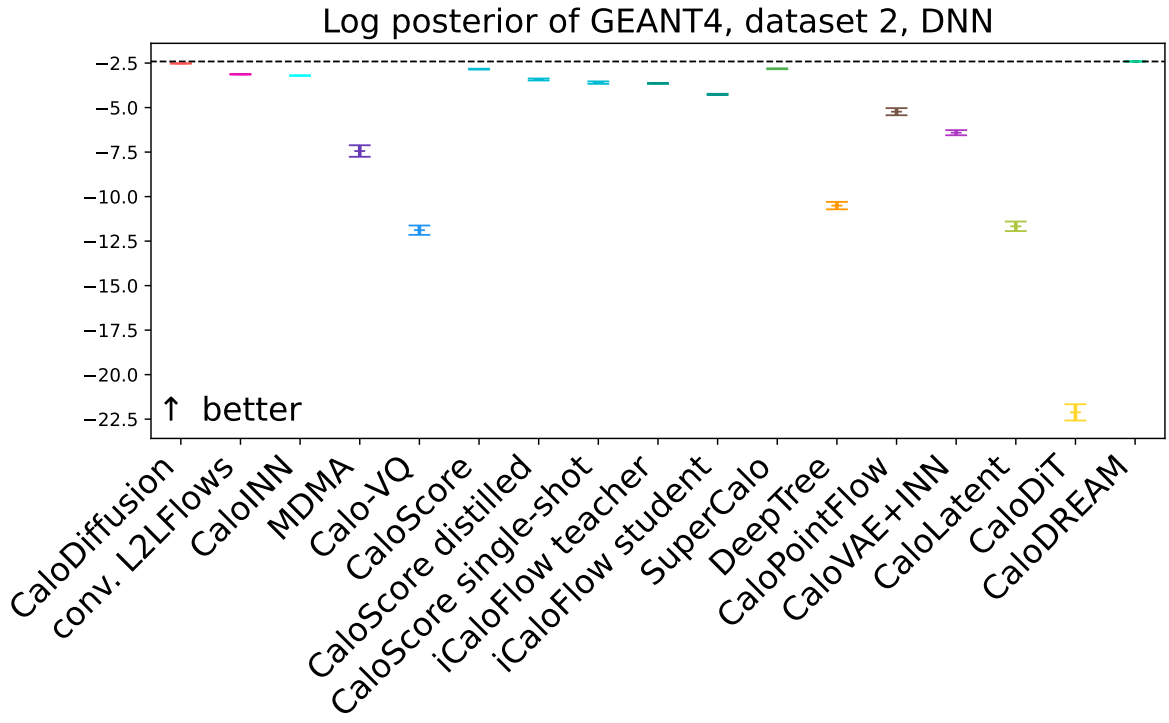



Figure 69: Log-posterior scores for ds 2 GEANT4 test data, averaged over 10 independent DNN classifier trainings. For the precise numbers, see Table C17.

that we use for the evaluation. This CNN-ResNet architecture is much more sensitive to differences in the distributions and it moves the AUC of many submissions close to 1.0. While `CaloDREAM` has the best scores in the DNN-based classifiers, it is tied with `CaloDiffusion` in the stronger CNN-ResNet classifier. However, as before, the submissions `CaloDREAM`, `CaloDiffusion`, and `CaloScore` show in general the best (lowest) binary AUC scores, independent of the classifier architecture used. Flow-based models follow, while VAE and GAN-based submissions have the highest AUC.

The computer science-inspired metrics KPD and FPD in figure 68 (with details in table C16) show results consistent with the classifier AUCs. `CaloDREAM`, `CaloDiffusion`, and `CaloScore` again have the best (lowest) scores, but now `CaloScore` is slightly better than `CaloDiffusion`, which is in fact overlapping with `conv. L2LFlows` now. At the other end of the spectrum we again see submissions based on GANs and VAEs.

For the multiclass classification we also employ a DNN and a CNN-ResNet architecture. Both of these have well-trained classifiers, as can be seen in figure B3 and figure B4. In figure 69 (as well as table C17), we see the results for the DNN architecture. `CaloDREAM` is again leading with `CaloDiffusion` at a very close second place. `CaloScore` with its distilled versions and the flow-based submissions of `SuperCalo`, `conv. L2LFlows`, `CaloINN`, and `iCaloFlow` follow with very small differences. Distilled submissions of `CaloScore` and `iCaloFlow` perform in general slightly worse than the original versions that they have been distilled from. Turning to

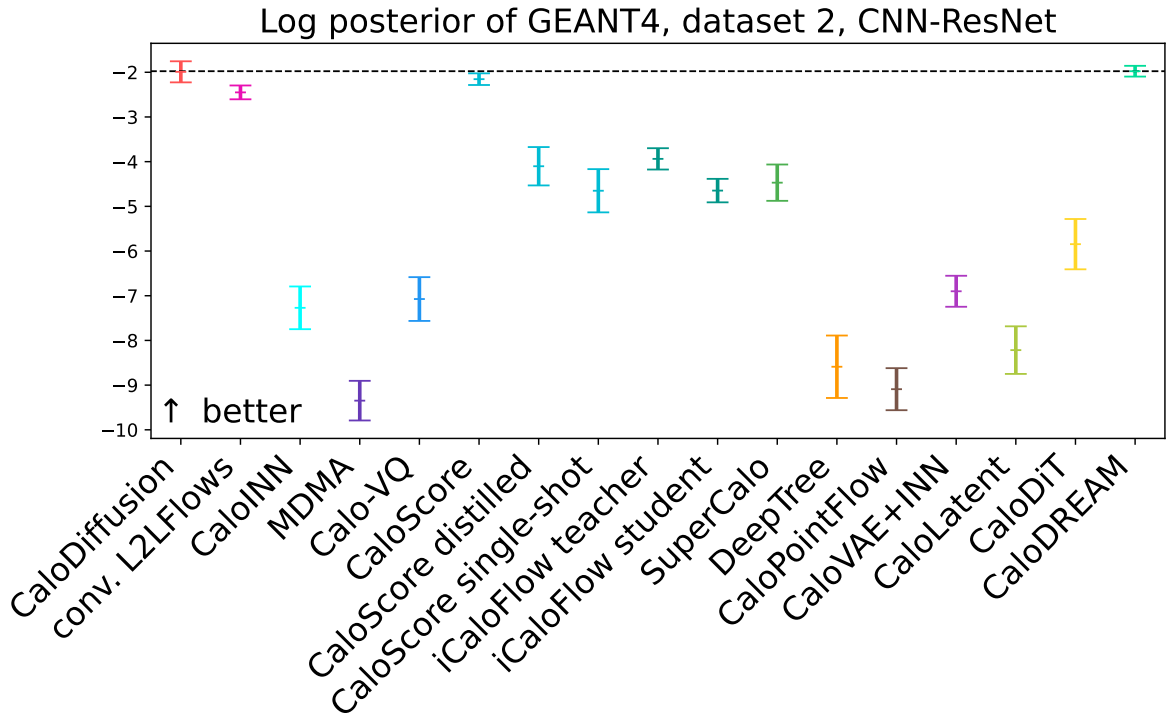



Figure 70: Log-posterior scores for ds 2 GEANT4 test data, averaged over 10 independent CNN ResNet classifier trainings. For the precise numbers, see Table C18.

the CNN-ResNet architecture in figure 70 (and table C18), the story is roughly the same as for the DNN before. Overall, we observe the errorbars becoming larger, indicating a larger spread of result in different trainings. However, the spread in the log posterior from the best to the worst model decreased by a factor two from about 20 to about 10. The three submissions `CaloDiffusion`, `CaloScore`, and `CaloDREAM` are on top and have comparable scores within their error bars. `conv. L2LFlows` follows closely and has a small gap to the midfield, which is composed of `iCaloFlow`, the distilled versions of `CaloScore`, and `SuperCalo`.

In figure 71 (with details in table C19) we show precision, density, recall, and coverage of the ds 2 submissions. We first notice that there is a group of submissions — consisting of `CaloDiffusion`, `conv. L2LFlows`, `CaloScore` and its distillations, `iCaloFlow`, and `CaloDREAM` — that gets all four metrics close to the GEANT4 reference. This is another indication that these models generate high-quality showers. Another group stands out with a large density value. These are `CaloINN`, `Calo-VQ`, `CaloVAE+INN`, and `CaloDiT`. The large density, together with the small recall that most of the submissions in this group have, suggests again that samples are generated very similar to each other, but not diverse enough. The GAN submissions `MDMA` and `DeepTree` stand out in a third group, with low precision, density and coverage, but large recall. We interpret this pattern as generating samples that are fairly spread out, but not really close to the reference samples. That way, precision and density are low, but the large distance

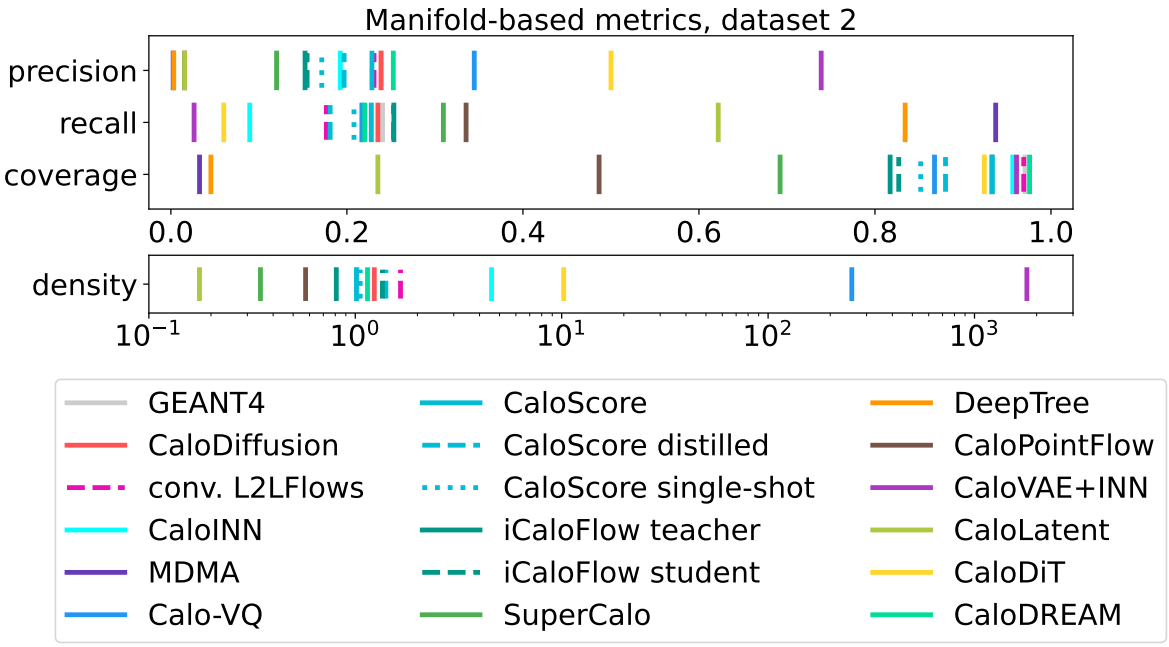



Figure 71: Precision, density, recall, and coverage for ds 2 submissions. For the precise numbers, see Table C19.

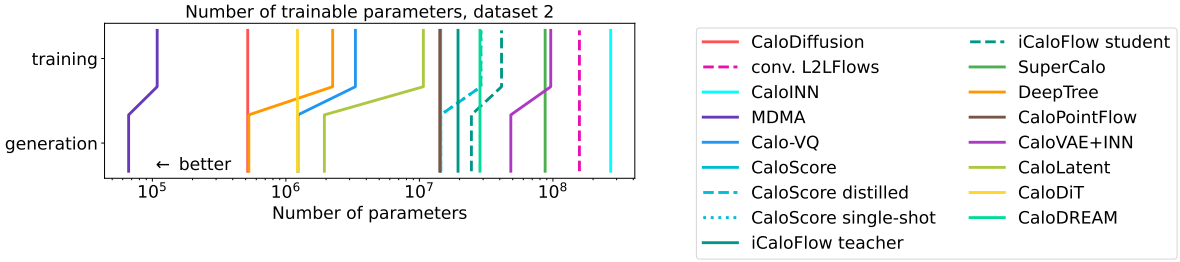



Figure 72: Number of trainable parameters for training and generation of ds 2 submissions. For the precise numbers, see Table C20.

between the submitted samples makes the recall manifold large enough to contain the references. The remaining submissions, `SuperCalo`, `CaloPointFlow`, and `CaloLatent` do not really fit in these groups, but are somehow similar to the GAN submissions with smaller precision, density, and coverage, but larger recall than the GEANT4 reference. However, the gap is smaller in these cases.

To summarize the shower quality, we see a similar pattern than already for dataset 1: The diffusion and conditional flow matching models have the best quality, followed then by Normalizing Flows and GAN and VAE-based models at the end.

Figure 72 compares the sizes of the submissions, with table C20 giving the precise numbers. The by-far smallest model is `MDMA`, with about an order of magnitude fewer parameters than the next submissions, `CaloDiffusion` and `DeepTree`. Normalizing-flow-based architectures like `conv. L2LFlows` and `CaloINN` have the most parameters,

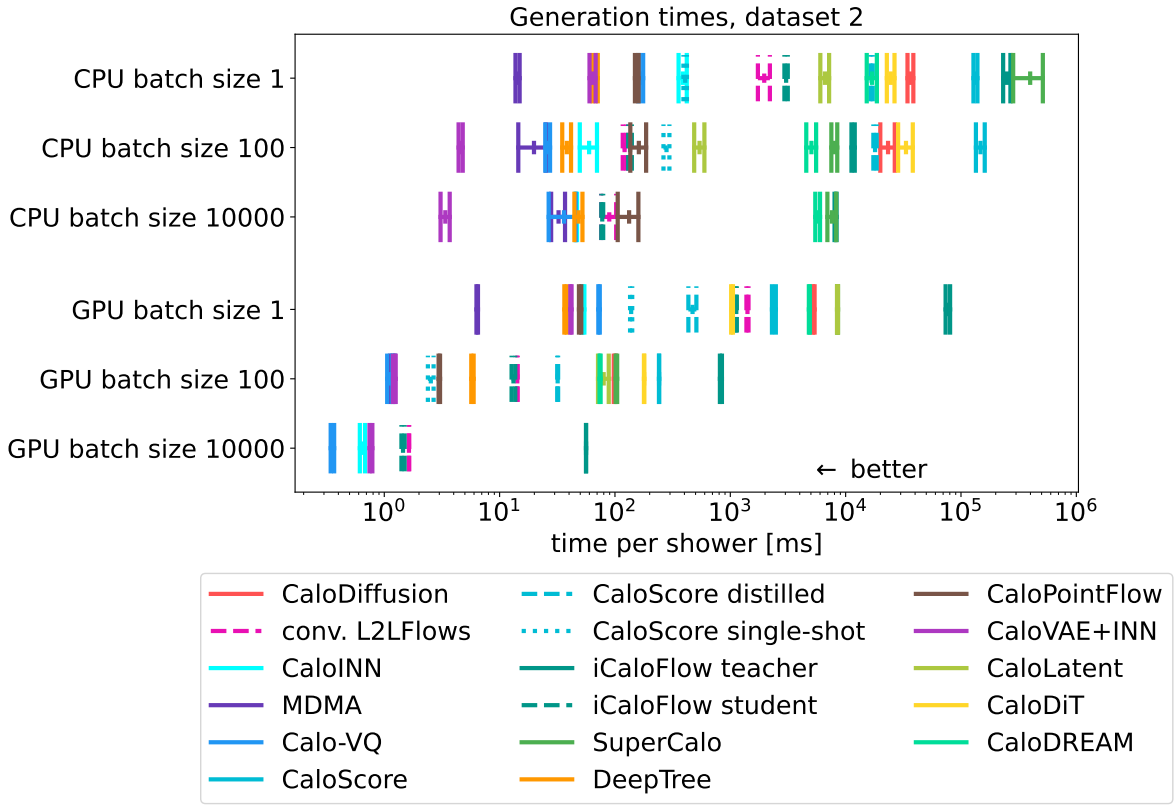



Figure 73: Timing of ds 2 submissions on CPU and GPU architectures. Not all submissions are shown everywhere due to memory and other constraints. More details are in table C21 and table C22.

so the bijective transformation in this 6480-dimensional space takes it toll on the required number of parameters.

Figure 73 (with detailed numbers in table C21 and table C22) show the generation times of the submissions normalized to generating a single shower. Overall, they span several orders of magnitude, even when only looking at one of the two architectures alone. This spread depends also on the batch size, with smaller batch sizes having a larger spread between slowest and fastest submission. For example, for a batch size of 1, we see four to five orders of magnitude difference. On the CPU, sample generation is in general slower and more spread out between slowest and fastest submission than on the GPU. Batching helps to speed up generation, but some of the models run into memory problems at very large batch sizes, even more on a GPU with limited VRAM. As for the DGM types, we see VAE and GAN-based models in the lead, with `MDMA`, `CaloVAE+INN`, and `Calo-VQ` being the fastest. The symmetric flow architecture of `CaloINN` is also fast in generation, but only on a GPU and for larger batch sizes. The diffusion models and MAF-based normalizing flows are the slowest submissions. Distillation of models clearly improves the generation speed in all cases, as expected. Generating showers in batches improves the generation speed in all cases, but also leads to out of memory errors in

11 out of 17 cases on the GPU. Given that some of the generation times (especially for smaller batch sizes) get considerably large, we restricted the number of samples used to time the models to fewer than 100 000 events, see the details in table C21 and table C22.

### *9.5. Dataset 3, Electrons (ds 3)*

Also for dataset 3, the minimal energy that can be read out is given by 15.15 keV. We again start our evaluation with the separation power of high-level observables, in particular with the energy depositions per layer and in total in figure 74. We notice that many models show the best performance around layers 3 – 10, and separation powers then grow towards the end of the detector. `CaloDREAM`, `CaloDiffusion`, `conv. L2LFlows`, and `L2LFlows-MAF` even reach the GEANT4 reference band in this region. Further, `CaloDREAM` matches the total energy deposition very well.

Moving on to centers of energy in $\eta$ and $\phi$ in figure 75 and figure 76, we see again two very similar distributions in both of these directions, indicating that the rotational symmetry was learned well by all submissions. At the level of the GEANT4 reference, we see `CaloDiffusion` and `CaloDREAM`, both with rather smooth separation powers from one calorimeter layer to the next. Slightly worse, we have `CaloPointFlow`, which was also smooth, and `conv. L2LFlows`, which shows some ups and downs from layer to layer. Among the rest, we notice a similar, but stronger up and down pattern for `Calo-VQ` (which is however not present in the improved version `Calo-VQ(norm)`). `MDMA` shows the largest spread between the lowest and largest separation power. The VAE-

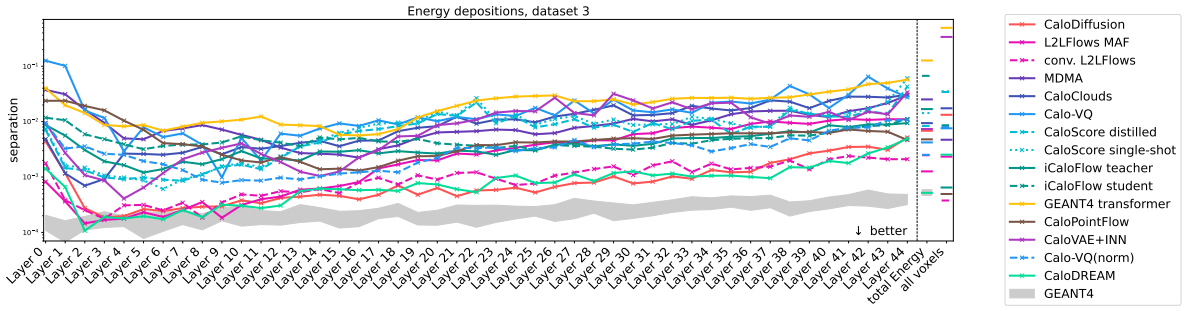



Figure 74: Separation power of energy depositions.

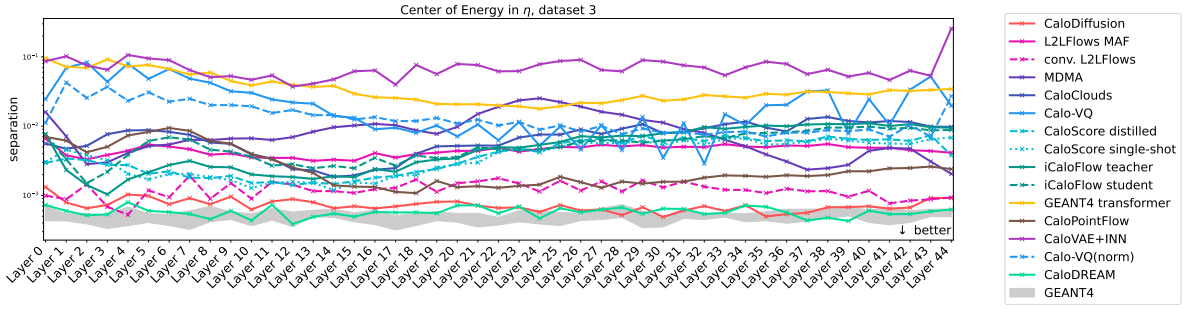



Figure 75: Separation power of centers of energy in $\eta$ direction.

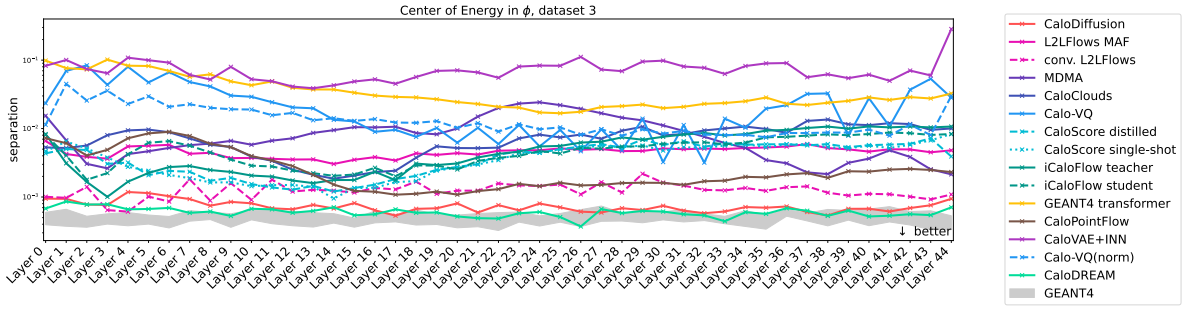



Figure 76: Separation power of centers of energy in $\phi$ direction.

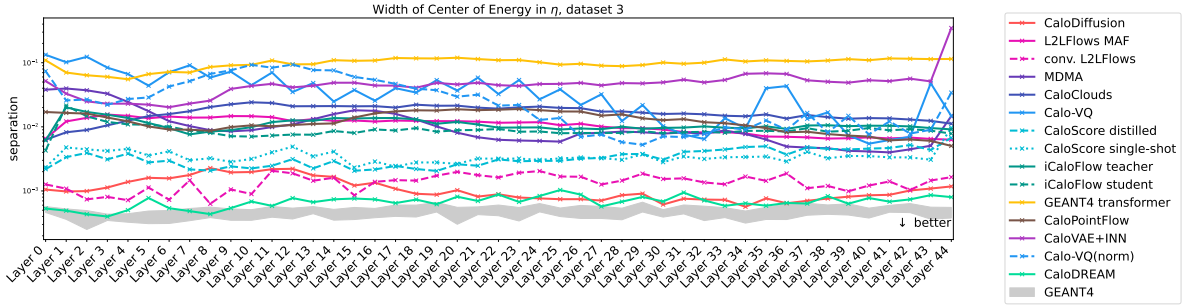



Figure 77: Separation power of widths of centers of energy in $\eta$ direction.

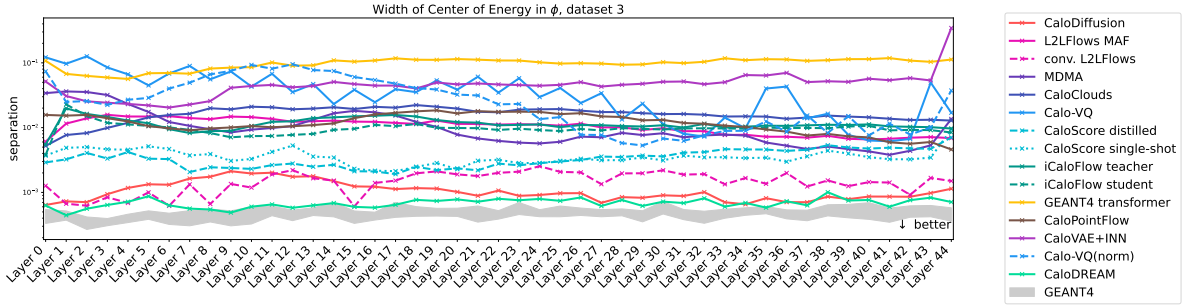



Figure 78: Separation power of widths of centers of energy in $\phi$ direction.

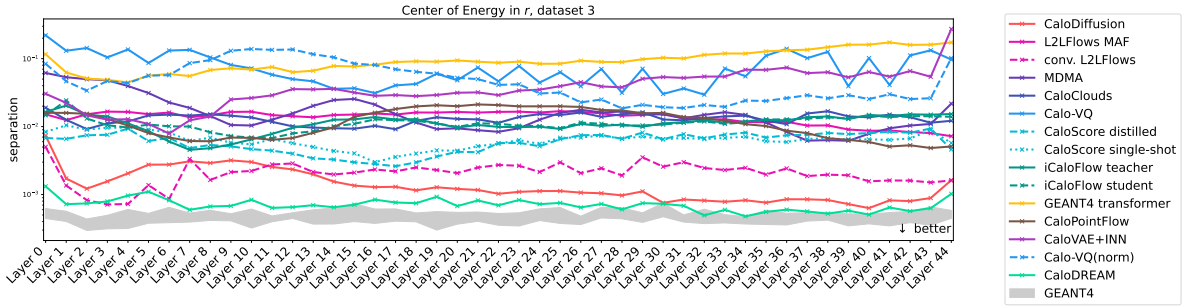



Figure 79: Separation power of centers of energy in radial direction.

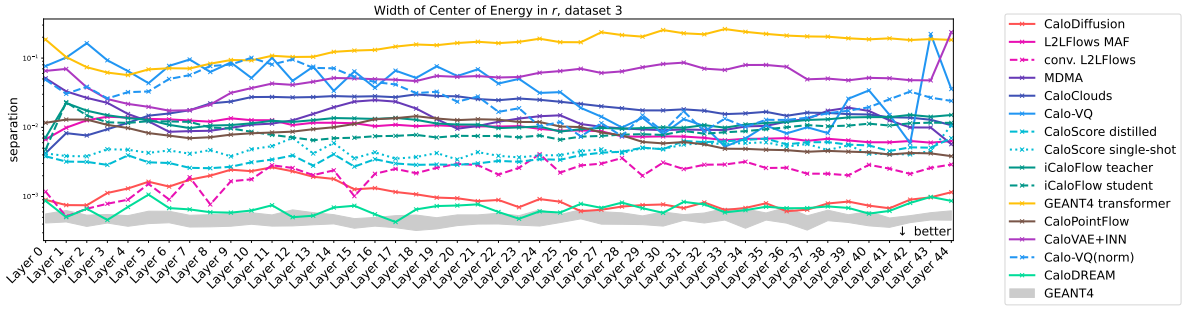



Figure 80: Separation power of widths of centers of energy in radial direction.

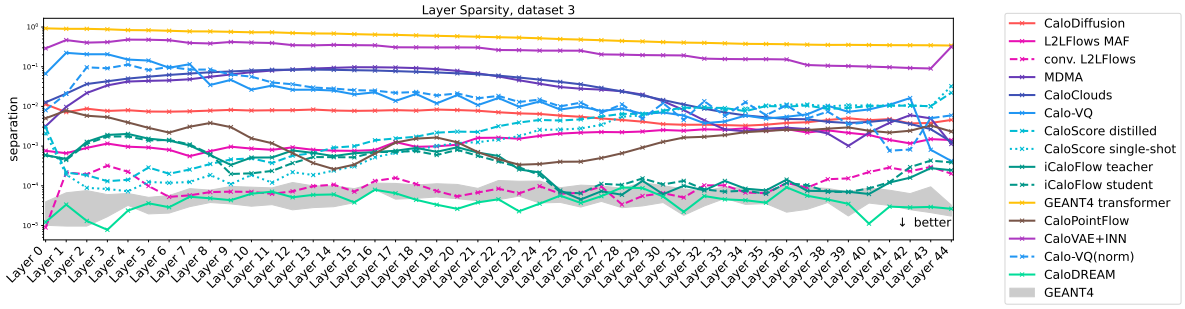



Figure 81: Separation power of the sparsity.

based submissions `CaloVAE+INN` and `Geant4-Transformer` have the most problems reproducing the GEANT4 data.

Most of the statements of the centers of energy also apply to their widths in figure 77 and figure 78. `CaloDREAM` has the smallest separation powers, close to GEANT4. `CaloDiffusion` comes second, but with a larger gap for earlier layers, where `conv. L2LFlows` shows a better match to the reference. In between these submissions and the bulk, we see `CaloScore distilled` and `CaloScore single-shot`. `Calo-VQ` again has an oscillating behavior over the entire size of the detector, and `Geant4-Transformer` and `CaloVAE+INN` have the largest separation powers. The order of submissions is also preserved when looking at the centers of energy in $r$ in figure 79 and their widths in figure 80.

Only for the sparsities in figure 81 we see a difference. `CaloDREAM` still shows the best performance, again at the level of GEANT4, but `CaloDiffusion` has a much harder time reproducing the correct distribution. Instead, `CaloScore distilled` and `CaloScore single-shot` (for early layers), `iCaloFlow` (for later layers), and `conv. L2LFlows` have small separation powers and get close to `CaloDREAM`. Also in this observable, the VAE-based submissions `Geant4-Transformer` and `CaloVAE+INN` show the largest separation powers. In fact, overall we see the separation powers ranging over five orders of magnitude between best and worst submission.

In figure 82 we look at the Pearson correlation coefficients of the layer energies. Also in this case we do not reproduce all findings of [20], again indicating that some of the observed patterns might fluctuate from training to training. Similar to what we have

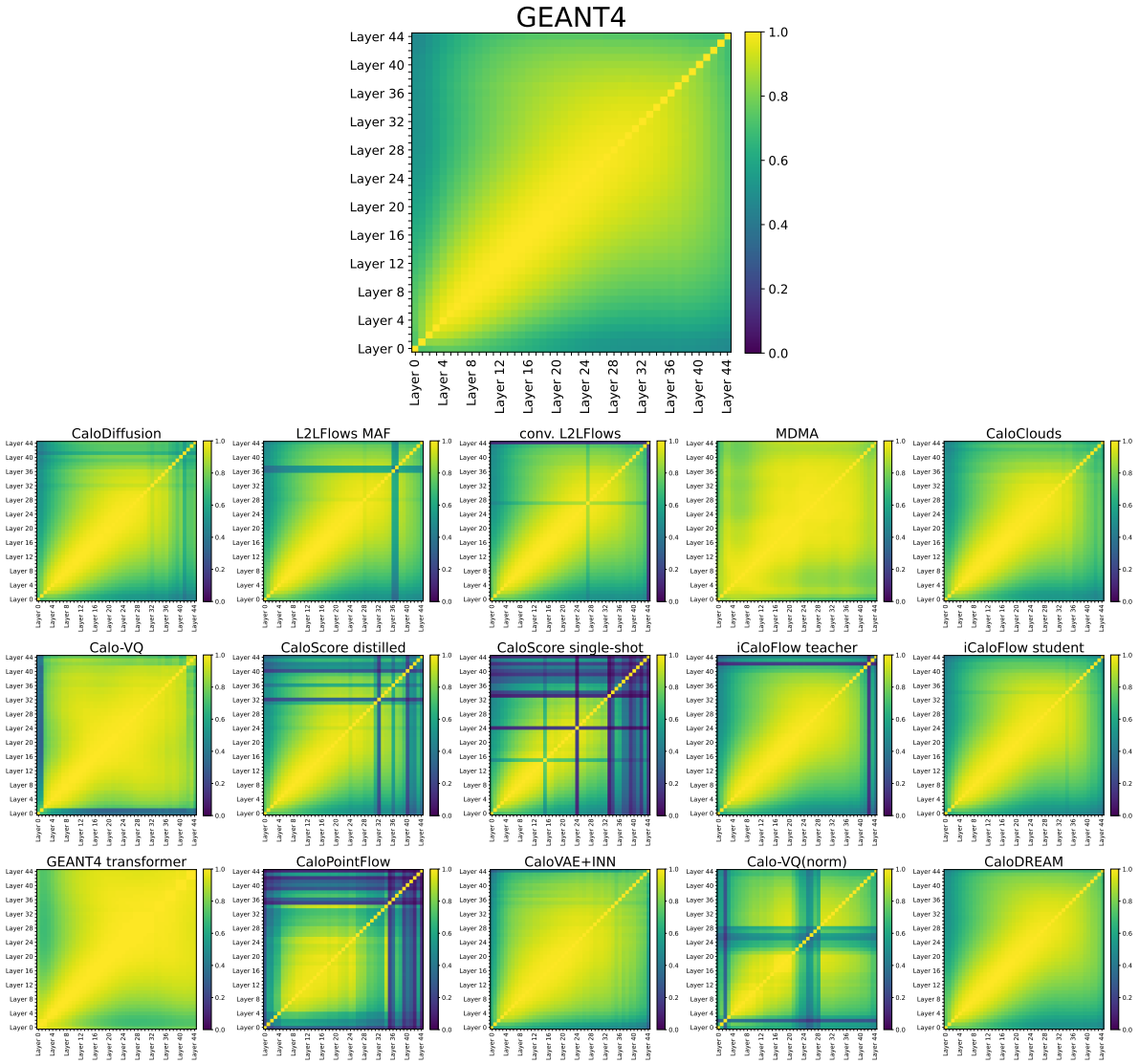



Figure 82: Pearson correlation coefficients of layer energies in ds 3.

observed for the other datasets, we see three different groups of correlation patterns. The first one reproduces the GEANT4 shape quite well and consists of `CaloClouds`, `iCaloFlow student`, and `CaloDREAM`. The submissions `MDMA`, `Geant4-Transformer` in the second group are also very smooth and only have small regions that appear slightly brighter than the reference. The third group consists of submissions that have single layers that do not have the correct correlation, indicated by stripes in the figures. While some are very faint and just a few (like for `CaloDiffusion`, `Calo-VQ`, `CaloVAE+INN`, or `iCaloFlow teacher`), others have more and a stronger contrast (like `CaloScore distilled`, `CaloScore single-shot`, `CaloPointFlow`, or `Calo-VQ(norm)`). We also find again the intriguing pattern that the distillation of `CaloScore` made the correlations worse, but the distillation of `iCaloFlow` improved the correlations.

The AUCs of the binary classifiers in figure 83 (and table C23) corroborate the

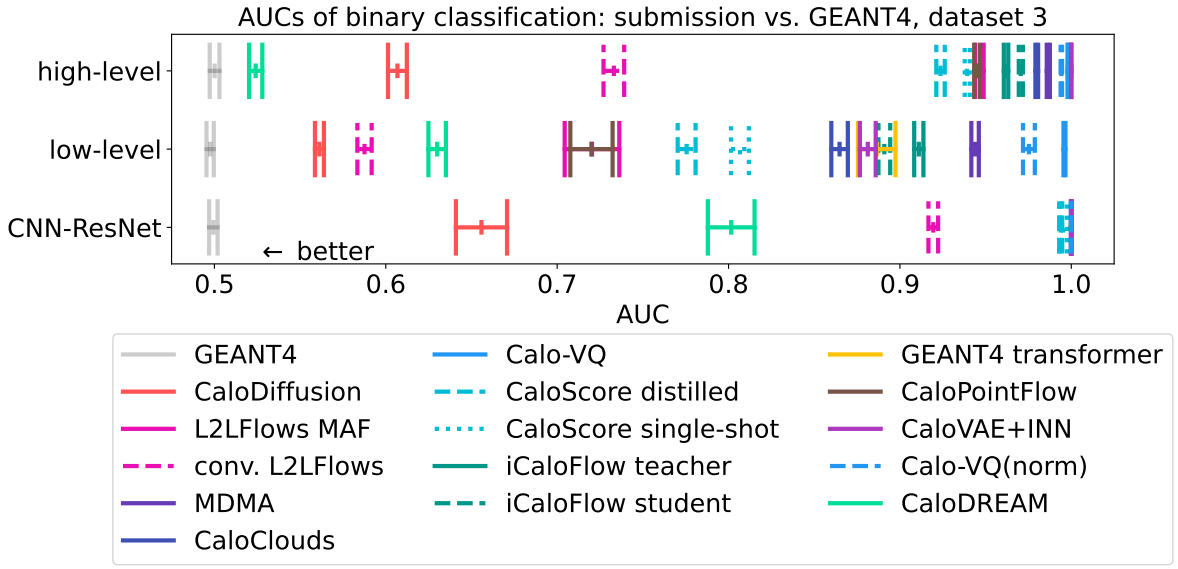



Figure 83: Low-level and high-level AUCs for evaluating GEANT4 vs. submission of ds 3, averaged over 10 independent evaluation runs. For the precise numbers, see Table C23.

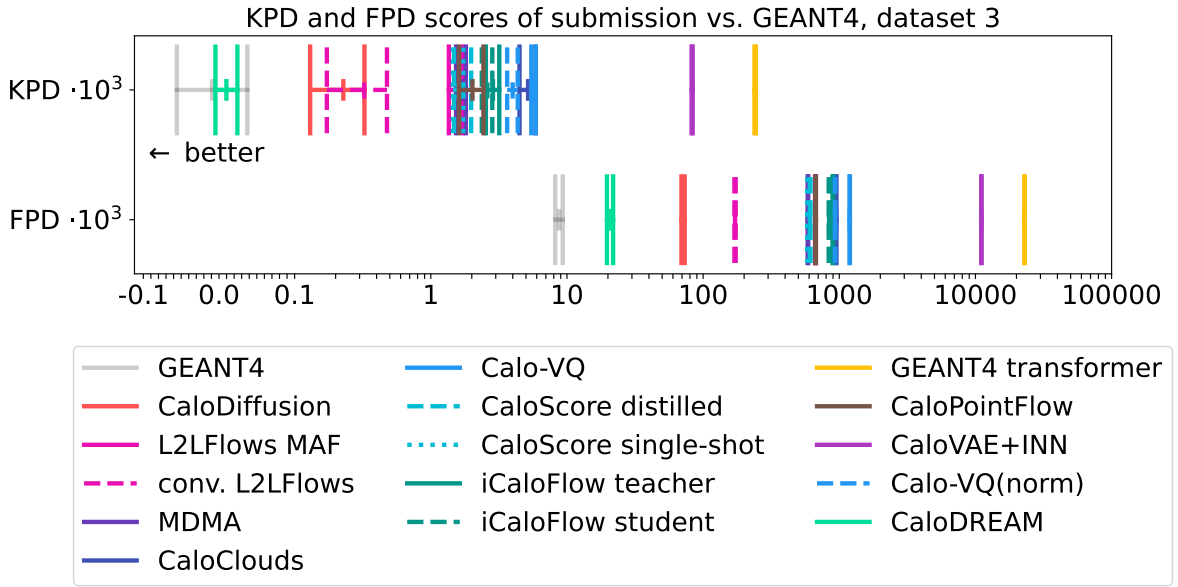



Figure 84: KPD and FPD for evaluating GEANT4 vs. submission of ds 3. For the precise numbers, see Table C24.

results of the separation power. For the high-level features, the best three models — `CaloDREAM`, `CaloDiffusion`, and `conv. L2LFlows`— are clearly separated from the other submissions. For low-level features, these three submissions still have the best performance, independent of the classifier architecture, but the ordering changed with `CaloDiffusion` having the best AUC. While the DNN indicates differences between submissions, yielding a spread between all the AUCs, the CNN-ResNet architecture

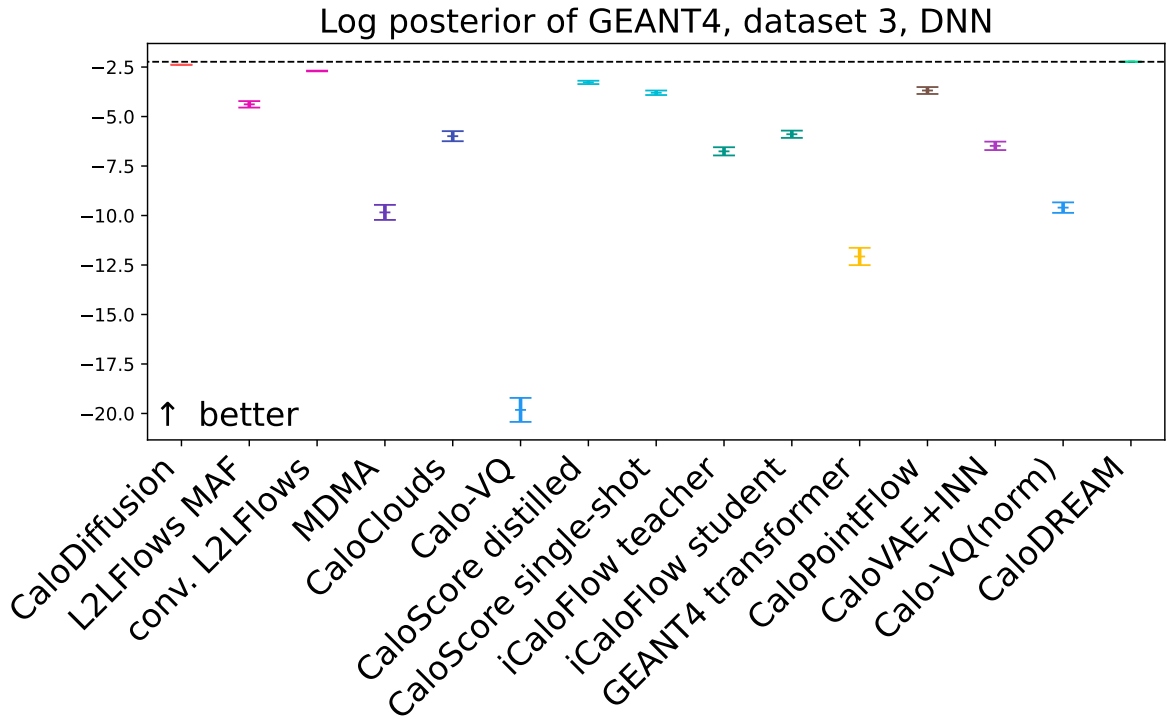



Figure 85: Log-posterior scores for ds 3 GEANT4 test data, averaged over 10 independent DNN classifier trainings. For the precise numbers, see Table C25.

essentially identifies the three best submissions — `CaloDiffusion`, `CaloDREAM`, and `conv. L2LFlows`— and gives all other submissions an AUC of 1.

A similar ordering, at least in terms of the top three models, is also seen in the KPD and FPD scores in figure 84 (with details in table C24). Now `CaloDREAM` is closest to the GEANT4 reference. Also for these scores (especially for the KPD), the bulk of all other submissions is very close to each other with scores overlapping within uncertainties.

The multiclass classifier metric, shown in figure 85, figure 86, table C25, and table C26 is consistent with the binary AUCs shown before. `CaloDREAM` and `CaloDiffusion` have the highest log-posterior, and `conv. L2LFlows` comes in third before there is a gap to the remaining submissions. Again, we see the CNN-ResNet being more powerful, giving low scores to almost all submissions when compared to GEANT4. As with ds 2, we also observe here that the spread in log-posterior between the best and worst model is smaller in the CNN-ResNet compared to the DNN architecture. However, both of the considered architectures have well-trained classifiers, as can be seen in figure B5 and figure B6. The size of the error bars, coming from ten independent retrainings of the classifier, seems to be correlated with the central value of the log-posterior, with smaller (worse) log-posterior scores having larger error bars.

In figure 87 and table C27 we show precision, density, recall, and coverage of the ds 3 submissions. The first thing we notice are the GEANT4 scores, which now have much smaller precision and recall compared to ds 2 in figure 71, maybe a

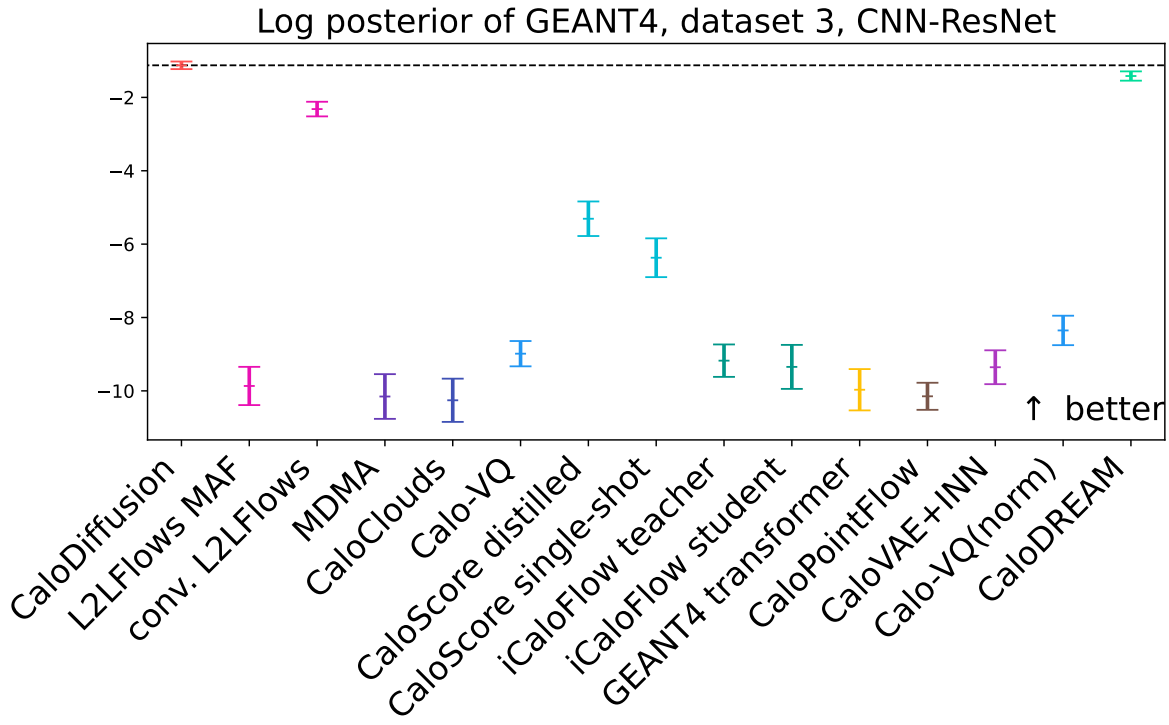



Figure 86: Log-posterior scores for ds 3 GEANT4 test data, averaged over 10 independent CNN ResNet classifier trainings. For the precise numbers, see Table C26.

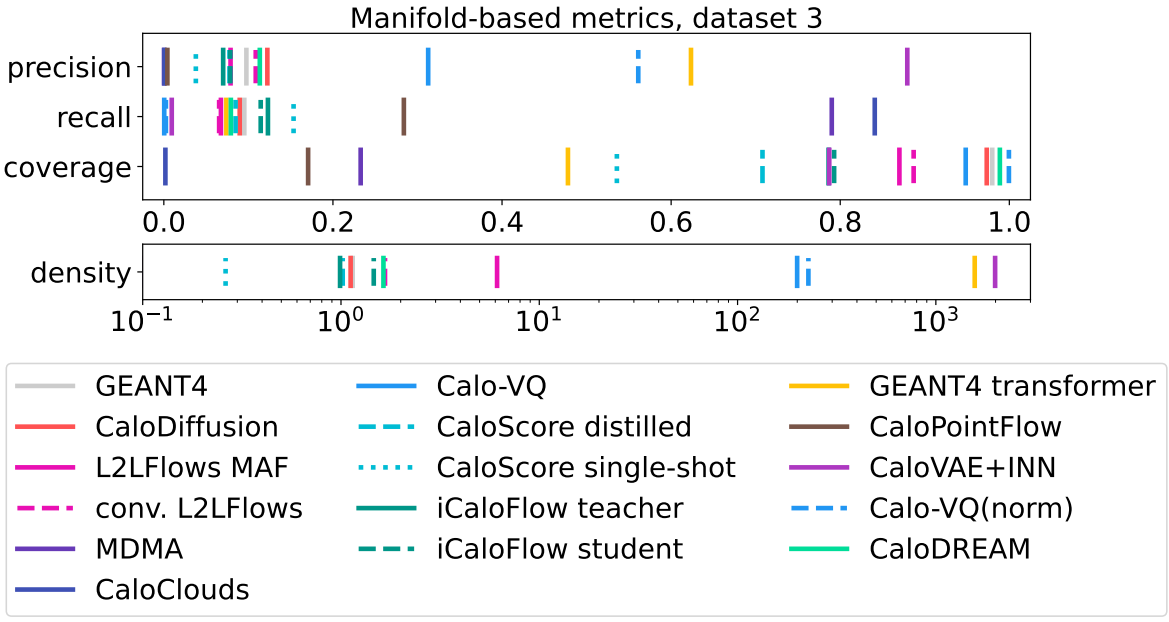



Figure 87: Precision, density, recall, and coverage for ds 3 submissions. For the precise numbers, see Table C27.

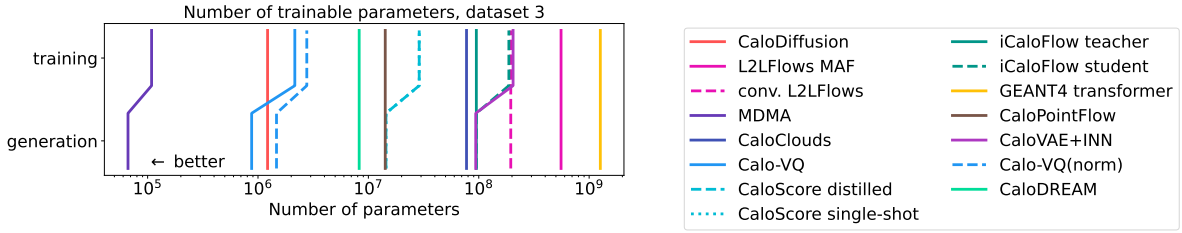



Figure 88: Number of trainable parameters for training and generation of ds 3 submissions. For the precise numbers, see Table C28.

sign for the much higher-dimensional dataset. When looking at the submissions, we observe groups with similar patterns as for the other datasets before. The first one of these are `CaloDiffusion`, `CaloScore distilled`, and `iCaloFlow teacher`, which have scores comparable to the GEANT4 reference. Similar to these, but with a slightly larger density are `CaloDREAM`, `conv. L2LFlows` and `iCaloFlow student`. These groups overlap to a large extend with the "winners" of the classifier-based metrics, but have with `iCaloFlow` also new members. With increasing density, we but otherwise similar scores is `L2LFlows-MAF`. These all indicate samples that are distributed similarly close to the validation data like the training data. Another group of submissions, consisting of `MDMA`, `CaloClouds`, `CaloScore single-shot`, and `CaloPointFlow`, have precision, density and coverage below the GEANT4 scores, and at the same time a very large recall. As already for ds 2, we interpret such a pattern as samples being generated fairly spread out, but not really close to the reference samples. Also these observations are consistent with what we saw for other metrics before. The last group, with a very large density, a larger precision and a low recall was also present in ds 2. In this group we have `Calo-VQ`, `Calo-VQ(norm)`, `Geant4-Transformer`, and `CaloVAE+INN`.

Figure 88 compares the sizes of the submissions, with table C28 giving the precise numbers. Overall, the entire span in number of parameters is more than four orders of magnitude. Similar to ds 2, `MDMA` has by far the fewest number of trainable parameters, making it a very economic submission. Following behind are with `Calo-VQ` and `CaloDiffusion` a VAE and a diffusion model, showing that these architectures can generate high-dimensional data much more economically than normalizing flows.

Lastly, we look at the generation time per shower in figure 89 (see table C29 for CPU and table C30 for GPU details). Overall, we see the same pattern as for all datasets before. Increasing the batch size and moving from a CPU to a GPU architecture speeds up generation. Depending on the architecture, sometimes by several orders of magnitude. However, the high dimensionality of ds 3 makes generation with large batch sizes sometimes impossible due to memory constraints. For example at batch size 10000, nine out of 15 submissions run into CUDA out of memory errors on the GPU. The large spread in generation times also required us to restrict the number of samples used to time the models to fewer than 100 000 events, especially for smaller batch sizes. For the largest batch size of 10 000, we had three cases on the CPU in which generation

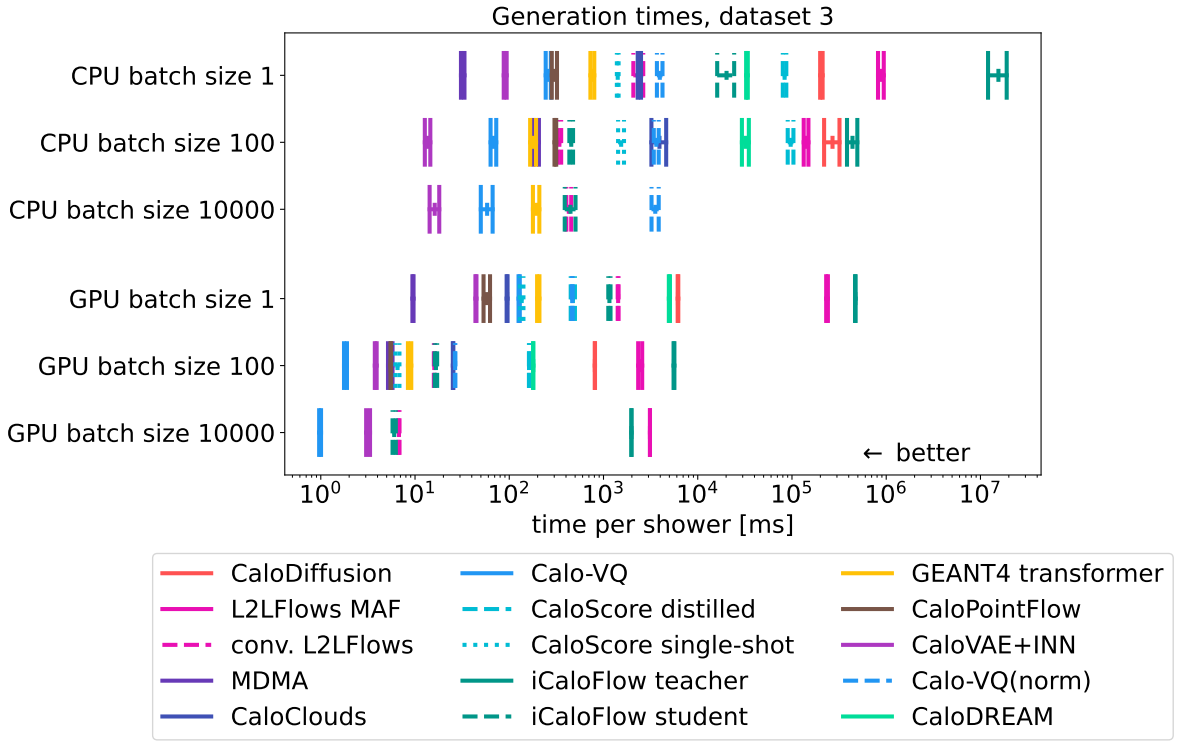



Figure 89: Timing of ds 3 submissions on CPU and GPU architectures. Not all submissions are shown everywhere due to memory and other constraints. More details are in table C29 and table C30.

of a single batch took longer than two days. Details for this are given in table C29 and table C30. The fastest models are again GAN-based submissions like `MDMA` and VAE-based submissions like `Calo-VQ` or `CaloVAE+INN`. As for the previous datasets, we observe that distillation worked and speeds up generation in all cases.

## 10. Results: Correlations Between Metrics

In this section we study how the scores in different metrics are related to each other. The goal of that is two-fold: First, in section 10.1, we study how various different metrics that all measure the same property correlate with each other. In the case of the sample quality, this will shed light on various aspects regarding the evaluation of generative models, a result of great importance beyond detector fast simulation. Second, in section 10.2, we are interested in the Pareto fronts in the "quality *vs.* speed *vs.* resource consumption" space, as these will be the ultimate results of the CaloChallenge. The observations made in the first part, *i.e.* how which quality metrics correlate with each other, will be especially important for the choice of metrics shown in the final evaluation Pareto Fronts.

### 10.1. Metric Comparison

As a nice side result of the challenge, we can evaluate how different metrics that measure the quality of the showers correlate with each other. These tests also justify that the Pareto fronts we will show below are representative.

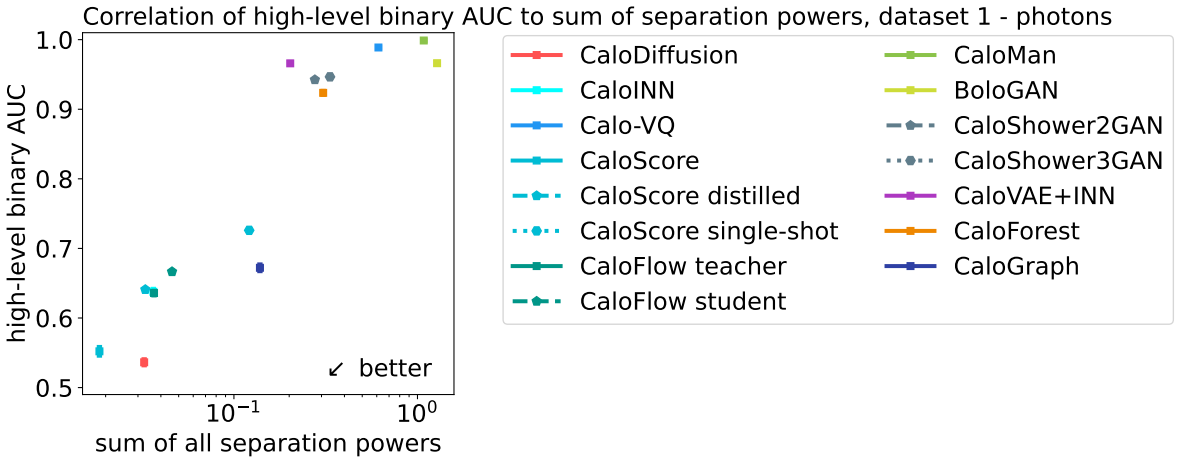



Figure 90: Correlation of two metrics based on high-level observables: the sum of all the separation powers (figure 32–figure 37) vs. the binary AUC (figure 39 and table C1).

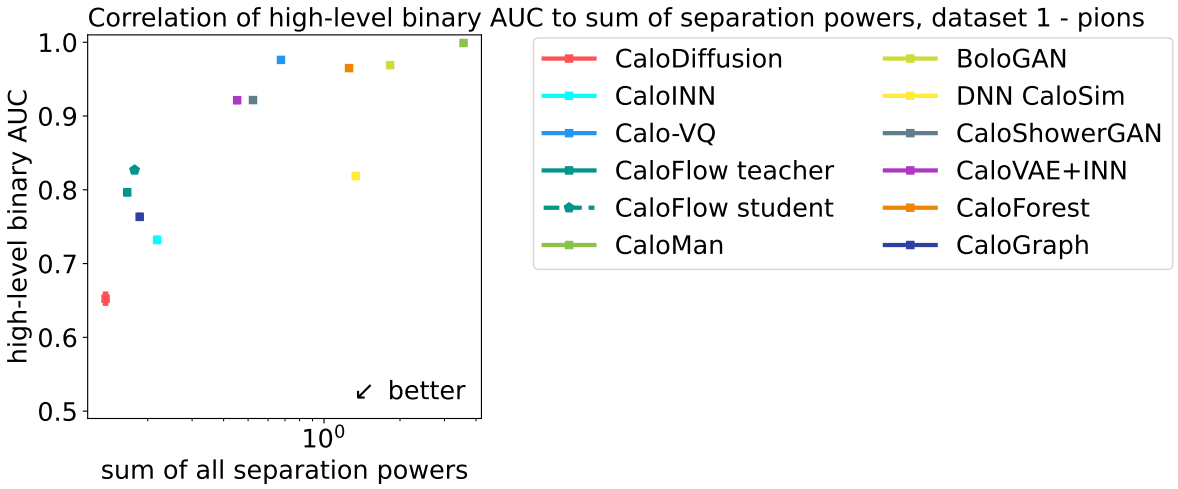



Figure 91: Correlation of two metrics based on high-level observables: the sum of all the separation powers (figure 45–figure 50) vs. the binary AUC (figure 52 and table C8).

The first of these tests looks at the high-level observables that were defined in section 8.1 and compares the sum of all separation powers to the AUC of the binary classifier. While the former is only sensitive to the distribution of the individual observables, the latter also captures correlations between them. We see in figure 90 that the results for ds $1-\gamma$ show a clear correlation. Submissions with a higher AUC also have a larger sum of their separation powers. The situation is similar for ds $1-\pi^+$

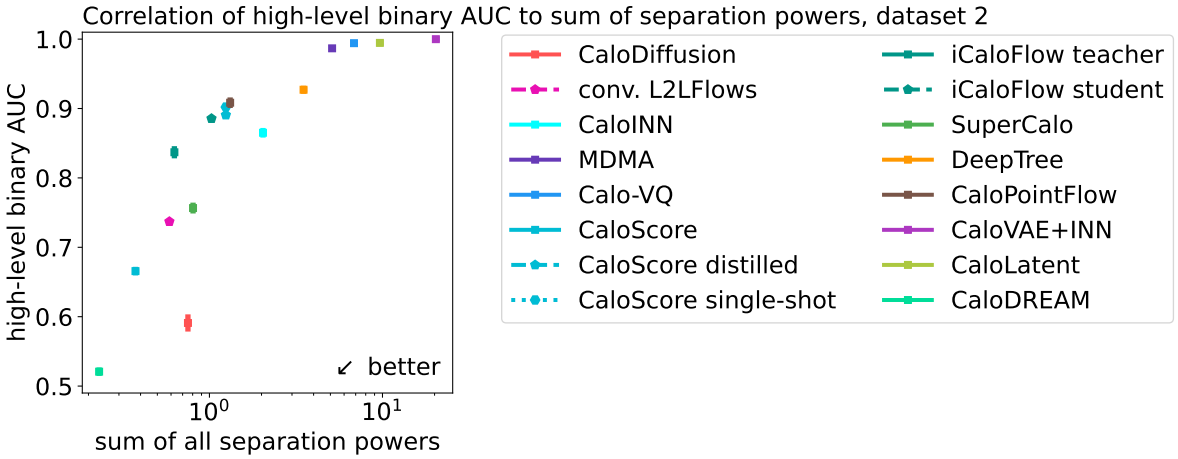



Figure 92: Correlation of two metrics based on high-level observables: the sum of all the separation powers (figure 58–figure 65) vs. the binary AUC (figure 67 and table C15).

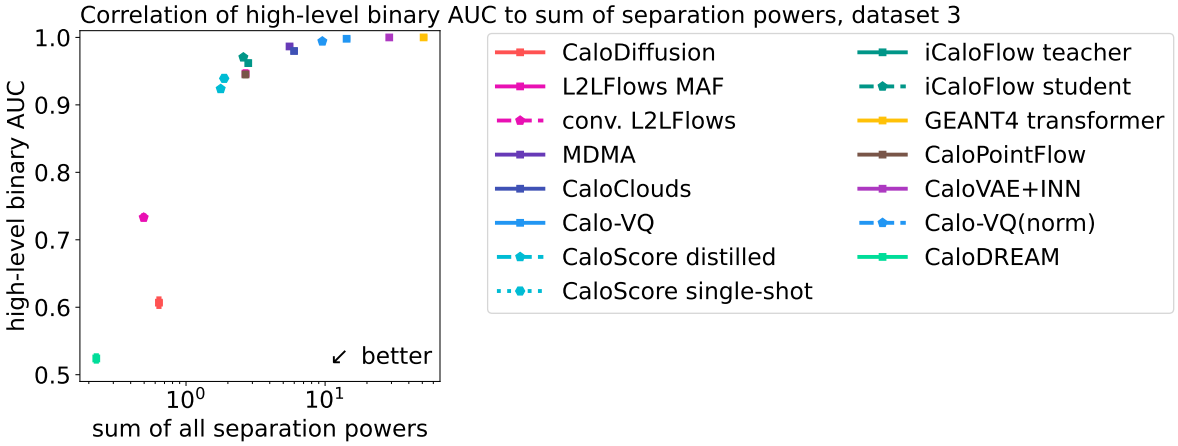



Figure 93: Correlation of two metrics based on high-level observables: the sum of all the separation powers (figure 74–figure 81) vs. the binary AUC (figure 83 and table C23).

in figure 91, but there the submissions are a little more spread out, indicating that some models struggled a bit more to capture all correlations between the observables. Also datasets 2 and 3 in figure 92 and figure 93 show a clear correlation of the two metrics, confirming that they both capture the essential features of the high-level observables.

Next we investigate how the choice of the input representation to the binary classifier influences the AUC. In particular, we look at the correlation of the AUC of the binary classifier with low-level inputs *vs.* the AUC of the binary classifier with high-level inputs. Figure 94 shows the result for ds $1-\gamma$. While there is a clear correlation between the two metrics visible, there is also a noticeable spread between submissions, for example when comparing `CaloINN` to `CaloScore`. The situation is more clear for ds $1-\pi^+$ in figure 95. Here, we see two different lines forming. One with `CaloDiffusion`, `CaloINN`, `CaloGraph`, and `CaloFlow`, where the low-level AUC is

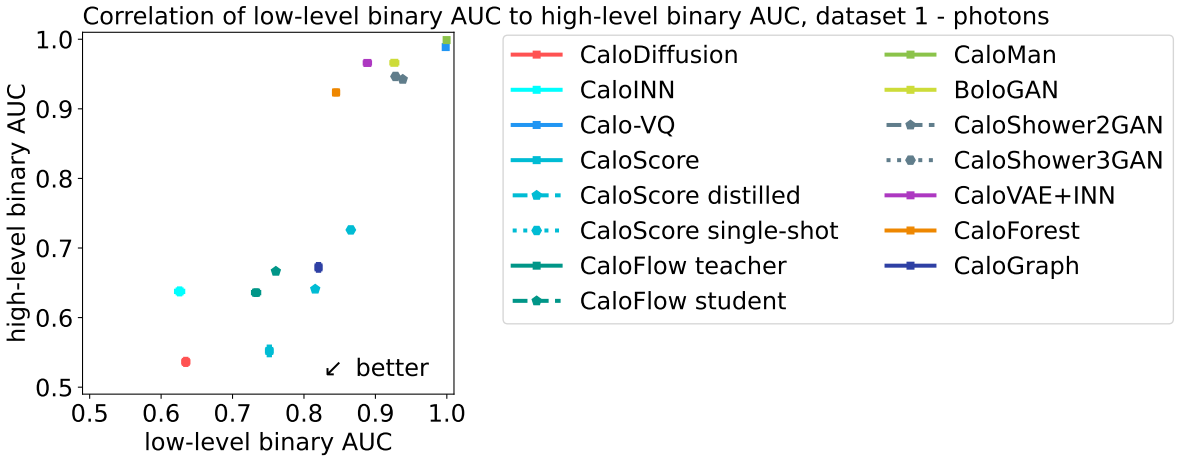



Figure 94: Correlation of two metrics based on binary classifiers (figure 39 and table C1): the AUC based on low-level observables vs. the AUC based on high-level observables.

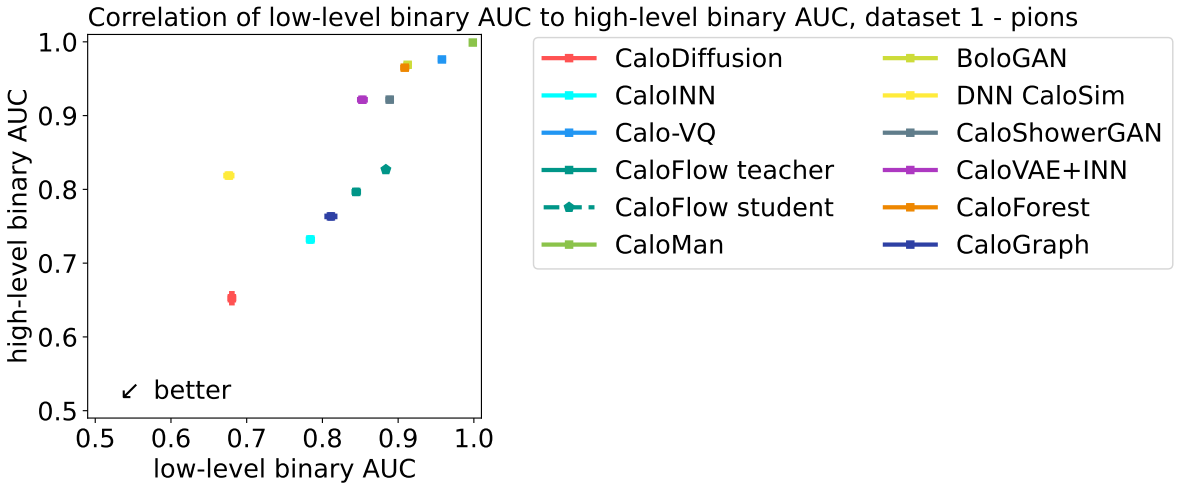



Figure 95: Correlation of two metrics based on binary classifiers (figure 52 and table C8): the AUC based on low-level observables vs. the AUC based on high-level observables.

slightly worse than the corresponding high-level AUC. The other one with `DNNCaloSim`, `CaloVAE+INN`, `CaloShowerGAN`, `CaloForest`, `Calo-VQ`, and `CaloMan`, where the high-level AUC is larger than the low-level AUC. Interestingly, the division in these two sets aligns with the underlying architectures, with the diffusion models and normalizing flows in the first group and the VAEs and GANs in the second group. We interpret these differences as follows: the first group (diffusion and Normalizing Flow-based) generates showers which better capture the correlations between voxels that form the high-level observables and the remaining mismodeling between the submissions and GEANT4 is in the lower-energetic, subleading voxels. The second group (VAE and GAN-based), however, already mismodels the correlations that form the high-level observables leading to a larger AUC for this classifier. The strong correlation between the AUCs is also

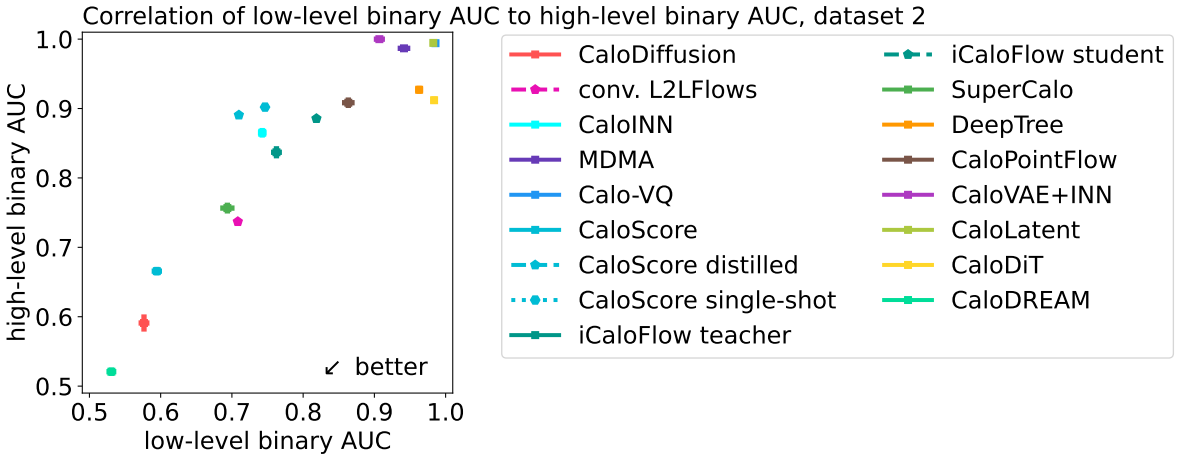



Figure 96: Correlation of two metrics based on binary classifiers (figure 67 and table C15): the AUC based on low-level observables vs. the AUC based on high-level observables.

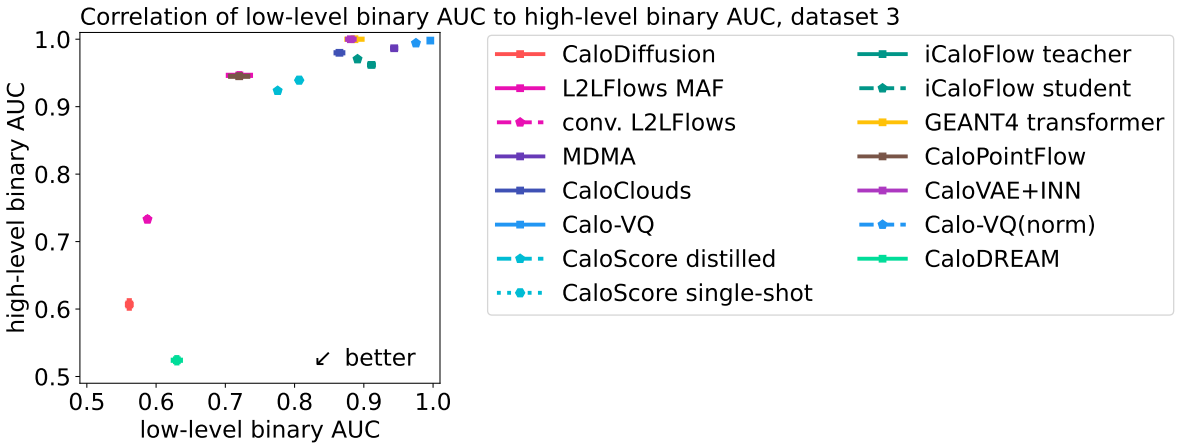



Figure 97: Correlation of two metrics based on binary classifiers (figure 83 and table C23): the AUC based on low-level observables vs. the AUC based on high-level observables.

present for dataset 2 in figure 96. For dataset 3 in figure 97 it is not as pronounced, but that is mostly due to the high-level AUCs being close to 1 for many submissions. The correlations between high and low-level AUCs also tell us something about the classifier metric itself. Since all the high-level observables are derived from the low-level ones, there cannot be any additional information in the high-level observables. The AUC based on low-level inputs should therefore be strictly larger, *i.e.* indicating a better classifier than the AUC based on high-level features alone. The fact that we do not see this here indicates that the DNN classifier used in this study is not at the Neyman-Pearson limit and additional studies based on the high-level observables are indeed necessary to get a better understanding on the quality of the generated samples.

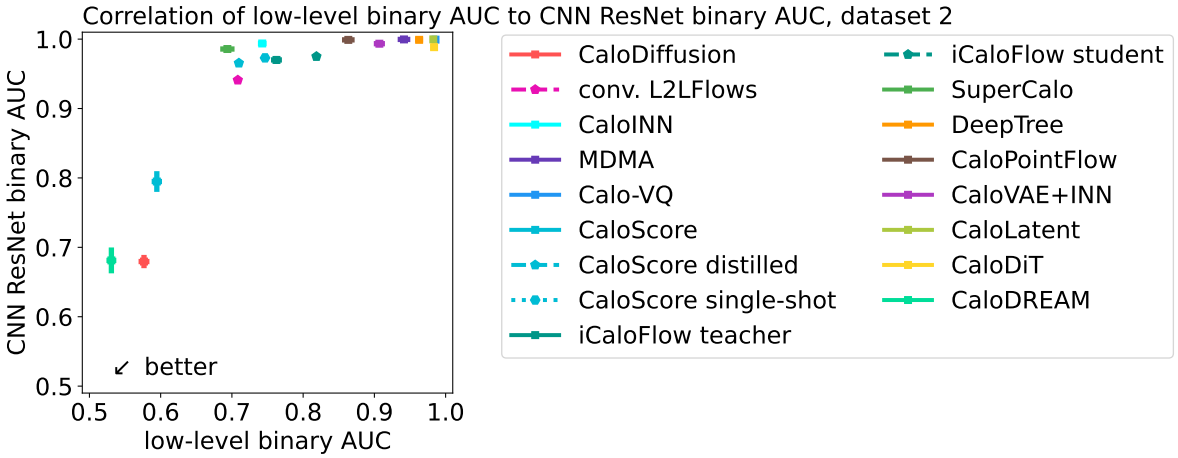



Figure 98: Correlation of two metrics based on binary classifiers and low-level observables (figure 67 and table C15): the AUC based on a DNN classifier vs. the AUC based on CNN ResNet classifier.

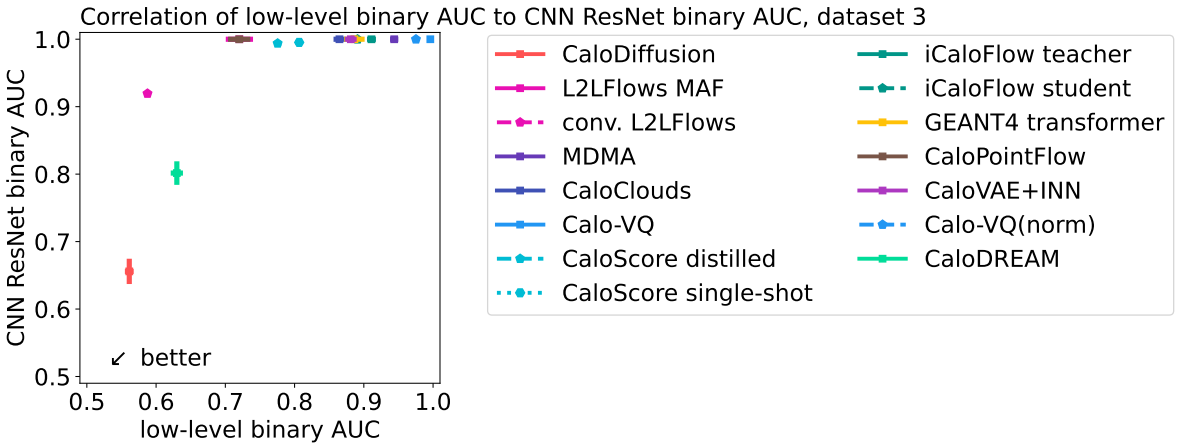



Figure 99: Correlation of two metrics based on binary classifiers and low-level observables (figure 83 and table C23): the AUC based on a DNN classifier vs. the AUC based on CNN ResNet classifier.

For datasets 2 and 3 we can also compare the AUCs obtained by the two different architectures used for the binary classification: the DNN and the CNN ResNet. The results are shown in figure 98 and figure 99. In both cases we see a correlation, but we also see many submissions having a CNN ResNet-based AUC close to 1, making it hard to order them by this metric. This is especially true for dataset 3. We therefore use the DNN architecture for the Pareto fronts below.

Lastly, we compare the results of the binary classification to the results of the multiclass classification. Also in these cases (ds $1-\gamma$ in figure 100, ds $1-\pi^+$ in figure 101, ds 2 in figure 102, and ds 3 in figure 103), we observe a clear correlation: submissions performing well in one metric also perform well in the other metric, indicating that

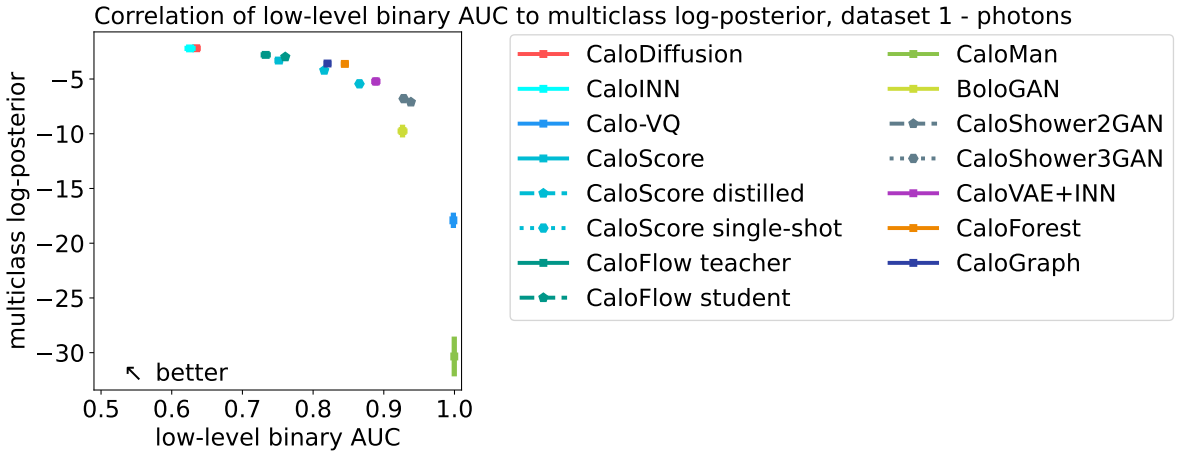



Figure 100: Correlation of two metrics based on classifiers : the log posterior (figure 41 and table C3) of the multiclass classification vs. the AUC of the binary classification (figure 39 and table C1).

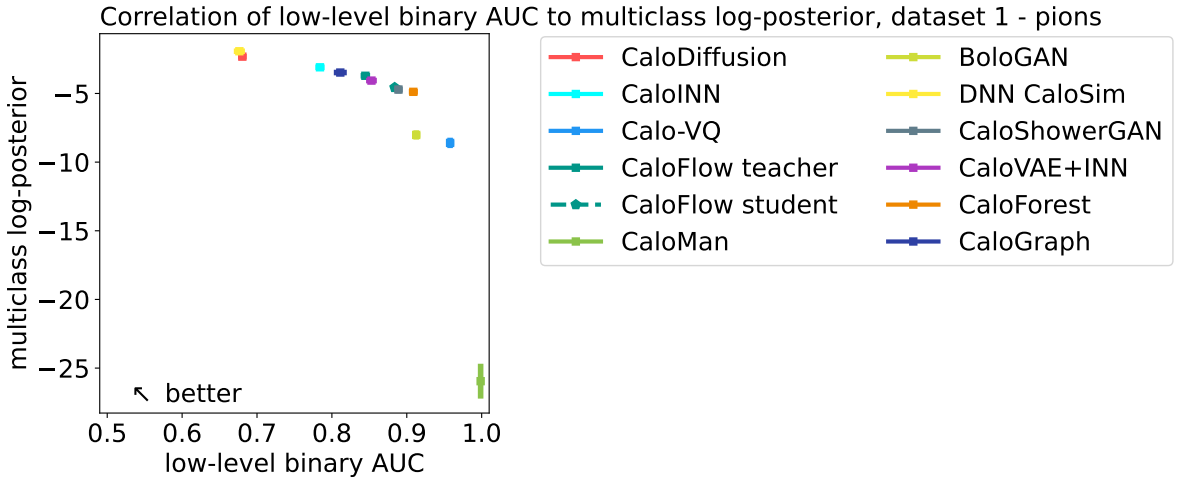



Figure 101: Correlation of two metrics based on classifiers : the log posterior (figure 54 and table C10) of the multiclass classification vs. the AUC of the binary classification (figure 52 and table C8).

both the binary and multiclass classification capture the main differences between the submissions. The spread for ds 3 in figure 103 is larger than for ds 2 in figure 102, which is maybe due to the rather small sample size compared to the high-dimensionality of ds 3. Overall, this implies that the binary classification can be used for further model development and it is not required to have all other submitted samples at hand to perform a multiclass classification for model evaluation.

In addition to the quality metrics, we also look at the correlation between the generation times per shower on CPU and GPU architectures. In particular, we consider generation batch sizes of 100 in figure 104 for ds $1-\gamma$, figure 105 for ds $1-\pi^+$, figure 106 for ds 2, and figure 107 for ds 3. In all cases, we see the scatter between fastest and

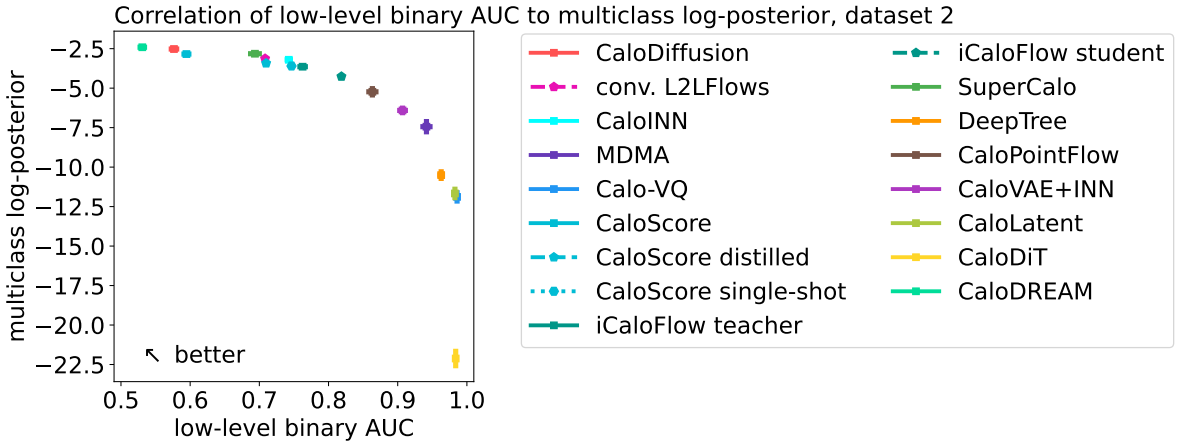



Figure 102: Correlation of two metrics based on classifiers : the log posterior (figure 69 and table C17) of the multiclass classification vs. the AUC of the binary classification (figure 67 and table C15).

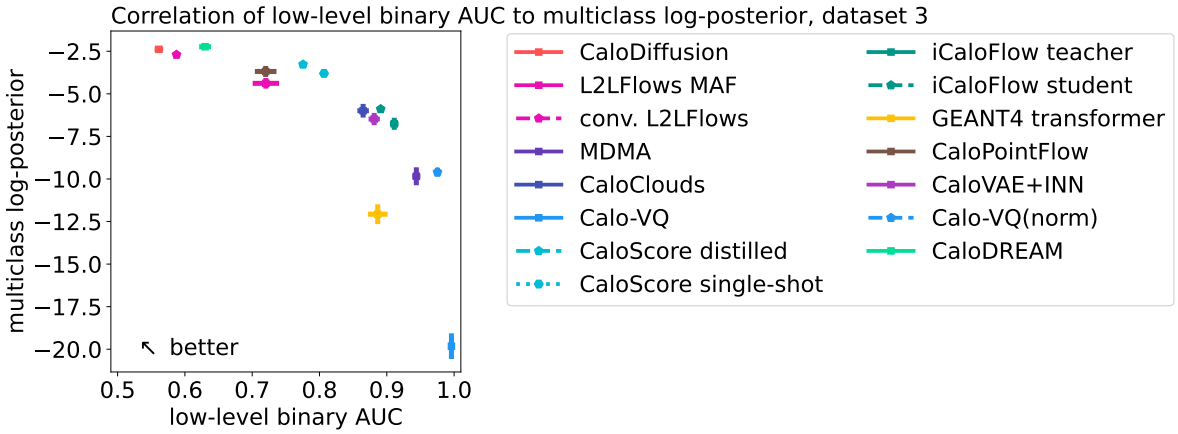



Figure 103: Correlation of two metrics based on classifiers : the log posterior (figure 85 and table C25) of the multiclass classification vs. the AUC of the binary classification (figure 83 and table C23).

slowest model to be much larger on the CPU than on the GPU. On top of the usual speed-up on the GPU, we observe the actual speed-up factor vary from model to model, depending on the specific building blocks of the models.

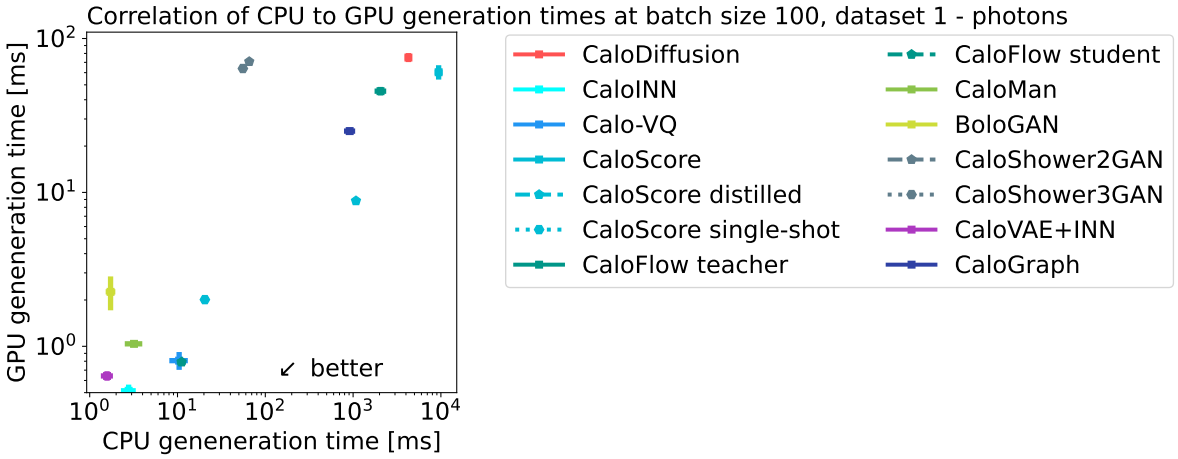



Figure 104: Correlation of generation times on CPU and GPU (see figure 44, table C6 and table C7).

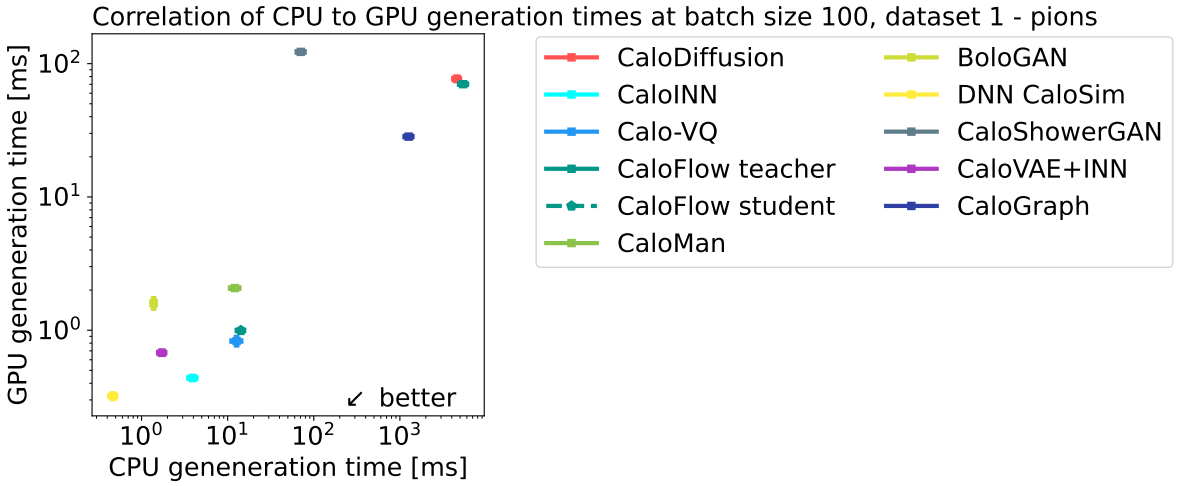



Figure 105: Correlation of generation times on CPU and GPU (see figure 57, table C13 and table C14).

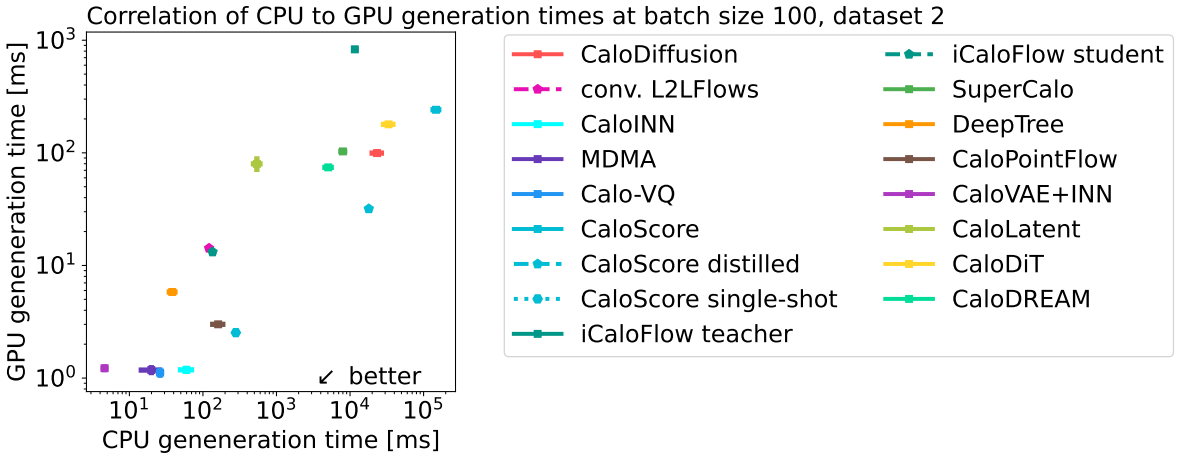



Figure 106: Correlation of generation times on CPU and GPU (see figure 73, table C21 and table C22).

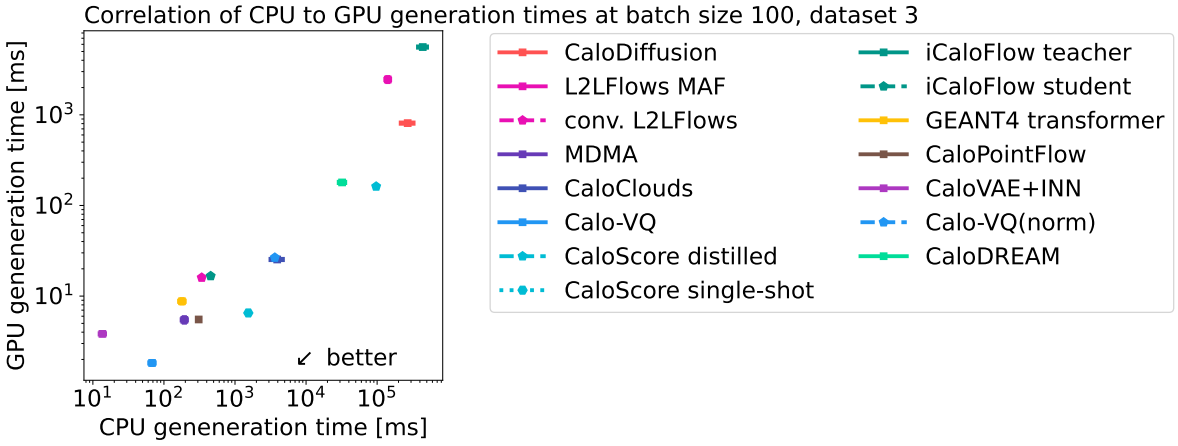



Figure 107: Correlation of generation times on CPU and GPU (see figure 89, table C29 and table C30).

### *10.2. Pareto Fronts*

This section compiles the main results of the "Fast Calorimeter Simulation Challenge 2022". We show the performance of the submissions in the abstract "quality *vs.* speed *vs.* resource consumption" space. We are interested in submissions which are lightweight (*i.e.* have few parameters), are fast in generation, and have good sample quality. In particular, we focus on two planes in which there is a trade-off between two properties: quality *vs.* resource consumption and quality *vs.* speed. The third option, speed *vs.* resource consumption, does not show a real trade-off, so we collect the figures in appendix D.

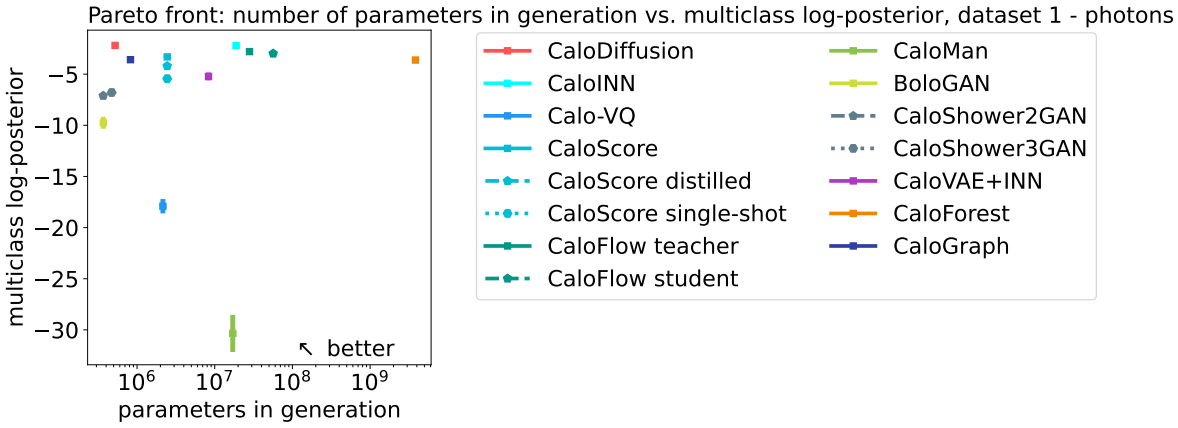



Figure 108: Pareto front in sample quality (from figure 41 and table C3) and number of parameters in generation (from figure 43 and table C5).

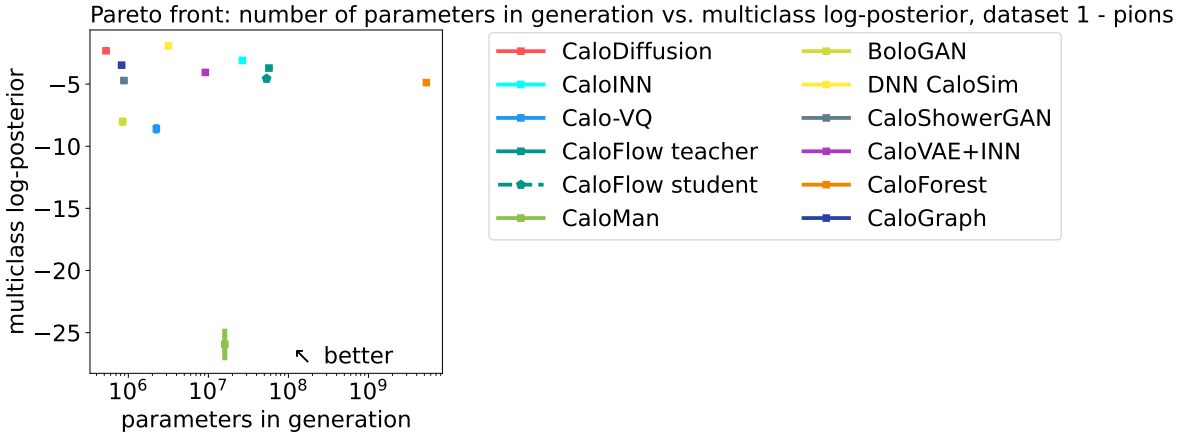



Figure 109: Pareto front in sample quality (from figure 54 and table C10) and number of parameters in generation (from figure 56 and table C12).

We start by comparing sample quality to model size by plotting the DNN multiclass log-posterior with respect to the number of trainable parameters in generation. While we expect models with more parameters to better learn the underlying probability

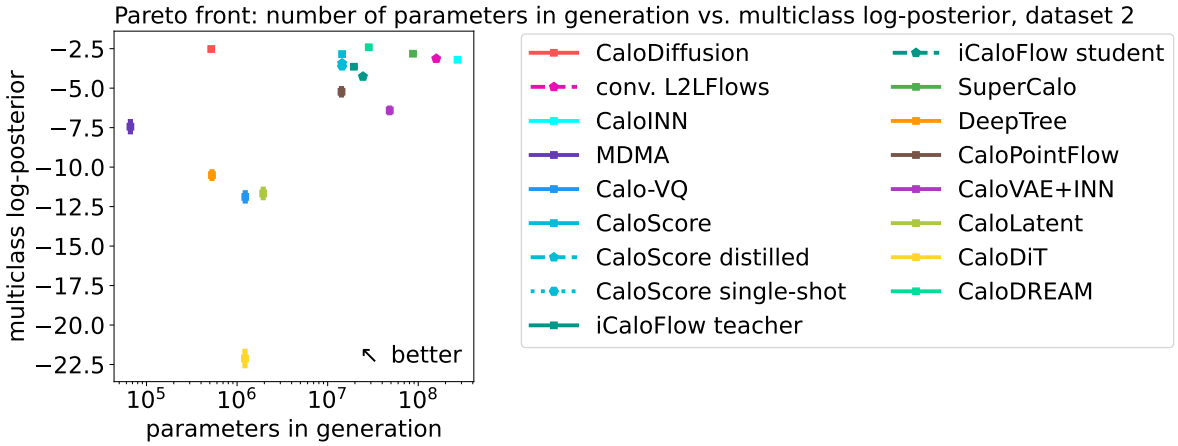



Figure 110: Pareto front in sample quality (from figure 69 and table C17) and number of parameters in generation (from figure 72 and table C20).

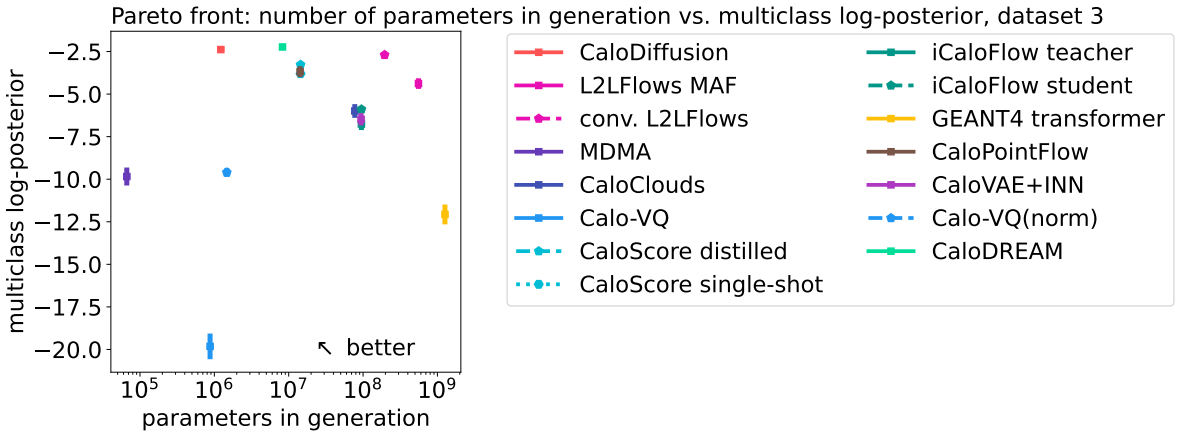



Figure 111: Pareto front in sample quality (from figure 85 and table C25) and number of parameters in generation (from figure 88 and table C28).

distribution that generates the showers, the use of the generative model inside a fast simulation framework prefers models that require less memory and are faster to load, *i.e.* have fewer parameters in generation. The figures for both particles in dataset 1 (photons in figure 108 and pions in figure 109) are very similar. In both cases we see `CaloDiffusion` in the top left corner, indicating that this diffusion model can generate high-quality showers with a comparatively small number of parameters. For dataset 2 in figure 110, we do not have a clear winner in the corner. Instead, we observe a cluster of various submissions (including `CaloScore`, its distillations, `iCaloFlow`, and `SuperCalo`) at good scores, but relatively large number of parameters. `CaloDiffusion` is part of the Pareto front, with similar or better quality than submissions of said cluster, but more than an order of magnitude fewer parameters. Sacrificing some quality moves the Pareto front to even fewer parameters with the submission `MDMA`. Dataset 3 in figure 111 shows a similar trade-off between `CaloDiffusion` and `MDMA` around the top-left corner,

but not such a large cluster of submission in the top-right.

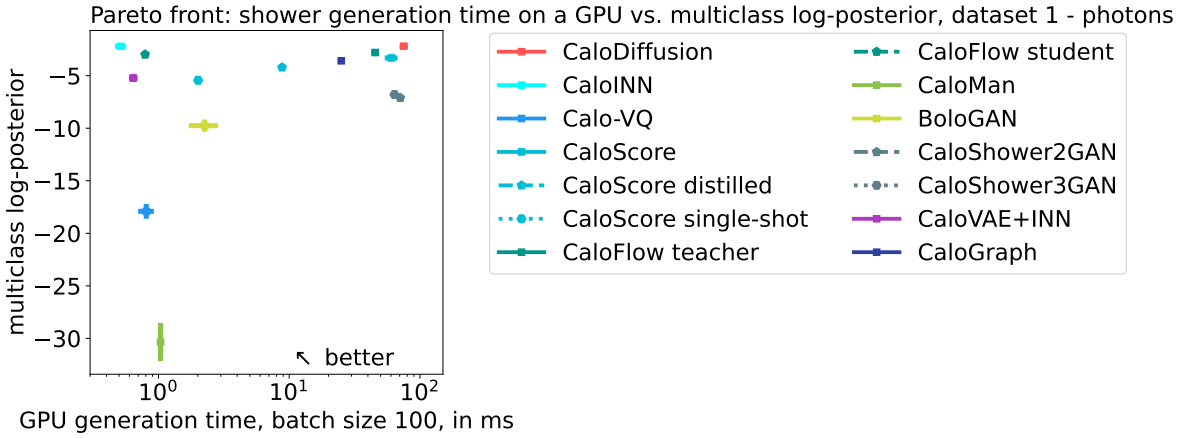



Figure 112: Pareto front in sample quality (from figure 41 and table C3) and generation speed (from figure 44 and table C7).

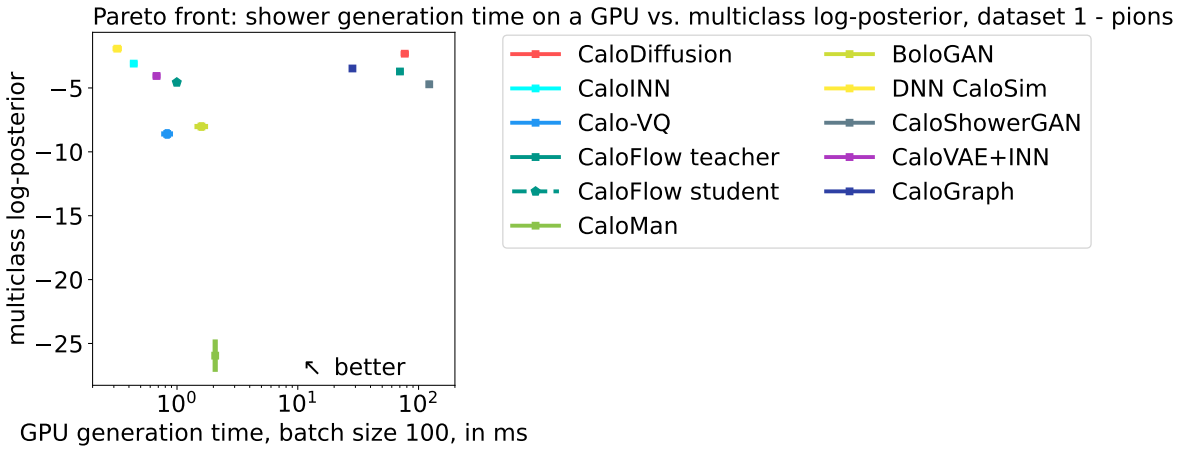



Figure 113: Pareto front in sample quality (from figure 54 and table C10) and generation speed (from figure 57 and table C14).

Next, we show the money plots in figure 112, figure 113, figure 114, and figure 115. Here, we compare the sample quality, measured by the DNN multiclass log-posterior, to the generation time, measured by the per-shower-time it takes to generate the entire dataset in batches of 100 on a GPU.

For ds $1 - \gamma$, we truly see a trade-off between the two metrics in figure 112. On the one side, we have submissions with good sample quality, *i.e.* a high log-posterior and large generation time in the top-right corner. The submissions `CaloDiffusion`, `CaloScore`, `CaloGraph`, and `CaloFlow teacher` belong to this group. The distillations `CaloScore distilled` and `CaloScore single-shot` for a line to smaller generation times at the expense of a little shower quality, as we had seen in the individual metrics before. On the other side, we have submissions with lower log-posterior score, but a much

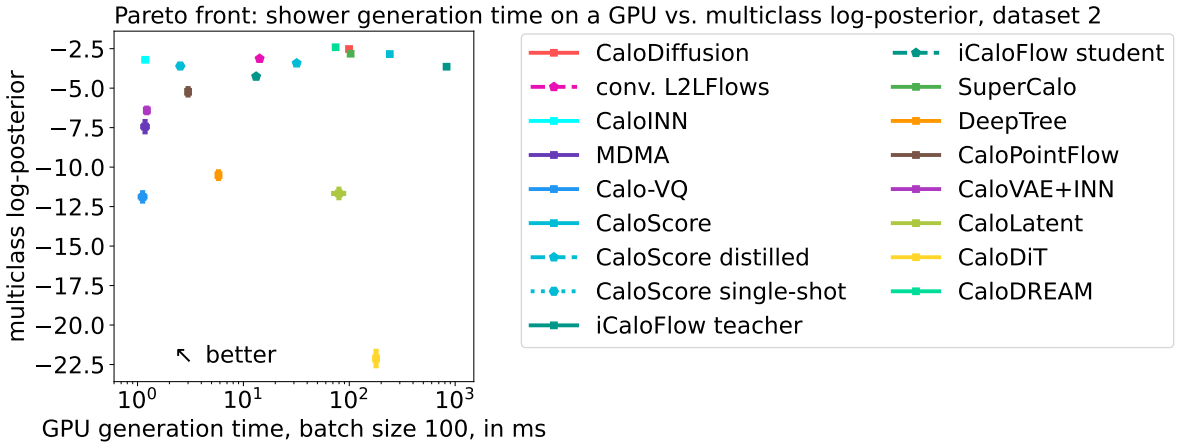



Figure 114: Pareto front in sample quality (from figure 69 and table C17) and generation speed (from figure 73 and table C22).

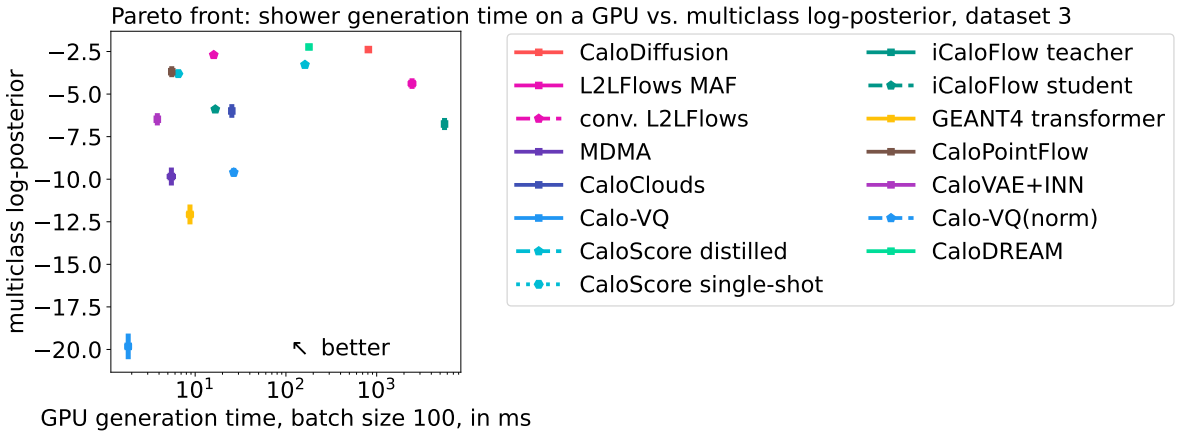



Figure 115: Pareto front in sample quality (from figure 85 and table C25) and generation speed (from figure 89 and table C30).

faster generation time per shower. The VAE-based submissions `CaloMan`, `Calo-VQ`, and `CaloVAE+INN` belong to this group. In the corner with the best scores of both dimensions, we have the Normalizing Flow-based submission `CaloINN`. Also `CaloFlow student` is close, indicating that the low-dimensional data of dataset 1 – photons can be described well with normalizing flows and that a good choice for the architecture can also make the generation fast.

With ds $1 - \pi^+$ in figure 113, the situation is similar than with ds $1 - \gamma$, given that it has comparable dimensionality. We again observe a cluster of submissions in the top-right at good shower quality and large generation time. Again, these are the diffusion models `CaloDiffusion`, `CaloGraph`, the MAF-based normalizing flow of `CaloFlow teacher`, and the GAN of `CaloShowerGAN`. Much faster, but also worse in quality we again see VAE and GAN-based models of `Calo-VQ`, `BoloGAN`, and `CaloMan`. In the corner of fast generation of good showers, we see four submissions that actually

form a line, making it easy to give them an order towards better showers at shorter times: From worst to best, these are CaloFlow student, CaloVAE+INN, CaloINN, and DNNCaloSim. Out of these, we have two normalizing flow-based submissions with CaloINN and CaloFlow student— as we had before for the photon dataset. However this time, we also have two VAE-based submissions at the Pareto front: DNNCaloSim and CaloVAE+INN. This indicates that the larger shower-to-shower variability of pion showers is better captured by VAEs as the rather uniform photon showers we had before.

Dataset 2 now increases the dimensionality of the samples by an order of magnitude. The Pareto front in figure 114, however, does show similar features as we have seen for dataset 1 before. There is a group of diffusion and normalizing flow-based submissions in the top right with very high log-posterior scores, but also rather big generation times per shower. In this group, we have CaloDREAM, CaloDiffusion, CaloScore, iCaloFlow, SuperCalo, and conv. L2LFlows. At the other end of the spectrum, we have again fast submissions with worse log-posterior scores. In this group we have GAN-based submissions MDMA and DeepTree, and VAE-based submissions Calo-VQ and CaloVAE+INN. In the corner with both scores being good, we have three submissions: CaloPointFlow, CaloScore single-shot, and CaloINN from "worst" to best. So also for this dataset, the normalizing flow-based submissions have the best trade-off between shower quality and generation speed.

In dataset 3, the Pareto front in figure 115 is a little more diffuse, with the individual groups more spread out and no single submission in the best corner. Nevertheless, the similar general trends than before also apply. Diffusion models like CaloDiffusion and CaloScore distilled, the CFM model CaloDREAM, and normalizing flow-based submissions L2LFlows-MAF and iCaloFlow teacher have good shower quality, but need longer to generate the showers. VAE and GAN-based submissions Calo-VQ, Calo-VQ(norm), Geant4-Transformer and MDMA are much faster in generation, but at the expense of shower quality. In the top-left corner, we see the remaining submissions. While CaloClouds and iCaloFlow student are outperformed by CaloVAE+INN, CaloScore single-shot, CaloPointFlow and conv. L2LFlows they still show a decent trade-off of quality and generation speed. The latter group now forms the Pareto front. The fastest among them is CaloVAE+INN. With a better shower quality, but at slightly bigger generation time, we have CaloPointFlow and CaloScore single-shot almost at the exact same spot, just slightly slower. Slowest of these four, but best in quality, is conv. L2LFlows. For this high-dimensional dataset finding the optimal point really influences the choice of generative architecture, since this group consists of normalizing flows, a diffusion model, and a VAE.

## 11. Conclusions and Outlook

In this document, we summarize the results of the Fast Calorimeter Simulation Challenge 2022. We present a broad survey of state-of-the-art generative AI architectures on four different calorimeter shower datasets with dimensionalities ranging

from a few hundred to a few tens of thousand voxels. The data has a few physics-specific characteristics, like a high degree of sparsity, energy depositions in voxels spanning several orders of magnitude, and correlations between voxels across several layers, that are not present in natural images and other datasets. With about 15 submissions per dataset, and at least one submission for each type of generative architecture (GAN, VAE, Normalizing Flow, Diffusion, and Conditional Flow Matching) per dataset, this document provides the most detailed and complete survey of generative AI for high-energy physics.

First announced in February 2022, the challenge quickly motivated the first publications using the dataset with `CaloScore` [50, 51], `CaloFlow` [34], and `CaloMan` [60]. More followed and were presented at the ML4Jets conference in November 2022 at Rutgers [17]. While we first planned to close the challenge with the dedicated meeting in Frascati [19] in May 2023, we saw a constant interest in the challenge with new submissions being presented at ML4Jets in Hamburg in November 2023 [18]. In total, we have received 59 submissions, sampled from 31 models, from 23 collaborations consisting of researchers from the theory and the experimental communities, as well as from outside academia. By now most of the submissions have been published by physics journals or ML conferences, highlighting the high quality of the individual works.

While the main focus of this challenge was on generative models for calorimeter showers, with the requirements of the (HL)-LHC and future colliders in mind, many of the results will likely translate to other domains in high-energy physics in which generative AI is used as well, such as generative unfolding [208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220], modeling of hadronization effects [221, 222, 223, 224, 225], end-to-end simulations like flashsim [226], or anomaly detection with generative aspects [227, 228, 229].

### *11.1. Overall Physics Results*

Since fast simulation frameworks are in the ideal case faithful, fast, and light-weight, it was expected that with such ambitious objectives there will be no clear winner of the CaloChallenge. Instead, our goal was to create a survey of different generative architectures, their advantages and disadvantages, and especially their scaling behavior when increasing the dimensionality of the dataset. Ultimately, the objectives of experiments will differ, with some in need of high fidelity simulation, others prioritizing the speed, trading off physics accuracy to a certain extent.

For low-dimensional datasets, *i.e.* dataset 1 – photons with 368 voxels, we saw that diffusion models like `CaloDiffusion` [44] and Normalizing Flow-based models like `CaloINN` [38] have the best quality, meaning they reproduce the GEANT4 distribution most faithfully. The diffusion model has a rather small number of trainable parameters, so it's also more lightweight than the normalizing flow, but since generation requires multiple steps and calls to the neural network, the diffusion model is much slower in generation. The invertible architecture of `CaloINN` does not require multiple calls to

the same neural network, it also avoids a more resource-consuming distillation step like `CaloFlow`, making `CaloINN` an optimal submission for ds $1-\gamma$, see figure 112. GAN and VAE-based architectures are in general also fast in generation, but do not produce high-quality showers, making them less favorable if high fidelity is top priority.

For dataset 1 – pions, the situation is very similar. With 533 voxels, the dataset is still relatively low-dimensional, so normalizing flows and diffusion models, namely `CaloINN` [38] and `CaloDiffusion` [44] show again a good performance. In addition, the VAE-based submission `DNNCaloSim` [62, 63] showed great performance in many of the quality metrics. The statement regarding model size and generation speed of dataset ds $1-\gamma$ also applies here. The diffusion model does not need a lot of trainable parameters, which makes it very lightweight. Generation speed, however, is lower due to the subsequent denoising steps in generation. The normalizing flow, on the other hand, is growing at least linearly in size with the dimensionality of the dataset, so it is now already about 1.4 times bigger than for the photons. Nevertheless, it is still very fast in generation, and, at least on the GPU, only marginally behind the well-performing submission `DNNCaloSim`. This VAE-based model is the fastest in generation on the CPU for all batch sizes and is only beaten marginally on the GPU for very large batch sizes. It is midrange in terms of model size, with some GANs having fewer parameters and most normalizing flows having more. It is also interesting to note that `DNNCaloSim` had the best scores in the low-level binary AUC and the multiclass log-posterior, but had worse scores for KPD and FPD, as well as the separation powers we looked at. In these latter cases, `CaloDiffusion`, `CaloFlow student` [34], and `CaloGraph` [54] performed better than `DNNCaloSim`.

Dataset 2 now increases the dimensionality by an order of magnitude to 6480. This also increases the number of parameters in the so-far well-performing normalizing flow of `CaloINN` [38] by an order of magnitude to about $2.7{\cdot}10^8$, making it the largest submission for dataset 2. Nevertheless, it still gives the best trade-off in quality and generation speed in figure 114. In terms of quality alone, the diffusion models `CaloDiffusion` [44] and `CaloScore` [50, 51] as well as the conditional flow matching model `CaloDREAM` [69] have better multiclass log-posteriors, KPD/FPD, and binary AUCs. However, in generation all of these require multiple steps and hence they are slower than `CaloINN`. With the use of distillation, `CaloScore` was able to speed up generation generation times by an order of magnitude to `CaloScore distilled` and another order of magnitude to `CaloScore single-shot` [50, 51] at the expense of a little shower quality. Similar techniques can also be applied to `CaloDiffusion` and `CaloDREAM`, which would bring them closer to `CaloINN` in figure 114. In terms of model size, `MDMA` [23, 24] needed by far the fewest parameters, making it also the fastest in generation, especially for small batch sizes.

Dataset 3 increases the complexity of the showers by another order of magnitude, to 40500 voxels. This was too big for the bijector in `CaloINN` [38], so it was not submitted to this dataset. Diffusion and conditional flow matching models `CaloDiffusion` [44], `CaloScore distilled` [50, 51], and `CaloDREAM` [69] show again the best shower quality, but not the fastest generation. Splitting the entire shower into individual calorimeter

layers makes the problem again manageable for a normalizing flow, as can be seen by the good shower quality of `conv. L2LFlows` [32]. In terms of model sizes, `MDMA` [23, 24] again is the smallest submission, followed by the VAE-based model `Calo-VQ` [58] and its variant `Calo-VQ(norm)` [58] and then `CaloDiffusion` and `CaloDREAM`. In terms of generation speed, GAN-based submission `MDMA` and VAE-based submissions `Calo-VQ` and `CaloVAE+INN` [38] are the fastest, which is correlated to the model sizes. When looking at the trade-off between quality and speed in figure 115, we see four submissions competing with each other. Fastest, but worst in quality of those four is `CaloVAE+INN`. In the center, we have `CaloPointFlow` [42] and `CaloScore single-shot` [50, 51], and slowest, but best in quality, is `conv. L2LFlows`. The potential speed-up of `CaloDiffusion` and `CaloDREAM` with model distillation, as discussed at the end of the dataset 2 paragraph, also applies here.

Summarizing, there is no single submission that excels in all three types of metrics: speed, quality, and size. Normalizing Flows show the best trade-off in sample quality and generation speed, but since they have to train a bijective mapping they do not scale well to higher dimensional datasets. Diffusion and conditional flow matching models have the highest sample quality, but suffer from a slow generation process. GAN and VAE-based submissions have fewer trainable parameters and are usually very fast in generation, but that comes at the expense of shower quality. Even the best performing model is not perfect for the high-dimensional datasets 2 and 3, so there is still a lot of room for improvement in generative architectures to be even more faithful and resource-efficient in the future.

Model distillation improves speed at expense of quality and we have seen some submissions that use this technique already, while it could be applied to others, too. Techniques like weight quantization or node pruning can have a large effect on the resource requirements with some or little effect on the sample quality. This has not been studied here and should be investigated more in the future.

### *11.2. Take-aways of the CaloChallenge beyond Detector Simulation*

This challenge triggered the development and adaptation of a lot of generative architectures to high-dimensional calorimeter shower data, leading to more than 20 publications in physics and ML journals or conferences, as well as talks at the central machine learning conference in particle physics, ML4Jets [17, 18] and other specialized workshops [19]. This collaborative effort was done by experimentalists, theorists, and scientists working outside academia in industry alike, but mostly outside of the big collaborations ATLAS and CMS. We hope that the presented results are useful for the experiment-specific development of fast simulation frameworks in the future.

The challenge also provided four datasets that will now serve as benchmarks for future generative models. Despite the large body of results we reported, there are a few questions that this challenge cannot answer. For example, some submissions (`CaloPointFlow` [42], `MDMA` [23, 24], `DeepTree` [28, 29], and `CaloClouds` [46, 47]) worked

with point clouds instead of with the voxelized data that we provided. Since they did not have access to the hits that GEANT4 simulated before we voxelized the data, they had to rely on suboptimal methods to create the point clouds. Further studies that directly use the point clouds coming from GEANT4 are needed to understand if that had an effect on shower quality.

Something else we noticed but were unable to disentangle and study in detail was the effect of model distillation. `CaloScore distilled` and `CaloScore single-shot` were distilled from `CaloScore` and `CaloFlow student` and `iCaloFlow student` were distilled from `CaloFlow teacher` and `iCaloFlow teacher` respectively. Since both, samples from the original model and samples from the distilled version of the same original model were submitted, there is a correlation between the scores of these submissions. This is most visible in the multiclass classification metric, where original and distilled model were sometimes confused with each other (see for example figure B3). We also see it in the Pearson correlation coefficients of layer energies in figure 66 and figure 82, where the distinct pattern of `CaloScore` got worse with distillation. The situation is, however, different for i`CaloFlow`, where the pattern got fainter with distillation. Other metrics were also sometimes better, sometimes worse in distilled versions, as previously seen also in [117]. We suspect that a smoothing that takes place in distillation can improve an incorrectly learned feature. One way of disentangling such effects would be to train multiple instances of the original model and use one for sample generation of the submission and the other one for training the distilled model.

The irregular geometry of datasets 1 posed a special challenge, in particular for models that were using 3-dimensional convolutions. While for example `conv. L2LFlows` decided not to work on ds $1-\gamma$ and ds $1-\pi^+$ for that reason, `CaloDiffusion` came up with a special solution to the problem. It also triggered some dedicated approaches for irregular geometries, like for example `CaloGraph`.

In addition, we also gained more insights in the evaluation of generative models for physics applications. We studied how different quality metrics, motivated by physics or coming from computer science, correlate with each other.

### *11.3. Outlook to the Future*

To serve as a benchmark for future developments in calorimeter simulation, we collected the raw data that went into all the figures of section 9 and tables in section C in a pandas [230] dataframe that we publish together with the jupyter notebook [231] required to reproduce the figures on the GitHub page of the CaloChallenge [73]. The 59 submitted samples are available for download at [232, 233, 234, 235].

For a better understanding on the resource requirements and best working point in the shower quality *vs.* generation speed trade-off, a full end-to-end implementation in fast simulation frameworks of experiments is needed. Since the generation times improve a lot for generating showers in batches, this should also be taken into account properly. In that sense, the results presented here focus only on one single step of

the full fast simulation chain. The produced showers still need to be projected back into the detector geometry and the generative model needs to be embedded in the appropriate software framework. These additional constraints go beyond the scope of this challenge, but they are required to get a full picture on the impact of generative AI in making the simulation faster. It could therefore very well be that sacrificing a little performance for a better speed, or sacrificing some speed advantage for a better performance is more beneficial when looking at the end-to-end performance. Also, conditioning on more initial conditions, like for example the incident angle *vs.* training more individual models can only be evaluated in a more complete framework. Another practical question that arises is the computing architecture which will run the final fast simulation. While inference clearly benefits GPU utilization and large batch sizes, this must be well incorporated in experiments' computing workflows so that speed-up factors can be maximized. In any case, further studies in all of these directions are therefore needed, and the corresponding results are applicable well beyond (HL)-LHC.

It is also important to stress that while many figures of merit are presented in this work, any experiment should not simply pick a technology but should carry out a careful evaluation of several models. The granularity of the calorimeters, the geometry of the cells in the sub-systems and the overall detector geometry will impose constraints on which models can be used. For example, ATLAS trains and runs 100 models (one per $\eta$ slice) but has a relatively low detector granularity while the CMS high-granularity calorimeter (HGCAL) covers only a small region of the detector but has a much higher granularity. Therefore, ATLAS may struggle to handle hundreds of models with many parameters but will be less affected by poor modeling of the shape, making some of the less performing models better candidates. For the HGCAL the opposite is true, although in this case the complex geometry of the calorimeter may require additional studies for the voxelisation strategy. What needs to be mentioned here is an important work of the LHCb on the implementation of the workflow presented in the Par04 example of GEANT4 into their simulation framework, featuring a Par04-inspired VAE model [236]. This allows them to test any of the models submitted to the CaloChallenge, looking not only at the simulation level observables, but at the broader spectrum of important variables that are typically a part of validation chain.

To summarize, we are very excited to have received so many different submissions to the CaloChallenge. We now have a full toolbox with publicly available models as well as a detailed set of comparisons of several different approaches for experiments and other interested users to try out. It will be highly exciting to see how these methods evolve in the future and how they are deployed in experiments, expanding our understanding of Nature by improved simulation techniques!

**Acknowledgments**

We thank James Tuan for the IT support at Rutgers University, where the intermediate evaluations and preparations were run. The main computational results presented were

obtained using the CLIP cluster (`https://clip.science`).

Oz Amram is supported by the U.S. CMS Software and Computing Operations Program under the U.S. CMS HL-LHC R&D Initiative. Erik Buhmann, Thorsten Buss, Gregor Kasieczka, and William Korcari are supported by the Deutsche Forschungsgemeinschaft under Germany's Excellence Strategy — EXC 2121 Quantum Universe — 390833306 and via the KISS consortium (05D23GU4, 13D22CH5) funded by the German Federal Ministry of Education and Research BMBF in the ErUM-Data action plan. Erik Buhmann further acknowledges funding through a scholarship by the Friedrich Naumann Foundation for Freedom. The work of Florian Ernst has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research (grant no. 13E18CHA). Michele Faucci Giannelli received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 754496. Luigi Favaro and Ayodele Ore are supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762 – TRR 257: *Particle Physics Phenomenology after the Higgs Discovery*. Benno Käch is funded by Helmholtz Association's Initiative and Networking Fund through Helmholtz AI (grant number: ZT-I-PF-5-64). Dmitrii Kobylianskii, Nathalie Soybelman, Etienne Dreyer, and Eilam Gross are supported by the Israel Science Foundation (ISF), Grant No. 2871/19 Centers of Excellence and BSF-NSF Grant No. 2020780. They are also grateful for the support provided by the collaborative Weizmann Institute and Mohamed bin Zayed University of Artificial Intelligence (MBZUAI) research grant, as well as the Benoziyo Center for High Energy Physics. Marco Letizia acknowledges the financial support of the European Research Council (grant SLING 819789). Qibin Liu and Shu Li acknowledge the support from National Key R&D Program of China (Grant No.: 2023YFA1606904 and 2023YFA1606900), National Natural Science Foundation of China (Grant No.: 12150006), and Shanghai Pilot Program for Basic Research—Shanghai Jiao Tong University (Grant No.: 21TQ1400209). Benjamin Nachman is supported by the U.S. Department of Energy (DOE), Office of Science under contract DE-AC02-05CH11231. Sofia Palacios Schweitzer is supported by the BMBF Junior Group Generative Precision Networks for Particle Physics (DLR 01IS22079). Matthew Buckley, Ian Pang, and David Shih are supported by the U.S. Department of Energy (DOE), Office of Science grant DOE-SC0010008. Kevin Pedro and Oz Amram are supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics. Piyush Raikwar, Dalila Salamani, and Anna Zaborowska were supported by the CERN Strategic R&D Programme on Technologies for Future Experiments and have received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 101004761. Humberto Reyes-Gonzalez is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762 – TRR 257: Particle Physics Phenomenology after the Higgs Discovery. He also acknowledges the support from the Italian PRIN grant 20172LNEEZ. Moritz

Scham is funded by Helmholtz Association's Initiative and Networking Fund through Helmholtz AI (grant number: ZT-I-PF-5-3). Rui Zhang is supported by US High-Luminosity Upgrade of the Large Hadron Collider (HL-LHC) under Work Authorization No. KA2102021.

## A. Histograms of high-level features

Here we show the histograms of the high-level features that were used to compute the separation powers in section 8.1. In particular, we show the distributions of the GEANT4 training and evaluation datasets. The exact same binning was chosen to compute the separation powers with (39).

### *A.1. Dataset 1, Photons (ds 1 – $\gamma$)*

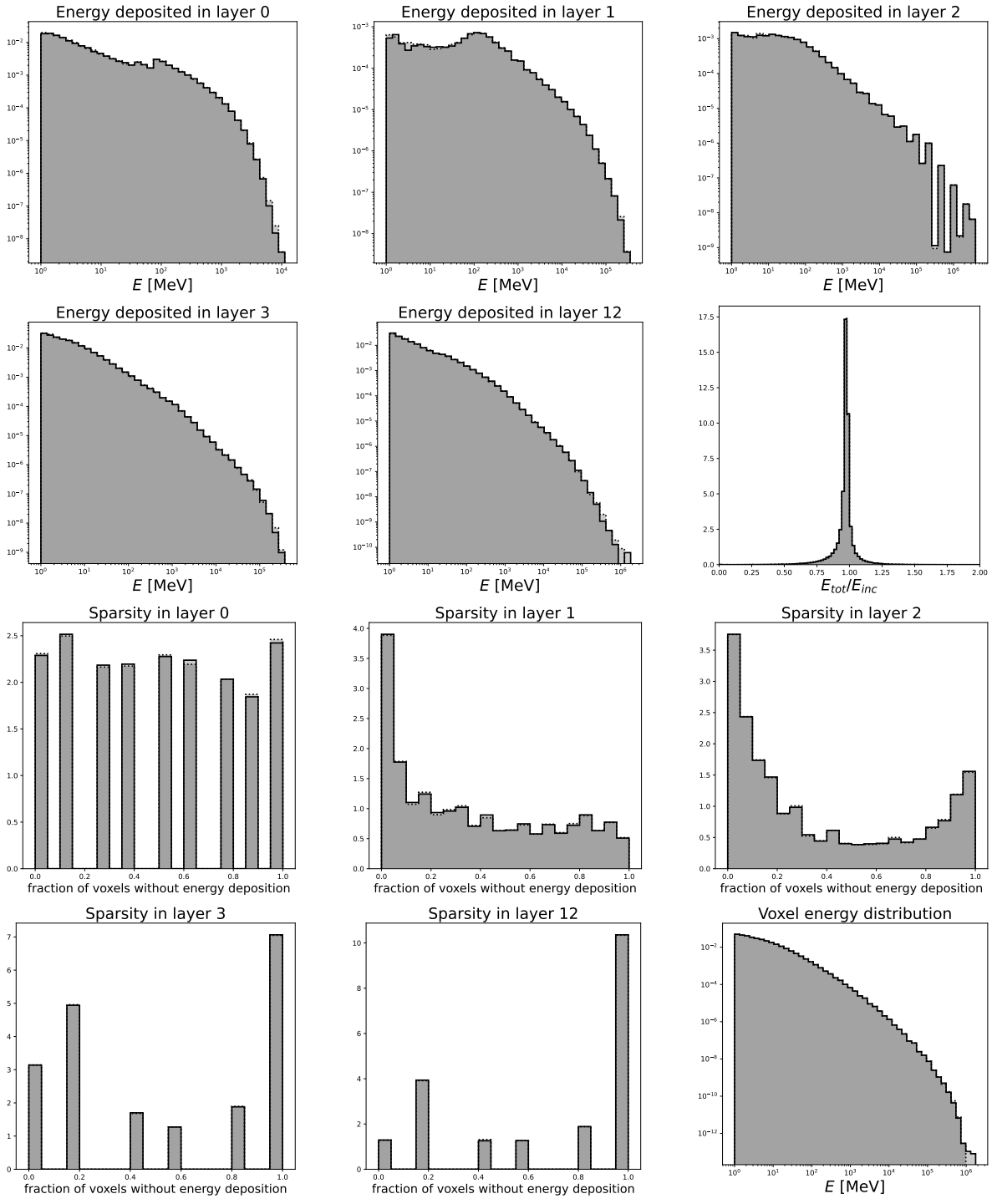



Figure A1: Distribution of GEANT4 training and evaluation data in layer energies $E_i$, ratio of total deposited energy to incident energy, sparsity, and energy per voxel for ds1 — photons.

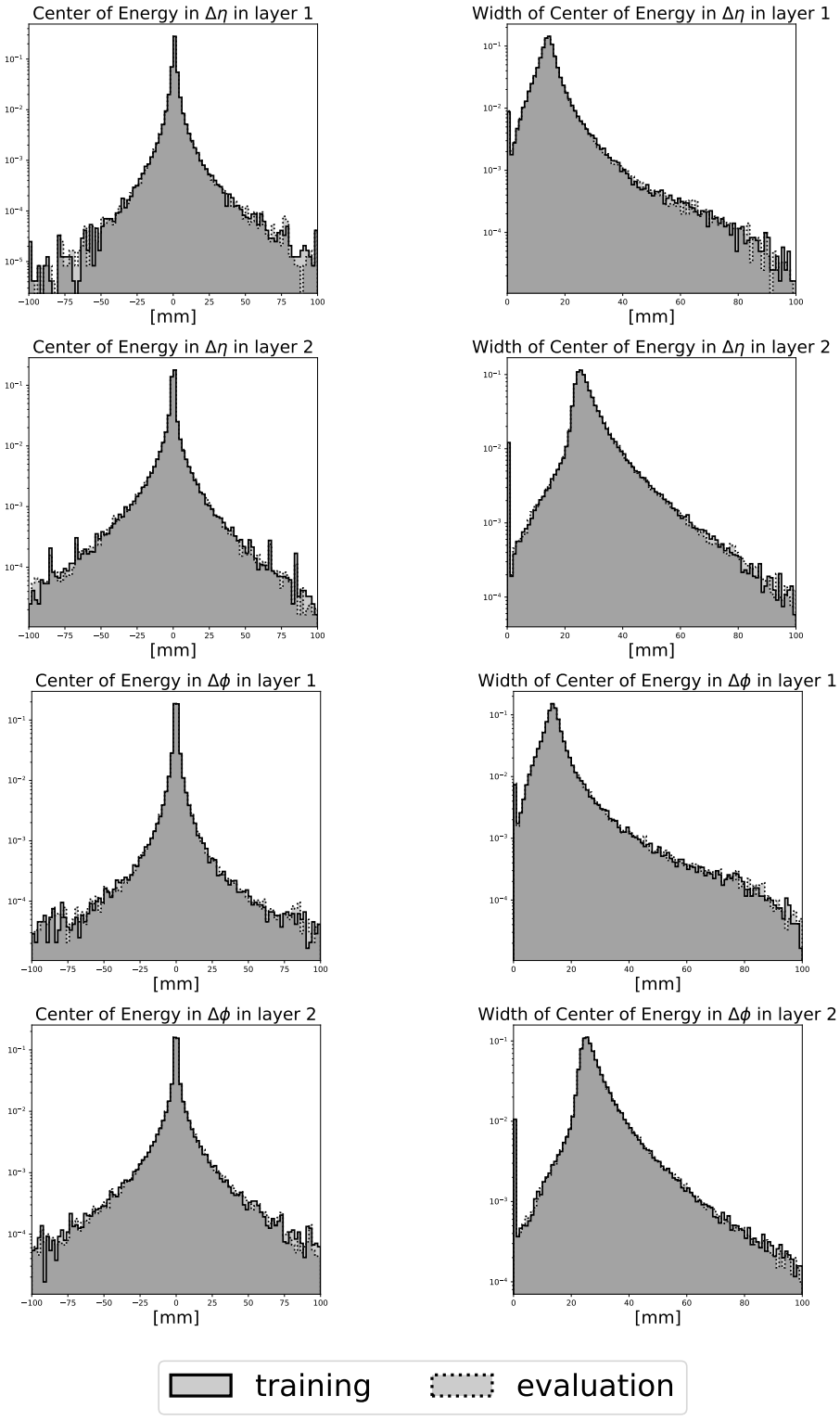



Figure A2: Distribution of GEANT4 training and evaluation data in centers of energy along $\eta$ and $\phi$, as well as the widths of these distributions for ds1 — photons.

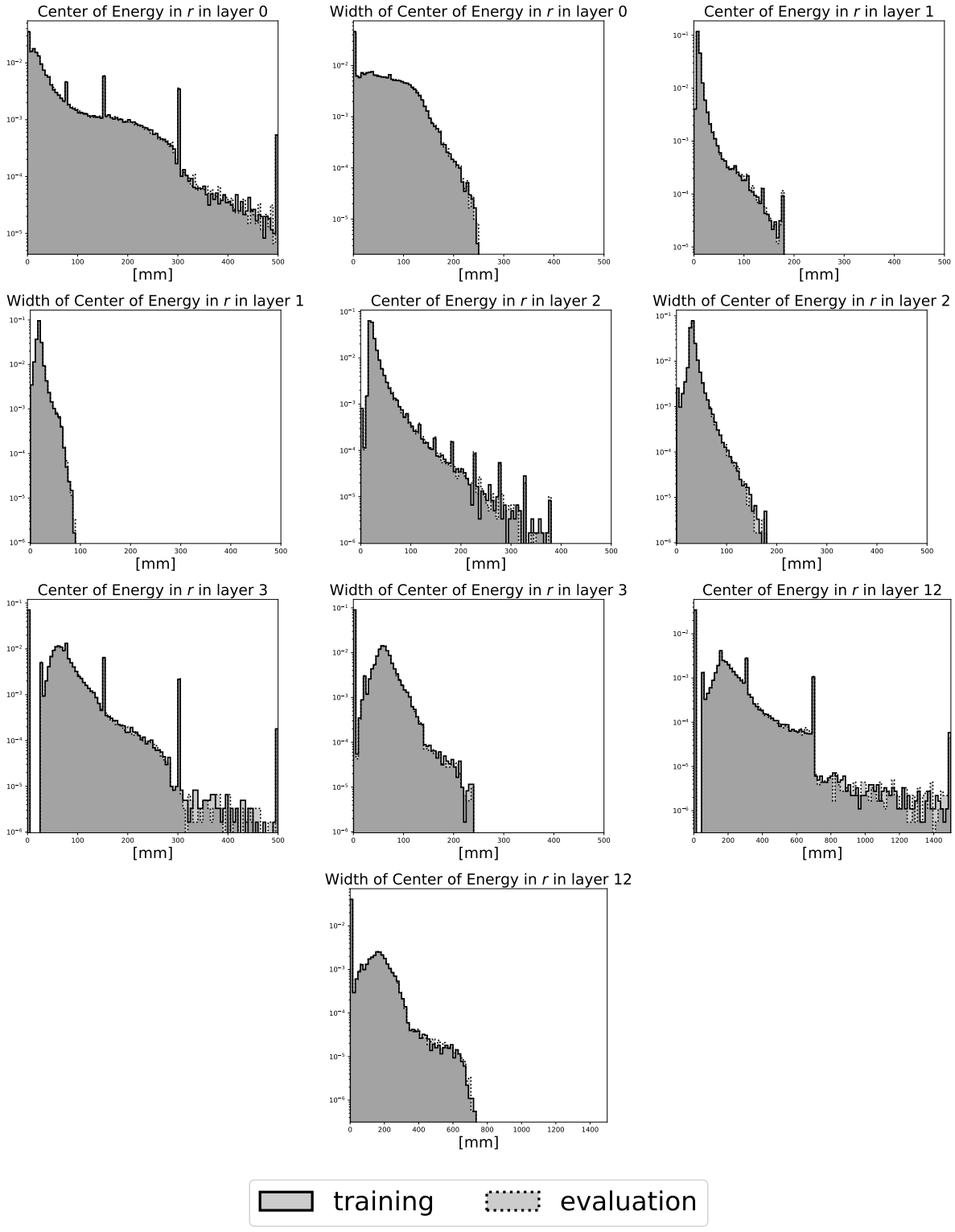



Figure A3: Distribution of GEANT4 training and evaluation data in centers of energy along the radial direction, as well as their widths for ds1 — photons.

*A.2. Dataset 1, Pions (ds 1 – $\pi^+$)*

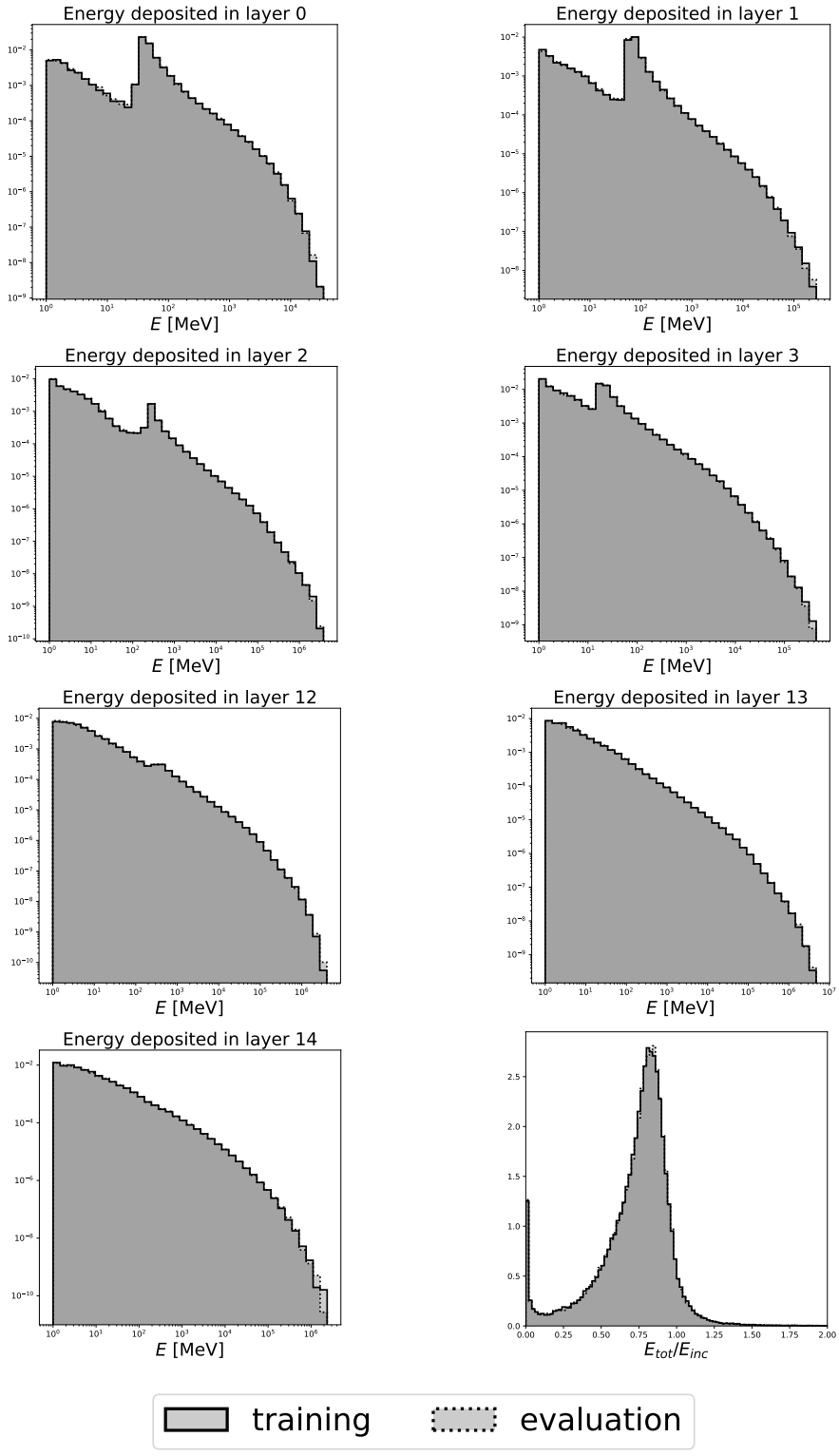



Figure A4: Distribution of GEANT4 training and evaluation data in layer energies $E_i$, and ratio of total deposited energy to incident energy for ds1 — pions.

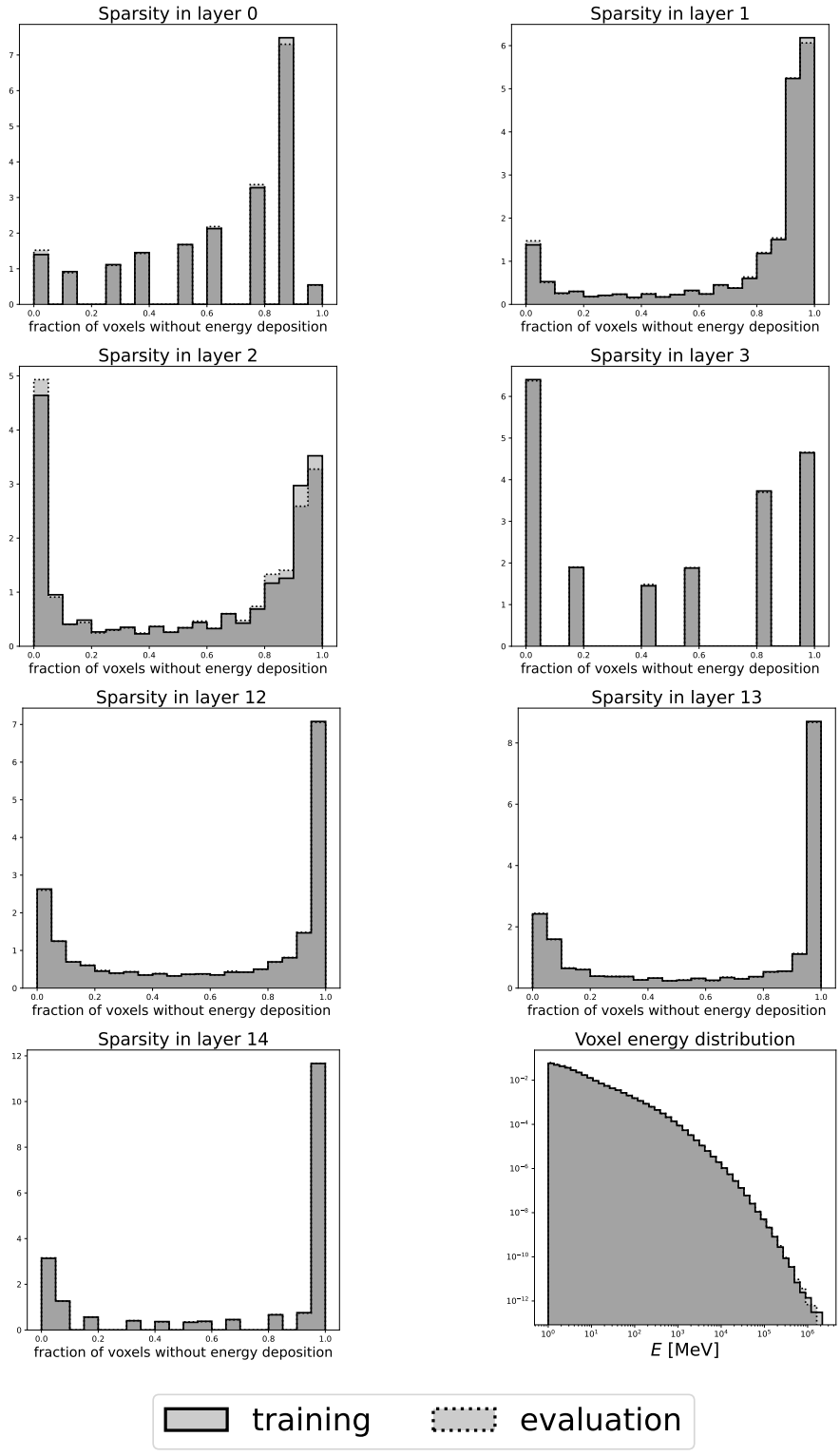



Figure A5: Distribution of GEANT4 training and evaluation data in sparsity and energy per voxel for ds1 — pions.

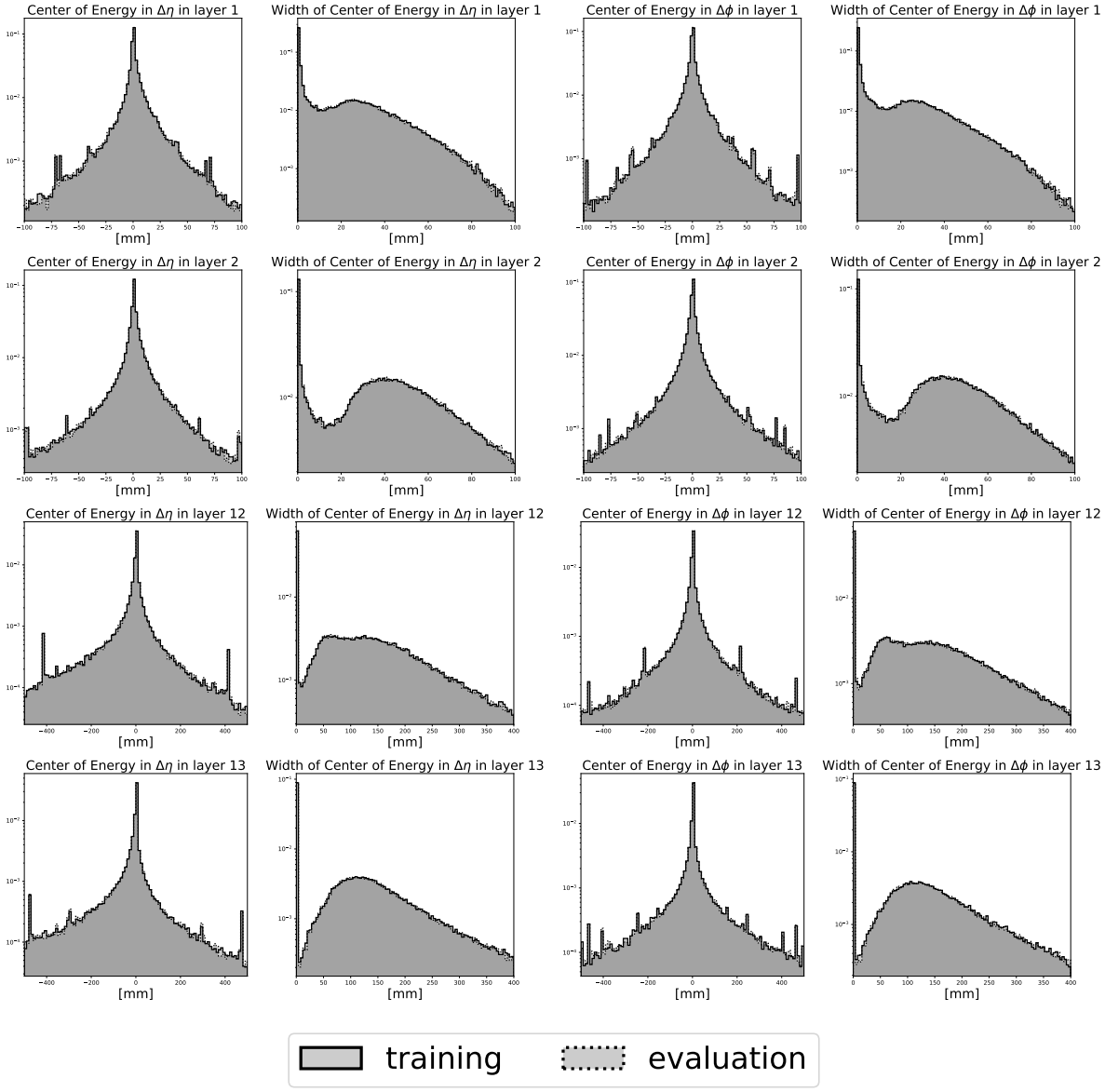



Figure A6: Distribution of GEANT4 training and evaluation data in centers of energy along $\eta$ and $\phi$, as well as their widths for ds1 — pions.

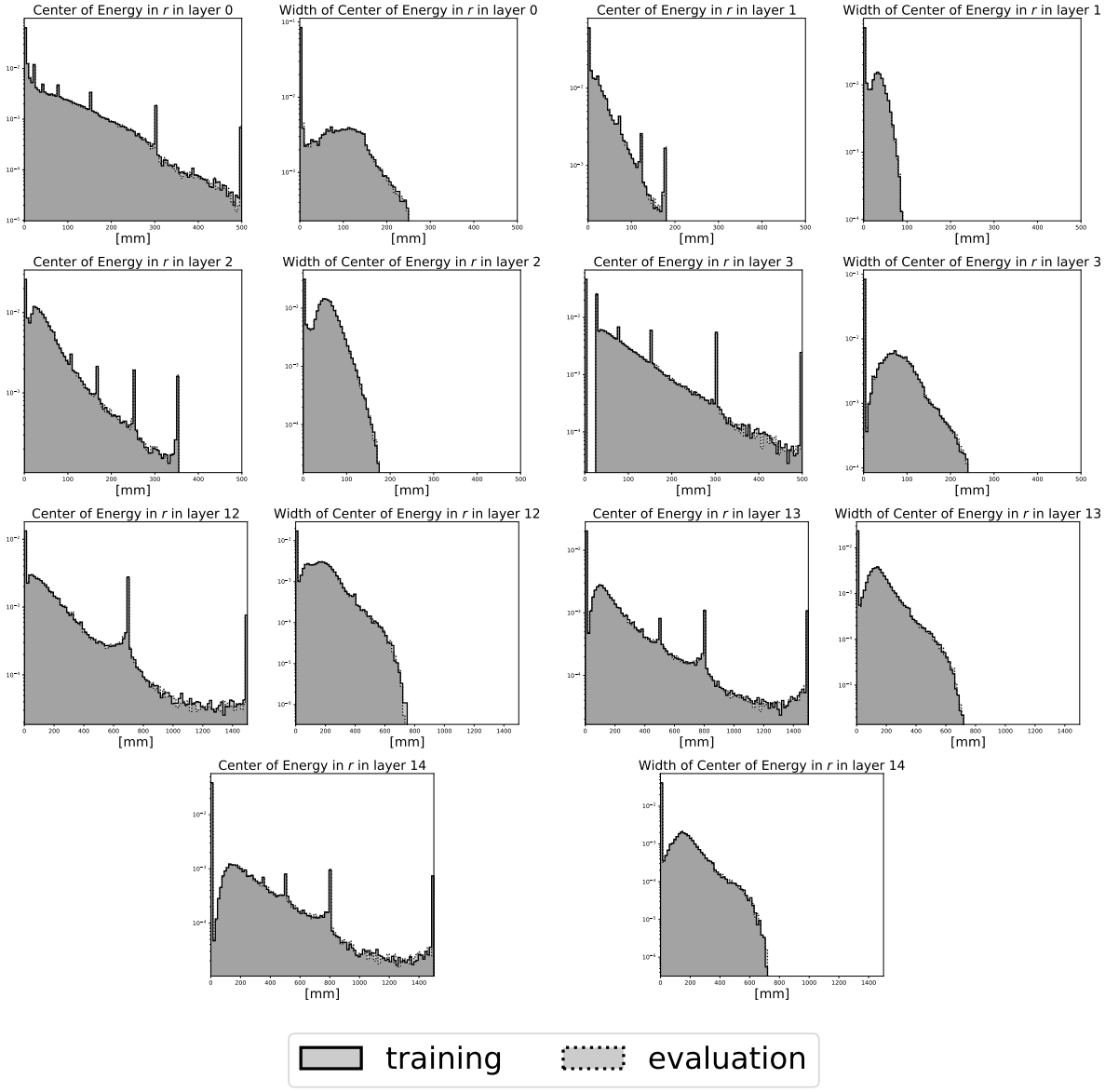



Figure A7: Distribution of GEANT4 training and evaluation data in centers of energy along the radial direction, as well as their widths for ds1 — pions.

*A.3. Dataset 2, Electrons (ds 2)*

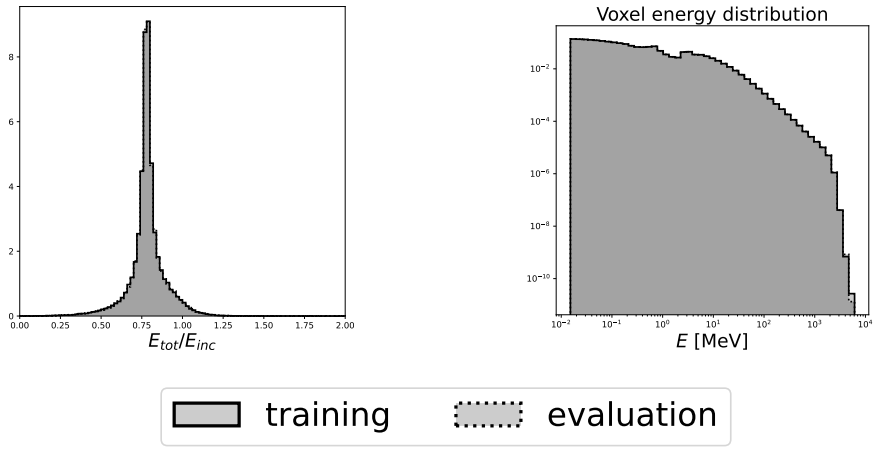



Figure A8: Distribution of GEANT4 training and evaluation data in ratio of total deposited energy to incident energy and energy per voxel for ds2.

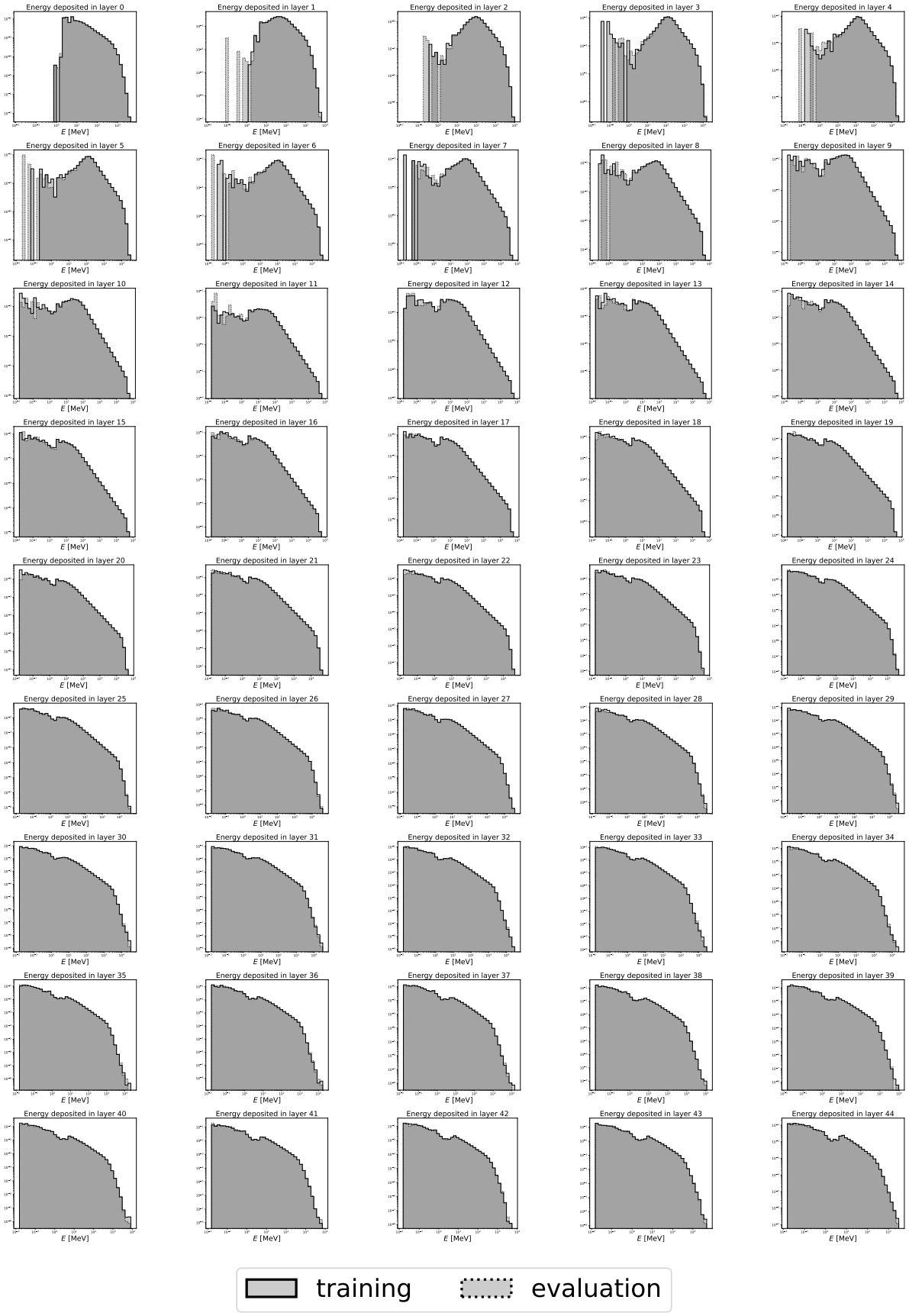



Figure A9: Distribution of GEANT4 training and evaluation data in layer energies $E_i$ for ds2.

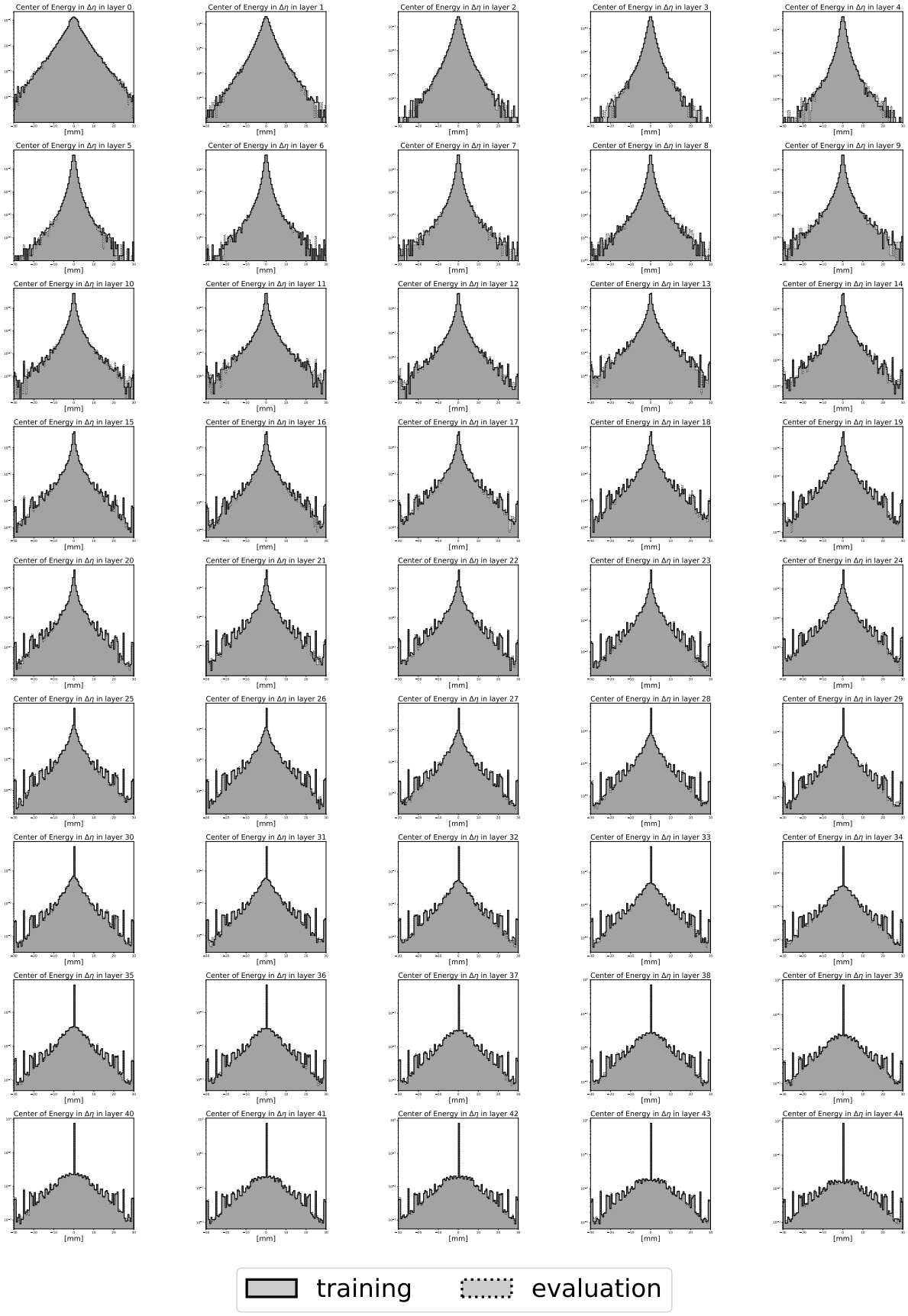



Figure A10: Distribution of GEANT4 training and evaluation data in centers of energy in $\eta$ direction for ds2.

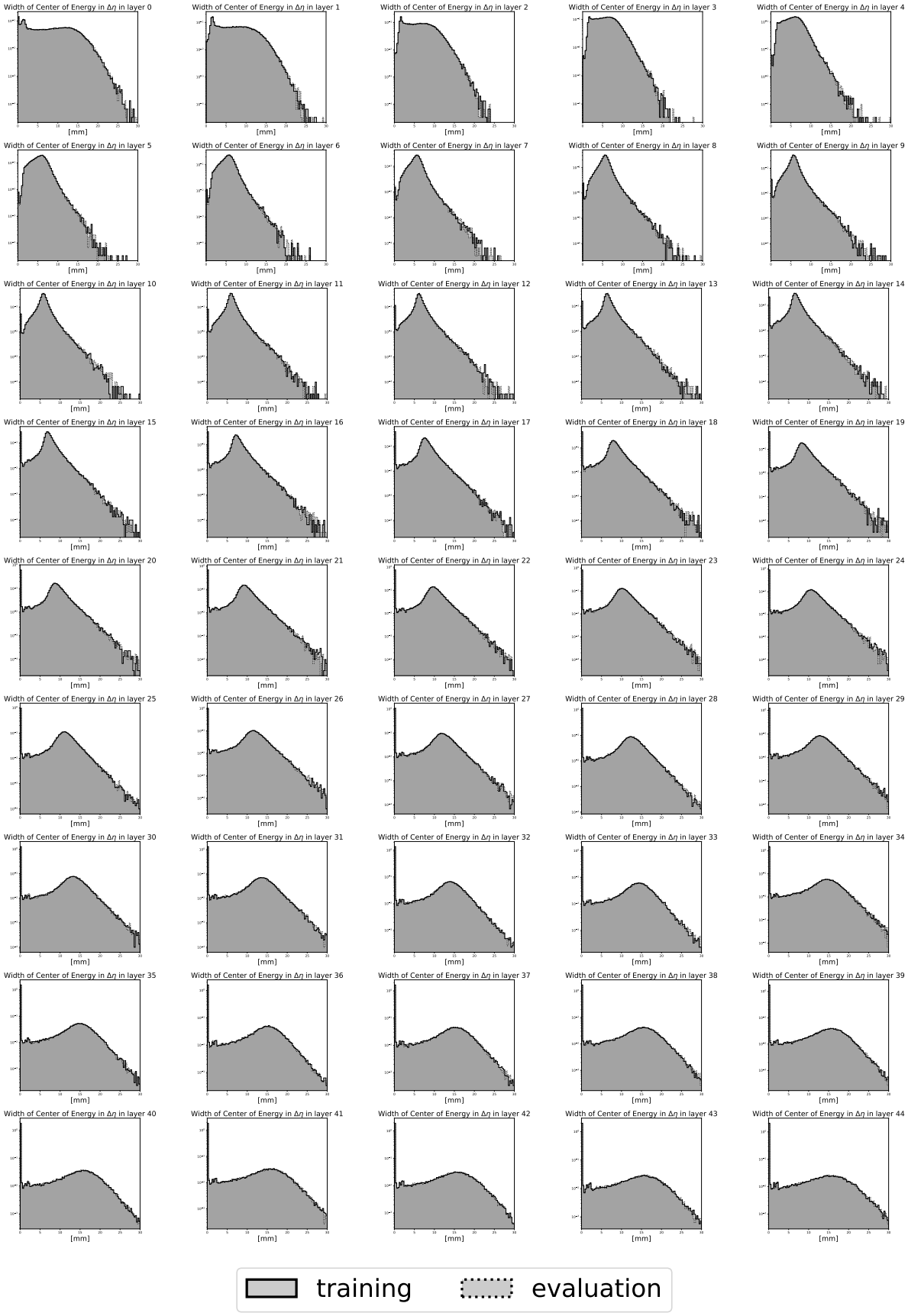



Figure A11: Distribution of GEANT4 training and evaluation data in width of the centers of energy in $\eta$ direction for ds2.

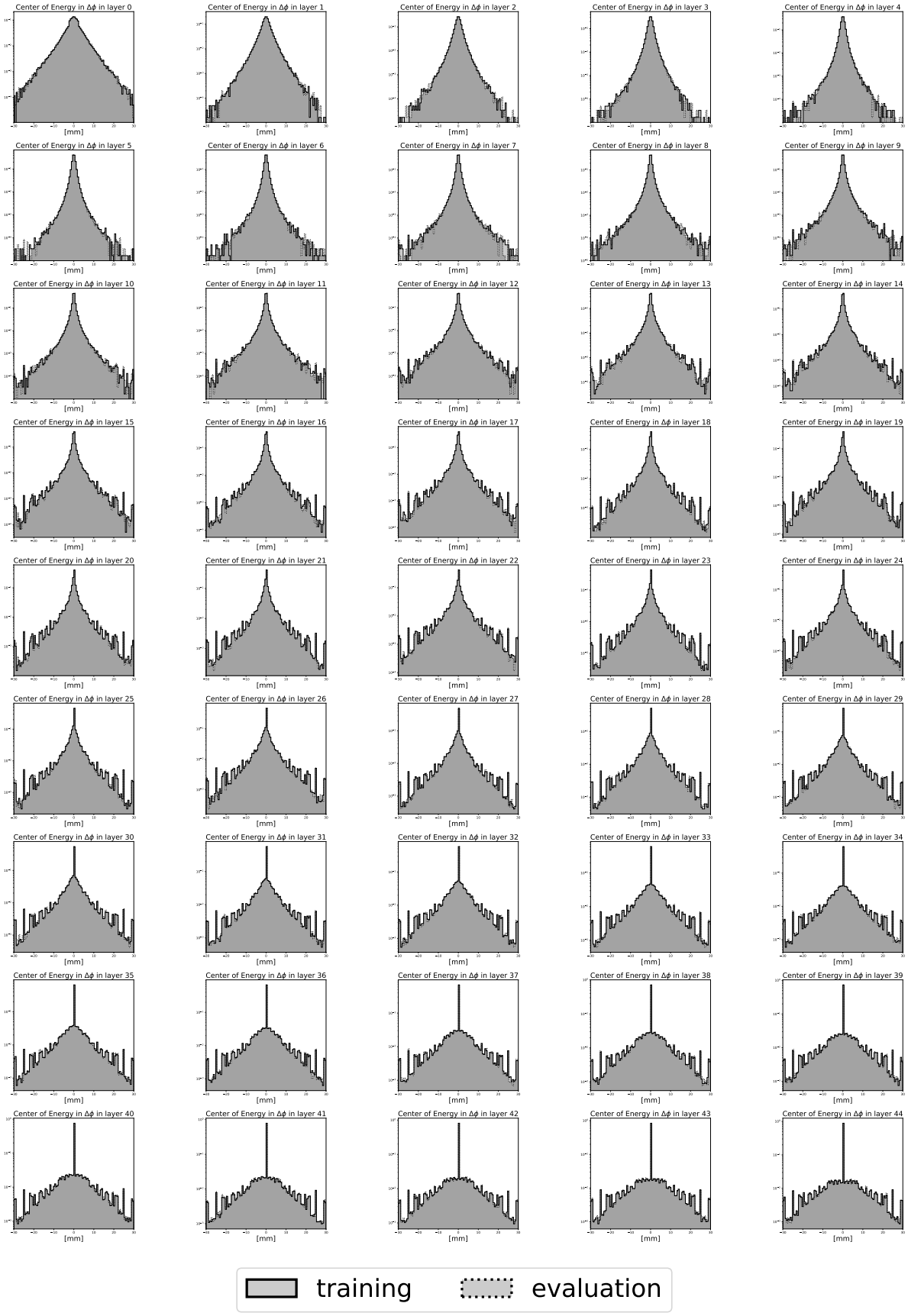



Figure A12: Distribution of GEANT4 training and evaluation data in centers of energy in $\phi$ direction for ds2.

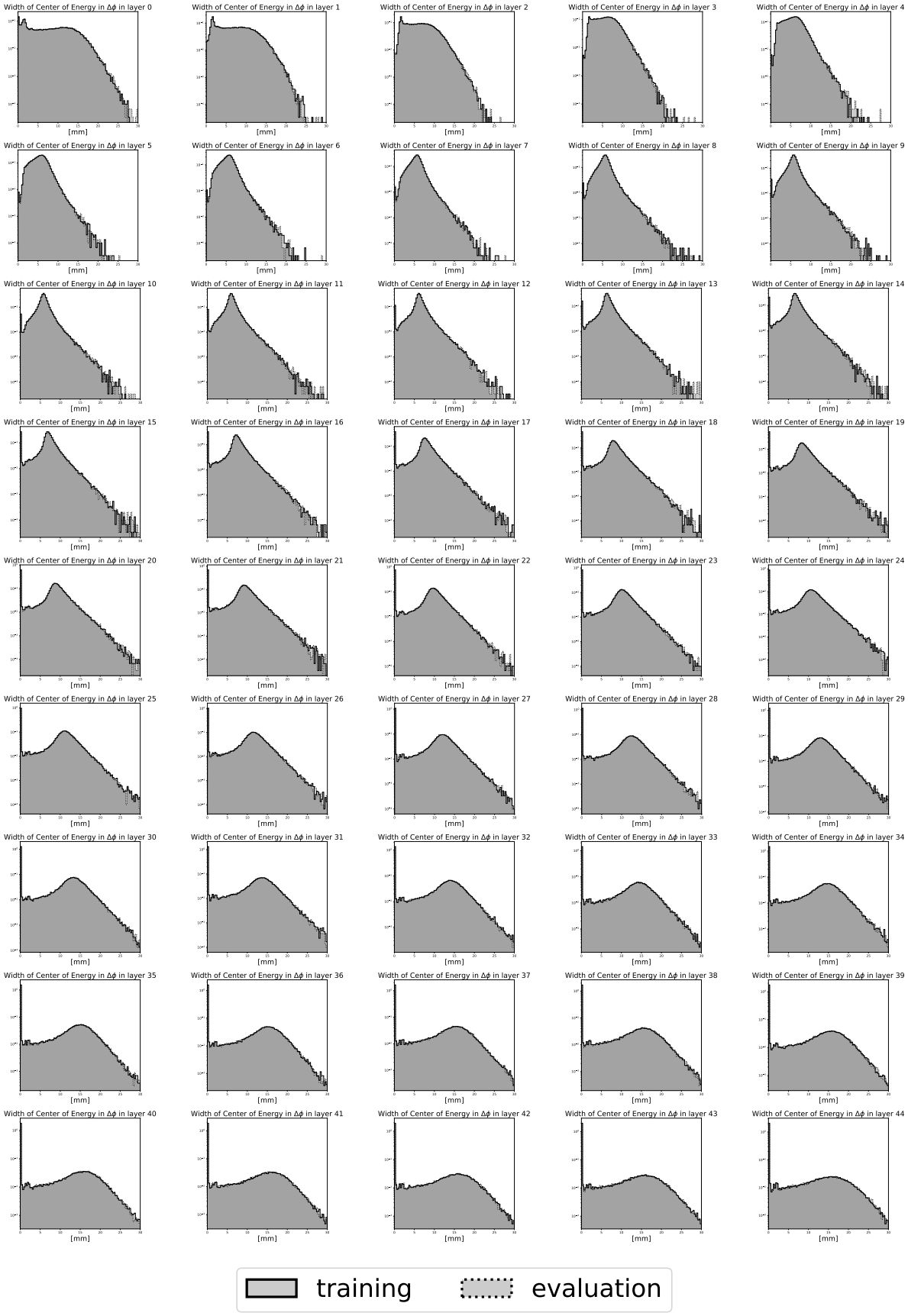



Figure A13: Distribution of GEANT4 training and evaluation data in width of the centers of energy in $\phi$ direction for ds2.

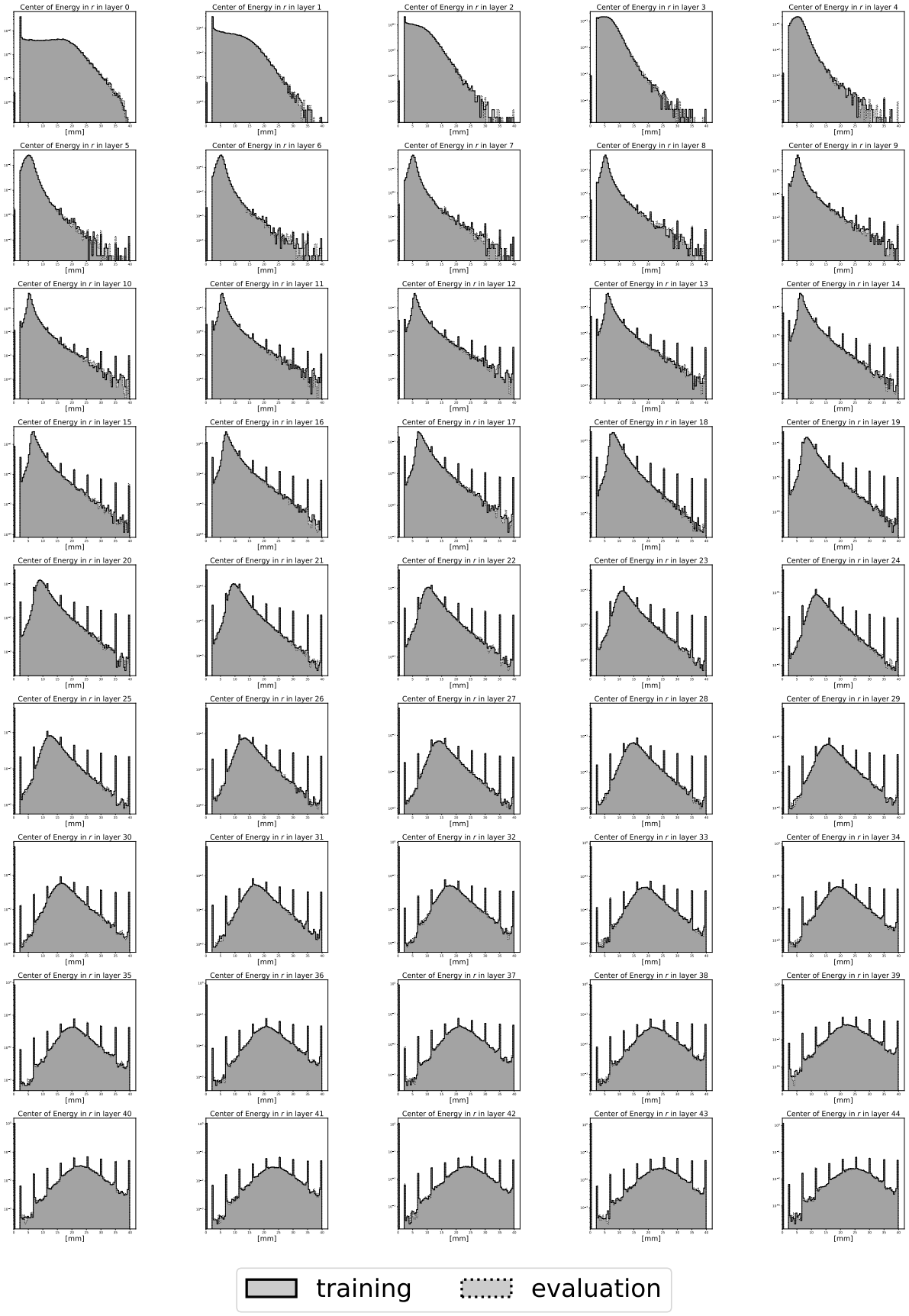



Figure A14: Distribution of GEANT4 training and evaluation data in centers of energy in $r$ direction for ds2.

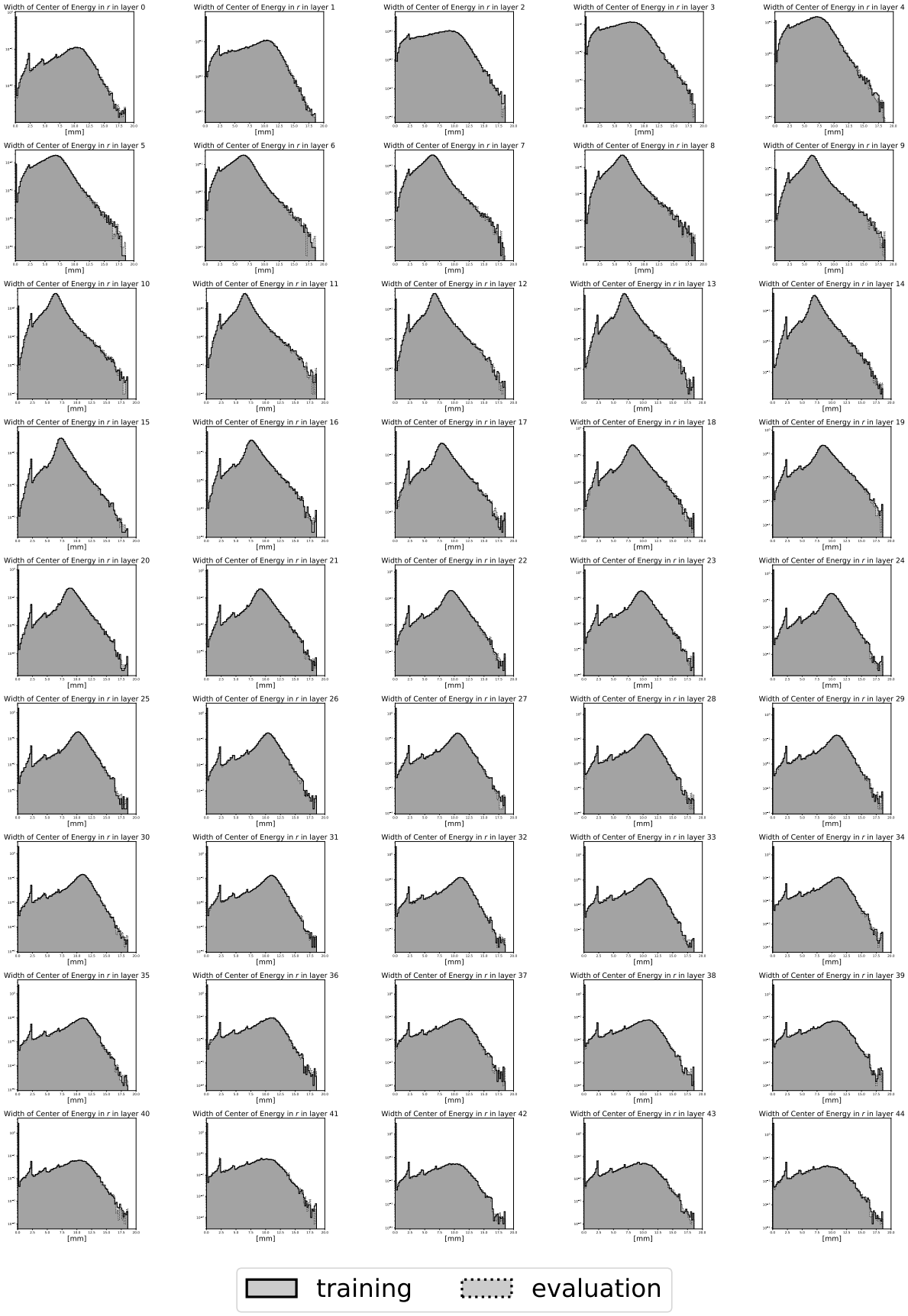



Figure A15: Distribution of GEANT4 training and evaluation data in width of the centers of energy in $r$ direction for ds2.

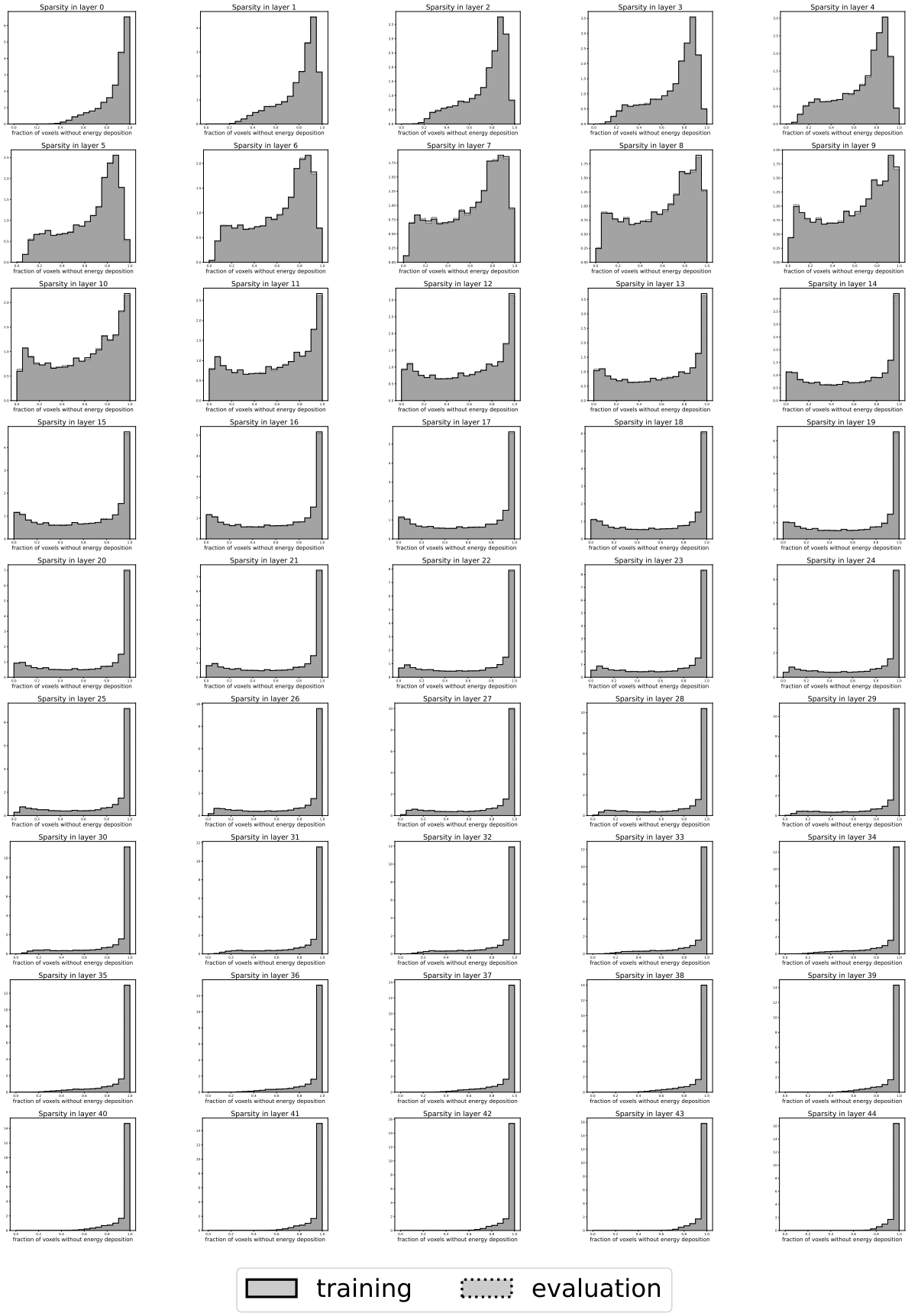



Figure A16: Distribution of GEANT4 training and evaluation data in sparsity for ds2.

*A.4. Dataset 3, Electrons (ds 3)*

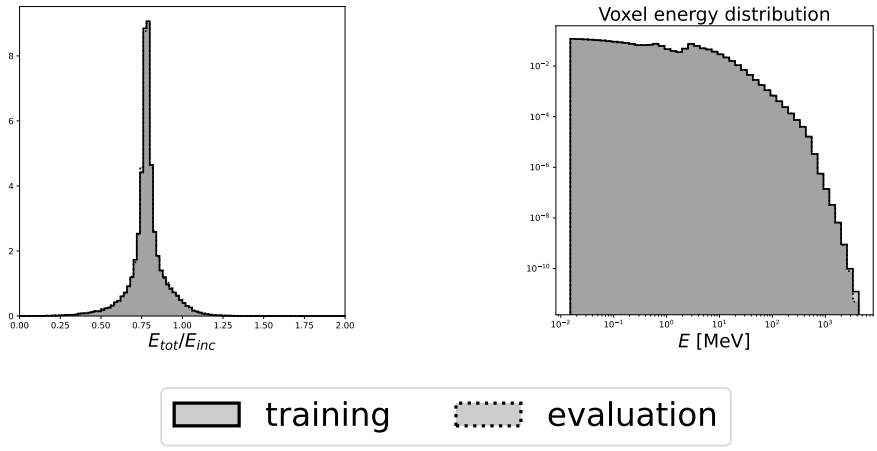



Figure A17: Distribution of GEANT4 training and evaluation data in ratio of total deposited energy to incident energy and energy per voxel for ds3.

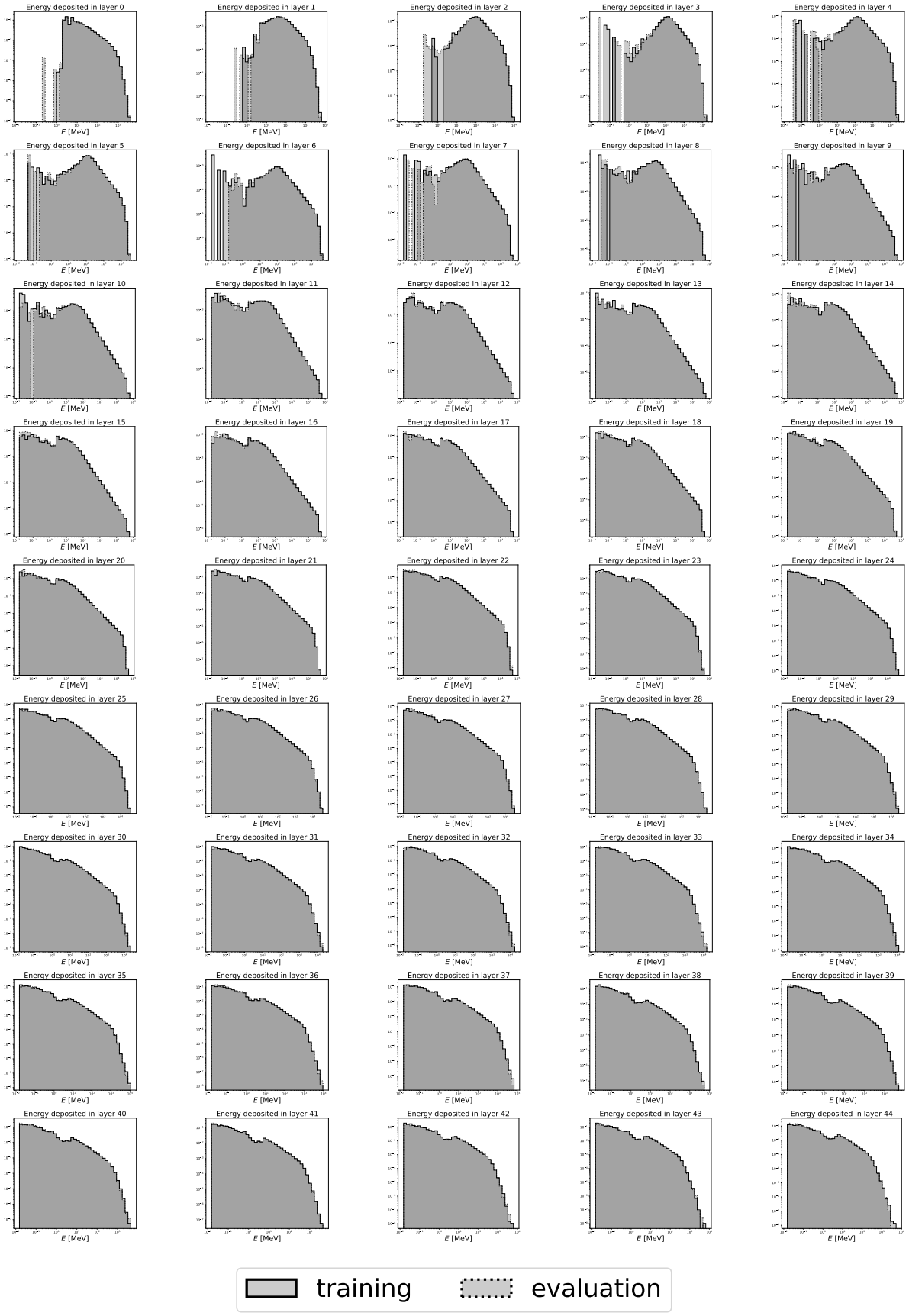



Figure A18: Distribution of GEANT4 training and evaluation data in layer energies $E_i$ for ds3.

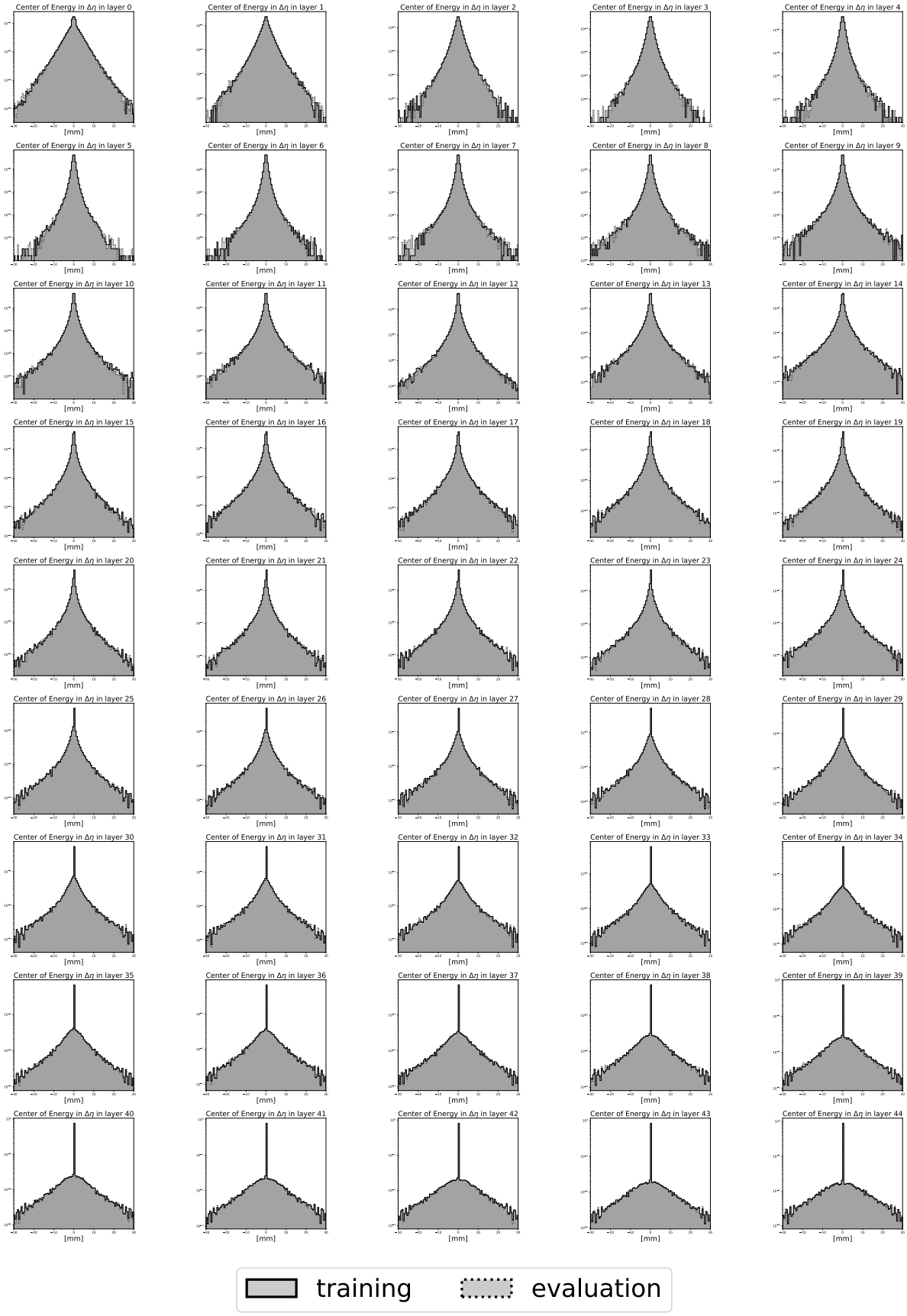



Figure A19: Distribution of GEANT4 training and evaluation data in centers of energy in $\eta$ direction for ds3.

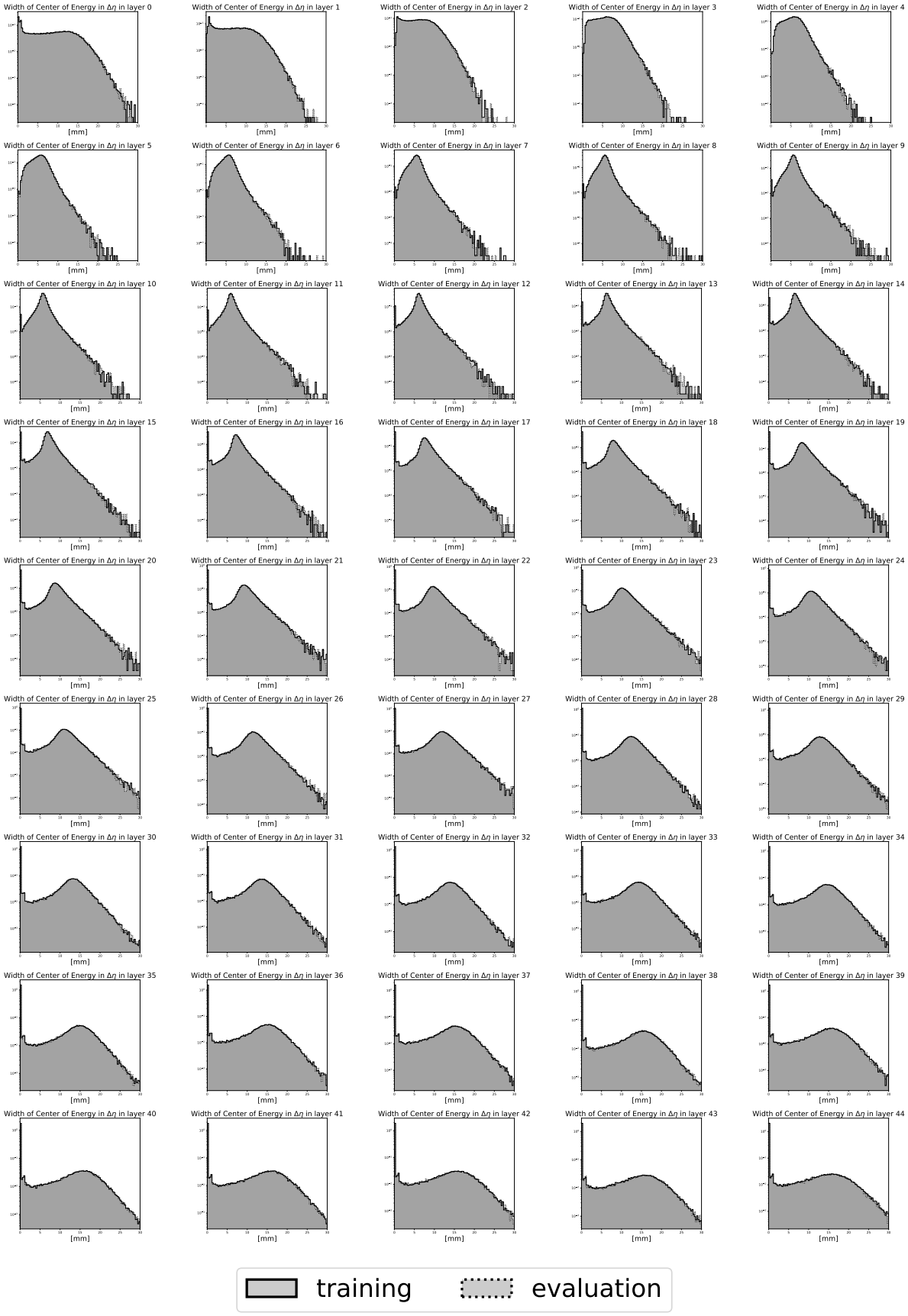



Figure A20: Distribution of GEANT4 training and evaluation data in width of the centers of energy in $\eta$ direction for ds3.

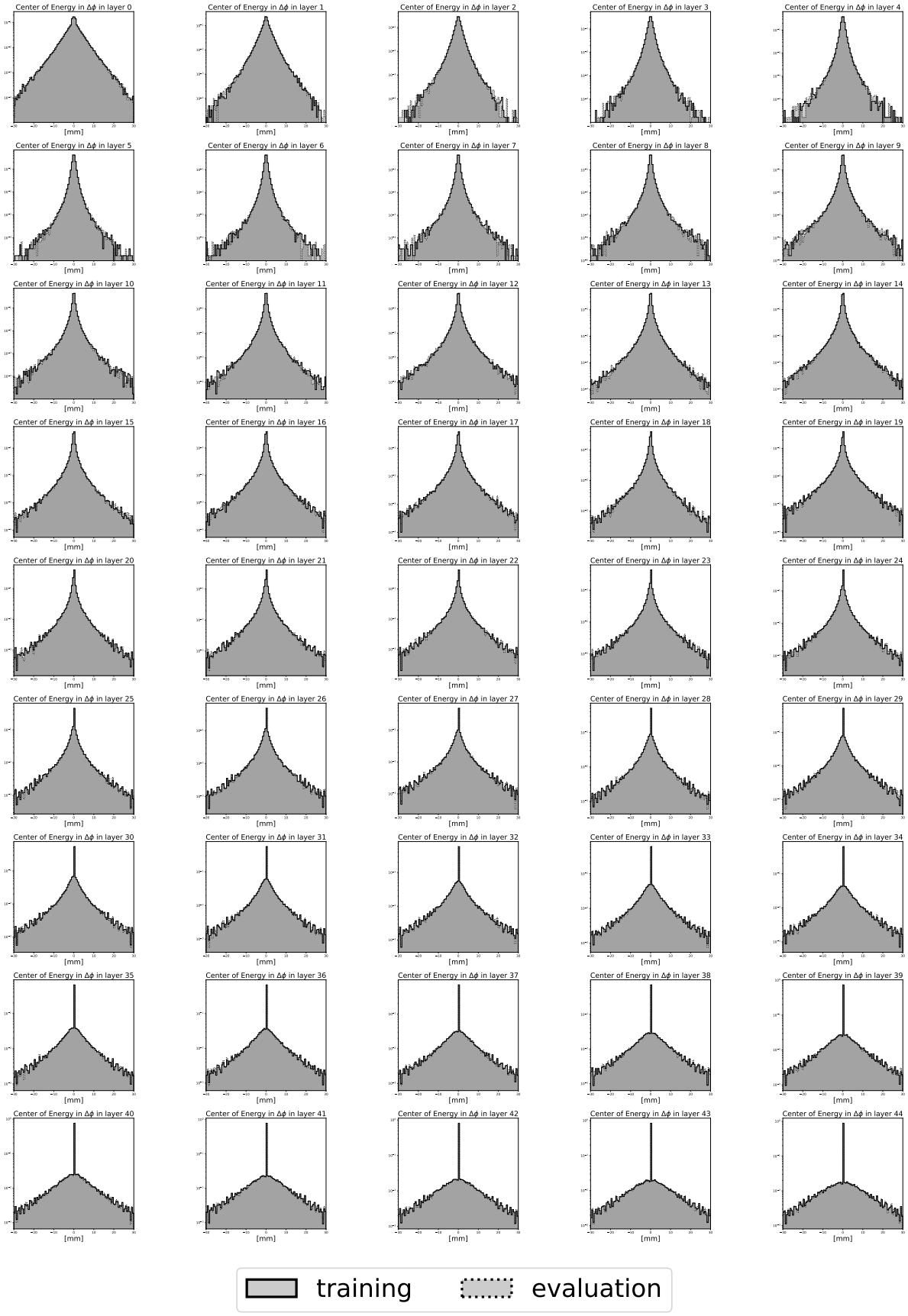



Figure A21: Distribution of GEANT4 training and evaluation data in centers of energy in $\phi$ direction for ds3.

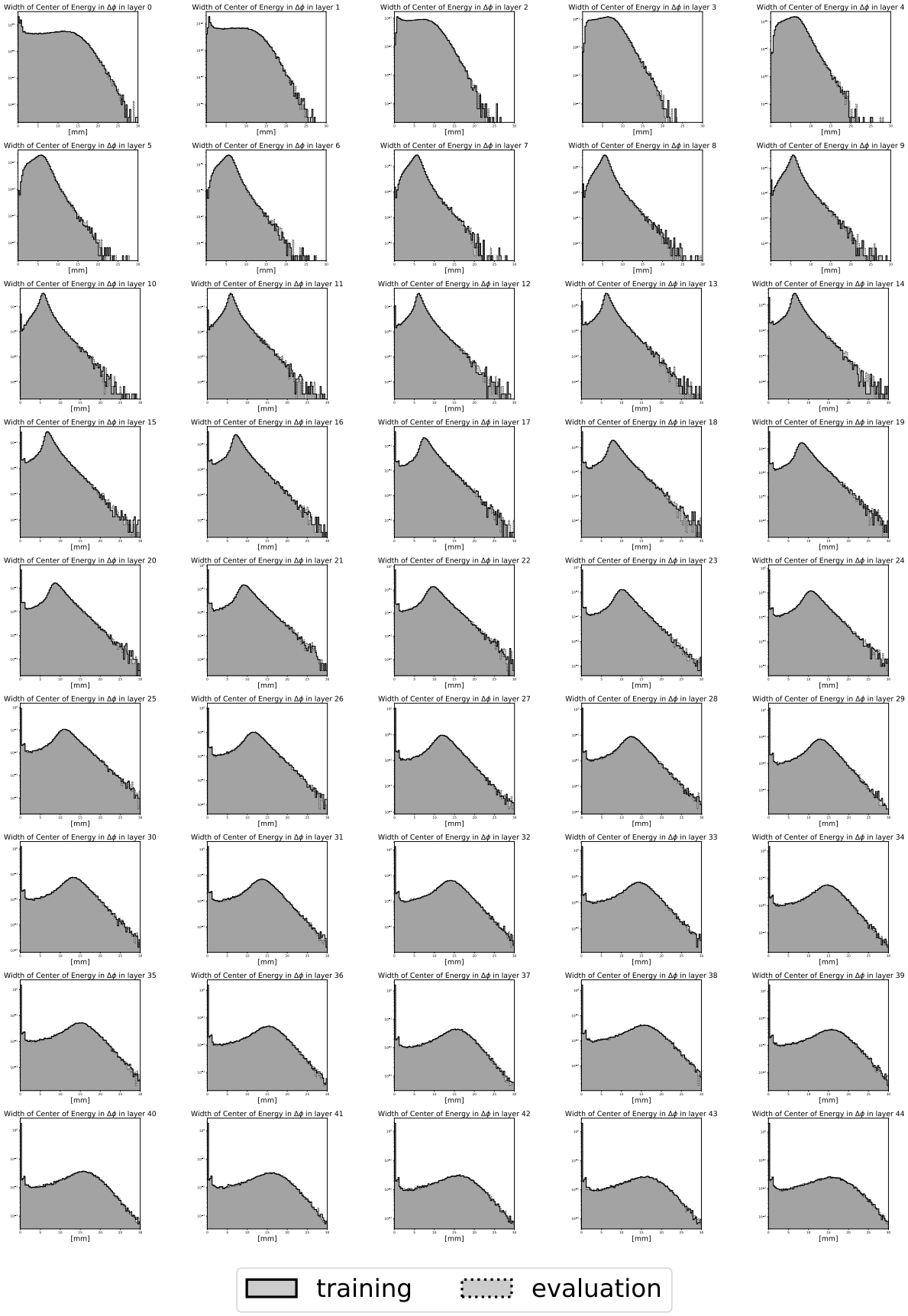



Figure A22: Distribution of GEANT4 training and evaluation data in width of the centers of energy in $\phi$ direction for ds3.

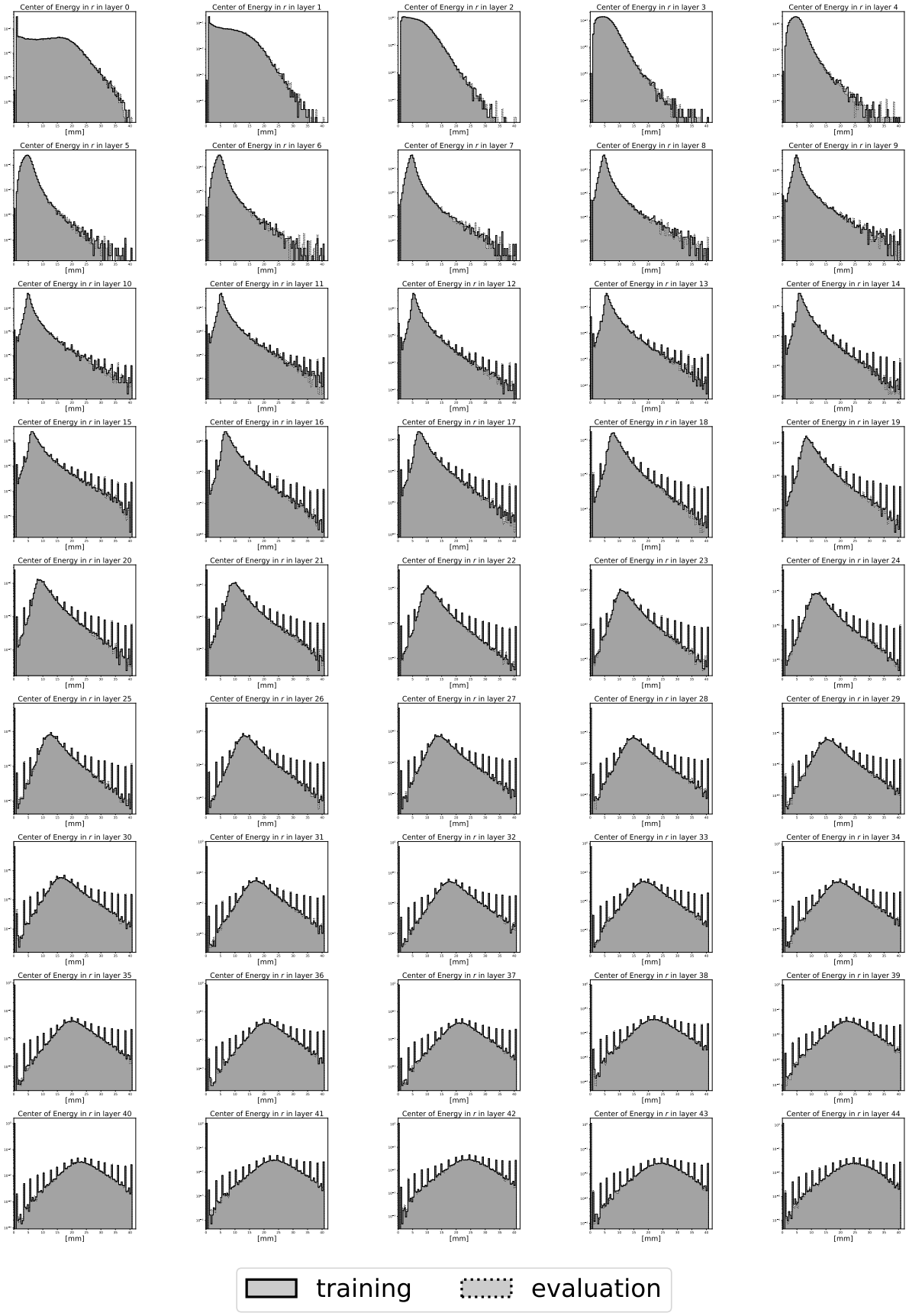



Figure A23: Distribution of GEANT4 training and evaluation data in centers of energy in $r$ direction for ds3.

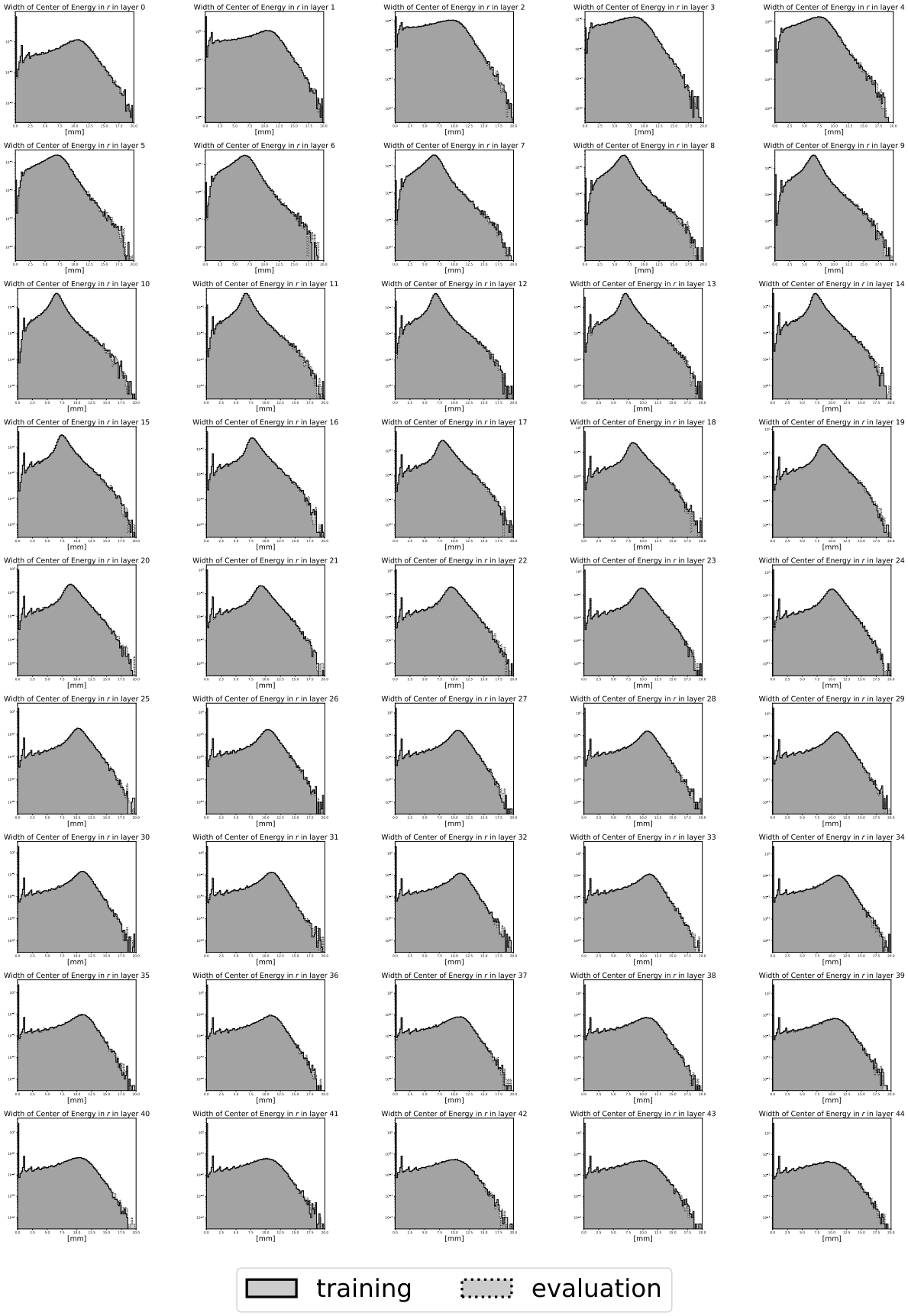



Figure A24: Distribution of GEANT4 training and evaluation data in width of the centers of energy in $r$ direction for ds3.

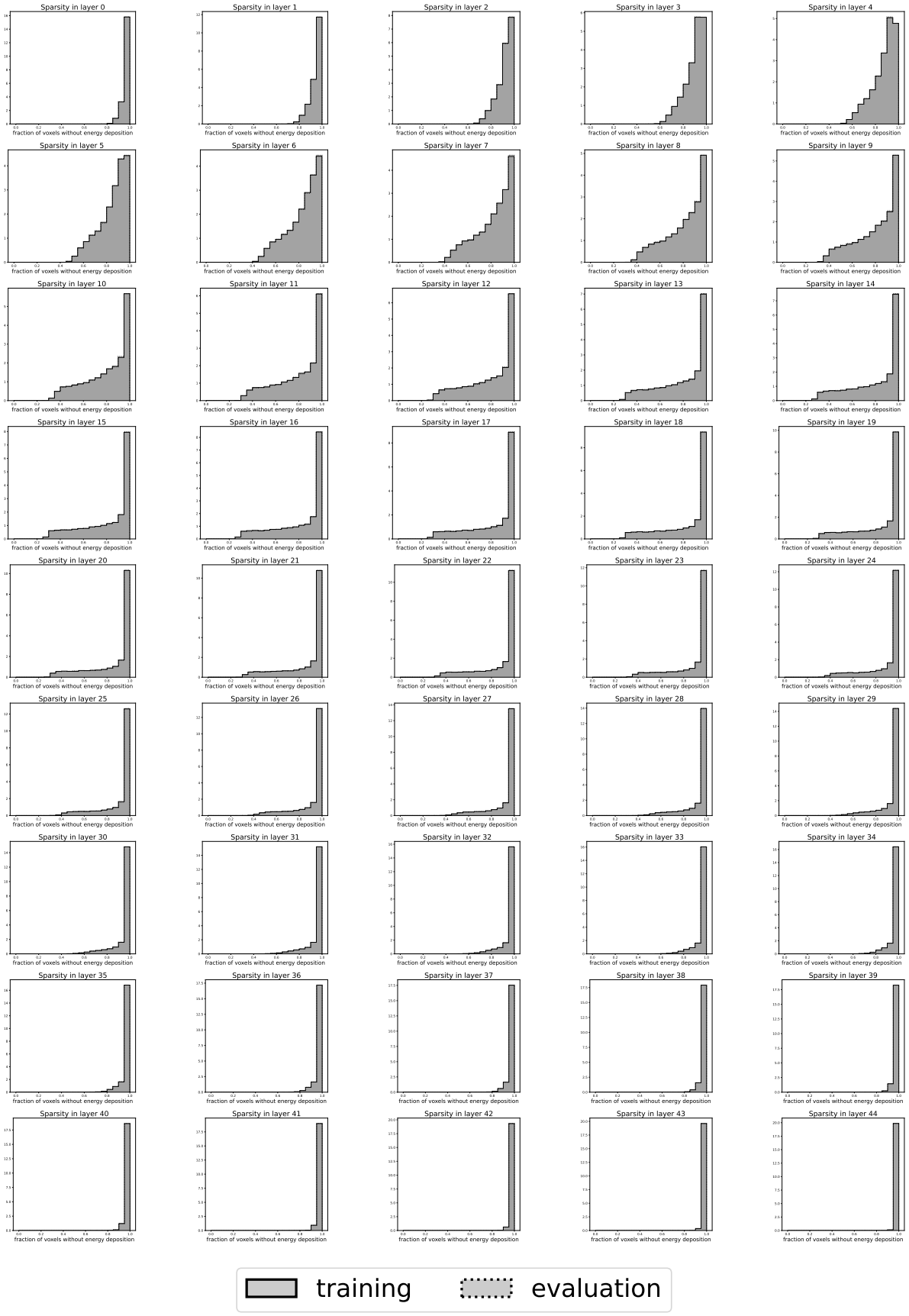



Figure A25: Distribution of GEANT4 training and evaluation data in sparsity for ds3.

### B. Consistency check of the multiclass classifier

A well-trained multiclass classifier identifies samples from each submission correctly. We show this test here in terms of the log posterior of (41). We show the mean and standard deviation of ten independent trainings and subsequent determination of the log posterior. In figure B1 to figure B6 we show all log posteriors in terms of confusion matrices. The consistency condition of (42) can be read line by line in them: In each line, the largest entry is in the diagonal position. This holds for all tests, except for the DNN classifier of dataset 2, where a `CaloScore` submission was confused as being `CaloDREAM`, which could bias the test with the GEANT4 dataset towards `CaloDREAM`. In addition, there are a few cases where a class confusion is within error bars, but these concern mostly models that are distilled versions from each other, most notably between `CaloScore distilled` and `CaloScore single-shot` for both, datasets 2 and 3.

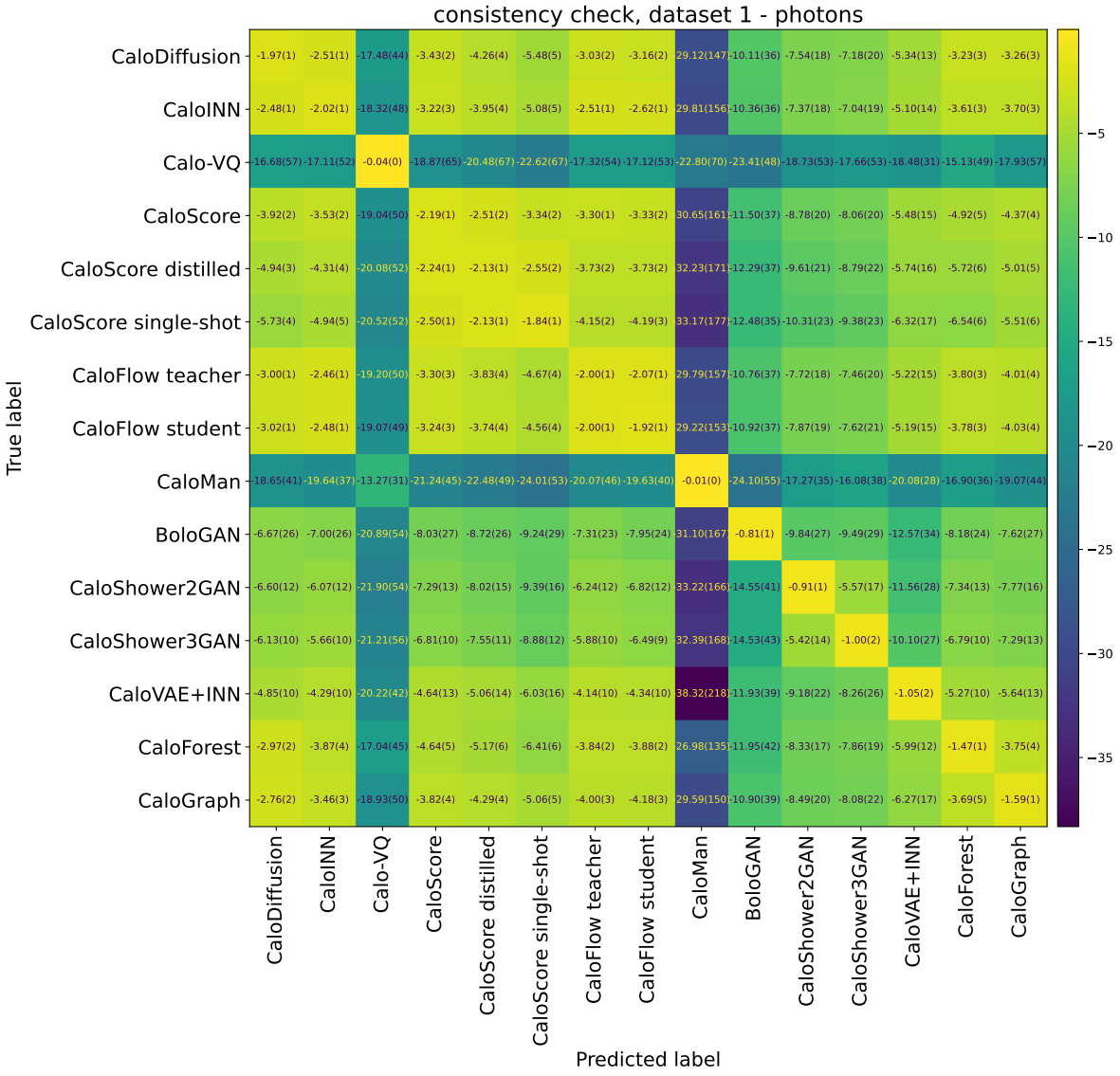



Figure B1: Log posteriors for evaluating the DNN multiclass classifier on submission test sets for ds $1-\gamma$.

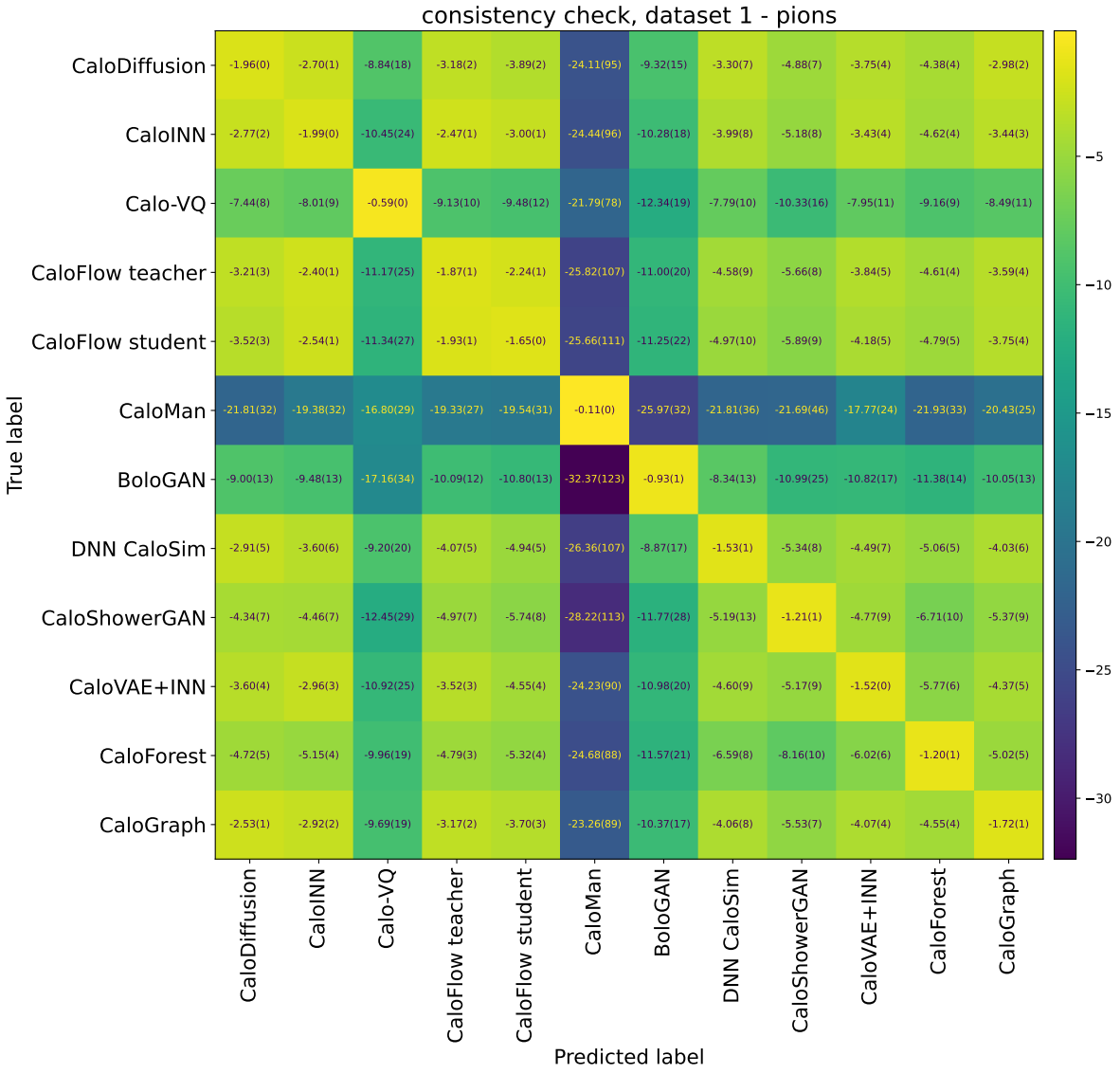



Figure B2: Log posteriors for evaluating the DNN multiclass classifier on submission test sets for ds $1 - \pi^+$.

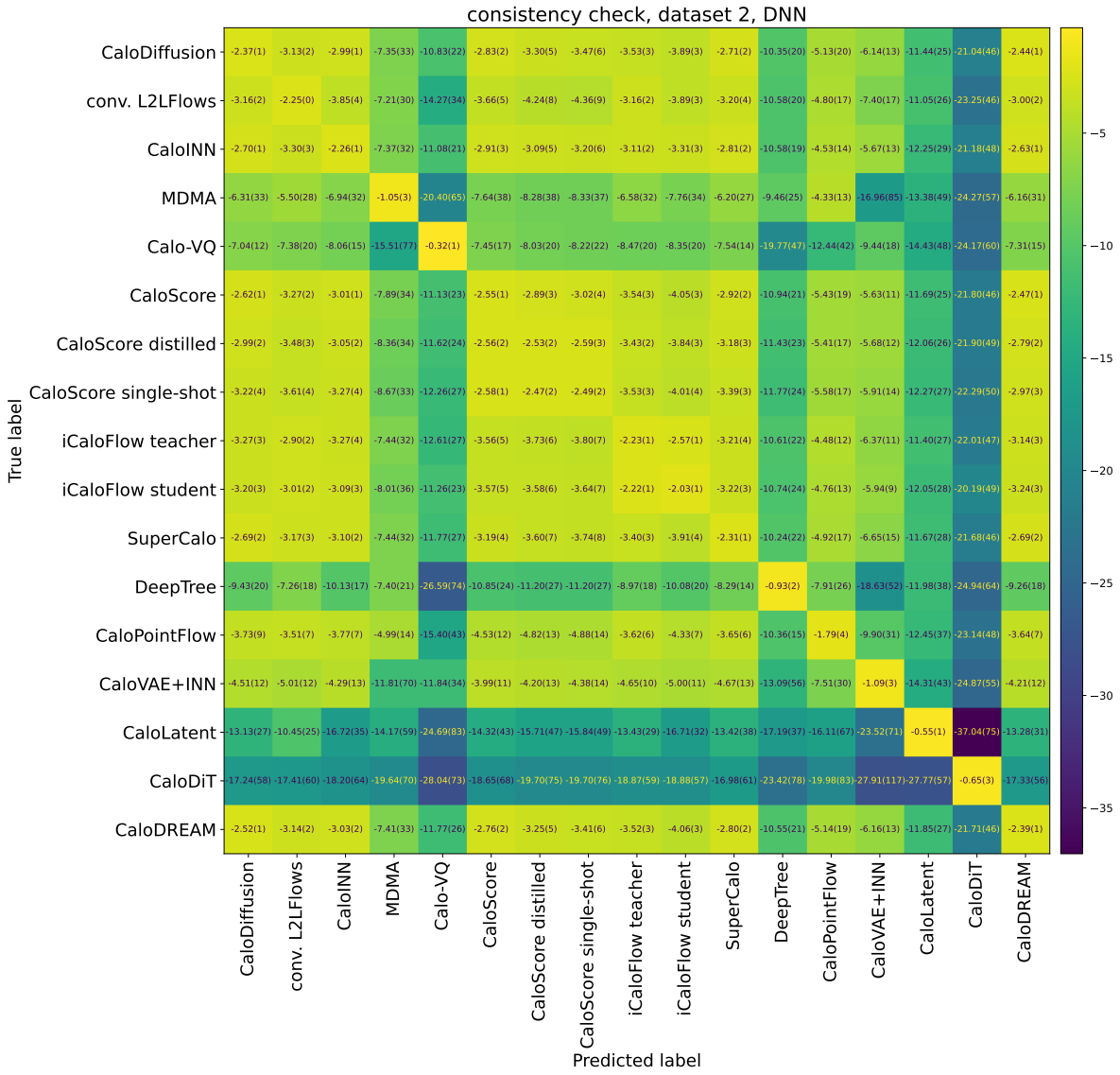



Figure B3: Log posteriors for evaluating the DNN multiclass classifier on submission test sets for ds 2.

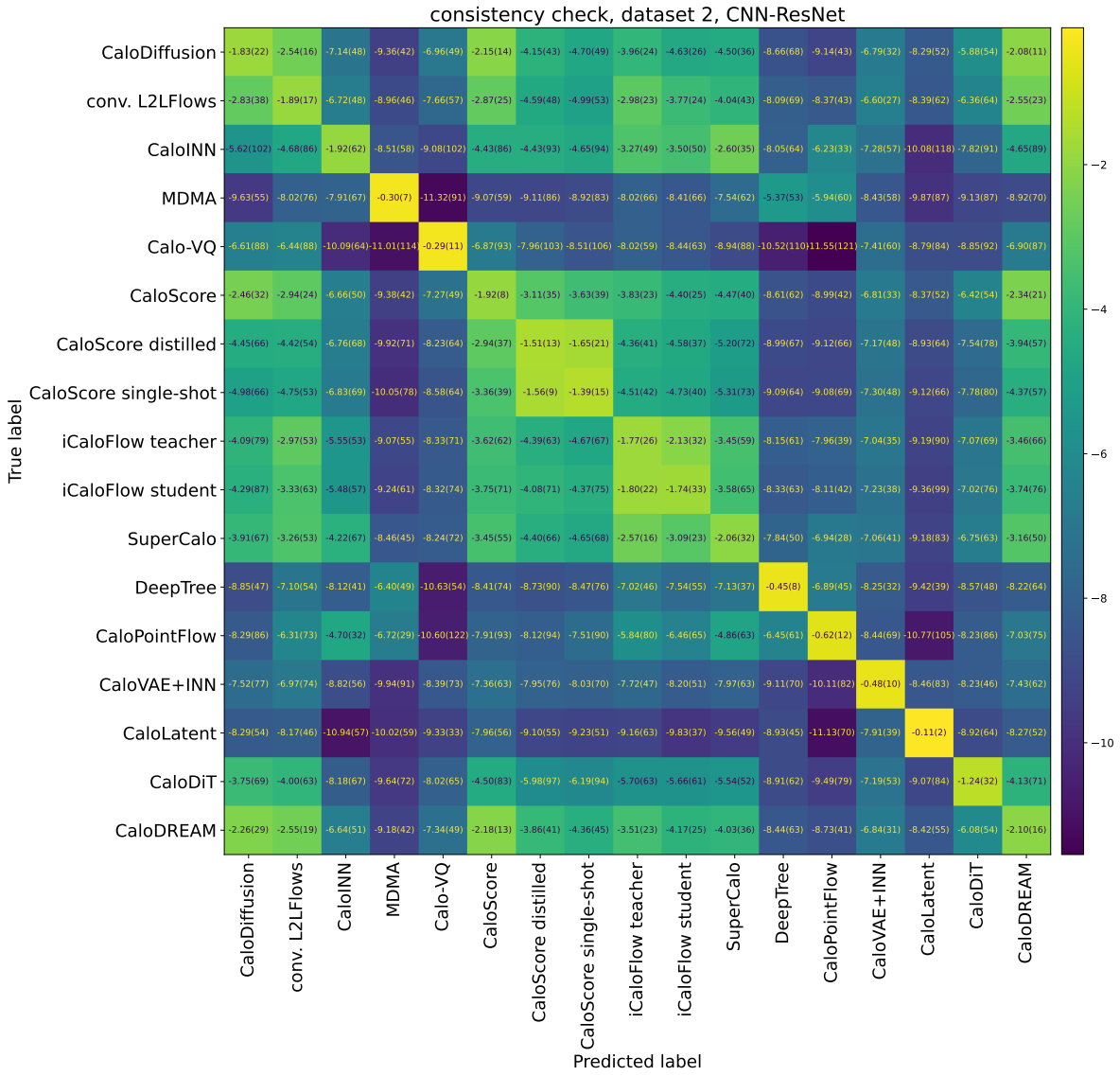



Figure B4: Log posteriors for evaluating the CNN ResNet multiclass classifier on submission test sets for ds 2.

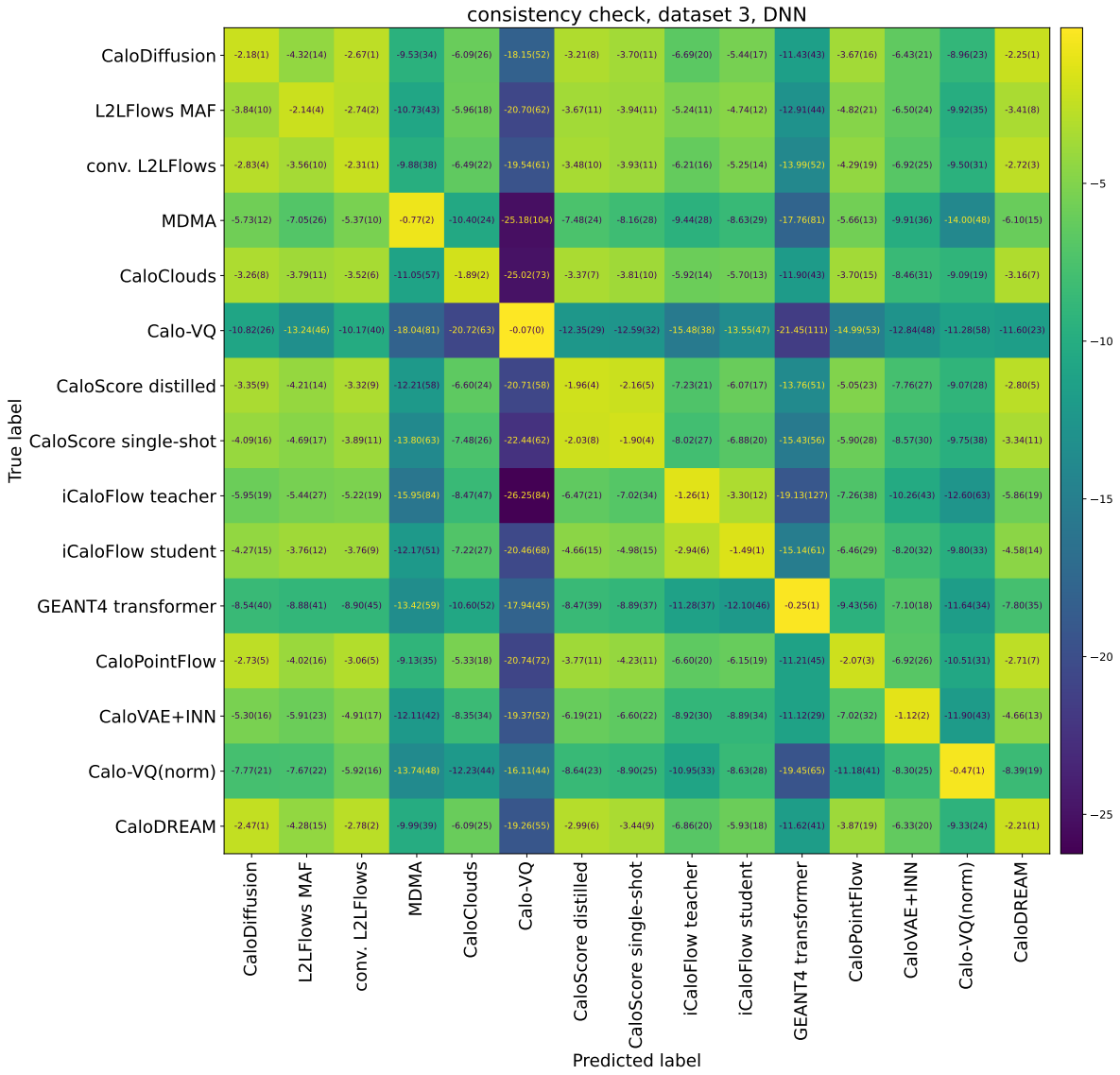



Figure B5: Log posteriors for evaluating the DNN multiclass classifier on submission test sets for ds 3.

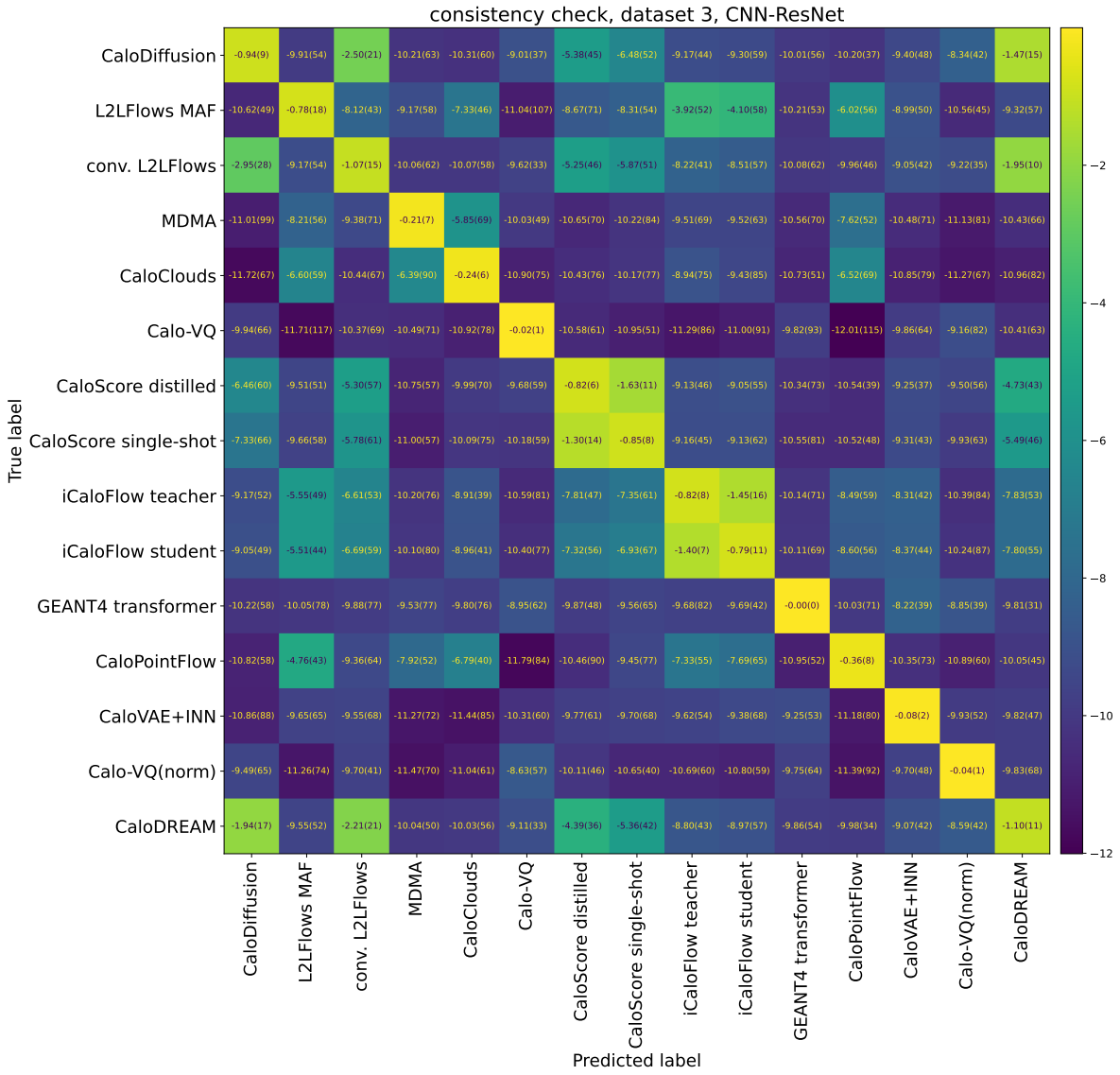



Figure B6: Log posteriors for evaluating the CNN ResNet multiclass classifier on submission test sets for ds 3.

## C. Numerical Results in Tables

In this appendix we show tables with all the results that went into the figures of section 9. Bold-faced entries indicate the best submission(s) within the given errorbars.

### *C.1. Dataset 1, Photons (ds 1 – $\gamma$)*

Table C1: Low-level and high-level AUCs for evaluating GEANT4 vs. submission of ds 1 – $\gamma$, averaged over 10 independent evaluation runs. For visualization, see Figure 39.

| Submission | low-level AUC ↓ | high-level AUC ↓ |
|---|---|---|
| GEANT4 | $0.499 \pm 0.002$ | $0.499 \pm 0.003$ |
| `CaloDiffusion` [44] | $0.635 \pm 0.003$ | $\mathbf{0.536 \pm 0.003}$ |
| `CaloINN` [38] | $\mathbf{0.626 \pm 0.004}$ | $0.638 \pm 0.003$ |
| `Calo-VQ` [58] | $0.998 \pm 0.000$ | $0.989 \pm 0.001$ |
| `CaloScore` [50, 51] | $0.751 \pm 0.002$ | $0.552 \pm 0.005$ |
| `CaloScore distilled` [50, 51] | $0.816 \pm 0.004$ | $0.641 \pm 0.003$ |
| `CaloScore single-shot` [50, 51] | $0.866 \pm 0.003$ | $0.726 \pm 0.004$ |
| `CaloFlow teacher` [34] | $0.733 \pm 0.003$ | $0.636 \pm 0.002$ |
| `CaloFlow student` [34] | $0.761 \pm 0.002$ | $0.667 \pm 0.004$ |
| `CaloMan` [60] | $1.000 \pm 0.000$ | $0.999 \pm 0.000$ |
| `BoloGAN` [26] | $0.927 \pm 0.003$ | $0.966 \pm 0.001$ |
| `CaloShower2GAN` [21] | $0.938 \pm 0.004$ | $0.942 \pm 0.002$ |
| `CaloShower3GAN` [21] | $0.928 \pm 0.004$ | $0.947 \pm 0.002$ |
| `CaloVAE+INN` [38] | $0.889 \pm 0.003$ | $0.966 \pm 0.001$ |
| `CaloForest` [71] | $0.845 \pm 0.002$ | $0.924 \pm 0.002$ |
| `CaloGraph` [54] | $0.820 \pm 0.002$ | $0.672 \pm 0.004$ |

Table C2: KPD and FPD for evaluating GEANT4 vs. submission of ds 1 – $\gamma$. For visualization, see Figure 40.

| Submission | KPD $\cdot 10^3 \downarrow$ | FPD $\cdot 10^3 \downarrow$ |
|---|---|---|
| GEANT4 | $0.0279 \pm 0.0592$ | $0.1192 \pm 0.0534$ |
| `CaloDiffusion` [44] | $\mathbf{-0.0100 \pm 0.0515}$ | $\mathbf{0.6497 \pm 0.1308}$ |
| `CaloINN` [38] | $\mathbf{0.2327 \pm 0.2165}$ | $3.8974 \pm 0.1707$ |
| `Calo-VQ` [58] | $2.4053 \pm 0.1363$ | $35.9531 \pm 0.3235$ |
| `CaloScore` [50, 51] | $\mathbf{0.0008 \pm 0.0579}$ | $\mathbf{0.4381 \pm 0.1296}$ |
| `CaloScore distilled` [50, 51] | $\mathbf{0.0319 \pm 0.0504}$ | $1.1200 \pm 0.1310$ |
| `CaloScore single-shot` [50, 51] | $0.2214 \pm 0.0537$ | $3.0866 \pm 0.1348$ |
| `CaloFlow teacher` [34] | $\mathbf{0.0824 \pm 0.0679}$ | $3.1124 \pm 0.0938$ |
| `CaloFlow student` [34] | $\mathbf{0.0902 \pm 0.0723}$ | $3.1512 \pm 0.1080$ |
| `CaloMan` [60] | $11.1063 \pm 0.5458$ | $141.3752 \pm 0.4676$ |
| `BoloGAN` [26] | $11.6268 \pm 1.1852$ | $142.4424 \pm 1.3268$ |
| `CaloShower2GAN` [21] | $2.8025 \pm 0.4946$ | $52.0972 \pm 0.3547$ |
| `CaloShower3GAN` [21] | $1.1561 \pm 0.2029$ | $20.5146 \pm 0.3082$ |
| `CaloVAE+INN` [38] | $1.2138 \pm 0.0542$ | $14.5877 \pm 0.1710$ |
| `CaloForest` [71] | $2.2297 \pm 0.3132$ | $33.5196 \pm 0.5523$ |
| `CaloGraph` [54] | $1.1558 \pm 0.2367$ | $15.7884 \pm 0.2592$ |

Table C3: Log-posterior scores for ds 1 – $\gamma$ GEANT4 test data, averaged over 10 independent classifier trainings. For visualization, see Figure 41.

| Submission | Log-posterior $\uparrow$ |
|---|---|
| `CaloDiffusion` [44] | $\mathbf{-2.1893 \pm 0.0053}$ |
| `CaloINN` [38] | $-2.2046 \pm 0.0083$ |
| `Calo-VQ` [58] | $-17.9096 \pm 0.4605$ |
| `CaloScore` [50, 51] | $-3.3126 \pm 0.0254$ |
| `CaloScore distilled` [50, 51] | $-4.2047 \pm 0.0408$ |
| `CaloScore single-shot` [50, 51] | $-5.4383 \pm 0.0507$ |
| `CaloFlow teacher` [34] | $-2.7946 \pm 0.0133$ |
| `CaloFlow student` [34] | $-2.9796 \pm 0.0143$ |
| `CaloMan` [60] | $-30.3461 \pm 1.5767$ |
| `BoloGAN` [26] | $-9.7551 \pm 0.3416$ |
| `CaloShower2GAN` [21] | $-7.1072 \pm 0.1705$ |
| `CaloShower3GAN` [21] | $-6.7926 \pm 0.1889$ |
| `CaloVAE+INN` [38] | $-5.2225 \pm 0.1405$ |
| `CaloForest` [71] | $-3.6188 \pm 0.0307$ |
| `CaloGraph` [54] | $-3.5833 \pm 0.0310$ |

Table C4: Precision, density, recall, and coverage for ds 1 – $\gamma$ submissions. A visualization is shown in figure 42.

| Submission | Precision ↑ | Density ↑ | Recall ↑ | Coverage ↑ |
|---|---|---|---|---|
| GEANT4 | 0.704 | 0.992 | 0.699 | 0.964 |
| `CaloDiffusion` [44] | 0.730 | 1.152 | 0.665 | 0.974 |
| `CaloINN` [38] | 0.607 | 0.831 | 0.716 | 0.911 |
| `Calo-VQ` [58] | 0.957 | 115.883 | 0.003 | 0.952 |
| `CaloScore` [50, 51] | 0.662 | 0.891 | 0.708 | 0.939 |
| `CaloScore distilled` [50, 51] | 0.613 | 0.781 | 0.722 | 0.907 |
| `CaloScore single-shot` [50, 51] | 0.482 | 0.441 | 0.807 | 0.782 |
| `CaloFlow teacher` [34] | 0.595 | 0.799 | 0.718 | 0.913 |
| `CaloFlow student` [34] | 0.617 | 0.859 | 0.701 | 0.912 |
| `CaloMan` [60] | 0.888 | 612.387 | 0.010 | 0.888 |
| `BoloGAN` [26] | 0.207 | 0.315 | 0.194 | 0.411 |
| `CaloShower2GAN` [21] | 0.761 | 72.665 | 0.036 | 0.832 |
| `CaloShower3GAN` [21] | 0.734 | 65.729 | 0.039 | 0.795 |
| `CaloVAE+INN` [38] | 0.873 | 73.873 | 0.139 | 0.973 |
| `CaloForest` [71] | 0.906 | 17.494 | 0.186 | 0.957 |
| `CaloGraph` [54] | 0.657 | 0.982 | 0.670 | 0.933 |

Table C5: Number of trainable parameters in training and for generation for ds $1-\gamma$ submissions. A visualization is shown in figure 43.

| Submission | number of parameters ↓ total | generator only |
|---|---|---|
| `CaloDiffusion` [44] | **521 581** | 521 581 |
| `CaloINN` [38] | 18 821 350 | 18 821 350 |
| `Calo-VQ` [58] | 4 060 878 | 2 152 637 |
| `CaloScore` [50, 51] | 2 447 366 | 2 447 366 |
| `CaloScore distilled` [50, 51] | 4 894 732 | 2 447 366 |
| `CaloScore single-shot` [50, 51] | 4 894 732 | 2 447 366 |
| `CaloFlow teacher` [34] | 28 043 810 | 28 043 810 |
| `CaloFlow student` [34] | 84 500 898 | 56 554 930 |
| `CaloMan` [60] | 19 276 658 | 17 061 148 |
| `BoloGAN` [26] | 1 185 520 | 368 558 |
| `CaloShower2GAN` [21] | 1 183 606 | **367 380** |
| `CaloShower3GAN` [21] | 1 696 459 | 472 120 |
| `CaloVAE+INN` [38] | 15 747 908 | 8 321 308 |
| `CaloForest` [71] | 3 837 598 845 | 3 837 598 845 |
| `CaloGraph` [54] | 823 617 | 823 617 |

Table C6: Timing of ds $1-\gamma$ submissions on a CPU. The symbols $^*$, $^\dagger$, $^\ddagger$, and $^\diamond$ indicate that only 1000 / 10 000 / 20 000 / 50 000 events were generated in timing the submission. A visualization of these timings is shown in figure 44.

| Submission | CPU [ms per shower] | | |
|---|---|---|---|
| | batch size 1 | batch size 100 | batch size 10 000 |
| `CaloDiffusion` [44] | $15\,677 \pm 633^\dagger$ | $4266 \pm 106^\dagger$ | $4500 \pm 357^\ddagger$ |
| `CaloINN` [38] | $\mathbf{37.9 \pm 3.2}$ | $2.78 \pm 0.34$ | $2.88 \pm 0.31$ |
| `Calo-VQ` [58] | $93.6 \pm 5.6$ | $10.4 \pm 1.8$ | $14.1 \pm 3.1$ |
| `CaloScore` [50, 51] | $19\,324 \pm 729^*$ | $9425 \pm 322^*$ | $12\,871 \pm 1197^\dagger$ |
| `CaloScore distilled` [50, 51] | $2456 \pm 178^\dagger$ | $1078 \pm 54^\dagger$ | $1276 \pm 190^\ddagger$ |
| `CaloScore single-shot` [50, 51] | $223.3 \pm 10.1$ | $20.5 \pm 2.0$ | $19.8 \pm 3.5$ |
| `CaloFlow teacher` [34] | $42\,875 \pm 3085^*$ | $2053 \pm 171^\dagger$ | $1912 \pm 141^\ddagger$ |
| `CaloFlow student` [34] | $575.9 \pm 22.2$ | $11.1 \pm 1.3$ | $6.11 \pm 0.60$ |
| `CaloMan` [60] | $186.5 \pm 31.8$ | $3.20 \pm 0.51$ | $1.46 \pm 0.16$ |
| `BoloGAN` [26] | $105.5 \pm 6.5$ | $\mathbf{1.73 \pm 0.10}$ | $\mathbf{0.55 \pm 0.03}$ |
| `CaloShower2GAN` [21] | $582.0 \pm 9.4^\dagger$ | $65.6 \pm 3.2$ | $1.28 \pm 0.07$ |
| `CaloShower3GAN` [21] | $480.0 \pm 19.5^\dagger$ | $55.5 \pm 2.9$ | $1.20 \pm 0.07$ |
| `CaloVAE+INN` [38] | $\mathbf{38.0 \pm 3.2}$ | $\mathbf{1.58 \pm 0.14}$ | $1.20 \pm 0.11$ |
| `CaloForest` [71] | $28\,400 \pm 916^*$ | $308.1 \pm 17.4$ | $27.3 \pm 1.7$ |
| `CaloGraph` [54] | $3250 \pm 277^\dagger$ | $914.4 \pm 70.2^\dagger$ | $1382 \pm 83^\diamond$ |

Table C7: Timing of ds 1 – $\gamma$ submissions on a GPU. The symbols $^*$ and $^\ddagger$ indicate that only 5000 or 10 000 events were generated in timing the submission; a – indicates a model that does not run on a GPU; and "CUDA o.o.m" ran out of VRAM on the GPU. A visualization of these timings is shown in figure 44.

| Submission | GPU [ms per shower] | | |
|---|---|---|---|
| | batch size 1 | batch size 100 | batch size 10 000 |
| `CaloDiffusion` [44] | $5593 \pm 64^*$ | $75.2 \pm 2.1$ | $24.4 \pm 0.1$ |
| `CaloINN` [38] | $\mathbf{24.6 \pm 1.6}$ | $\mathbf{0.51 \pm 0.03}$ | $0.19 \pm 0.01$ |
| `Calo-VQ` [58] | $48.2 \pm 0.6$ | $0.81 \pm 0.08$ | $\mathbf{0.16 \pm 0.01}$ |
| `CaloScore` [50, 51] | $4706 \pm 171^*$ | $60.4 \pm 4.1$ | $36.00 \pm 0.01$ |
| `CaloScore distilled` [50, 51] | $756.6 \pm 22.8$ | $8.8 \pm 0.2$ | $4.96 \pm 0.02$ |
| `CaloScore single-shot` [50, 51] | $189.4 \pm 12.5$ | $2.0 \pm 0.0$ | $0.56 \pm 0.02$ |
| `CaloFlow teacher` [34] | $4193 \pm 130^*$ | $45.5 \pm 1.1$ | $8.13 \pm 0.03$ |
| `CaloFlow student` [34] | $56.9 \pm 0.5$ | $0.79 \pm 0.01$ | $0.26 \pm 0.10$ |
| `CaloMan` [60] | $76.0 \pm 1.1$ | $1.04 \pm 0.08$ | $0.30 \pm 0.25$ |
| `BoloGAN` [26] | $286.1 \pm 12.3$ | $2.26 \pm 0.48$ | $0.53 \pm 0.03$ |
| `CaloShower2GAN` [21] | $611.6 \pm 44^\ddagger$ | $70.8 \pm 1.8$ | $1.71 \pm 0.67$ |
| `CaloShower3GAN` [21] | $518.8 \pm 44.7^\ddagger$ | $63.8 \pm 1.2$ | $1.49 \pm 0.05$ |
| `CaloVAE+INN` [38] | $34.0 \pm 0.4$ | $0.64 \pm 0.02$ | $0.26 \pm 0.01$ |
| `CaloForest` [71] | – | – | – |
| `CaloGraph` [54] | $1633 \pm 25^\ddagger$ | $25.0 \pm 0.4$ | CUDA o.o.m. |

Table C8: Low-level and high-level AUCs for evaluating GEANT4 vs. submission of ds 1 – $\pi^+$, averaged over 10 independent evaluation runs. For visualization, see Figure 52.

| Submission | low-level AUC ↓ | high-level AUC ↓ |
|---|---|---|
| GEANT4 | $0.609 \pm 0.004$ | $0.558 \pm 0.002$ |
| `CaloDiffusion` [44] | $\mathbf{0.680 \pm 0.002}$ | $\mathbf{0.652 \pm 0.006}$ |
| `CaloINN` [38] | $0.784 \pm 0.002$ | $0.732 \pm 0.002$ |
| `Calo-VQ` [58] | $0.958 \pm 0.002$ | $0.976 \pm 0.001$ |
| `CaloFlow teacher` [34] | $0.845 \pm 0.002$ | $0.797 \pm 0.002$ |
| `CaloFlow student` [34] | $0.884 \pm 0.002$ | $0.827 \pm 0.004$ |
| `CaloMan` [60] | $0.999 \pm 0.000$ | $0.999 \pm 0.000$ |
| `BoloGAN` [26] | $0.913 \pm 0.002$ | $0.969 \pm 0.001$ |
| `DNNCaloSim` [62, 63] | $\mathbf{0.676 \pm 0.004}$ | $0.819 \pm 0.002$ |
| `CaloShowerGAN` [21] | $0.889 \pm 0.002$ | $0.922 \pm 0.001$ |
| `CaloVAE+INN` [38] | $0.853 \pm 0.003$ | $0.921 \pm 0.002$ |
| `CaloForest` [71] | $0.909 \pm 0.002$ | $0.965 \pm 0.001$ |
| `CaloGraph` [54] | $0.811 \pm 0.005$ | $0.763 \pm 0.002$ |

Table C9: KPD and FPD for evaluating GEANT4 vs. submission of ds 1 – $\pi^+$. For visualization, see Figure 53.

| Submission | KPD $\cdot 10^3$ ↓ | FPD $\cdot 10^3$↓ |
|---|---|---|
| GEANT4 | $-0.0075 \pm 0.0362$ | $0.5110 \pm 0.0730$ |
| `CaloDiffusion` [44] | $\mathbf{0.0893 \pm 0.0572}$ | $\mathbf{2.7746 \pm 0.0922}$ |
| `CaloINN` [38] | $1.4781 \pm 0.2448$ | $29.2598 \pm 0.2133$ |
| `Calo-VQ` [58] | $6.2679 \pm 0.2652$ | $126.9924 \pm 0.6750$ |
| `CaloFlow teacher` [34] | $0.8083 \pm 0.0923$ | $25.6634 \pm 0.3002$ |
| `CaloFlow student` [34] | $0.9937 \pm 0.0846$ | $25.6868 \pm 0.2292$ |
| `CaloMan` [60] | $31.1636 \pm 0.9840$ | $524.4263 \pm 0.9067$ |
| `BoloGAN` [26] | $25.2457 \pm 0.9748$ | $498.6887 \pm 2.9504$ |
| `DNNCaloSim` [62, 63] | $20.0149 \pm 1.0029$ | $464.6882 \pm 2.2500$ |
| `CaloShowerGAN` [21] | $3.0518 \pm 0.1067$ | $113.2271 \pm 0.6673$ |
| `CaloVAE+INN` [38] | $4.3241 \pm 0.2868$ | $82.9985 \pm 0.8746$ |
| `CaloForest` [71] | $9.5250 \pm 0.9264$ | $204.7435 \pm 1.6359$ |
| `CaloGraph` [54] | $0.8500 \pm 0.2796$ | $22.3235 \pm 0.4428$ |

Table C10: Log-posterior scores for ds 1 – $\pi^+$ GEANT4 test data, averaged over 10 independent classifier trainings. For visualization, see Figure 54.

| Submission | Log-posterior ↑ |
|---|---|
| `CaloDiffusion` [44] | $-2.3189 \pm 0.0135$ |
| `CaloINN` [38] | $-3.0949 \pm 0.0234$ |
| `Calo-VQ` [58] | $-8.5998 \pm 0.1748$ |
| `CaloFlow teacher` [34] | $-3.7110 \pm 0.0316$ |
| `CaloFlow student` [34] | $-4.5623 \pm 0.0365$ |
| `CaloMan` [60] | $-25.9528 \pm 1.0659$ |
| `BoloGAN` [26] | $-8.0202 \pm 0.1331$ |
| `DNNCaloSim` [62, 63] | $\mathbf{-1.9262 \pm 0.0239}$ |
| `CaloShowerGAN` [21] | $-4.7157 \pm 0.0807$ |
| `CaloVAE+INN` [38] | $-4.0618 \pm 0.0511$ |
| `CaloForest` [71] | $-4.8811 \pm 0.0406$ |
| `CaloGraph` [54] | $-3.4762 \pm 0.0360$ |

Table C11: Precision, density, recall, and coverage for ds 1 – $\pi^+$ submissions. A visualization is shown in figure 55.

| Submission | Precision ↑ | Density ↑ | Recall ↑ | Coverage ↑ |
|---|---|---|---|---|
| GEANT4 | 0.643 | 0.782 | 0.739 | 0.913 |
| `CaloDiffusion` [44] | 0.632 | 0.750 | 0.743 | 0.895 |
| `CaloINN` [38] | 0.474 | 0.474 | 0.789 | 0.734 |
| `Calo-VQ` [58] | 0.954 | 54.862 | 0.134 | 0.947 |
| `CaloFlow teacher` [34] | 0.394 | 0.390 | 0.799 | 0.621 |
| `CaloFlow student` [34] | 0.414 | 0.445 | 0.772 | 0.625 |
| `CaloMan` [60] | 0.669 | 27.709 | 0.133 | 0.584 |
| `BoloGAN` [26] | 0.268 | 0.487 | 0.335 | 0.386 |
| `DNNCaloSim` [62, 63] | 0.945 | 0.788 | 1.000 | 0.931 |
| `CaloShowerGAN` [21] | 0.710 | 2.803 | 0.185 | 0.855 |
| `CaloVAE+INN` [38] | 0.709 | 5.857 | 0.416 | 0.808 |
| `CaloForest` [71] | 0.643 | 1.625 | 0.490 | 0.661 |
| `CaloGraph` [54] | 0.626 | 0.800 | 0.687 | 0.827 |

Table C12: Number of trainable parameters in training and for generation for ds 1 – $\pi^+$ submissions. A visualization is shown in figure 56.

| Submission | number of parameters ↓ total | generator only |
|---|---|---|
| `CaloDiffusion` [44] | **525 901** | **525 901** |
| `CaloINN` [38] | 26 592 624 | 26 592 624 |
| `Calo-VQ` [58] | 4 314 739 | 2 237 538 |
| `CaloFlow teacher` [34] | 57 079 326 | 57 079 326 |
| `CaloFlow student` [34] | 110 389 398 | 53 426 622 |
| `CaloMan` [60] | 18 452 248 | 16 032 327 |
| `BoloGAN` [26] | 1 678 334 | 848 733 |
| `DNNCaloSim` [62, 63] | 6 052 063 | 3 169 663 |
| `CaloShowerGAN` [21] | 1 715 742 | 880 541 |
| `CaloVAE+INN` [38] | 17 426 875 | 9 165 275 |
| `CaloForest` [71] | 5 297 822 388 | 5 297 822 388 |
| `CaloGraph` [54] | 823 617 | 823 617 |

Table C13: Timing of ds 1 – $\pi^+$ submissions on a CPU. The symbols $^\diamond$, $^\ddagger$, $^*$, and $^\dagger$ indicate that only 100 / 1000 / 10 000 / 20 000 events were generated in timing the submission. A visualization of these timings is shown in figure 57.

| Submission | CPU [ms per shower] batch size 1 | batch size 100 | batch size 10 000 |
|---|---|---|---|
| `CaloDiffusion` [44] | $15\,144 \pm 1126^\ddagger$ | $4571 \pm 333^*$ | $4501 \pm 181^\dagger$ |
| `CaloINN` [38] | $42.8 \pm 2.9$ | $3.92 \pm 0.36$ | $4.57 \pm 0.3$ |
| `Calo-VQ` [58] | $108.2 \pm 10.5$ | $12.7 \pm 1.4$ | $16.5 \pm 4.3$ |
| `CaloFlow teacher` [34] | $197\,570 \pm 34\,423^\diamond$ | $5430 \pm 489^*$ | $3509 \pm 177^\dagger$ |
| `CaloFlow student` [34] | $620.1 \pm 18.4$ | $14.2 \pm 2.2$ | $10.2 \pm 0.6$ |
| `CaloMan` [60] | $605.5 \pm 38.3$ | $12.1 \pm 1.3$ | $7.38 \pm 0.54$ |
| `BoloGAN` [26] | $79.6 \pm 3.1$ | $1.38 \pm 0.05$ | $0.49 \pm 0.03$ |
| `DNNCaloSim` [62, 63] | $\mathbf{3.85 \pm 0.96}$ | $\mathbf{0.47 \pm 0.04}$ | $\mathbf{0.39 \pm 0.03}$ |
| `CaloShowerGAN` [21] | $1163 \pm 301^*$ | $70.7 \pm 6.3$ | $1.60 \pm 0.07$ |
| `CaloVAE+INN` [38] | $40.8 \pm 1.1$ | $1.72 \pm 0.14$ | $1.20 \pm 0.14$ |
| `CaloForest` [71] | $37\,876 \pm 1905^*$ | $432.8 \pm 22.9$ | $41.2 \pm 4.2$ |
| `CaloGraph` [54] | $3168 \pm 135^*$ | $1263 \pm 112^*$ | $2419 \pm 153^\dagger$ |

Table C14: Timing of ds 1 – $\pi^+$ submissions on a GPU. The symbols $^*, ^\ddagger$, and $^\dagger$ indicate that only 4000 / 5000 / 10 000 events were generated in timing the submission; a – indicates a model that does not run on a GPU; and "CUDA o.o.m" ran out of VRAM on the GPU. A visualization of these timings is shown in figure 57.

| Submission | GPU [ms per shower] | | |
|---|---|---|---|
| | batch size 1 | batch size 100 | batch size 10 000 |
| `CaloDiffusion` [44] | $5673 \pm 49^\ddagger$ | $76.9 \pm 2.2$ | $27.0 \pm 0.2$ |
| `CaloINN` [38] | $24.7 \pm 2.0$ | $0.44 \pm 0.01$ | $\mathbf{0.20 \pm 0.01}$ |
| `Calo-VQ` [58] | $53.3 \pm 0.8$ | $0.83 \pm 0.05$ | $\mathbf{0.18 \pm 0.01}$ |
| `CaloFlow teacher` [34] | $6166 \pm 112^*$ | $70.1 \pm 1.0$ | $17.7 \pm 0.0$ |
| `CaloFlow student` [34] | $77.4 \pm 3.6$ | $1.00 \pm 0.02$ | $0.25 \pm 0.08$ |
| `CaloMan` [60] | $181.4 \pm 3.6$ | $2.07 \pm 0.02$ | $0.27 \pm 0.01$ |
| `BoloGAN` [26] | $209.5 \pm 12.9$ | $1.59 \pm 0.13$ | $0.48 \pm 0.03$ |
| `DNNCaloSim` [62, 63] | $\mathbf{2.34 \pm 0.17}$ | $\mathbf{0.32 \pm 0.01}$ | $0.29 \pm 0.01$ |
| `CaloShowerGAN` [21] | $1119 \pm 56^\dagger$ | $122.6 \pm 1.5$ | $2.06 \pm 0.06$ |
| `CaloVAE+INN` [38] | $34.2 \pm 0.5$ | $0.68 \pm 0.02$ | $0.27 \pm 0.01$ |
| `CaloForest` [71] | – | – | – |
| `CaloGraph` [54] | $1475 \pm 22^\dagger$ | $28.4 \pm 0.02$ | CUDA o.o.m. |

*C.3. Dataset 2, Electrons (ds 2)*

Table C15: Low-level and high-level AUCs for evaluating GEANT4 vs. submission of ds 2, averaged over 10 independent evaluation runs. For visualization, see Figure 67.

| Submission | AUC ↓ low-level | high-level | ResNet |
|---|---|---|---|
| GEANT4 | $0.500 \pm 0.002$ | $0.499 \pm 0.002$ | $0.500 \pm 0.004$ |
| `CaloDiffusion` [44] | $0.577 \pm 0.004$ | $0.591 \pm 0.009$ | $\mathbf{0.680 \pm 0.006}$ |
| `conv. L2LFlows` [32] | $0.708 \pm 0.004$ | $0.737 \pm 0.002$ | $0.941 \pm 0.003$ |
| `CaloINN` [38] | $0.743 \pm 0.002$ | $0.865 \pm 0.003$ | $0.994 \pm 0.000$ |
| `MDMA` [23, 24] | $0.942 \pm 0.005$ | $0.987 \pm 0.001$ | $1.000 \pm 0.000$ |
| `Calo-VQ` [58] | $0.986 \pm 0.001$ | $0.994 \pm 0.000$ | $0.999 \pm 0.000$ |
| `CaloScore` [50, 51] | $0.595 \pm 0.003$ | $0.666 \pm 0.002$ | $0.795 \pm 0.011$ |
| `CaloScore distilled` [50, 51] | $0.710 \pm 0.002$ | $0.891 \pm 0.003$ | $0.965 \pm 0.002$ |
| `CaloScore single-shot` [50, 51] | $0.747 \pm 0.003$ | $0.902 \pm 0.002$ | $0.973 \pm 0.002$ |
| `iCaloFlow teacher` [35] | $0.763 \pm 0.004$ | $0.837 \pm 0.005$ | $0.970 \pm 0.002$ |
| `iCaloFlow student` [35] | $0.819 \pm 0.004$ | $0.886 \pm 0.003$ | $0.975 \pm 0.002$ |
| `SuperCalo` [40] | $0.694 \pm 0.006$ | $0.757 \pm 0.004$ | $0.986 \pm 0.001$ |
| `DeepTree` [28, 29] | $0.963 \pm 0.002$ | $0.927 \pm 0.002$ | $0.999 \pm 0.000$ |
| `CaloPointFlow` [42] | $0.863 \pm 0.005$ | $0.908 \pm 0.004$ | $0.999 \pm 0.000$ |
| `CaloVAE+INN` [38] | $0.907 \pm 0.004$ | $1.000 \pm 0.000$ | $0.993 \pm 0.001$ |
| `CaloLatent` [67] | $0.983 \pm 0.001$ | $0.995 \pm 0.001$ | $1.000 \pm 0.000$ |
| `CaloDiT` [56] | $0.984 \pm 0.001$ | $0.912 \pm 0.002$ | $0.988 \pm 0.001$ |
| `CaloDREAM` [69] | $\mathbf{0.531 \pm 0.003}$ | $\mathbf{0.521 \pm 0.002}$ | $\mathbf{0.681 \pm 0.015}$ |

Table C16: KPD and FPD for evaluating GEANT4 vs. submission of ds 2. For visualization, see Figure 68.

| Submission | KPD $\cdot 10^3 \downarrow$ | FPD $\cdot 10^3 \downarrow$ |
|---|---|---|
| GEANT4 | $-0.0276 \pm 0.0215$ | $10.7760 \pm 0.7901$ |
| `CaloDiffusion` [44] | $0.1741 \pm 0.0422$ | $146.9334 \pm 0.8703$ |
| `conv. L2LFlows` [32] | $0.2705 \pm 0.0897$ | $157.4047 \pm 0.9684$ |
| `CaloINN` [38] | $2.8210 \pm 0.4194$ | $732.8274 \pm 5.3303$ |
| `MDMA` [23, 24] | $4.9624 \pm 0.2728$ | $864.9781 \pm 5.1452$ |
| `Calo-VQ` [58] | $8.5212 \pm 0.5043$ | $1315.7233 \pm 7.0344$ |
| `CaloScore` [50, 51] | $0.1486 \pm 0.0568$ | $112.4790 \pm 0.9080$ |
| `CaloScore distilled` [50, 51] | $1.0129 \pm 0.0738$ | $638.8525 \pm 1.5996$ |
| `CaloScore single-shot` [50, 51] | $0.9294 \pm 0.0684$ | $546.2661 \pm 1.9396$ |
| `iCaloFlow teacher` [35] | $0.5679 \pm 0.1375$ | $377.0613 \pm 1.8961$ |
| `iCaloFlow student` [35] | $1.0406 \pm 0.2190$ | $449.2585 \pm 3.2844$ |
| `SuperCalo` [40] | $0.5564 \pm 0.1900$ | $300.8183 \pm 2.7275$ |
| `DeepTree` [28, 29] | $0.6803 \pm 0.1285$ | $292.6319 \pm 2.9330$ |
| `CaloPointFlow` [42] | $0.3241 \pm 0.0392$ | $494.0547 \pm 1.7906$ |
| `CaloVAE+INN` [38] | $45.6091 \pm 0.8315$ | $5443.4295 \pm 27.2305$ |
| `CaloLatent` [67] | $2.8791 \pm 0.1998$ | $962.9750 \pm 2.4089$ |
| `CaloDiT` [56] | $11.0322 \pm 0.4274$ | $1690.9873 \pm 6.7650$ |
| `CaloDREAM` [69] | $\mathbf{0.0231 \pm 0.0364}$ | $\mathbf{24.6488 \pm 1.0350}$ |

Table C17: Log-posterior scores for ds 2 GEANT4 test data, averaged over 10 independent DNN classifier trainings. For visualization, see Figure 69.

| Submission | Log-posterior ↑ |
|---|---|
| `CaloDiffusion` [44] | $-2.5226 \pm 0.0094$ |
| `conv. L2LFlows` [32] | $-3.1295 \pm 0.0219$ |
| `CaloINN` [38] | $-3.2032 \pm 0.0153$ |
| `MDMA` [23, 24] | $-7.4399 \pm 0.3234$ |
| `Calo-VQ` [58] | $-11.8863 \pm 0.2627$ |
| `CaloScore` [50, 51] | $-2.8415 \pm 0.0238$ |
| `CaloScore distilled` [50, 51] | $-3.4226 \pm 0.0556$ |
| `CaloScore single-shot` [50, 51] | $-3.5974 \pm 0.0685$ |
| `CaloDREAM` [69] | $\mathbf{-2.4102 \pm 0.0105}$ |
| `iCaloFlow teacher` [35] | $-3.6423 \pm 0.0286$ |
| `iCaloFlow student` [35] | $-4.2617 \pm 0.0360$ |
| `SuperCalo` [40] | $-2.8204 \pm 0.0237$ |
| `CaloDiT` [56] | $-22.1206 \pm 0.4549$ |
| `DeepTree` [28, 29] | $-10.5062 \pm 0.2094$ |
| `CaloPointFlow` [42] | $-5.2306 \pm 0.1996$ |
| `CaloVAE+INN` [38] | $-6.4103 \pm 0.1439$ |
| `CaloLatent` [67] | $-11.6683 \pm 0.2679$ |

Table C18: Log-posterior scores for ds 2 GEANT4 test data, averaged over 10 independent CNN ResNet classifier trainings. For visualization, see Figure 70.

| Submission | Log-posterior ↑ |
|---|---|
| `CaloDiffusion` [44] | $\mathbf{-1.9901 \pm 0.2358}$ |
| `conv. L2LFlows` [32] | $-2.4500 \pm 0.1546$ |
| `CaloINN` [38] | $-7.2706 \pm 0.4780$ |
| `MDMA` [23, 24] | $-9.3476 \pm 0.4435$ |
| `Calo-VQ` [58] | $-7.0740 \pm 0.4898$ |
| `CaloScore` [50, 51] | $\mathbf{-2.1544 \pm 0.1303}$ |
| `CaloScore distilled` [50, 51] | $-4.1027 \pm 0.4292$ |
| `CaloScore single-shot` [50, 51] | $-4.6509 \pm 0.4837$ |
| `CaloDREAM` [69] | $\mathbf{-1.9761 \pm 0.1203}$ |
| `iCaloFlow teacher` [35] | $-3.9376 \pm 0.2385$ |
| `iCaloFlow student` [35] | $-4.6476 \pm 0.2627$ |
| `SuperCalo` [40] | $-4.4702 \pm 0.4064$ |
| `CaloDiT` [56] | $-5.8461 \pm 0.5629$ |
| `DeepTree` [28, 29] | $-8.5889 \pm 0.6987$ |
| `CaloPointFlow` [42] | $-9.0910 \pm 0.4704$ |
| `CaloVAE+INN` [38] | $-6.9001 \pm 0.3468$ |
| `CaloLatent` [67] | $-8.2169 \pm 0.5325$ |

Table C19: Precision, density, recall, and coverage for ds 2 submissions. A visualization is shown in figure 71.

| Submission | Precision ↑ | Density ↑ | Recall ↑ | Coverage ↑ |
|---|---|---|---|---|
| GEANT4 | 0.239 | 1.021 | 0.241 | 0.971 |
| `CaloDiffusion` [44] | 0.239 | 1.236 | 0.235 | 0.933 |
| `conv. L2LFlows` [32] | 0.231 | 1.656 | 0.177 | 0.969 |
| `CaloINN` [38] | 0.193 | 4.573 | 0.090 | 0.957 |
| `MDMA` [23, 24] | 0.003 | 0.009 | 0.937 | 0.033 |
| `Calo-VQ` [58] | 0.345 | 254.397 | 0.217 | 0.868 |
| `CaloScore` [50, 51] | 0.228 | 1.013 | 0.228 | 0.933 |
| `CaloScore distilled` [50, 51] | 0.197 | 1.407 | 0.181 | 0.880 |
| `CaloScore single-shot` [50, 51] | 0.171 | 1.056 | 0.208 | 0.852 |
| `iCaloFlow teacher` [35] | 0.152 | 0.809 | 0.253 | 0.817 |
| `iCaloFlow student` [35] | 0.155 | 1.354 | 0.253 | 0.827 |
| `SuperCalo` [40] | 0.120 | 0.347 | 0.310 | 0.692 |
| `DeepTree` [28, 29] | 0.003 | 0.013 | 0.834 | 0.045 |
| `CaloPointFlow` [42] | 0.016 | 0.575 | 0.335 | 0.487 |
| `CaloVAE+INN` [38] | 0.739 | 1793.855 | 0.026 | 0.961 |
| `CaloLatent` [67] | 0.016 | 0.176 | 0.622 | 0.235 |
| `CaloDiT` [56] | 0.500 | 10.228 | 0.060 | 0.924 |
| `CaloDREAM` [69] | 0.253 | 1.146 | 0.220 | 0.976 |

Table C20: Number of trainable parameters in training and for generation for ds 2 submissions. A visualization is shown in figure 72.

| Submission | number of parameters ↓ total | generator only |
|---|---|---|
| `CaloDiffusion` [44] | 517 969 | 517 969 |
| `conv. L2LFlows` [32] | 158 017 226 | 158 017 226 |
| `CaloINN` [38] | 270 999 370 | 270 999 370 |
| `MDMA` [23, 24] | **108 656** | **66 416** |
| `Calo-VQ` [58] | 3 317 546 | 1 231 433 |
| `CaloScore` [50, 51] | 14 436 206 | 14 436 206 |
| `CaloScore distilled` [50, 51] | 28 872 412 | 14 436 206 |
| `CaloScore single-shot` [50, 51] | 28 872 412 | 14 436 206 |
| `iCaloFlow teacher` [35] | 19 470 168 | 19 470 168 |
| `iCaloFlow student` [35] | 41 237 080 | 24 519 512 |
| `SuperCalo` [40] | 87 465 608 | 87 465 608 |
| `DeepTree` [28, 29] | 2 240 496 | 527 676 |
| `CaloPointFlow` [42] | 14 215 334 | 14 215 334 |
| `CaloVAE+INN` [38] | 96 356 674 | 48 393 824 |
| `CaloLatent` [67] | 10 707 408 | 1 942 402 |
| `CaloDiT` [56] | 1 221 544 | 1 221 544 |
| `CaloDREAM` [69] | 28 427 393 | 28 427 393 |

Table C21: Timing of ds 2 submissions on a CPU. Superscripts indicate if fewer than 100 000 events were generated in timing the submission. "LLVM: o.o.m." crashes with an LLVM out of memory error, "o.o.m." crashes with an memory allocation error, "> 17 280" translates to >48h/batch. A visualization of these timings is shown in figure 73.

| Submission | CPU [ms per shower] | | |
|---|---|---|---|
| | batch size 1 | batch size 100 | batch size 10 000 |
| `CaloDiffusion` [44] | $36\,502 \pm 2127^{(1000)}$ | $23\,317 \pm 3343^{(1000)}$ | $> 17\,280$ |
| `conv. L2LFlows` [32] | $1969 \pm 234^{(10\,000)}$ | $121.2 \pm 4.4$ | $89.2 \pm 13.2$ |
| `CaloINN` [38] | $387.7 \pm 31.6$ | $59.7 \pm 10.1$ | $46.2 \pm 0.7$ |
| `MDMA` [23, 24] | $\mathbf{14.3 \pm 0.6}$ | $19.9 \pm 5.3$ | $32.4 \pm 4.5$ |
| `Calo-VQ` [58] | $168.4 \pm 7.3$ | $26.1 \pm 1.2$ | $36.2 \pm 9.6$ |
| `CaloScore` [50, 51] | $133\,549 \pm 5286^{(1000)}$ | $147\,694 \pm 12\,981^{(500)}$ | LLVM: o.o.m. |
| `CaloScore distilled` [50, 51] | $16\,785 \pm 388^{(10\,000)}$ | $17\,989 \pm 595^{(1000)}$ | LLVM: o.o.m. |
| `CaloScore single-shot` [50, 51] | $406.6 \pm 14.3$ | $278.0 \pm 16.6$ | LLVM: o.o.m. |
| `iCaloFlow teacher` [35] | $250\,171 \pm 18\,156^{(100)}$ | $11\,614 \pm 380^{(1000)}$ | $8179 \pm 164^{(20\,000)}$ |
| `iCaloFlow student` [35] | $3048 \pm 102^{(10\,000)}$ | $135.2 \pm 4.9$ | $77.6 \pm 1.5$ |
| `SuperCalo` [40] | $397\,940 \pm 114\,537^{(100)}$ | $7988 \pm 470^{(1000)}$ | $7609 \pm 663^{(20\,000)}$ |
| `DeepTree` [28, 29] | $67.5 \pm 3.4$ | $38.3 \pm 3.3$ | $48.4 \pm 3.9$ |
| `CaloPointFlow` [42] | $154.6 \pm 5.6$ | $161.1 \pm 25.2$ | $132.6 \pm 27.0$ |
| `CaloVAE+INN` [38] | $64.1 \pm 3.9$ | $\mathbf{4.60 \pm 0.19}$ | $\mathbf{3.38 \pm 0.31}$ |
| `CaloLatent` [67] | $6611 \pm 577^{(10\,000)}$ | $541.3 \pm 54.9$ | LLVM: o.o.m. |
| `CaloDiT` [56] | $24\,642 \pm 1883^{(1000)}$ | $33\,355 \pm 4854^{(1000)}$ | o.o.m. |
| `CaloDREAM` [69] | $16\,942 \pm 1707^{(1000)}$ | $5052 \pm 496^{(10\,000)}$ | $5727 \pm 271^{(20\,000)}$ |

Table C22: Timing of ds 2 submissions on a GPU. Superscripts indicate if fewer than 100 000 events were generated in timing the submission, "CUDA o.o.m." ran out of VRAM on the GPU; and "array o.o.m." crashed because created arrays were too large. A visualization of these timings is shown in figure 73.

| Submission | GPU [ms per shower] | | |
|---|---|---|---|
| | batch size 1 | batch size 100 | batch size 10 000 |
| `CaloDiffusion` [44] | $5291 \pm 88^{(5000)}$ | $99.5 \pm 1.8$ | CUDA o.o.m. |
| `conv. L2LFlows` [32] | $1409 \pm 29^{(10\,000)}$ | $14.2 \pm 0.1$ | $1.64 \pm 0.01$ |
| `CaloINN` [38] | $53.4 \pm 0.9$ | $\mathbf{1.18 \pm 0.03}$ | $0.65 \pm 0.03$ |
| `MDMA` [23, 24] | $\mathbf{6.4 \pm 0.1}$ | $\mathbf{1.2 \pm 0.1}$ | CUDA o.o.m. |
| `Calo-VQ` [58] | $72.8 \pm 1.0$ | $\mathbf{1.1 \pm 0.1}$ | $\mathbf{0.36 \pm 0.01}$ |
| `CaloScore` [50, 51] | $2389 \pm 92^{(5000)}$ | $241.3 \pm 0.6$ | array o.o.m. |
| `CaloScore distilled` [50, 51] | $470.6 \pm 37.5^{(50\,000)}$ | $31.9 \pm 0.3$ | array o.o.m. |
| `CaloScore single-shot` [50, 51] | $138.2 \pm 3.0$ | $2.5 \pm 0.1$ | array o.o.m. |
| `iCaloFlow teacher` [35] | $77\,016 \pm 3447^{(100)}$ | $829.4 \pm 16.7^{(10\,000)}$ | $56.1 \pm 0.1$ |
| `iCaloFlow student` [35] | $1127 \pm 12^{(10\,000)}$ | $13.2 \pm 0.5$ | $1.45 \pm 0.05$ |
| `SuperCalo` [40] | $8508 \pm 77^{(1000)}$ | $103.0 \pm 1.6$ | CUDA o.o.m. |
| `DeepTree` [28, 29] | $37.3 \pm 0.9$ | $5.82 \pm 0.14$ | CUDA o.o.m. |
| `CaloPointFlow` [42] | $49.9 \pm 1.2$ | $3.00 \pm 0.04$ | CUDA o.o.m. |
| `CaloVAE+INN` [38] | $41.5 \pm 0.5$ | $1.22 \pm 0.03$ | $0.77 \pm 0.02$ |
| `CaloLatent` [67] | $8497 \pm 79^{(2500)}$ | $79.8 \pm 8.5$ | array o.o.m. |
| `CaloDiT` [56] | $1036 \pm 18^{(25\,000)}$ | $179.0 \pm 0.7$ | CUDA o.o.m. |
| `CaloDREAM` [69] | $4846 \pm 47^{(10\,000)}$ | $74.3 \pm 0.8$ | CUDA o.o.m. |

Table C23: Low-level and high-level AUCs for evaluating GEANT4 vs. submission of ds 3, averaged over 10 independent evaluation runs. For visualization, see Figure 83.

| Submission | AUC ↓ low-level | high-level | ResNet |
|---|---|---|---|
| GEANT4 | $0.498 \pm 0.002$ | $0.500 \pm 0.003$ | $0.499 \pm 0.002$ |
| `CaloDiffusion` [44] | $\mathbf{0.561 \pm 0.003}$ | $0.607 \pm 0.005$ | $\mathbf{0.656 \pm 0.015}$ |
| `L2LFlows-MAF` [31, 32] | $0.720 \pm 0.016$ | $0.946 \pm 0.002$ | $1.000 \pm 0.000$ |
| `conv. L2LFlows` [32] | $0.588 \pm 0.004$ | $0.733 \pm 0.006$ | $0.919 \pm 0.003$ |
| `MDMA` [23, 24] | $0.944 \pm 0.002$ | $0.987 \pm 0.001$ | $1.000 \pm 0.000$ |
| `CaloClouds` [46, 47] | $0.865 \pm 0.005$ | $0.980 \pm 0.001$ | $1.000 \pm 0.000$ |
| `Calo-VQ` [58] | $0.996 \pm 0.001$ | $0.998 \pm 0.000$ | $1.000 \pm 0.000$ |
| `Calo-VQ(norm)` [58] | $0.975 \pm 0.003$ | $0.994 \pm 0.000$ | $1.000 \pm 0.000$ |
| `CaloScore distilled` [50, 51] | $0.776 \pm 0.005$ | $0.924 \pm 0.002$ | $0.994 \pm 0.001$ |
| `CaloScore single-shot` [50, 51] | $0.807 \pm 0.005$ | $0.939 \pm 0.001$ | $0.995 \pm 0.002$ |
| `iCaloFlow teacher` [35] | $0.911 \pm 0.003$ | $0.962 \pm 0.001$ | $1.000 \pm 0.000$ |
| `iCaloFlow student` [35] | $0.891 \pm 0.003$ | $0.971 \pm 0.001$ | $1.000 \pm 0.000$ |
| `Geant4-Transformer` [65] | $0.886 \pm 0.011$ | $1.000 \pm 0.000$ | $1.000 \pm 0.000$ |
| `CaloPointFlow` [42] | $0.720 \pm 0.012$ | $0.945 \pm 0.002$ | $1.000 \pm 0.000$ |
| `CaloVAE+INN` [38] | $0.881 \pm 0.005$ | $1.000 \pm 0.000$ | $1.000 \pm 0.000$ |
| `CaloDREAM` [69] | $0.630 \pm 0.005$ | $\mathbf{0.524 \pm 0.004}$ | $0.802 \pm 0.014$ |

Table C24: KPD and FPD for evaluating GEANT4 vs. submission of ds 3. For visualization, see Figure 84.

| Submission | KPD $\cdot 10^3 \downarrow$ | FPD $\cdot 10^3 \downarrow$ |
|---|---|---|
| GEANT4 | $-0.0091 \pm 0.0466$ | $8.7578 \pm 0.5587$ |
| `CaloDiffusion` [44] | $0.2278 \pm 0.0978$ | $71.2380 \pm 1.9208$ |
| `L2LFlows-MAF` [31, 32] | $1.5398 \pm 0.1831$ | $665.4975 \pm 1.6930$ |
| `conv. L2LFlows` [32] | $0.3245 \pm 0.1521$ | $171.6365 \pm 1.7965$ |
| `MDMA` [23, 24] | $1.6705 \pm 0.1370$ | $588.6035 \pm 2.5358$ |
| `CaloClouds` [46, 47] | $5.1826 \pm 0.7016$ | $948.2275 \pm 4.6265$ |
| `Calo-VQ` [58] | $5.6838 \pm 0.2075$ | $1193.9149 \pm 2.8258$ |
| `Calo-VQ(norm)` [58] | $3.9937 \pm 0.3564$ | $930.8472 \pm 3.4598$ |
| `CaloScore distilled` [50, 51] | $1.7304 \pm 0.2490$ | $610.8560 \pm 4.0175$ |
| `CaloScore single-shot` [50, 51] | $1.5934 \pm 0.1380$ | $584.0234 \pm 2.9294$ |
| `iCaloFlow teacher` [35] | $2.8602 \pm 0.3240$ | $897.5908 \pm 5.2608$ |
| `iCaloFlow student` [35] | $2.5991 \pm 0.2299$ | $841.1136 \pm 5.1413$ |
| `Geant4-Transformer` [65] | $241.0380 \pm 2.6919$ | $22947.3168 \pm 23.3703$ |
| `CaloPointFlow` [42] | $2.0229 \pm 0.4123$ | $670.7538 \pm 3.3806$ |
| `CaloVAE+INN` [38] | $83.0692 \pm 0.7260$ | $11060.7266 \pm 13.9947$ |
| `CaloDREAM` [69] | $\mathbf{0.0098 \pm 0.0145}$ | $\mathbf{20.7469 \pm 1.0767}$ |

Table C25: Log-posterior scores for ds 3 GEANT4 test data, averaged over 10 independent DNN classifier trainings. For visualization, see Figure 85.

| Submission | Log-posterior $\uparrow$ |
|---|---|
| `CaloDiffusion` [44] | $-2.3860 \pm 0.0063$ |
| `L2LFlows-MAF` [31, 32] | $-4.3836 \pm 0.1634$ |
| `conv. L2LFlows` [32] | $-2.6992 \pm 0.0171$ |
| `MDMA` [23, 24] | $-9.8424 \pm 0.3800$ |
| `CaloClouds` [46, 47] | $-5.9925 \pm 0.2542$ |
| `Calo-VQ` [58] | $-19.8196 \pm 0.6060$ |
| `Calo-VQ(norm)` [58] | $-9.6010 \pm 0.2645$ |
| `CaloScore distilled` [50, 51] | $-3.2759 \pm 0.0844$ |
| `CaloScore single-shot` [50, 51] | $-3.8002 \pm 0.1135$ |
| `CaloDREAM` [69] | $\mathbf{-2.2337 \pm 0.0132}$ |
| `iCaloFlow teacher` [35] | $-6.7583 \pm 0.2071$ |
| `iCaloFlow student` [35] | $-5.8949 \pm 0.1835$ |
| `Geant4-Transformer` [65] | $-12.0696 \pm 0.4366$ |
| `CaloPointFlow` [42] | $-3.6853 \pm 0.1748$ |
| `CaloVAE+INN` [38] | $-6.4805 \pm 0.2134$ |

Table C26: Log-posterior scores for ds 3 GEANT4 test data, averaged over 10 independent CNN ResNet classifier trainings. For visualization, see Figure 86.

| Submission | Log-posterior ↑ |
|---|---|
| `CaloDiffusion` [44] | $\mathbf{-1.1254 \pm 0.1035}$ |
| `L2LFlows-MAF` [31, 32] | $-9.8665 \pm 0.5211$ |
| `conv. L2LFlows` [32] | $-2.3183 \pm 0.1994$ |
| `MDMA` [23, 24] | $-10.1538 \pm 0.6095$ |
| `CaloClouds` [46, 47] | $-10.2566 \pm 0.5892$ |
| `Calo-VQ` [58] | $-8.9864 \pm 0.3448$ |
| `Calo-VQ(norm)` [58] | $-8.3526 \pm 0.4014$ |
| `CaloScore distilled` [50, 51] | $-5.3080 \pm 0.4710$ |
| `CaloScore single-shot` [50, 51] | $-6.3708 \pm 0.5287$ |
| `CaloDREAM` [69] | $-1.4174 \pm 0.1267$ |
| `iCaloFlow teacher` [35] | $-9.1767 \pm 0.4420$ |
| `iCaloFlow student` [35] | $-9.3465 \pm 0.5997$ |
| `Geant4-Transformer` [65] | $-9.9694 \pm 0.5631$ |
| `CaloPointFlow` [42] | $-10.1480 \pm 0.3690$ |
| `CaloVAE+INN` [38] | $-9.3558 \pm 0.4615$ |

Table C27: Precision, density, recall, and coverage for ds 3 submissions. A visualization is shown in figure 87.

| Submission | Precision ↑ | Density ↑ | Recall ↑ | Coverage ↑ |
|---|---|---|---|---|
| GEANT4 | 0.098 | 1.145 | 0.095 | 0.980 |
| `CaloDiffusion` [44] | 0.122 | 1.118 | 0.090 | 0.973 |
| `L2LFlows-MAF` [31, 32] | 0.079 | 6.126 | 0.068 | 0.870 |
| `conv. L2LFlows` [32] | 0.109 | 1.667 | 0.065 | 0.887 |
| `MDMA` [23, 24] | 0.000 | 0.097 | 0.790 | 0.233 |
| `CaloClouds` [46, 47] | 0.001 | 0.000 | 0.841 | 0.002 |
| `Calo-VQ` [58] | 0.313 | 199.747 | 0.000 | 0.948 |
| `Calo-VQ(norm)` [58] | 0.561 | 227.616 | 0.002 | 1.000 |
| `CaloScore distilled` [50, 51] | 0.078 | 1.020 | 0.085 | 0.708 |
| `CaloScore single-shot` [50, 51] | 0.038 | 0.262 | 0.153 | 0.536 |
| `iCaloFlow teacher` [35] | 0.070 | 0.989 | 0.123 | 0.786 |
| `iCaloFlow student` [35] | 0.079 | 1.463 | 0.115 | 0.793 |
| `Geant4-Transformer` [65] | 0.623 | 1570.369 | 0.074 | 0.478 |
| `CaloPointFlow` [42] | 0.004 | 0.045 | 0.284 | 0.171 |
| `CaloVAE+INN` [38] | 0.879 | 1990.146 | 0.009 | 0.787 |
| `CaloDREAM` [69] | 0.114 | 1.637 | 0.079 | 0.989 |

Table C28: Number of trainable parameters in training and for generation for ds 3 submissions. A visualization is shown in figure 88.

| Submission | number of parameters ↓ total | generator only |
|---|---|---|
| `CaloDiffusion` [44] | 1 221 153 | 1 221 153 |
| `L2LFlows-MAF` [31, 32] | 556 526 578 | 556 526 578 |
| `conv. L2LFlows` [32] | 194 964 482 | 194 964 482 |
| `MDMA` [23, 24] | **108 656** | **66 416** |
| `CaloClouds` [46, 47] | 77 475 856 | 77 475 856 |
| `Calo-VQ` [58] | 2 155 763 | 876 050 |
| `Calo-VQ(norm)` [58] | 2 767 443 | 1 471 282 |
| `CaloScore distilled` [50, 51] | 28 872 412 | 14 436 206 |
| `CaloScore single-shot` [50, 51] | 28 872 412 | 14 436 206 |
| `iCaloFlow teacher` [35] | 95 088 152 | 95 088 152 |
| `iCaloFlow student` [35] | 187 423 704 | 95 088 152 |
| `Geant4-Transformer` [65] | 1 262 921 306 | 1 262 921 306 |
| `CaloPointFlow` [42] | 14 215 334 | 14 215 334 |
| `CaloVAE+INN` [38] | 204 609 270 | 93 935 070 |
| `CaloDREAM` [69] | 8 253 575 | 8 253 575 |

Table C29: Timing of ds 3 submissions on a CPU. Superscripts indicate if fewer than 100 000 events were generated in timing the submission, "LLVM: o.o.m." crashes with an LLVM out of memory error, "o.o.m." crashes with an memory allocation error, "> 17 280" translates to >48h/batch. A visualization of these timings is shown in figure 89.

| Submission | CPU [ms per shower] | | |
|---|---|---|---|
| | batch size 1 | batch size 100 | batch size 10 000 |
| `CaloDiffusion` [44] | $205\,235 \pm 6154^{(100)}$ | $269\,230 \pm 50\,007^{(100)}$ | o.o.m. |
| `L2LFlows-MAF` [31, 32] | $880\,131 \pm 61\,699^{(100)}$ | $141\,317 \pm 8151^{(1000)}$ | $> 17\,280$ |
| `conv. L2LFlows` [32] | $2357 \pm 283^{(10\,000)}$ | $340.2 \pm 15.2^{(10\,000)}$ | $428.7 \pm 25.7$ |
| `MDMA` [23, 24] | $\mathbf{32.1 \pm 1.4}$ | $193.7 \pm 12.7$ | o.o.m. |
| `CaloClouds` [46, 47] | $2404 \pm 96^{(10\,000)}$ | $3924 \pm 701^{(10\,000)}$ | o.o.m. |
| `Calo-VQ` [58] | $260.0 \pm 16.3$ | $68.1 \pm 4.7$ | $58.2 \pm 8.4$ |
| `Calo-VQ(norm)` [58] | $3957 \pm 269^{(10\,000)}$ | $3635 \pm 219^{(10\,000)}$ | $3543 \pm 311^{(20\,000)}$ |
| `CaloScore distilled` [50, 51] | $83\,500 \pm 3567^{(1000)}$ | $96\,869 \pm 6748^{(1000)}$ | o.o.m. |
| `CaloScore single-shot` [50, 51] | $1416 \pm 30$ | $1539 \pm 111^{(10\,000)}$ | o.o.m. |
| `iCaloFlow teacher` [35] | $15\,512\,081 \pm 3\,471\,602^{(10)}$ | $438\,642 \pm 55\,017^{(100)}$ | $> 17\,280$ |
| `iCaloFlow student` [35] | $20\,217 \pm 4133^{(1000)}$ | $454.7 \pm 20.5^{(10\,000)}$ | $446.3 \pm 58.4$ |
| `Geant4-Transformer` [65] | $762.3 \pm 35.9$ | $179.5 \pm 12.9$ | $193.4 \pm 15.1$ |
| `CaloPointFlow` [42] | $301.7 \pm 19.2$ | $308.5 \pm 6.4$ | o.o.m. |
| `CaloVAE+INN` [38] | $90.6 \pm 3.2$ | $\mathbf{13.6 \pm 0.9}$ | $\mathbf{16.2 \pm 1.9}$ |
| `CaloDREAM` [69] | $33\,292 \pm 672^{(1000)}$ | $32\,138 \pm 2721^{(1000)}$ | $> 17\,280$ |

Table C30: Timing of ds 3 submissions on a GPU. Superscripts indicate if fewer than 100 000 events were generated in timing the submission, "CUDA o.o.m." ran out of VRAM on the GPU. A visualization of these timings is shown in figure 89.

| Submission | GPU [ms per shower] | | |
|---|---|---|---|
| | batch size 1 | batch size 100 | batch size 10 000 |
| `CaloDiffusion` [44] | $6171 \pm 45^{(2500)}$ | $810.2 \pm 2.4^{(10\,000)}$ | CUDA o.o.m. |
| `L2LFlows-MAF` [31, 32] | $235\,537 \pm 4910^{(100)}$ | $2454 \pm 120^{(10\,000)}$ | $3112 \pm 1^{(10\,000)}$ |
| `conv. L2LFlows` [32] | $1430 \pm 16^{(10\,000)}$ | $16.0 \pm 0.2$ | $6.8 \pm 0.1$ |
| `MDMA` [23, 24] | $\mathbf{9.51 \pm 0.13}$ | $5.47 \pm 0.30$ | CUDA o.o.m. |
| `CaloClouds` [46, 47] | $94.9 \pm 1.1$ | $25.3 \pm 0.2$ | CUDA o.o.m. |
| `Calo-VQ` [58] | $127.4 \pm 1.9$ | $\mathbf{1.8 \pm 0.1}$ | $0.98 \pm 0.02$ |
| `Calo-VQ(norm)` [58] | $466.0 \pm 5.0^{(10\,000)}$ | $26.6 \pm 0.2$ | CUDA o.o.m. |
| `CaloScore distilled` [50, 51] | $473.2 \pm 24.3^{(50\,000)}$ | $162.2 \pm 0.5$ | CUDA o.o.m. |
| `CaloScore single-shot` [50, 51] | $135.5 \pm 5.9$ | $6.5 \pm 0.3$ | CUDA o.o.m. |
| `iCaloFlow teacher` [35] | $470\,081 \pm 3379^{(50)}$ | $5596 \pm 56^{(5000)}$ | $1979 \pm 1^{(10\,000)}$ |
| `iCaloFlow student` [35] | $1156 \pm 31^{(10\,000)}$ | $16.7 \pm 0.5$ | $6.0 \pm 0.2$ |
| `Geant4-Transformer` [65] | $203.2 \pm 5.8$ | $8.77 \pm 0.36$ | CUDA o.o.m. |
| `CaloPointFlow` [42] | $57.9 \pm 4.6$ | $5.52 \pm 0.03$ | CUDA o.o.m. |
| `CaloVAE+INN` [38] | $44.3 \pm 0.6$ | $3.83 \pm 0.09$ | $\mathbf{3.18 \pm 0.15}$ |
| `CaloDREAM` [69] | $5003 \pm 67^{(10\,000)}$ | $179.6 \pm 0.5$ | CUDA o.o.m. |

## D. Generation time vs. number of parameters

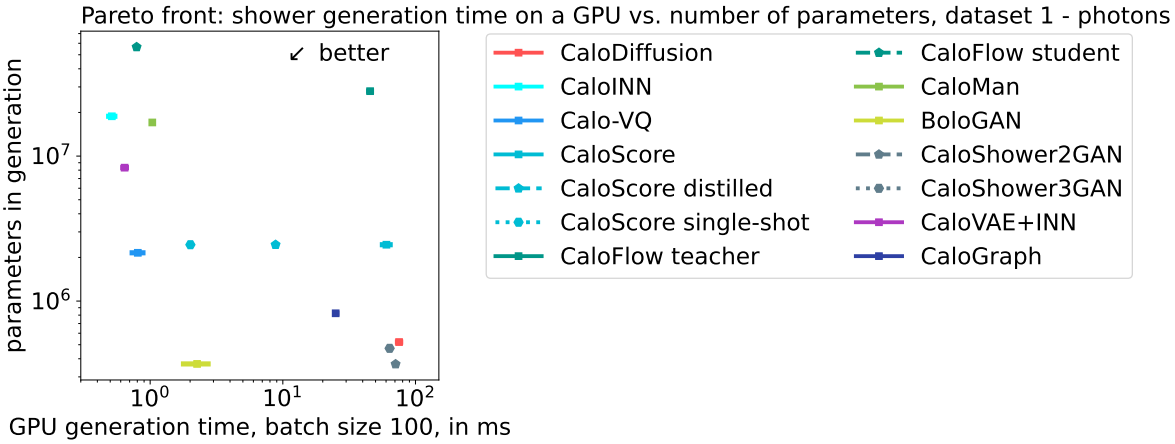



Figure D1: Pareto front in number of trainable parameters in generation (from figure 43 and table C5) and generation speed (from figure 44 and table C7).

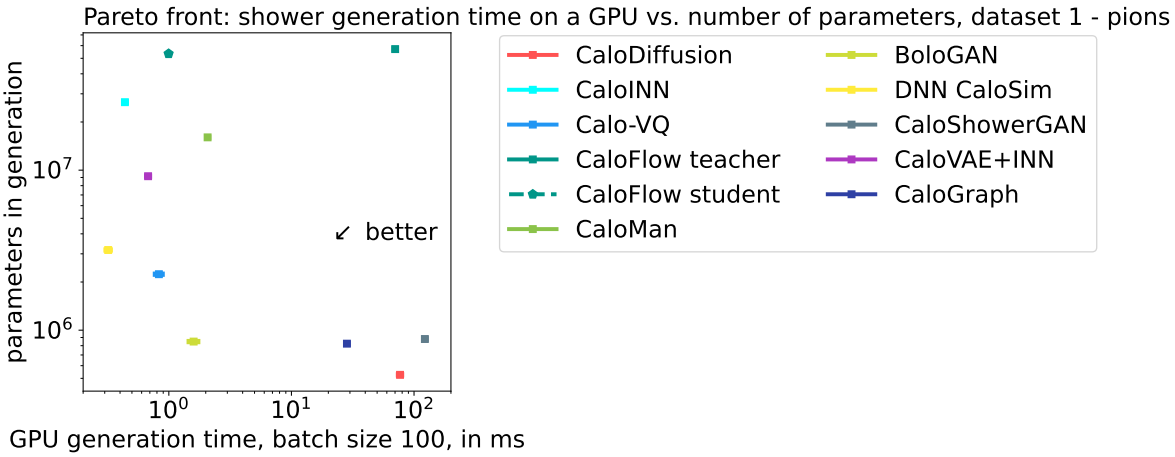



Figure D2: Pareto front in number of trainable parameters in generation (from figure 56 and table C12) and generation speed (from figure 57 and table C14).

When looking at the Pareto front in generation speed (here taken as the time it takes to generate in batches of 100 on a GPU) and the model size (in terms of number of trainable parameters in generation), we barely see an actual front emerging. The generation time strongly depends on the model architecture and not so much on the actual size of the submissions, as can be seen for example by the `CaloFlow` examples in ds $1 - \gamma$ in figure D1: `CaloFlow student` has more parameters than `CaloFlow teacher`, but is almost 2 orders of magnitude faster in sampling. The diffusion model `CaloDiffusion` even has the fewest number of parameters in figure D1, but is one of the slowest in sampling. Datasets 2 (in figure D3) and 3 (in figure D4) show even less

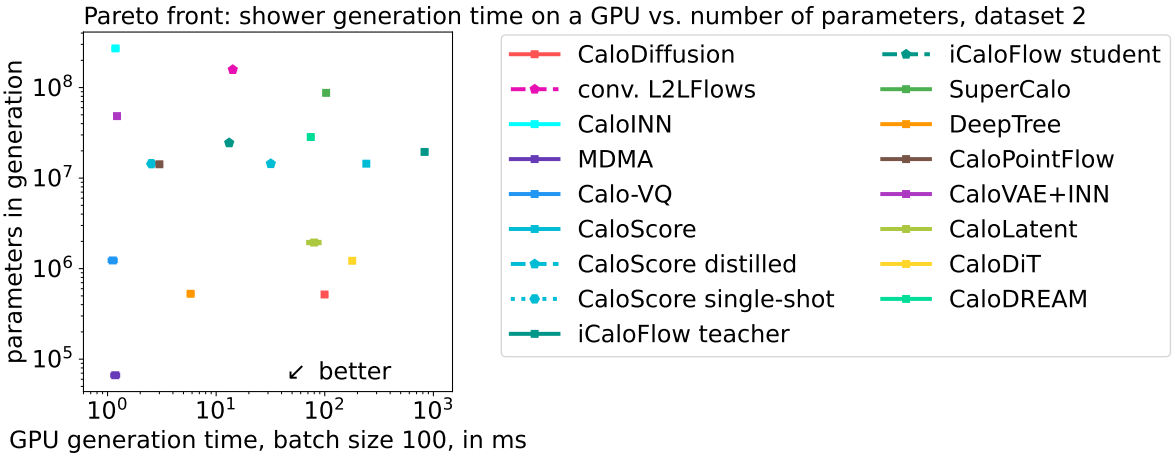



Figure D3: Pareto front in number of trainable parameters in generation (from figure 72 and table C20) and generation speed (from figure 73 and table C22).

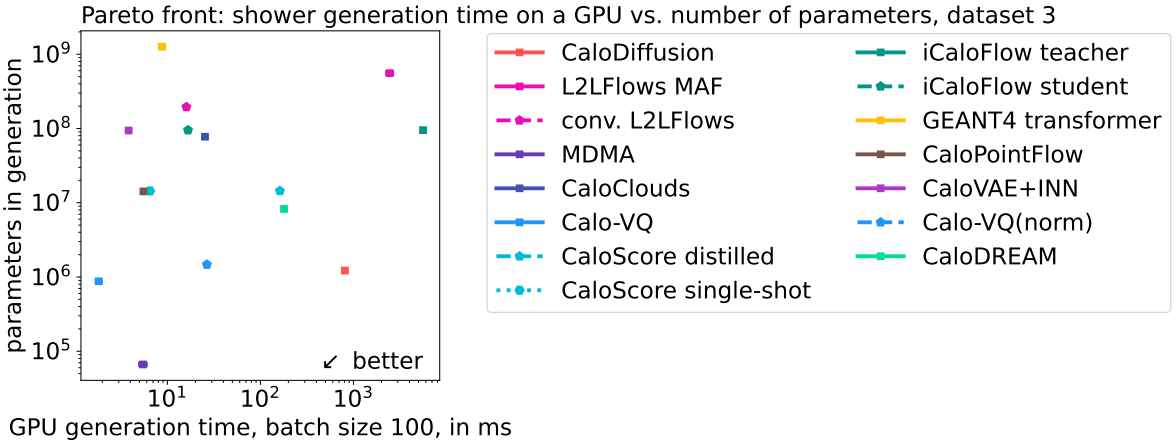



Figure D4: Pareto front in number of trainable parameters in generation (from figure 88 and table C28) and generation speed (from figure 89 and table C30).

of a front, but with `MDMA` a submission clearly in the sweet spot at few parameters and fast generation.

## References


[1] ATLAS Collaboration 2022 ATLAS Software and Computing HL-LHC Roadmap Tech. rep. CERN Geneva URL https://cds.cern.ch/record/2802918

[2] CMS Offline Software and Computing 2022 CMS Phase-2 Computing Model: Update Document Tech. rep. CERN Geneva URL https://cds.cern.ch/record/2815292

[3] Bozzi C 2024 LHCb Computing Resources: 2025 requests Tech. rep. CERN Geneva URL https://cds.cern.ch/record/2888939

[4] Agostinelli S, Allison J, Amako K, Apostolakis J, Araujo H, Arce P, Asai M, Axen D, Banerjee S, Barrand G, Behner F, Bellagamba L, Boudreau J, Broglia L, Brunengo A, Burkhardt H, Chauvie S, Chuma J, Chytracek R, Cooperman G, Cosmo G, Degtyarenko P and Dell'Acqua A 2003 *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **506** 250–303 ISSN 0168-9002 URL https://www.sciencedirect.com/science/article/pii/S0168900203013688

[5] Allison J, Amako K, Apostolakis J, Araujo H, Arce Dubois P, Asai M, Barrand G, Capra R, Chauvie S, Chytracek R, Cirrone G, Cooperman G, Cosmo G, Cuttone G, Daquino G, Donszelmann M, Dressel M, Folger G, Foppiano F, Generowicz J, Grichine V, Guatelli S, Gumplinger P, Heikkinen A, Hrivnacova I, Howard A, Incerti S, Ivanchenko V, Johnson T, Jones F, Koi T, Kokoulin R, Kossov M, Kurashige H, Lara V, Larsson S, Lei F, Link O, Longo F, Maire M, Mantero A, Mascialino B, McLaren I, Mendez Lorenzo P, Minamimoto K, Murakami K, Nieminen P, Pandola L, Parlati S, Peralta L, Perl J, Pfeiffer A, Pia M, Ribon A, Rodrigues P, Russo G, Sadilov S, Santin G, Sasaki T, Smith D, Starkov N, Tanaka S, Tcherniaev E, Tome B, Trindade A, Truscott P, Urban L, Verderi M, Walkden A, Wellisch J, Williams D, Wright D and Yoshida H 2006 *IEEE Transactions on Nuclear Science* **53** 270–278

[6] Allison J, Amako K, Apostolakis J, Arce P, Asai M, Aso T, Bagli E, Bagulya A, Banerjee S, Barrand G, Beck B, Bogdanov A, Brandt D, Brown J, Burkhardt H, Canal P, Cano-Ott D, Chauvie S, Cho K, Cirrone G, Cooperman G, Cortés-Giraldo M, Cosmo G, Cuttone G, Depaola G, Desorgher L, Dong X, Dotti A, Elvira V, Folger G, Francis Z, Galoyan A, Garnier L, Gayer M, Genser K, Grichine V, Guatelli S, Guèye P, Gumplinger P, Howard A, Hřivnáčová I, Hwang S, Incerti S, Ivanchenko A, Ivanchenko V, Jones F, Jun S, Kaitaniemi P, Karakatsanis N, Karamitros M, Kelsey M, Kimura A, Koi T, Kurashige H, Lechner A, Lee S, Longo F, Maire M, Mancusi D, Mantero A, Mendoza E, Morgan B, Murakami K, Nikitina T, Pandola L, Paprocki P, Perl J, Petrović I, Pia M, Pokorski W, Quesada J, Raine M, Reis M, Ribon A, Ristić Fira A, Romano F, Russo G, Santin G, Sasaki T, Sawkey D, Shin J, Strakovsky I, Taborda A, Tanaka S, Tomé B, Toshito T, Tran H, Truscott P, Urban L, Uzhinsky V, Verbeke J, Verderi M, Wendt B, Wenzel H, Wright D, Wright D, Yamashita T, Yarba J and Yoshida H 2016 *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **835** 186–225 ISSN 0168-9002 URL https://www.sciencedirect.com/science/article/pii/S0168900216306957

[7] ATLAS Collaboration 2010 The simulation principle and performance of the ATLAS fast calorimeter simulation FastCaloSim ATL-PHYS-PUB-2010-013 URL https://cds.cern.ch/record/1300517

[8] ATLAS Collaboration 2014 Performance of the Fast ATLAS Tracking Simulation (FATRAS) and the ATLAS Fast Calorimeter Simulation (FastCaloSim) with single particles ATL-SOFT-PUB-2014-001 URL https://cds.cern.ch/record/1669341

[9] ATLAS Collaboration 2021 *Comput. Softw. Big Sci.* **6** 7 (*Preprint* 2109.02551)

[10] Abdullin S, Azzi P, Beaudette F, Janot P and Perrotta A (CMS) 2011 *J. Phys. Conf. Ser.* **331** 032049

[11] Hildreth M, Ivanchenko V N and Lange D J (CMS) 2017 *J. Phys.: Conf. Ser.* **898** 042040 URL https://cds.cern.ch/record/2297284

[12] Paganini M, de Oliveira L and Nachman B 2018 *Phys. Rev. Lett.* **120** 042003 (*Preprint*

1705.02355)
[13] Hashemi B and Krause C 2024 *Rev. Phys.* **12** 100092 (*Preprint* 2312.09597)
[14] Barbetti M 2023 Lamarr: LHCb ultra-fast simulation based on machine learning models deployed within Gauss *21th International Workshop on Advanced Computing and Analysis Techniques in Physics Research: AI meets Reality* (*Preprint* 2303.11428)
[15] Butter A *et al.* 2019 *SciPost Phys.* **7** 014 (*Preprint* 1902.09914)
[16] Kasieczka G *et al.* 2021 *Rept. Prog. Phys.* **84** 124201 (*Preprint* 2101.08320)
[17] 2022 ML4Jets 2022 Conference, Rutgers University URL https://indico.cern.ch/event/1159913/
[18] 2023 ML4Jets 2023 Conference, DESY Hamburg URL https://indico.cern.ch/event/1253794/
[19] 2023 CaloChallenge Workshop, Villa Mondragone Frascati URL https://agenda.infn.it/event/34036/
[20] Ahmad F Y, Venkataswamy V and Fox G 2024 (*Preprint* 2406.12898)
[21] Faucci Giannelli M and Zhang R 2024 *Eur. Phys. J. Plus* **139** 597 (*Preprint* 2309.06515)
[22] Faucci Giannelli M and Zhang R 2024 GitHub repository https://gitlab.cern.ch/zhangruiPhysics/FastCaloChallenge
[23] Käch B and Melzer-Pellmann I 2023 (*Preprint* 2305.15254)
[24] Käch B, Melzer-Pellmann I and Krücker D 2024 (*Preprint* 2408.04997)
[25] Käch B and Melzer-Pellmann I 2023 GitHub repository https://github.com/kaechb/MDMA/tree/NeurIPS
[26] ATLAS Collaboration 2020 Fast simulation of the ATLAS calorimeter system with Generative Adversarial Networks ATL-SOFT-PUB-2020-006 URL https://cds.cern.ch/record/2746032
[27] Corchia F A G, Rinaldi L and Franchini M 2024 GitHub repository https://github.com/FedericoCorchia/BoloGAN
[28] Scham M A W, Krücker D, Käch B and Borras K 2023 *EPJ Web Conf.* (*Preprint* 2311.12616) URL http://arxiv.org/abs/2311.12616
[29] Scham M A W, Krücker D and Borras K 2023 (*Preprint* 2312.00042) URL http://arxiv.org/abs/2312.00042
[30] Scham M A W, Krücker D and Borras K 2023 GitHub repository https://github.com/DeGeSim/nips23DeepTreeGANv2
[31] Diefenbacher S, Eren E, Gaede F, Kasieczka G, Krause C, Shekhzadeh I and Shih D 2023 *JINST* **18** P10017 (*Preprint* 2302.11594)
[32] Buss T, Gaede F, Kasieczka G, Krause C and Shih D 2024 *JINST* **19** P09003 (*Preprint* 2405.20407)
[33] Buss T, Gaede F, Kasieczka G, Krause C and Shih D 2024 GitHub repository https://github.com/FLC-QU-hep/ConvL2LFlow
[34] Krause C, Pang I and Shih D 2024 *SciPost Phys.* **16** 126 (*Preprint* 2210.14245)
[35] Buckley M R, Krause C, Pang I and Shih D 2024 *Phys. Rev. D* **109** 033006 (*Preprint* 2305.11934)
[36] Krause C, Pang I and Shih D 2022 GitHub repository https://github.com/Ian-Pang/caloflow-for-calochallenge
[37] Buckley M R, Krause C, Pang I and Shih D 2023 GitHub repository https://github.com/claudius-krause/inductive-caloflow
[38] Ernst F, Favaro L, Krause C, Plehn T and Shih D 2025 *SciPost Phys.* **18** 081 (*Preprint* 2312.09290)
[39] Ernst F, Favaro L, Krause C, Plehn T and Shih D 2023 GitHub repository https://github.com/heidelberg-hepml/CaloINN
[40] Pang I, Shih D and Raine J A 2024 *Phys. Rev. D* **109** 092009 (*Preprint* 2308.11700)
[41] Pang I, Shih D and Raine J A 2023 GitHub repository https://github.com/Ian-Pang/supercalo
[42] Schnake S, Krücker D and Borras K 2024 (*Preprint* 2403.15782)

[43] Schnake S, Krücker D and Borras K 2024 GitHub repository https://github.com/simonschnake/CaloPointFlow
[44] Amram O and Pedro K 2023 *Phys. Rev. D* **108** 072014 (*Preprint* 2308.03876)
[45] Amram O and Pedro K 2023 GitHub repository https://github.com/OzAmram/CaloDiffusionPaper
[46] Buhmann E, Diefenbacher S, Eren E, Gaede F, Kasieczka G, Korol A, Korcari W, Krüger K and McKeown P 2023 *JINST* **18** P11025 (*Preprint* 2305.04847)
[47] Buhmann E, Gaede F, Kasieczka G, Korol A, Korcari W, Krüger K and McKeown P 2024 *JINST* **19** P04020 (*Preprint* 2309.05704)
[48] Buhmann E, Diefenbacher S, Eren E, Gaede F, Kasieczka G, Korol A, Korcari W, Krüger K and McKeown P 2023 GitHub repository https://github.com/FLC-QU-hep/CaloClouds
[49] Buhmann E, Gaede F, Kasieczka G, Korol A, Korcari W, Krüger K and McKeown P 2023 GitHub repository https://github.com/FLC-QU-hep/CaloClouds-2
[50] Mikuni V and Nachman B 2022 *Phys. Rev. D* **106** 092009 (*Preprint* 2206.11898)
[51] Mikuni V and Nachman B 2024 *JINST* **19** P02001 (*Preprint* 2308.03847)
[52] Mikuni V and Nachman B 2022 GitHub repository https://github.com/ViniciusMikuni/CaloScore
[53] Mikuni V and Nachman B 2023 GitHub repository https://github.com/ViniciusMikuni/CaloScoreV2
[54] Kobylianskii D, Soybelman N, Dreyer E and Gross E 2024 *Phys. Rev. D* **110**(7) 072003 URL https://link.aps.org/doi/10.1103/PhysRevD.110.072003
[55] Kobylianskii D, Soybelman N, Dreyer E and Gross E 2024 GitHub repository https://github.com/dkobylianskii/CaloGraph
[56] Cardoso R P D C, Raikwar P, Zaborowska A, Salamani D, Jaruskova K, Vallecorsa S, Yeo K, Ekambaram V, Nguyen N, Kalagnanam J and Srivatsa M 2024 CaloDiT: Diffusion with transformers for fast shower simulation URL https://indico.cern.ch/event/1330797/contributions/5796591/
[57] Cardoso R P D C, Raikwar P, Zaborowska A, Salamani D, Jaruskova K, Vallecorsa S, Yeo K, Ekambaram V, Nguyen N, Kalagnanam J and Srivatsa M 2024 GitLab repository https://gitlab.cern.ch/redacost/calo_submission_renato
[58] Liu Q, Shimmin C, Liu X, Shlizerman E, Li S and Hsu S C 2024 (*Preprint* 2405.06605)
[59] Liu Q, Shimmin C, Liu X, Shlizerman E, Li S and Hsu S C 2024 GitHub repository https://github.com/qibin2020/calo-VQ
[60] Cresswell J C, Ross B L, Loaiza-Ganem G, Reyes-Gonzalez H, Letizia M and Caterini A L 2022 CaloMan: Fast generation of calorimeter showers with density estimation on learned manifolds *NeurIPS Workshop on Machine Learning and the Physical Sciences* (*Preprint* 2211.15380)
[61] Cresswell J C, Ross B L, Loaiza-Ganem G, Reyes-Gonzalez H, Letizia M and Caterini A L 2022 GitHub repository https://github.com/layer6ai-labs/calo-man
[62] Salamani D 2021 *Machine Learning Techniques for Fast Shower Simulation at the ATLAS Experiment* Ph.D. thesis Geneva U. URL https://doi.org/10.13097/archive-ouverte/unige:158540
[63] ATLAS Collaboration 2024 *Comput. Softw. Big Sci.* **8** 7 (*Preprint* 2210.06204)
[64] Salamani D 2024 GitHub repository https://github.com/DalilaSalamani/ATLASDNNCaloSim
[65] Raikwar P, Cardoso R, Chernyavskaya N, Jaruskova K, Pokorski W, Salamani D, Srivatsa M, Tsolaki K, Vallecorsa S and Zaborowska A 2024 *EPJ Web of Conf.* **295** 09039 URL https://doi.org/10.1051/epjconf/202429509039
[66] Raikwar P, Cardoso R, Chernyavskaya N, Jaruskova K, Pokorski W, Salamani D, Srivatsa M, Tsolaki K, Vallecorsa S and Zaborowska A 2024 GitLab repository https://gitlab.cern.ch/praikwar/ml4fastsim/-/tree/updated?ref_type=heads
[67] Madula T and Mikuni V M 2023 CaloLatent: Score-based Generative Modelling in the Latent Space for Calorimeter Shower Generation *NeurIPS Workshop on Machine Learning and the Physical Sciences* URL https://ml4physicalsciences.github.io/2023/files/NeurIPS_ML4PS_2023_19.pdf
[68] Madula T and Mikuni V M 2023 GitHub repository https://github.com/thandi1908/CaloLatent

[69] Favaro L, Ore A, Schweitzer S P and Plehn T 2025 *SciPost Phys.* **18** 088 (*Preprint* 2405.09629)
[70] Favaro L, Ore A, Schweitzer S P and Plehn T 2024 GitHub repository https://github.com/luigifvr/calo_dreamer
[71] Cresswell J C and Kim T 2024 Scaling Up Diffusion and Flow-based XGBoost Models *ICML Workshop on AI for Science* (*Preprint* 2408.16046) URL https://openreview.net/forum?id=Jnu4adAVeA
[72] Cresswell J C and Kim T 2024 GitHub repository https://github.com/layer6ai-labs/calo-forest
[73] Faucci Giannelli M, Kasieczka G, Krause C, Nachman B, Salamani D, Shih D and Zaborowska A 2022 Fast calorimeter simulation challenge 2022 github page https://github.com/CaloChallenge/homepage
[74] Collette A 2013 *Python and HDF5* (O'Reilly) URL https://www.h5py.org/
[75] Faucci Giannelli M, Kasieczka G, Krause C, Nachman B, Salamani D, Shih D and Zaborowska A 2022 Fast Calorimeter Simulation Challenge 2022 - Dataset 1 URL https://doi.org/10.5281/zenodo.6368338
[76] Faucci Giannelli M, Kasieczka G, Krause C, Nachman B, Salamani D, Shih D and Zaborowska A 2023 Fast Calorimeter Simulation Challenge 2022 - Updated Dataset 1 URL https://doi.org/10.5281/zenodo.8099322
[77] ATLAS Collaboration 2023 Datasets used to train the Generative Adversarial Networks used in ATLFast3 URL https://doi.org/10.7483/OPENDATA.ATLAS.UXKX.TXBN
[78] ATLAS Collaboration 2017 *Eur. Phys. J. C* **77** 490 (*Preprint* 1603.02934)
[79] Geant4 Collaboration Par04 example URL https://gitlab.cern.ch/geant4/geant4/-/tree/master/examples/extended/parameterisations/Par04
[80] Salamani D and Zaborowska A 2022 High Granularity Electromagnetic Calorimeter Shower Images URL https://doi.org/10.5281/zenodo.6082201
[81] Particle Data Group Atomic and nuclear properties of tungsten (w) URL https://pdg.lbl.gov/2020/AtomicNuclearProperties/HTML/tungsten_W.html
[82] Faucci Giannelli M, Kasieczka G, Krause C, Nachman B, Salamani D, Shih D and Zaborowska A 2022 Fast Calorimeter Simulation Challenge 2022 - Dataset 2 URL https://doi.org/10.5281/zenodo.6366271
[83] Faucci Giannelli M, Kasieczka G, Krause C, Nachman B, Salamani D, Shih D and Zaborowska A 2022 Fast Calorimeter Simulation Challenge 2022 - Dataset 3 URL https://doi.org/10.5281/zenodo.6366324
[84] Goodfellow I J, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A and Bengio Y 2014 *arXiv e-prints* arXiv:1406.2661 (*Preprint* 1406.2661)
[85] Karras T, Laine S and Aila T 2019 A style-based generator architecture for generative adversarial networks (*Preprint* 1812.04948)
[86] Arjovsky M and Bottou L 2017 *arXiv e-prints* arXiv:1701.04862 (*Preprint* 1701.04862)
[87] Arjovsky M, Chintala S and Bottou L 2017 *arXiv e-prints* arXiv:1701.07875 (*Preprint* 1701.07875)
[88] Paganini M, de Oliveira L and Nachman B 2018 *Phys. Rev. D* **97** 014021 (*Preprint* 1712.10321)
[89] ATLAS Collaboration (ATLAS) 2021 FastCaloGAN Training Project (1.0), URL https://doi.org/10.5281/zenodo.5589623
[90] Kingma D P and Ba J 2014 *ICLR* arXiv:1412.6980 (*Preprint* 1412.6980)
[91] Kansal R, Duarte J, Su H, Orzari B, Tomei T, Pierini M, Touranakou M, Vlimant J R and Gunopulos D 2022 Jetnet URL https://doi.org/10.5281/zenodo.6975118
[92] Gulrajani I, Ahmed F, Arjovsky M, Dumoulin V and Courville A 2017 Improved training of Wasserstein GANs *Advances in Neural Information Processing Systems* vol 30 ed Guyon I, Luxburg U V, Bengio S, Wallach H, Fergus R, Vishwanathan S and Garnett R (Curran Associates, Inc.) p 5767 (*Preprint* 1704.00028) URL http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.pdf
[93] Xiang S and Li H 2017 On the effects of batch and weight normalization in generative adversarial

networks (*Preprint* 1704.03971)

[94] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado G S, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y and Zheng X 2015 TensorFlow: Large-scale machine learning on heterogeneous systems software available from tensorflow.org URL https://www.tensorflow.org/

[95] Xu K, Hu W, Leskovec J and Jegelka S 2018 *arXiv e-prints* arXiv:1810.00826 (*Preprint* 1810.00826)

[96] Fey M and Lenssen J E 2019 *ICLR Workshop on Representation Learning on Graphs and Manifolds* arXiv:1903.02428 (*Preprint* 1903.02428)

[97] Ioffe S and Szegedy C 2015 Batch normalization: Accelerating deep network training by reducing internal covariate shift (*Preprint* 1502.03167)

[98] Brody S, Alon U and Yahav E 2021 *International Conference on Learning Representations* arXiv:2105.14491 (*Preprint* 2105.14491) URL https://openreview.net/forum?id=F72ximsx7C1

[99] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M and Duchesnay E 2011 *Journal of Machine Learning Research* **12** 2825–2830

[100] Kobyzev I, Prince S J D and Brubaker M A 2021 *IEEE Trans. Pattern Anal. Machine Intell.* **43** 3964–3979 (*Preprint* 1908.09257)

[101] Papamakarios G, Nalisnick E, Rezende D J, Mohamed S and Lakshminarayanan B 2021 *J. Machine Learning Res.* **22** 2617–2680 (*Preprint* 1912.02762)

[102] Ross B and Cresswell J 2021 Tractable density estimation on learned manifolds with conformal embedding flows *Advances in Neural Information Processing Systems* vol 34 (Curran Associates, Inc.) pp 26635–26648 (*Preprint* 2106.05275) URL https://proceedings.neurips.cc/paper/2021/hash/dfd786998e082758be12670d856df755-Abstract.html

[103] Papamakarios G, Pavlakou T and Murray I 2017 *arXiv e-prints* arXiv:1705.07057 (*Preprint* 1705.07057)

[104] Kingma D P, Salimans T, Jozefowicz R, Chen X, Sutskever I and Welling M 2016 *Advances in neural information processing systems* **29** arXiv:1606.04934 (*Preprint* 1606.04934)

[105] Dinh L, Sohl-Dickstein J and Bengio S 2016 (*Preprint* 1605.08803)

[106] Kingma D P and Dhariwal P 2018 *Advances in Neural Information Processing Systems* **31** (*Preprint* 1807.03039) URL https://proceedings.neurips.cc/paper/2018/file/d139db6a236200b21cc7f752979132d0-Paper.pdf

[107] Ardizzone L, Kruse J, Wirkert S, Rahner D, Pellegrini E W, Klessen R S, Maier-Hein L, Rother C and Köthe U 2019 Analyzing inverse problems with invertible neural networks (*Preprint* 1808.04730) URL https://arxiv.org/abs/1808.04730

[108] Germain M, Gregor K, Murray I and Larochelle H 2015 *arXiv e-prints* arXiv:1502.03509 (*Preprint* 1502.03509)

[109] Krause C and Shih D 2023 *Phys. Rev. D* **107** 113003 (*Preprint* 2106.05285)

[110] Durkan C, Bekasov A, Murray I and Papamakarios G 2019 *Advances in Neural Information Processing Systems* **32** (*Preprint* 1906.04032) URL https://proceedings.neurips.cc/paper/2019/file/7ac71d433f282034e088473244df8c02-Paper.pdf

[111] Ronneberger O, Fischer P and Brox T 2015 U-net: Convolutional networks for biomedical image segmentation *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* ed Navab N, Hornegger J, Wells W M and Frangi A F (Cham: Springer International Publishing) p 234 ISBN 978-3-319-24574-4 (*Preprint* 1505.04597)

[112] Dinh L, Krueger D and Bengio Y 2014 *arXiv e-prints* arXiv:1410.8516 (*Preprint* 1410.8516)

[113] Smith L N and Topin N 2017 *arXiv e-prints* arXiv:1708.07120 (*Preprint* 1708.07120)

[114] Loshchilov I and Hutter F 2017 *arXiv e-prints* arXiv:1711.05101 (*Preprint* 1711.05101)
[115] Paszke A *et al.* 2019 *Advances in Neural Information Processing Systems 32* 8024–8035 Advances in Neural Information Processing Systems 32 pp. 8024–8035 URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[116] Durkan C, Bekasov A, Murray I and Papamakarios G 2020 nflows: normalizing flows in PyTorch URL https://doi.org/10.5281/zenodo.4296287
[117] Krause C and Shih D 2023 *Phys. Rev. D* **107** 113004 (*Preprint* 2110.11377)
[118] van den Oord A, Li Y, Babuschkin I, Simonyan K, Vinyals O, Kavukcuoglu K, van den Driessche G, Lockhart E, Cobo L, Stimberg F, Casagrande N, Grewe D, Noury S, Dieleman S, Elsen E, Kalchbrenner N, Zen H, Graves A, King H, Walters T, Belov D and Hassabis D 2018 Parallel WaveNet: Fast high-fidelity speech synthesis *Proceedings of the 35th International Conference on Machine Learning* (*Proceedings of Machine Learning Research* vol 80) ed Dy J and Krause A (PMLR) pp 3918–3926 URL https://proceedings.mlr.press/v80/oord18a.html
[119] Ardizzone L, Bungert T, Draxler F, Köthe U, Kruse J, Schmier R and Sorrenson P 2018-2022 Framework for Easily Invertible Architectures (FrEIA) URL https://github.com/vislearn/FrEIA
[120] Yang G, Huang X, Hao Z, Liu M Y, Belongie S and Hariharan B 2019 *Proceedings of the IEEE/CVF international conference on computer vision* arXiv:1906.12320 (*Preprint* 1906.12320)
[121] Zaheer M, Kottur S, Ravanbakhsh S, Poczos B, Salakhutdinov R and Smola A 2017 *Advances in Neural Information Processing Systems* **30** arXiv:1703.06114 (*Preprint* 1703.06114) URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf
[122] Uria B, Murray I and Larochelle H 2013 RNADE: The real-valued neural autoregressive density-estimator *In Advances in Neural Information Processing Systems 26 (NIPS 26)* p arXiv:1306.0186 (*Preprint* 1306.0186)
[123] Vincent P 2011 *Neural Computation* **23** 1661–1674 ISSN 0899-7667 (*Preprint* https://direct.mit.edu/neco/article-pdf/23/7/1661/851298/neco_a_00142.pdf) URL https://doi.org/10.1162/NECO_a_00142
[124] Ho J, Jain A and Abbeel P 2020 *Advances in Neural Information Processing Systems* **33** arXiv:2006.11239 (*Preprint* 2006.11239)
[125] Luo C 2022 *arXiv e-prints* arXiv:2208.11970 (*Preprint* 2208.11970)
[126] Nichol A Q and Dhariwal P 2021 Improved denoising diffusion probabilistic models *Proceedings of the 38th International Conference on Machine Learning* (*Proceedings of Machine Learning Research* vol 139) ed Meila M and Zhang T (PMLR) p 8162 (*Preprint* 2102.09672) URL https://proceedings.mlr.press/v139/nichol21a.html
[127] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* pp 770–778 (*Preprint* 1512.03385)
[128] Katharopoulos A, Vyas A, Pappas N and Fleuret F 2020 *ICML 2020* (*Preprint* 2006.16236)
[129] Song Y, Dhariwal P, Chen M and Sutskever I 2023 *arXiv e-prints* arXiv:2303.01469 (*Preprint* 2303.01469)
[130] Karras T, Aittala M, Aila T and Laine S 2022 *arXiv e-prints* arXiv:2206.00364 (*Preprint* 2206.00364)
[131] Luo S and Hu W 2021 *arXiv e-prints* arXiv:2103.01458 (*Preprint* 2103.01458)
[132] Song J, Meng C and Ermon S 2020 *CoRR* **abs/2010.02502** (*Preprint* 2010.02502) URL https://arxiv.org/abs/2010.02502
[133] Salimans T and Ho J 2022 *arXiv preprint arXiv:2202.00512*
[134] Ramachandran P, Zoph B and Le Q V 2017 *arXiv preprint arXiv:1710.05941*
[135] Liu L, Ren Y, Lin Z and Zhao Z 2022 *arXiv e-prints* arXiv:2202.09778 (*Preprint* 2202.09778)

[136] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser L and Polosukhin I 2017 *Advances in neural information processing systems* **30** arXiv:1706.03762 (*Preprint* 1706.03762)

[137] Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, Uszkoreit J and Houlsby N 2020 *ICLR 2021* arXiv:2010.11929 (*Preprint* 2010.11929)

[138] Liu Z, Lin Y, Cao Y, Hu H, Wei Y, Zhang Z, Lin S and Guo B 2021 arXiv:2103.14030 (*Preprint* 2103.14030)

[139] Brooks T, Peebles B, Holmes C, DePue W, Guo Y, Jing L, Schnurr D, Taylor J, Luhman T, Luhman E, Ng C, Wang R and Ramesh A 2024 URL https://openai.com/research/video-generation-models-as-world-simulators

[140] Peebles W and Xie S 2023 arXiv:2212.09748 (*Preprint* 2212.09748)

[141] Kingma D P and Welling M 2013 *arXiv e-prints* arXiv:1312.6114 (*Preprint* 1312.6114)

[142] Rezende D J, Mohamed S and Wierstra D 2014 Stochastic backpropagation and approximate inference in deep generative models *International conference on machine learning* (PMLR) pp 1278–1286

[143] Jordan M I, Ghahramani Z, Jaakkola T S and Saul L K 1999 *Machine learning* **37** 183–233

[144] van den Oord A, Vinyals O and Kavukcuoglu K 2017 *Advances in neural information processing systems* **30** arXiv:1711.00937 (*Preprint* 1711.00937)

[145] Karpathy A 2020 mingpt: A pytorch re-implementation of gpt https://github.com/karpathy/minGPT

[146] Esser P, Rombach R and Ommer B 2020 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* arXiv:2012.09841 (*Preprint* 2012.09841)

[147] Bengio Y, Courville A and Vincent P 2013 *IEEE transactions on pattern analysis and machine intelligence* **35** 1798–1828

[148] Pope P, Zhu C, Abdelkader A, Goldblum M and Goldstein T 2021 The Intrinsic Dimension of Images and Its Impact on Learning *International Conference on Learning Representations* (*Preprint* 2104.08894)

[149] Brown B C, Caterini A L, Ross B L, Cresswell J C and Loaiza-Ganem G 2023 Verifying the union of manifolds hypothesis for image data *The Eleventh International Conference on Learning Representations* (*Preprint* 2207.02862) URL https://openreview.net/forum?id=Rvee9CAX4fi

[150] Loaiza-Ganem G, Ross B L, Cresswell J C and Caterini A L 2022 Diagnosing and Fixing Manifold Overfitting in Deep Generative Models *Transactions on Machine Learning Research* (*Preprint* 2204.07172)

[151] Loaiza-Ganem G, Ross B L, Hosseinzadeh R, Caterini A L and Cresswell J C 2024 Deep generative models through the lens of the manifold hypothesis: A survey and new connections *Transactions on Machine Learning Research* (*Preprint* 2404.02954)

[152] Rumelhart D E, Hinton G E and Williams R J 1985 Learning internal representations by error propagation Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science

[153] Tolstikhin I, Bousquet O, Gelly S and Schoelkopf B 2017 *arXiv e-prints* arXiv:1711.01558 (*Preprint* 1711.01558)

[154] Donahue J, Krähenbühl P and Darrell T 2016 *arXiv e-prints* arXiv:1605.09782 (*Preprint* 1605.09782)

[155] Dumoulin V, Belghazi I, Poole B, Mastropietro O, Lamb A, Arjovsky M and Courville A 2016 *arXiv e-prints* arXiv:1606.00704 (*Preprint* 1606.00704)

[156] Mescheder L, Nowozin S and Geiger A 2017 Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks *International Conference on Machine Learning* (PMLR) pp 2391–2400 (*Preprint* 1701.04722)

[157] Du Y and Mordatch I 2019 *Advances in Neural Information Processing Systems* **32** 3608–3618

[158] Song Y and Ermon S 2019 Generative Modeling by Estimating Gradients of the Data Distribution

*Advances in Neural Information Processing Systems* vol 32 p arXiv:1907.05600 (*Preprint* 1907.05600)

[159] Campadelli P, Casiraghi E, Ceruti C and Rozza A 2015 *Mathematical Problems in Engineering* **2015** 1–21

[160] Johnsson K, Soneson C and Fontes M 2015 *IEEE Transactions on Pattern Analysis and Machine Intelligence* **37** 196–202

[161] Bac J, Mirkes E M, Gorban A N, Tyukin I and Zinovyev A 2021 *Entropy* **23** 1368 ISSN 1099-4300 (*Preprint* 2109.02596)

[162] Tempczyk P, Michaluk R, Garncarek Ł, Spurek P, Tabor J and Goliński A 2022 *Proceedings of the 39th International Conference on Machine Learning* **162** arXiv:2206.14882 (*Preprint* 2206.14882)

[163] Kamkari H, Ross B L, Hosseinzadeh R, Cresswell J C and Loaiza-Ganem G 2024 A geometric view of data complexity: Efficient local intrinsic dimension estimation with diffusion models *Advances in Neural Information Processing Systems* vol 37 (*Preprint* 2406.03537)

[164] Brown T, Mann B, Ryder N, Subbiah M, Kaplan J D, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I and Amodei D 2020 *Advances in Neural Information Processing Systems* **33** arXiv:2005.14165 (*Preprint* 2005.14165) URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[165] Ramesh A, Pavlov M, Goh G, Gray S, Voss C, Radford A, Chen M and Sutskever I 2021 *Proceedings of the 38th International Conference on Machine Learning* **139** arXiv:2102.12092 (*Preprint* 2102.12092) URL https://proceedings.mlr.press/v139/ramesh21a.html

[166] Higgins I, Matthey L, Pal A, Burgess C, Glorot X, Botvinick M, Mohamed S and Lerchner A 2017 beta-VAE: Learning basic visual concepts with a constrained variational framework *International Conference on Learning Representations* URL https://openreview.net/forum?id=Sy2fzU9gl

[167] Burgess C P, Higgins I, Pal A, Matthey L, Watters N, Desjardins G and Lerchner A 2018 Understanding disentangling in $\beta$-VAE (*Preprint* 1804.03599)

[168] Liu X, Gong C and qiang liu 2023 Flow straight and fast: Learning to generate and transfer data with rectified flow *The Eleventh International Conference on Learning Representations* (*Preprint* 2209.03003) URL https://openreview.net/forum?id=XVjTT1nw5z

[169] Albergo M S and Vanden-Eijnden E 2023 Building normalizing flows with stochastic interpolants *The Eleventh International Conference on Learning Representations* (*Preprint* 2209.15571) URL https://openreview.net/forum?id=li7qeBbCR1t

[170] Lipman Y, Chen R T Q, Ben-Hamu H, Nickel M and Le M 2023 Flow matching for generative modeling *The Eleventh International Conference on Learning Representations* (*Preprint* 2210.02747) URL https://openreview.net/forum?id=PqvMRDCJT9t

[171] Tong A, Fatras K, Malkin N, Huguet G, Zhang Y, Rector-Brooks J, Wolf G and Bengio Y 2024 *Transactions on Machine Learning Research* arXiv:2302.00482 ISSN 2835-8856 (*Preprint* 2302.00482)

[172] Chen R T Q, Rubanova Y, Bettencourt J and Duvenaud D K 2018 Neural ordinary differential equations *Advances in Neural Information Processing Systems* vol 31 (*Preprint* 1806.07366)

[173] Heimel T, Huetsch N, Winterhalder R, Plehn T and Butter A 2024 *SciPost Phys.* **17** 129 (*Preprint* 2310.07752)

[174] Perez E, Strub F, de Vries H, Dumoulin V and Courville A 2018 *Proceedings of the AAAI Conference on Artificial Intelligence* **32** arXiv:1709.07871 (*Preprint* 1709.07871) URL https://ojs.aaai.org/index.php/AAAI/article/view/11671

[175] Grinsztajn L, Oyallon E and Varoquaux G 2022 Why do tree-based models still outperform deep learning on typical tabular data? *Advances in Neural Information Processing Systems* vol 35

(*Preprint* 2207.08815)

[176] McElfresh D, Khandagale S, Valverde J, Prasad C V, Feuer B, Hegde C, Ramakrishnan G, Goldblum M and White C 2023 When do neural nets outperform boosted trees on tabular data? *Advances in Neural Information Processing Systems* vol 36 (*Preprint* 2305.02997)

[177] Ma J, Thomas V, Hosseinzadeh R, Kamkari H, Labach A, Cresswell J C, Golestan K, Yu G, Volkovs M and Caterini A L 2024 TabDPT: Scaling Tabular Foundation Models (*Preprint* 2410.18164)

[178] Shapley L S 1951 *Notes on the n-person game—ii: The value of an n-person game* (*Notes on the N-person Game* no Bd. 2) (Rand Corporation) URL https://books.google.at/books?id=uMoWzwEACAAJ

[179] Lundberg S M and Lee S I 2017 A unified approach to interpreting model predictions *Advances in Neural Information Processing Systems* vol 30 (*Preprint* 1705.07874)

[180] Lundberg S M, Erion G G and Lee S I 2018 *arXiv:1802.03888* (*Preprint* 1802.03888)

[181] Chen T and Guestrin C 2016 Xgboost: A scalable tree boosting system *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* ISBN 9781450342322 (*Preprint* 1603.02754)

[182] Friedman J H 2001 *The Annals of Statistics* **29**

[183] Nock R and Guillame-Bert M 2022 Generative trees: Adversarial and copycat *Proceedings of the 39th International Conference on Machine Learning* (*Preprint* 2201.11205) URL https://proceedings.mlr.press/v162/nock22a.html

[184] Jolicoeur-Martineau A, Fatras K and Kachman T 2024 Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics* (*Preprint* 2309.09968) URL https://proceedings.mlr.press/v238/jolicoeur-martineau24a.html

[185] Kansal R, Li A, Duarte J, Chernyavskaya N, Pierini M, Orzari B and Tomei T 2023 *Phys. Rev. D* **107** 076017 (*Preprint* 2211.10295)

[186] Das R, Favaro L, Heimel T, Krause C, Plehn T and Shih D 2024 *SciPost Phys.* **16** 031 (*Preprint* 2305.16774)

[187] Diefenbacher S, Eren E, Kasieczka G, Korol A, Nachman B and Shih D 2020 *JINST* **15** P11004 (*Preprint* 2009.03796)

[188] Pearson K 1900 *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **50** 157–175

[189] Cramér H 1946 *Mathematical Methods of Statistics* Goldstine Printed Materials (Princeton University Press) ISBN 9780691080048

[190] Gagunashvili N D 2007 *PoS* **ACAT** 054 (*Preprint* physics/0605123)

[191] Lopez-Paz D and Oquab M 2016 *arXiv e-prints* arXiv:1610.06545 (*Preprint* 1610.06545)

[192] Neyman J and Pearson E S 1933 *Phil. Trans. Roy. Soc. Lond. A* **231** 289–337

[193] Guo C, Pleiss G, Sun Y and Weinberger K Q 2017 *arXiv e-prints* arXiv:1706.04599 (*Preprint* 1706.04599)

[194] Lim S H, Raman K A, Buckley M R and Shih D 2024 *Mon. Not. Roy. Astron. Soc.* **533** 143–164 (*Preprint* 2211.11765)

[195] Golling T, Kasieczka G, Krause C, Mastandrea R, Nachman B, Raine J A, Sengupta D, Shih D and Sommerhalder M 2024 *Eur. Phys. J. C* **84** 241 (*Preprint* 2307.11157)

[196] Defazio A, Xingyu, Yang, Mehta H, Mishchenko K, Khaled A and Cutkosky A 2024 *arXiv e-prints* arXiv:2405.15682 (*Preprint* 2405.15682)

[197] Hara K, Kataoka H and Satoh Y 2018 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* arXiv:1711.09577 (*Preprint* 1711.09577)

[198] Heusel M, Ramsauer H, Unterthiner T, Nessler B and Hochreiter S 2017 *Advances in Neural Information Processing Systems* **30** arXiv:1706.08500 (*Preprint* 1706.08500) URL https://proceedings.neurips.cc/paper_files/paper/2017/file/8a1d694707eb0fefe65871369074926d-Paper.pdf

[199] Kansal R, Duarte J, Su H, Orzari B, Tomei T, Pierini M, Touranakou M, Vlimant J R and Gunopulos D 2021 Particle Cloud Generation with Message Passing Generative Adversarial Networks *35th Conference on Neural Information Processing Systems* (*Preprint* 2106.11535)
[200] Bińkowski M, Sutherland D J, Arbel M and Gretton A 2018 *International Conference on Learning Representations* arXiv:1801.01401 (*Preprint* 1801.01401) URL https://openreview.net/forum?id=r1lUOzWCW
[201] Kansal R, Pareja C, Hao Z and Duarte J 2023 *Journal of Open Source Software* **8** 5789 URL https://joss.theoj.org/papers/10.21105/joss.05789
[202] Astolfi P, Careil M, Hall M, Mañas O, Muckley M, Verbeek J, Romero Soriano A and Drozdzal M 2024 *arXiv e-prints* arXiv:2406.10429 (*Preprint* 2406.10429)
[203] Sajjadi M S M, Bachem O, Lucic M, Bousquet O and Gelly S 2018 *arXiv e-prints* arXiv:1806.00035 (*Preprint* 1806.00035)
[204] Kynkäänniemi T, Karras T, Laine S, Lehtinen J and Aila T 2019 *arXiv e-prints* arXiv:1904.06991 (*Preprint* 1904.06991)
[205] Ferjad Naeem M, Oh S J, Uh Y, Choi Y and Yoo J 2020 *arXiv e-prints* arXiv:2002.09797 (*Preprint* 2002.09797)
[206] Kurtzer G M, Sochat V and Bauer M W 2017 *PLOS ONE* **12** 1–20 URL https://doi.org/10.1371/journal.pone.0177459
[207] Cloud Infrastructure Platform (CLIP) Vienna URL https://www.clip.science/
[208] Datta K, Kar D and Roy D 2018 (*Preprint* 1806.00433)
[209] Bellagente M, Butter A, Kasieczka G, Plehn T and Winterhalder R 2020 *SciPost Phys.* **8** 070 (*Preprint* 1912.00477)
[210] Bellagente M, Butter A, Kasieczka G, Plehn T, Rousselot A, Winterhalder R, Ardizzone L and Köthe U 2020 *SciPost Phys.* **9** 074 (*Preprint* 2006.06685)
[211] Vandegar M, Kagan M, Wehenkel A and Louppe G 2020 (*Preprint* 2011.05836)
[212] Howard J N, Mandt S, Whiteson D and Yang Y 2022 *Sci. Rep.* **12** 7567 (*Preprint* 2101.08944)
[213] Leigh M, Raine J A, Zoch K and Golling T 2023 *SciPost Phys.* **14** 159 (*Preprint* 2207.00664)
[214] Backes M, Butter A, Dunford M and Malaescu B 2024 *SciPost Phys. Core* **7** 007 (*Preprint* 2212.08674)
[215] Shmakov A, Greif K, Fenton M, Ghosh A, Baldi P and Whiteson D 2023 (*Preprint* 2305.10399)
[216] Ackerschott J, Barman R K, Gonçalves D, Heimel T and Plehn T 2024 *SciPost Phys.* **17** 001 (*Preprint* 2308.00027)
[217] Diefenbacher S, Liu G H, Mikuni V, Nachman B and Nie W 2024 *Phys. Rev. D* **109** 076011 (*Preprint* 2308.12351)
[218] Butter A, Jezo T, Klasen M, Kuschick M, Palacios Schweitzer S and Plehn T 2024 *SciPost Phys. Core* **7** 064 (*Preprint* 2311.17175)
[219] Shmakov A, Greif K T, Fenton M J, Ghosh A, Baldi P and Whiteson D 2025 *SciPost Phys.* **18** 117 (*Preprint* 2404.14332)
[220] Huetsch N *et al.* 2025 *SciPost Phys.* **18** 070 (*Preprint* 2404.18807)
[221] Ilten P, Menzo T, Youssef A and Zupan J 2023 *SciPost Phys.* **14** 027 (*Preprint* 2203.04983)
[222] Ghosh A, Ju X, Nachman B and Siodmok A 2022 *Phys. Rev. D* **106** 096020 (*Preprint* 2203.12660)
[223] Chan J, Ju X, Kania A, Nachman B, Sangli V and Siodmok A 2023 *JHEP* **09** 084 (*Preprint* 2305.17169)
[224] Bierlich C, Ilten P, Menzo T, Mrenna S, Szewc M, Wilkinson M K, Youssef A and Zupan J 2024 *SciPost Phys.* **17** 045 (*Preprint* 2311.09296)
[225] Chan J, Ju X, Kania A, Nachman B, Sangli V and Siodmok A 2025 *Phys. Rev. D* **111** 116015 (*Preprint* 2312.08453)
[226] Vaselli F, Rizzi A, Cattafesta F and Cicconofri G 2024 *EPJ Web Conf.* **295** 09020
[227] Hallin A, Isaacson J, Kasieczka G, Krause C, Nachman B, Quadfasel T, Schlaffer M, Shih D and Sommerhalder M 2022 *Phys. Rev. D* **106** 055006 (*Preprint* 2109.00546)

[228] Raine J A, Klein S, Sengupta D and Golling T 2023 *Front. Big Data* **6** 899345 (*Preprint* 2203.09470)

[229] Golling T, Klein S, Mastandrea R and Nachman B 2023 *Phys. Rev. D* **107** 096025 (*Preprint* 2212.11285)

[230] pandas development team T 2021 pandas-dev/pandas: Pandas 1.3.5 URL https://doi.org/10.5281/zenodo.5774815

[231] Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic J, Kelley K, Hamrick J, Grout J, Corlay S, Ivanov P, Avila D, Abdalla S and Willing C 2016 Jupyter notebooks – a publishing format for reproducible computational workflows *Positioning and Power in Academic Publishing: Players, Agents and Agendas* ed Loizides F and Schmidt B (IOS Press) pp 87 – 90

[232] Faucci Giannelli M, Kasieczka G, Krause C, Nachman B, Salamani D, Shih D, Zaborowska A, Amram O, Caterini A, Corchia F A G, Cresswell J, Dreyer E, Ernst F, Favaro L, Franchini M, Gross E, Hsu S C, Kim T, Kobylianskii D, Letizia M, Liu Q, Liu X, Loaiza-Ganem G, Mikuni V, Pang I, Pedro K, Plehn T, Reyes-González H, Rinaldi L, Ross B L, Shimmin C, Shlizerman E, Shu L, Soybelman N and Zhang R 2025 Fast calorimeter simulation challenge 2022 - submissions dataset 1 photons URL https://doi.org/10.5281/zenodo.15961728

[233] Faucci Giannelli M, Kasieczka G, Krause C, Nachman B, Salamani D, Shih D, Zaborowska A, Amram O, Ernst F, Caterini A, Corchia F A G, Cresswell J, Dreyer E, Favaro L, Franchini M, Gross E, Hsu S C, Kim T, Kobylianskii D, Letizia M, Liu Q, Liu X, Loaiza-Ganem G, Pang I, Pedro K, Plehn T, Reyes-González H, Rinaldi L, Ross B L, Shimmin C, Shlizerman E, Shu L, Soybelman N and Zhang R 2025 Fast calorimeter simulation challenge 2022 - submissions dataset 1 pions URL https://doi.org/10.5281/zenodo.15961924

[234] Faucci Giannelli M, Kasieczka G, Krause C, Nachman B, Salamani D, Shih D, Zaborowska A, Amram O, Borras K, Buckley M, Buss T, Da Costa Cardoso R P, Ekambaram V, Ernst F, Favaro L, Gaede F, Hsu S C, Jaruskova K, Käch B, Kalagnanam J, Krücker D, Liu Q, Liu X, Madula T, Melzer-Pellmann I A, Mikuni V, Nguyen N, Ore A, Palacios Schweitzer S, Pang I, Pedro K, Plehn T, Raikwar P, Raine J, Scham M A W, Schnake S, Shimmin C, Shlizerman E, Shu L, Srivatsa M, Vallecorsa S and Yeo K 2025 Fast calorimeter simulation challenge 2022 - submissions dataset 2 URL https://doi.org/10.5281/zenodo.15962050

[235] Faucci Giannelli M, Kasieczka G, Krause C, Nachman B, Salamani D, Shih D, Zaborowska A, Amram O, Borras K, Buckley M, Buhmann E, Buss T, Chernyavskaya N, Da Costa Cardoso R P, Diefenbacher S, Eren E, Ernst F, Favaro L, Gaede F, Hsu S C, Jaruskova K, Käch B, Korcari W, Korol A, Krücker D, Krüger K S, Liu Q, Liu X, McKeown P, Melzer-Pellmann I A, Mikuni V, Ore A, Palacios Schweitzer S, Pang I, Pedro K, Plehn T, Pokorski W, Raikwar P, Scham M A W, Schnake S, Shlizerman E, Shimmin C, Shu L, Srivatsa M, Tsolaki K and Vallecorsa S 2025 Fast calorimeter simulation challenge 2022 - submissions dataset 3 URL https://doi.org/10.5281/zenodo.15962527

[236] Mazurek M, Corti G and Kmiec M 2024 Performance of the Gaussino CaloChallenge-compatible infrastucture for ML-based fast simulation in the LHCb Experiment URL https://indico.cern.ch/event/1330797/contributions/5796650/