Generation of Calorimeter Showers with Normalizing Flows: CaloFlow

— 1st MODE workshop, UC Louvain —

Claudius Krause

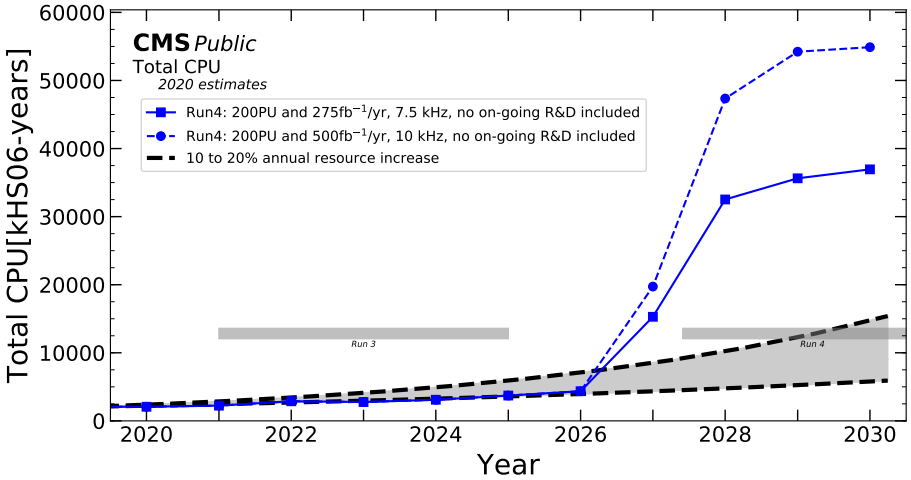
Rutgers, The State University of New Jersey

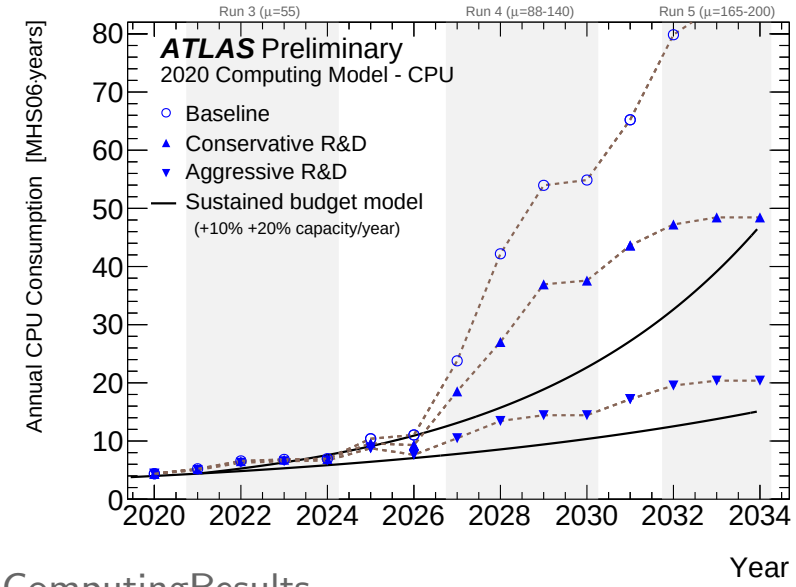
September 7, 2021

RUTGERS
UNIVERSITY | NEW BRUNSWICK

In collaboration with David Shih
arXiv: 2106.05285

Why do we need Deep Generative Models?





https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults
https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults

- At the end of LHC Run 3, the computational needs will exceed the available budget.

⇒ Deep Generative Models will be crucial for the LHC.

- MODE will need a faithful surrogate

CERN-LHCC-2020-015; LHCC-G-178

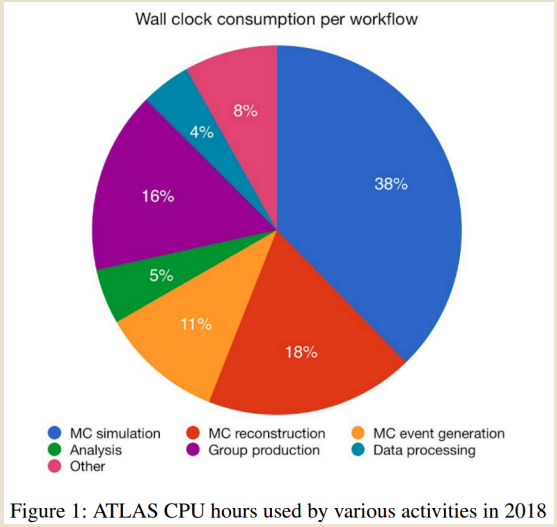
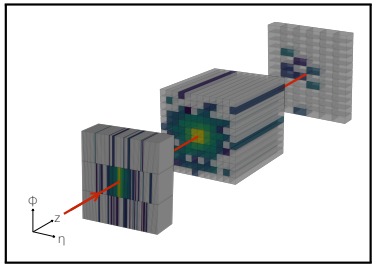
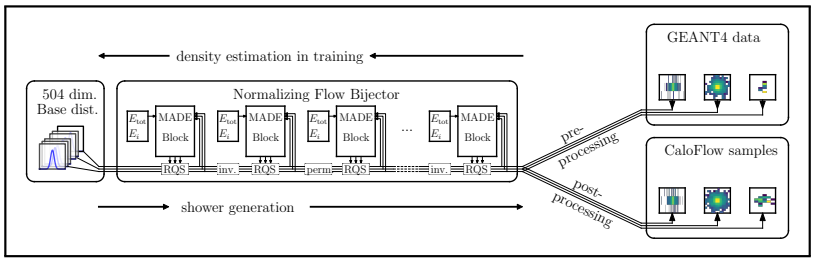


Figure 1: ATLAS CPU hours used by various activities in 2018

Generation of Calorimeter Showers with Normalizing Flows: CaloFlow

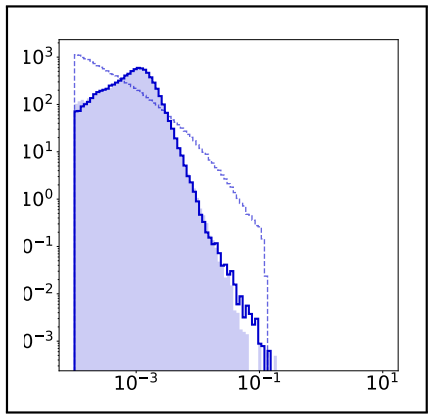


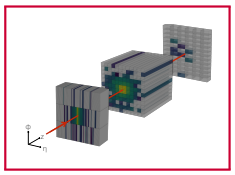
Part I: The Calorimeter Dataset



Part II: Generative Modeling with Normalizing Flows

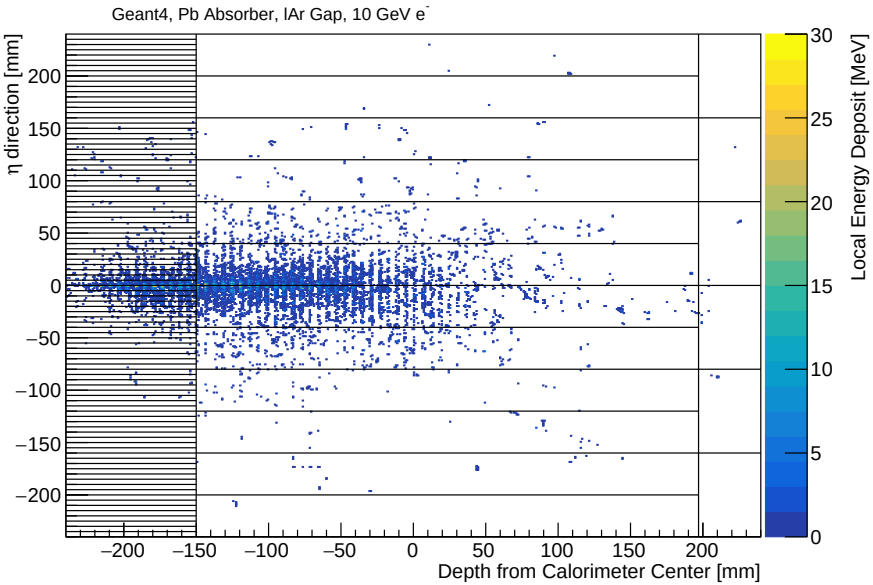
Part III: Performance of CaloFlow

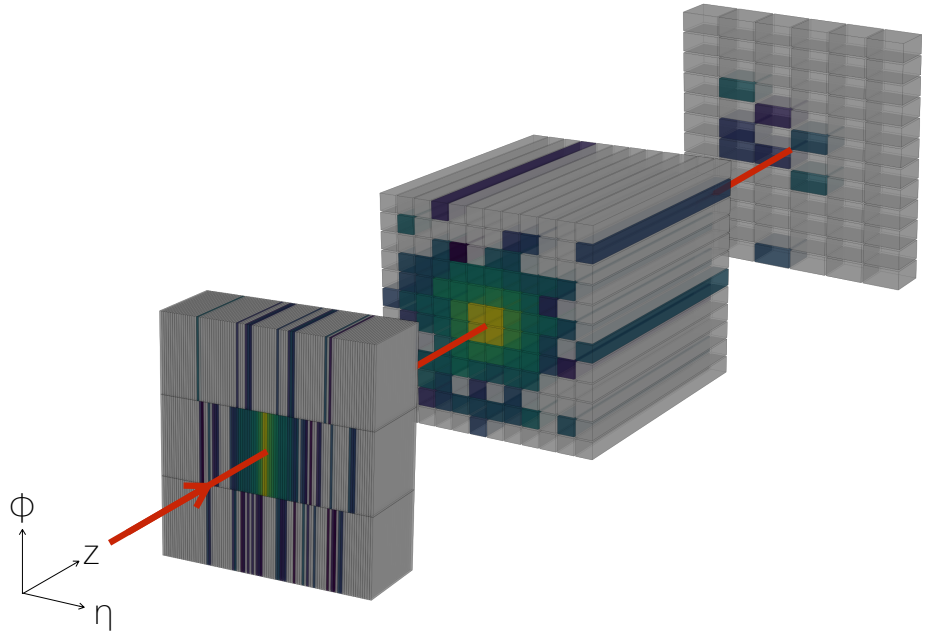




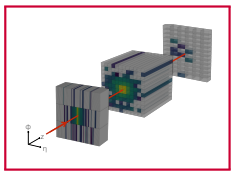
I: We use the same calorimeter geometry as CaloGAN

- We consider a simplified version of the ATLAS ECal: flat alternating layers of lead and LAr
- They form three instrumented layers of dimension 3×96, 12×12, and 12×6



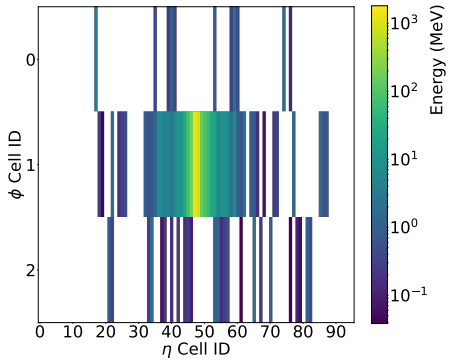


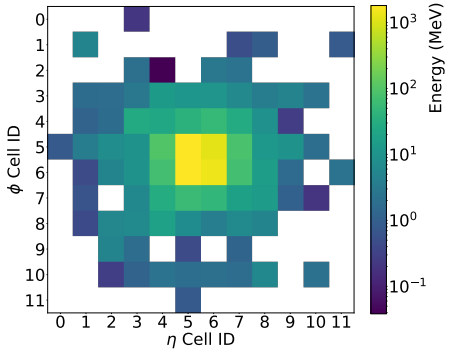
CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD; PRL]

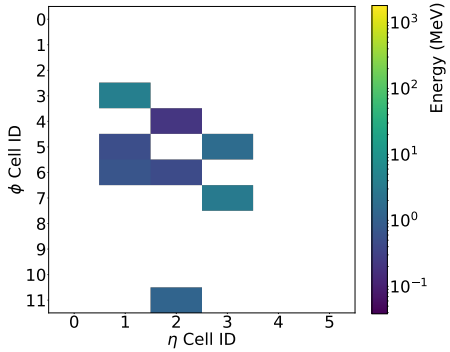


I: We use the same calorimeter geometry as CaloGAN

- The Geant4 configuration of CaloGAN is available at
  https://github.com/hep-lbdl/CaloGAN
- We produce our own dataset:
- Showers of e^+, γ, and π^+ (100k each)
- All are centered and perpendicular
- E_{tot} is uniform in $[1, 100]$ GeV and given in addition to the energy deposits per voxel:

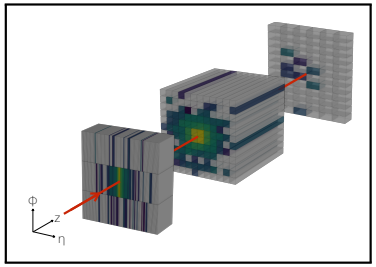




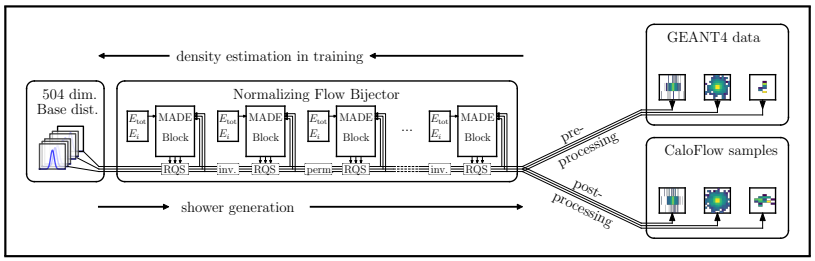


CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD; PRL]

Generation of Calorimeter Showers with Normalizing Flows: CaloFlow

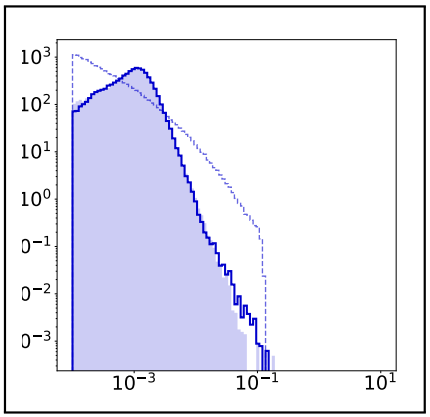


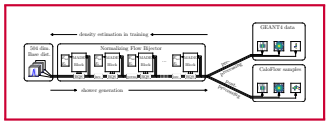
Part I: The Calorimeter Dataset



Part II: Generative Modeling with Normalizing Flows

Part III: Performance of CaloFlow





II: Normalizing Flows learn a change-of-coordinates efficiently.

"easy" base distribution $\Leftrightarrow$ bijective transformation $\Leftrightarrow$ "target" distribution, data

density estimation, $p(x)$ ←

sample generation →

Normalizing Flows ...

Dinh et al. [arXiv:1410.8516], Rezende/Mohamed [arXiv:1505.05770], Review: Papamakarios et al. [arXiv:1912.02762]

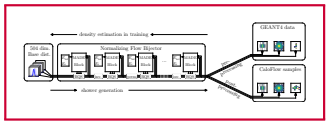
- ...learn the parameters of a series of easy transformations.
- Each transformation is bijective and has an analytic Jacobian and inverse.

$\Rightarrow$ We use a piecewise Rational Quadratic Spline. Durkan et al. [arXiv:1906.04032]

- An autoregressive architecture ensures a triangular Jacobian.

$\Rightarrow$ We use a Masked Autoregressive Flow (MAF) architecture.

Germain et al. [arXiv:1502.03509], Papamakarios et al. [arXiv:1705.07057]



II: Normalizing Flows resolve a few challenges of Deep Generative Models.

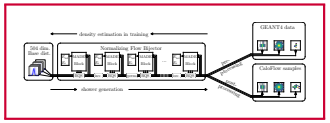
General challenges of deep generative models:

- $\Rightarrow$ By which metric can we monitor the quality of the generator?
- $\Rightarrow$ Energy conservation and other constraints on samples
- $\Rightarrow$ Sparse data and "sharp edges"
- $\Rightarrow$ Fast sampling vs. (long) training times (not important for MODE)

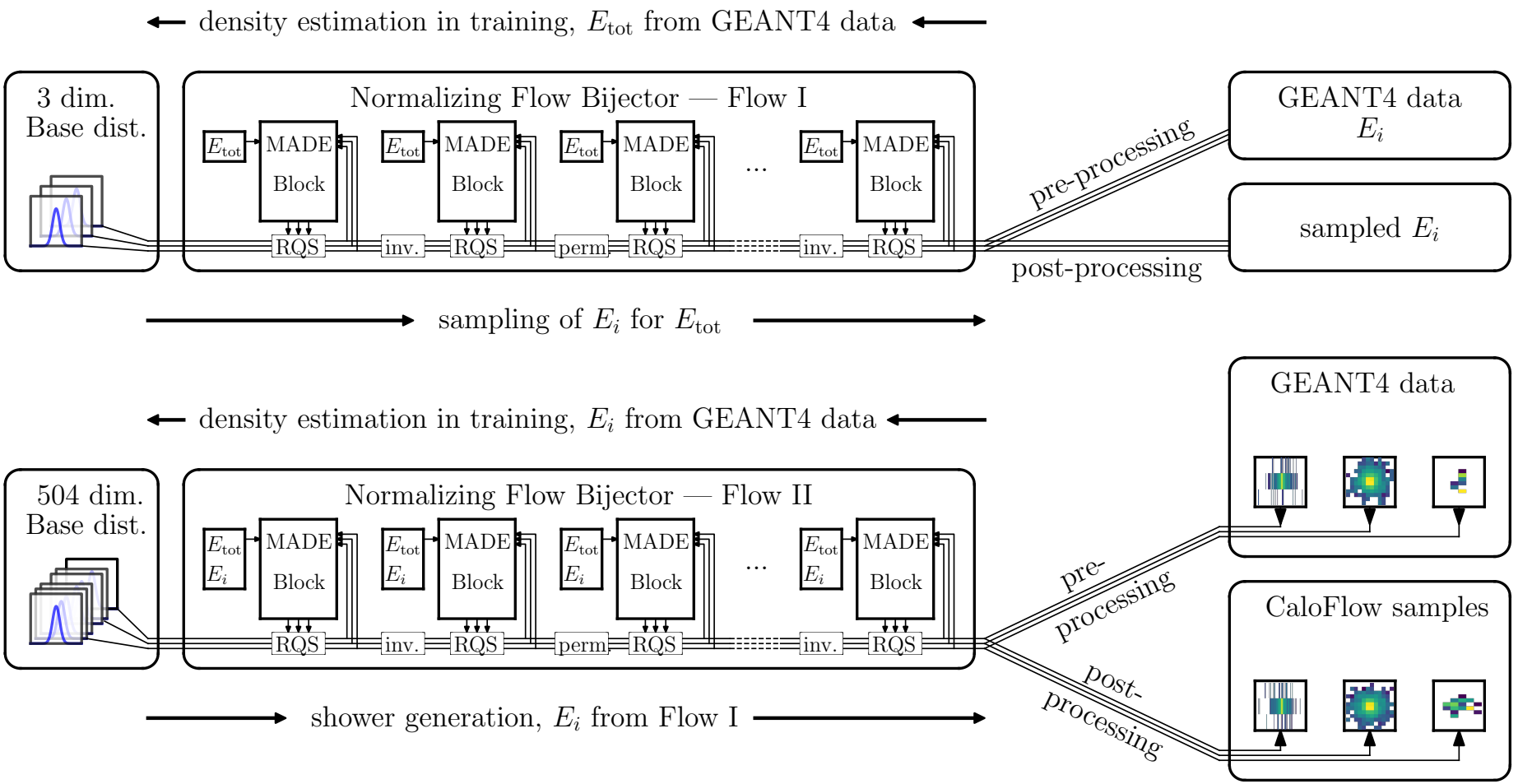
Normalizing Flows:

- ✓ learn $p(x)$ explicitly
- ✓ training is more stable
- ✓ model selection is straightforward
- ✓ no mode collapse and artefacts in samples
- ✓ Can be conditioned on external parameters

- × sparse data is hard to model
- × MAF can be trained with $\log p(x)$, but samples slow

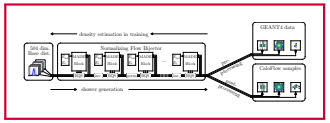


II: CaloFlow uses a 2-step approach.

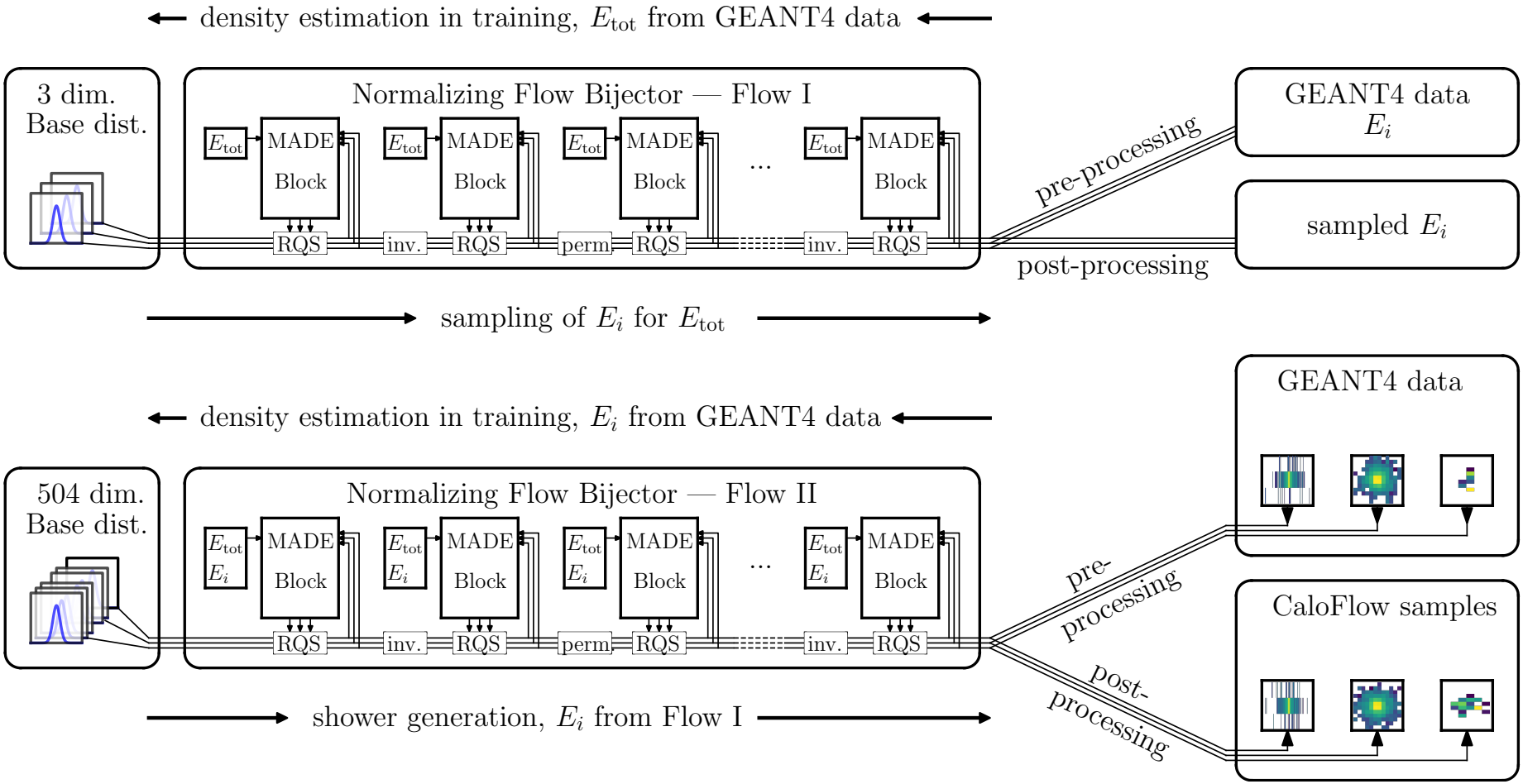


Data processing Flow I

- "←" map E_i to $[0, 1]$
- "←" work in logit space
- "→" invert logit
- "→" map back to E_i



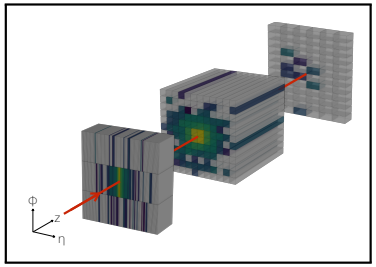
II: CaloFlow uses a 2-step approach.



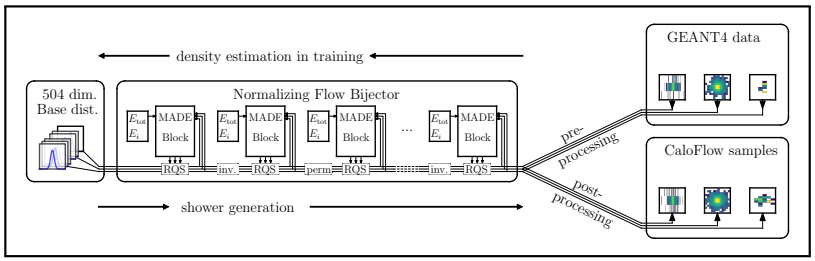
Data processing Flow II

"←" add noise	"→" invert logit
"←" normalize layers to 1	"→" renormalize to E_i
"←" work in logit space	"→" apply threshold

Generation of Calorimeter Showers with Normalizing Flows: CaloFlow

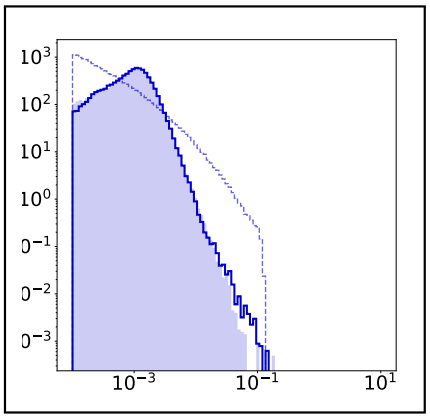


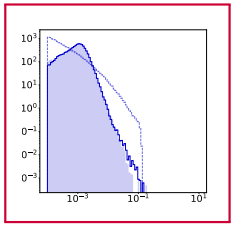
Part I: The Calorimeter Dataset



Part II: Generative Modeling with Normalizing Flows

Part III: Performance of CaloFlow



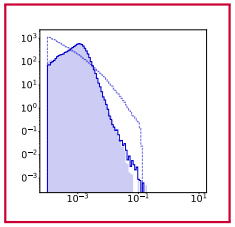


III: A classifier is the "ultimate metric".

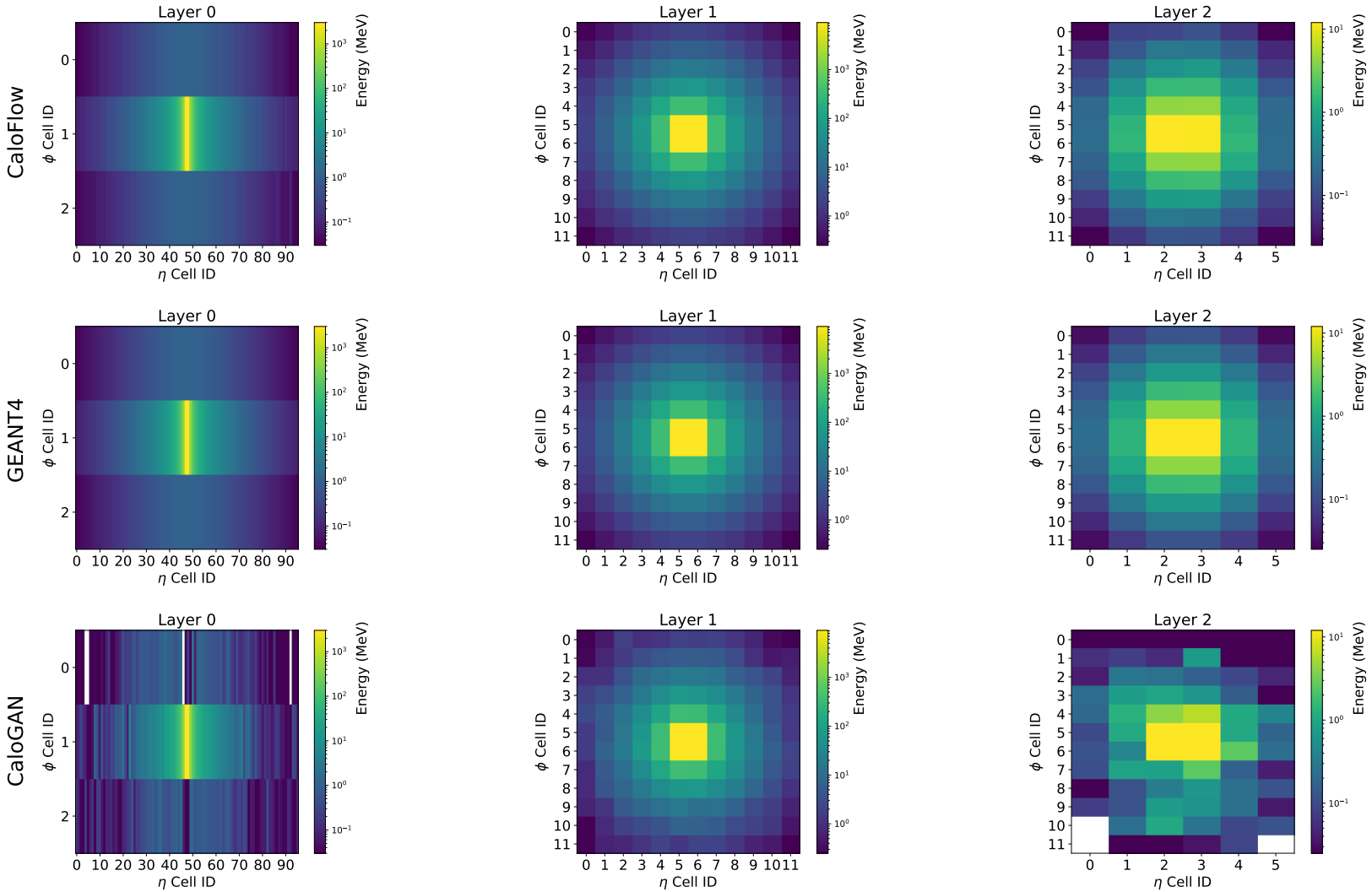
According to the Neyman-Pearson Lemma we have:
$p_{\text{data}} = p_{\text{gen}}$ if a classifier cannot distinguish data from generated samples.

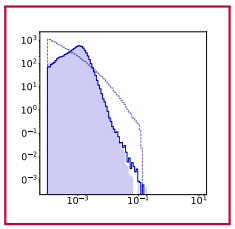
AUC / JSD		DNN		CNN	
		GEANT4 vs. CALOGAN	GEANT4 vs. CALOFLOW	GEANT4 vs. CALOGAN	GEANT4 vs. CALOFLOW
e^+	unnorm.	1.000(0) / 0.993(1)	0.847(8) / 0.345(12)	0.952(6) / 0.613(19)	0.504(2) / 0.002(1)
	norm.	1.000(0) / 0.997(0)	0.869(2) / 0.376(4)	1.000(0) / 0.979(1)	0.736(92) /0.168(134)
γ	unnorm.	1.000(0) / 0.996(1)	0.660(6) / 0.067(4)	0.975(5) / 0.712(31)	0.516(1) / 0.002(1)
	norm.	1.000(0) / 0.994(1)	0.794(4) / 0.213(7)	1.000(0) / 0.989(1)	0.678(50) / 0.082(57)
π^+	unnorm.	1.000(0) / 0.988(1)	0.632(2) / 0.048(1)	0.970(18) /0.714(119)	0.517(2) / 0.001(0)
	norm.	1.000(0) / 0.997(0)	0.751(4) / 0.148(4)	1.000(0) / 0.997(1)	0.864(7) / 0.340(16)

AUC ($\in [0.5, 1]$): Area Under the ROC Curve
JSD ($\in [0, 1]$): Jensen-Shannon divergence based on the binary cross entropy

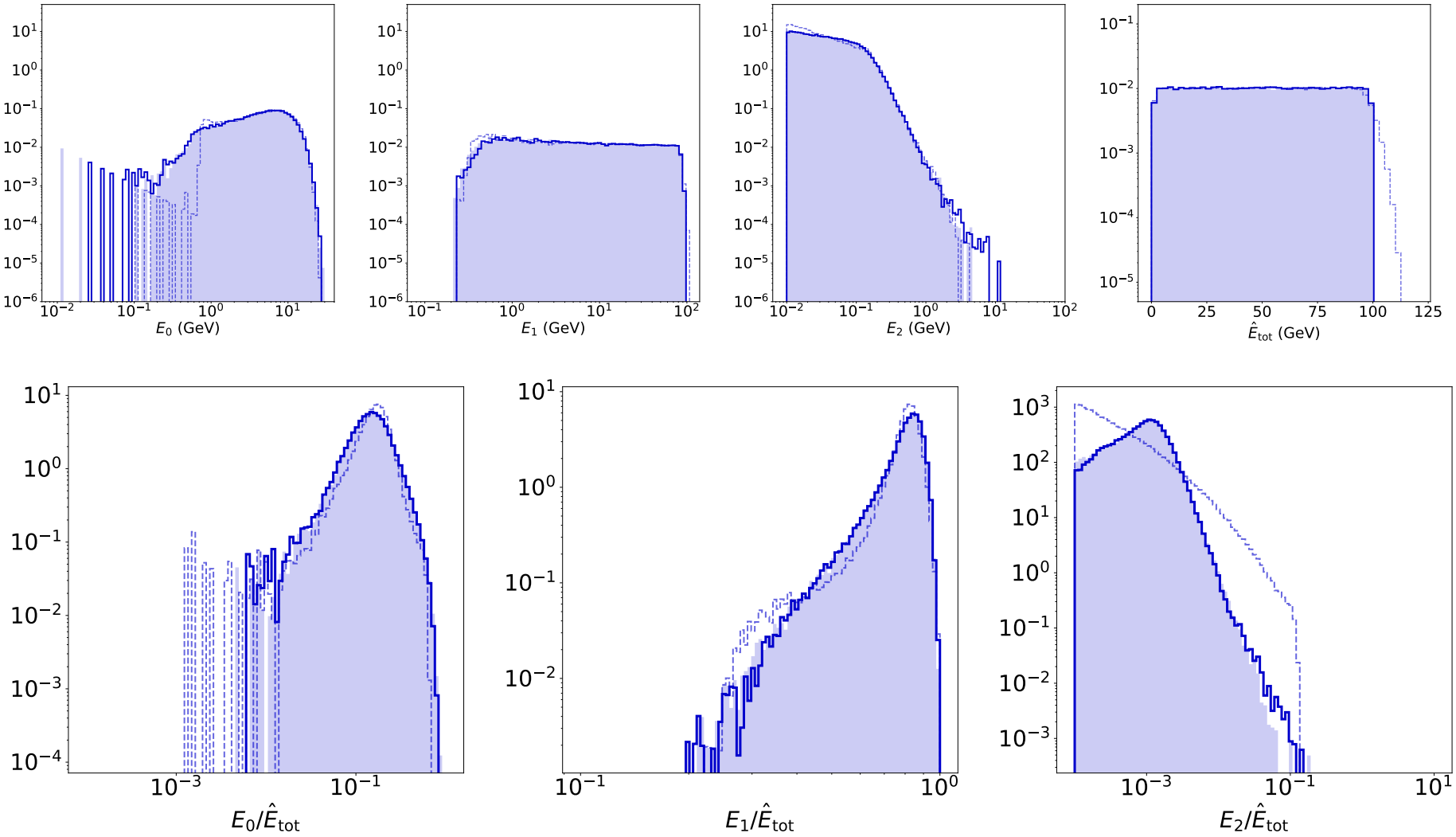


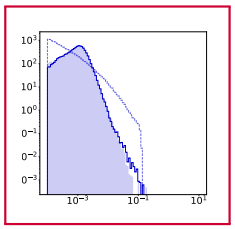
III: Comparing Shower Averages: e^+



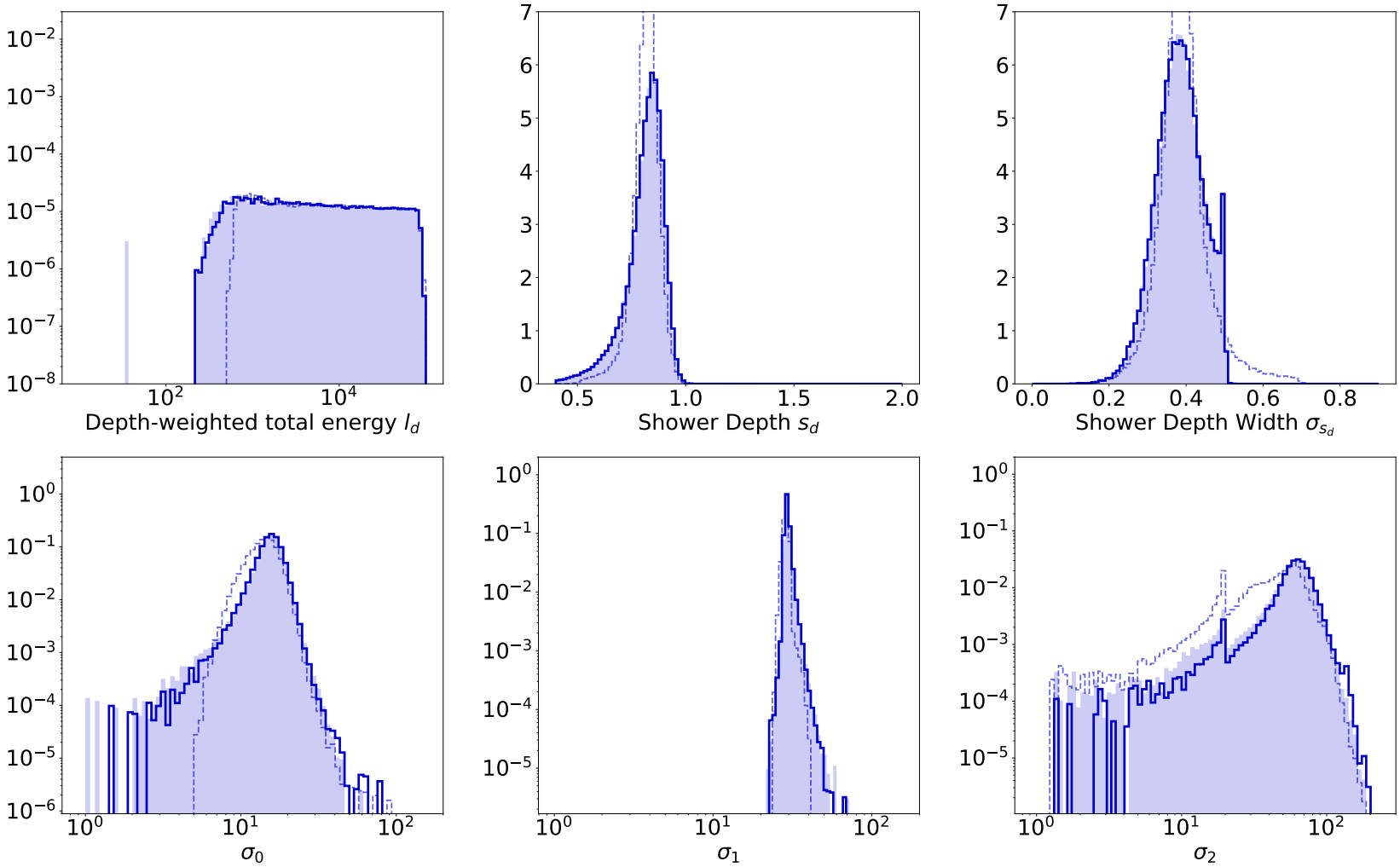


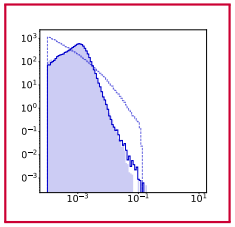
III: Flow I histograms: e^+



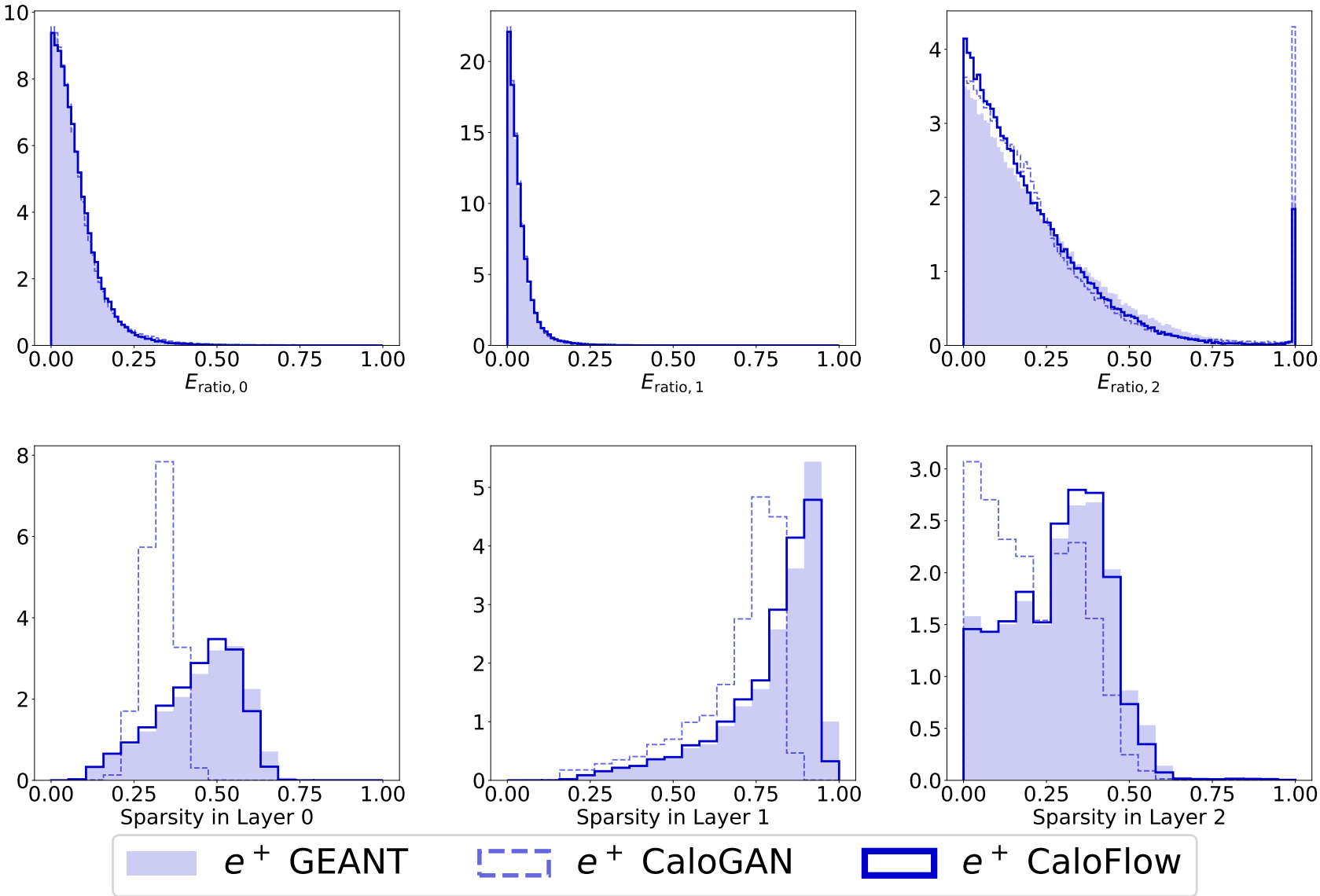


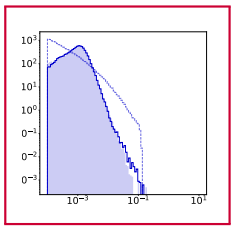
III: Flow I+II histograms: e^+



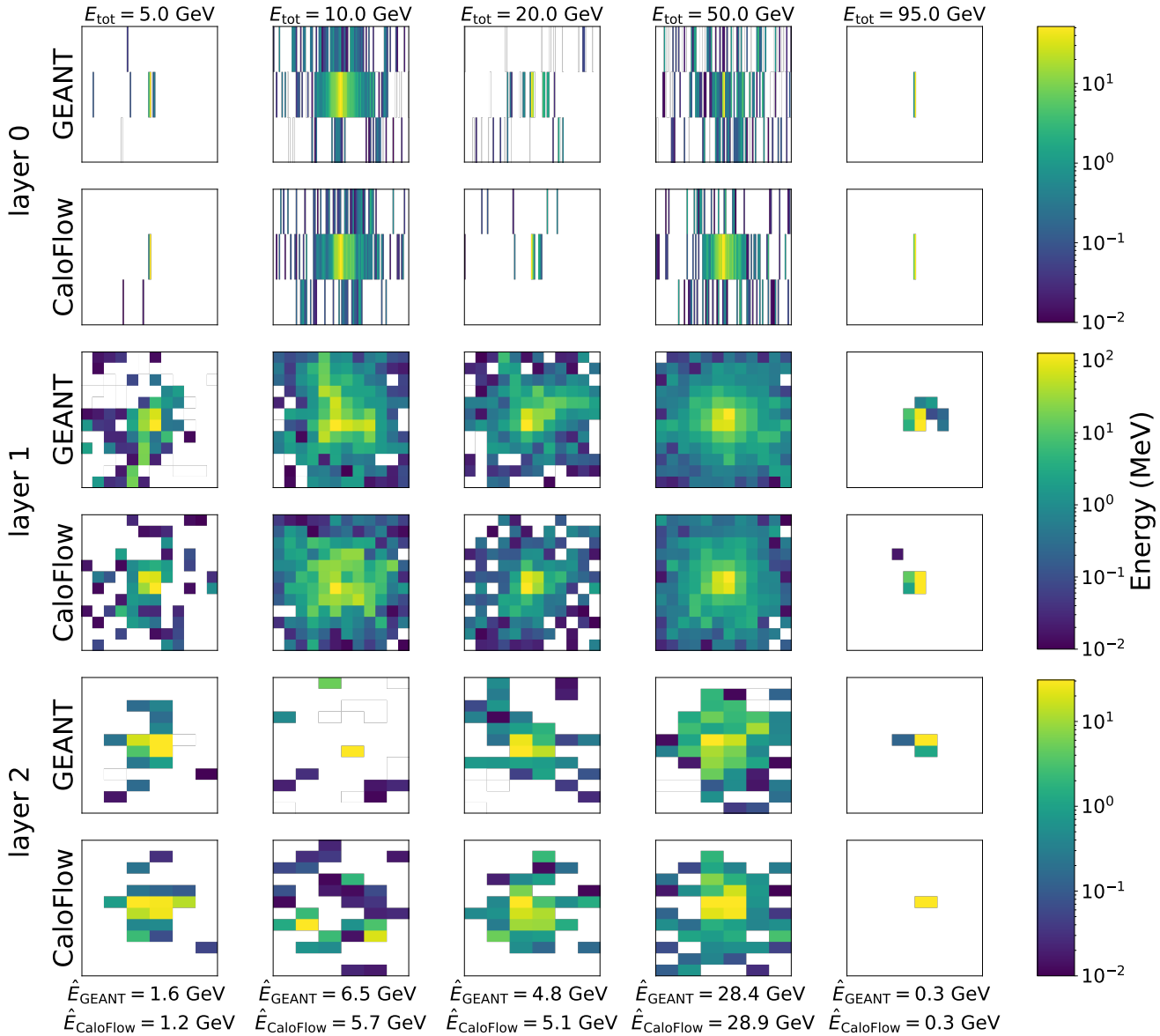


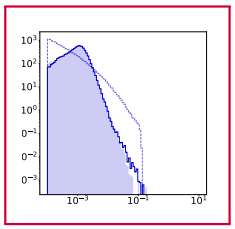
III: Flow II histograms: e^+



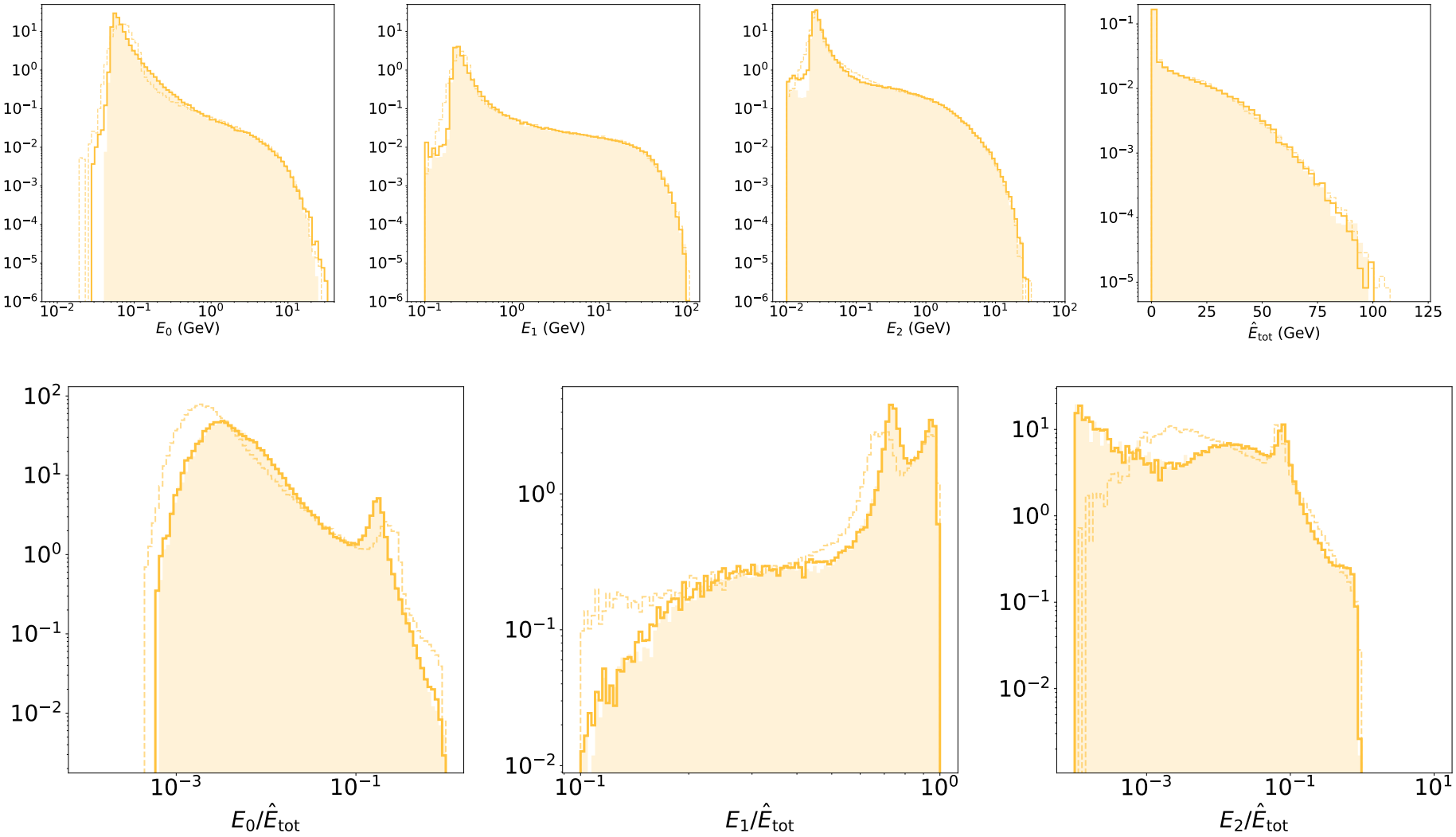


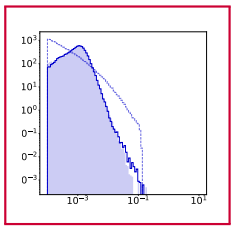
III: Nearest Neighbors: π^+



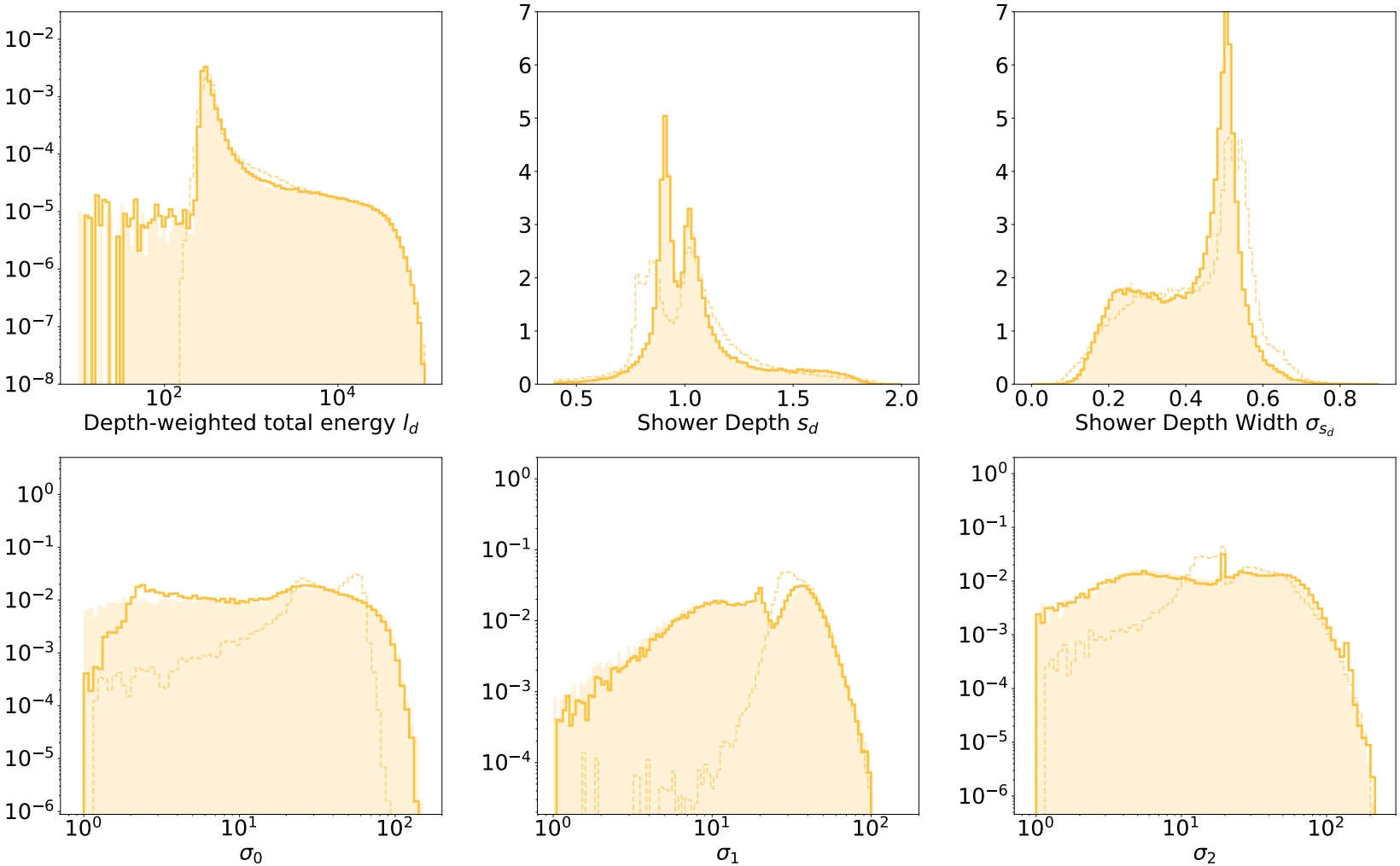


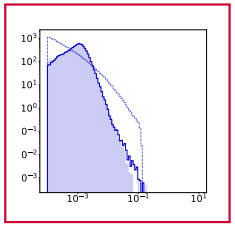
III: Flow I histograms: π^+



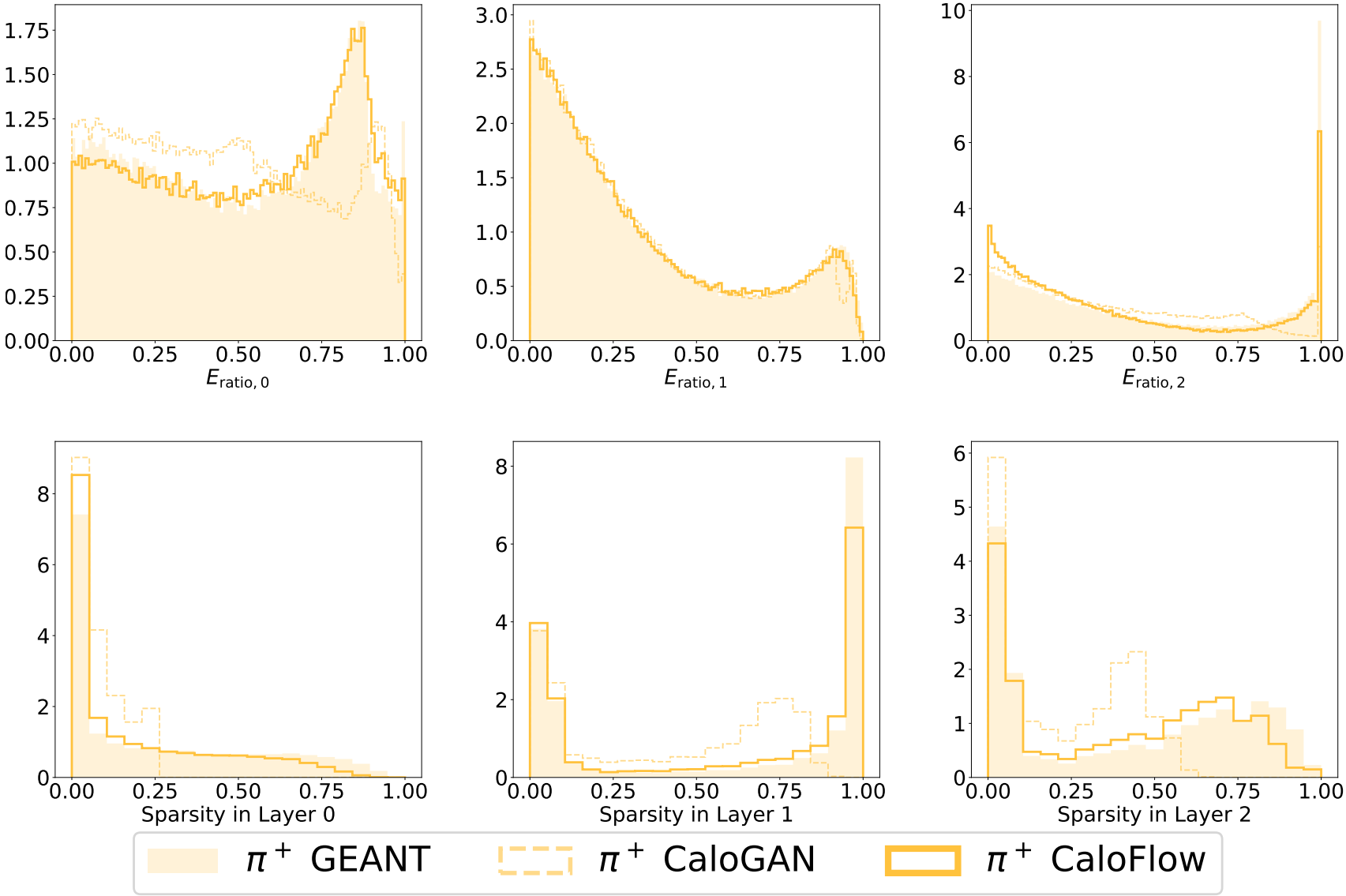


III: Flow I+II histograms: π^+





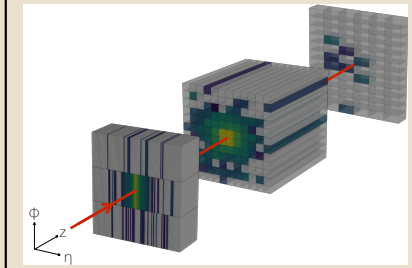
III: Flow II histograms: π^+



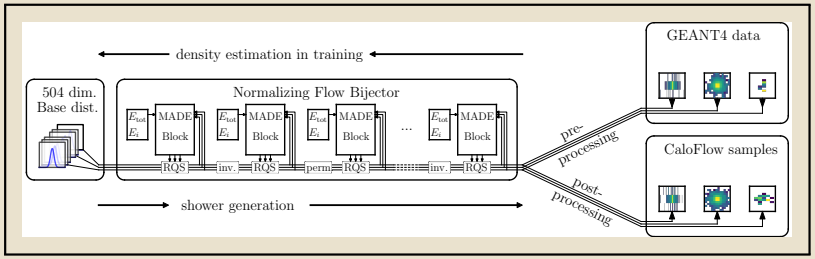
Generation of Calorimeter Showers with Normalizing Flows: CaloFlow

- We use the same calorimeter and Geant4 setup as the original CaloGAN.
- Events are 504-dim. showers of e^+, γ, and π^+

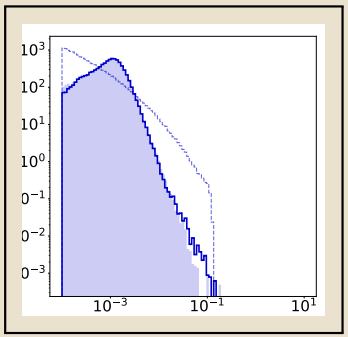
$\Rightarrow$ First time application of Normalizing Flows!



- Having $\log p(x)$ allows stable training and straightforward model selection.
- We use a 2-step setup to ensure energy conservation.
- The setup can easily be conditioned on external parameters.

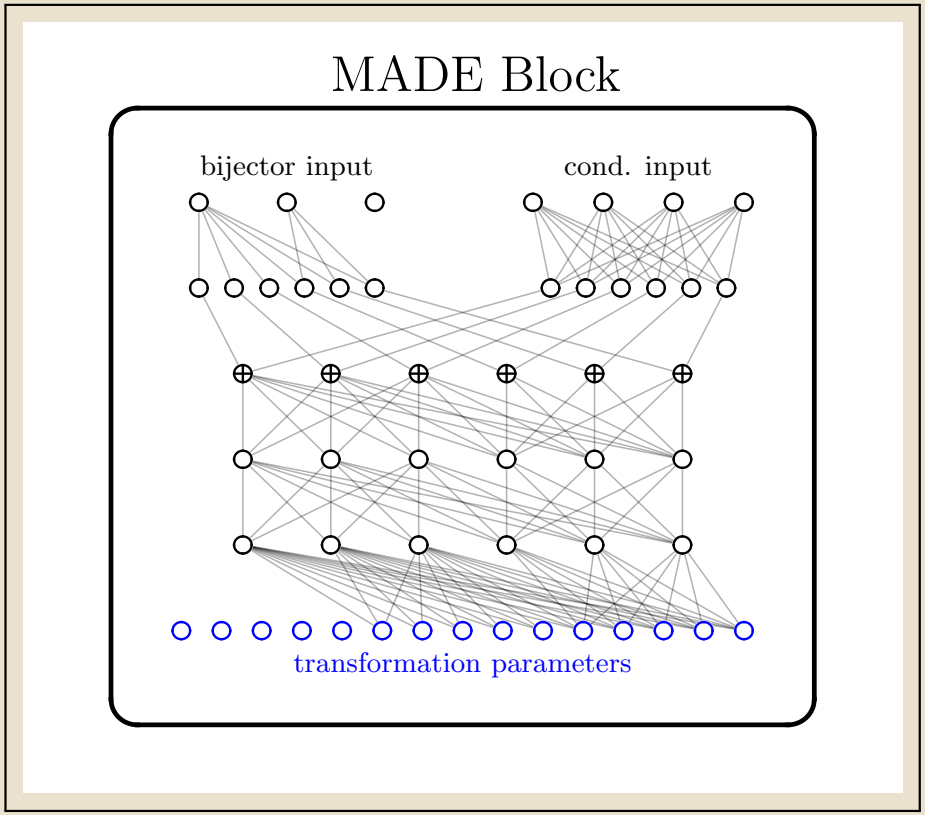


- I argued that a classifier provides the "ultimate test" of a generative model.
- I showed how CaloFlow passes this stringent test, along with more qualitative comparisons (histograms, images).



Backup

Ensuring the Autoregressive Property.



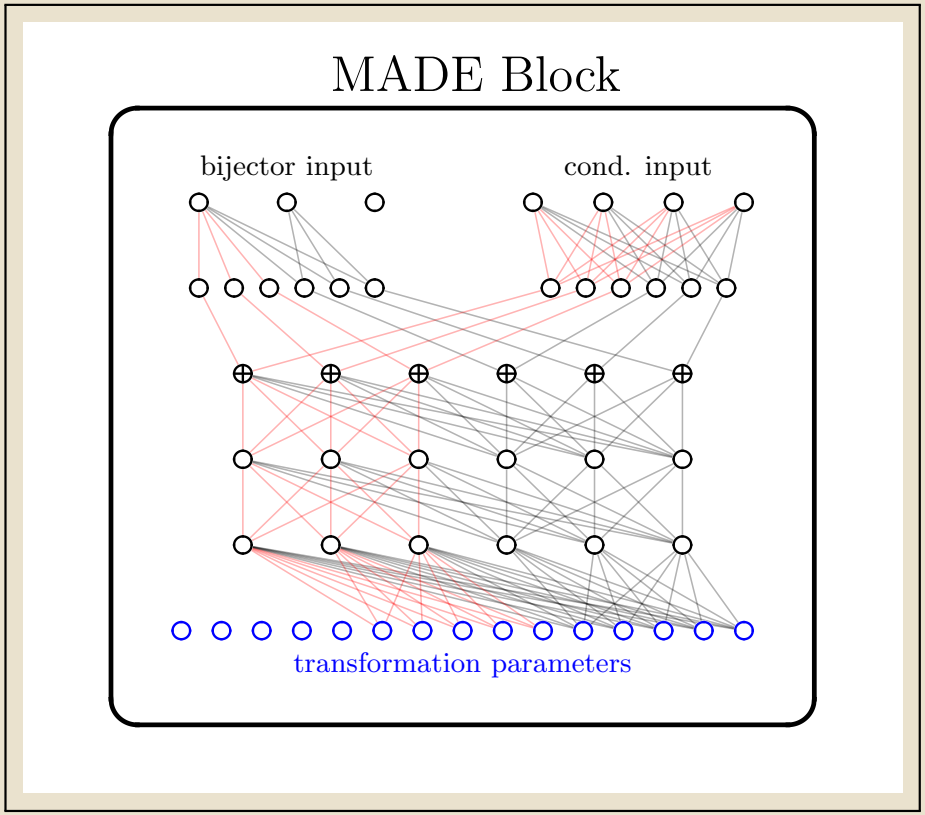
Implementation via masking:

- a single "forward" pass gives the full output of all $p(x_i|x_{i-1}\dots x_1)$.
  $\Rightarrow$ very fast
- the "inverse" needs to loop through all dimensions and gets a single $p(x_i|x_{i-1}\dots x_1)$ each time.
  $\Rightarrow$ very slow

Germain/Gregor/Murray/Larochelle [arXiv:1502.03509]

- Inverse Autoregressive Flow (IAF), introduced in Kingma et al. [arXiv:1606.04934], are fast in sampling and slow in inference.
- Masked Autoregressive Flow (MAF), introduced in Papamakarios et al. [arXiv:1705.07057], are slow in sampling and fast in inference.

Ensuring the Autoregressive Property.



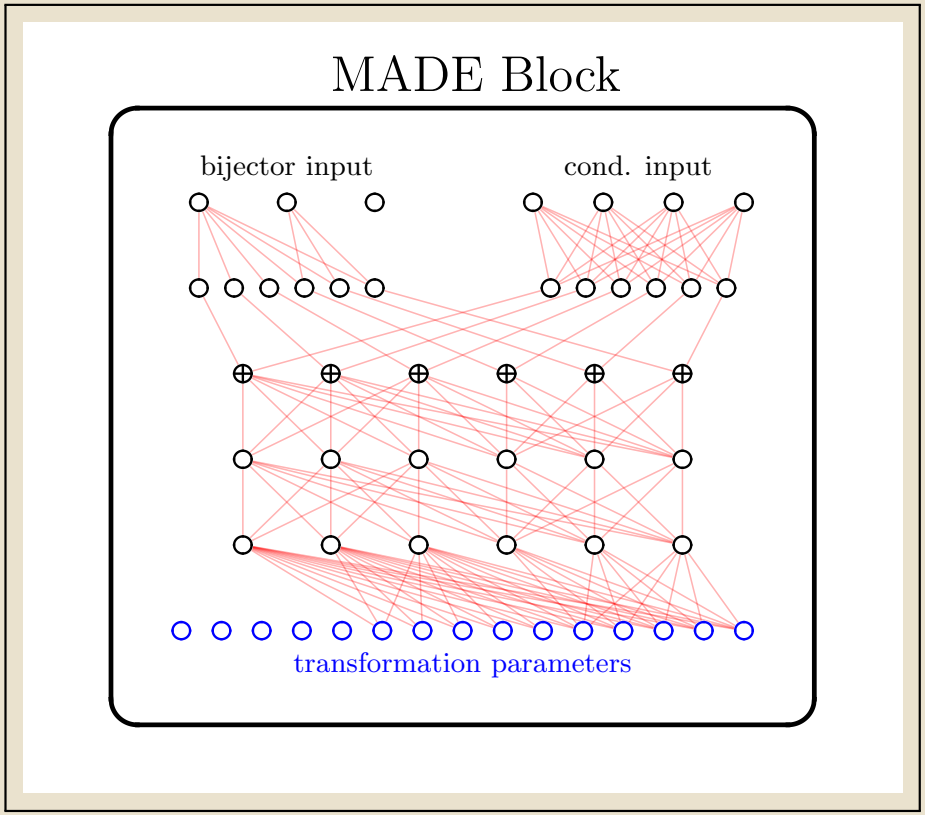
Implementation via masking:

- a single "forward" pass gives the full output of all $p(x_i|x_{i-1}\dots x_1)$.
  $\Rightarrow$ very fast
- the "inverse" needs to loop through all dimensions and gets a single $p(x_i|x_{i-1}\dots x_1)$ each time.
  $\Rightarrow$ very slow

Germain/Gregor/Murray/Larochelle [arXiv:1502.03509]

- Inverse Autoregressive Flow (IAF), introduced in Kingma et al. [arXiv:1606.04934], are fast in sampling and slow in inference.
- Masked Autoregressive Flow (MAF), introduced in Papamakarios et al. [arXiv:1705.07057], are slow in sampling and fast in inference.

Ensuring the Autoregressive Property.



Implementation via masking:

- a single "forward" pass gives the full output of all $p(x_i|x_{i-1}\dots x_1)$.
  $\Rightarrow$ very fast
- the "inverse" needs to loop through all dimensions and gets a single $p(x_i|x_{i-1}\dots x_1)$ each time.
  $\Rightarrow$ very slow

Germain/Gregor/Murray/Larochelle [arXiv:1502.03509]

- Inverse Autoregressive Flow (IAF), introduced in Kingma et al. [arXiv:1606.04934], are fast in sampling and slow in inference.
- Masked Autoregressive Flow (MAF), introduced in Papamakarios et al. [arXiv:1705.07057], are slow in sampling and fast in inference.

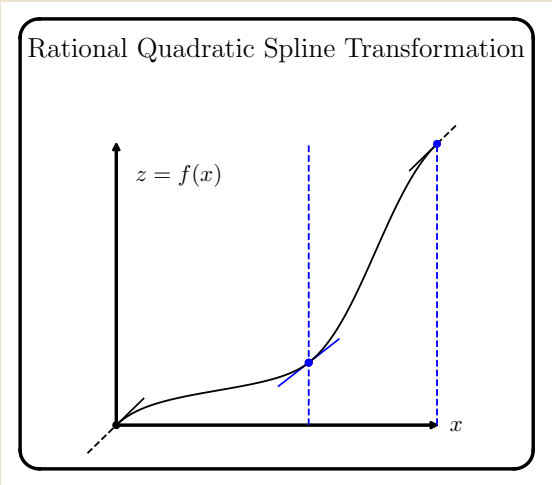
The Coupling Function defines the bijection.

The coupling function (transformation)

- must be invertible and expressive
- is chosen to factorize:
  $\vec{C}(\vec{x};\vec{p}) = (C_1(x_1;p_1), C_2(x_2;p_2), \ldots, C_n(x_n;p_n))^T$,
  where $\vec{x}$ are the coordinates to be transformed and $\vec{p}$ the parameters of the transformation.

rational quadratic spline coupling function: Durkan et al. [arXiv:1906.04032]

Gregory/Delbourgo [IMA Journal of Numerical Analysis, '82]

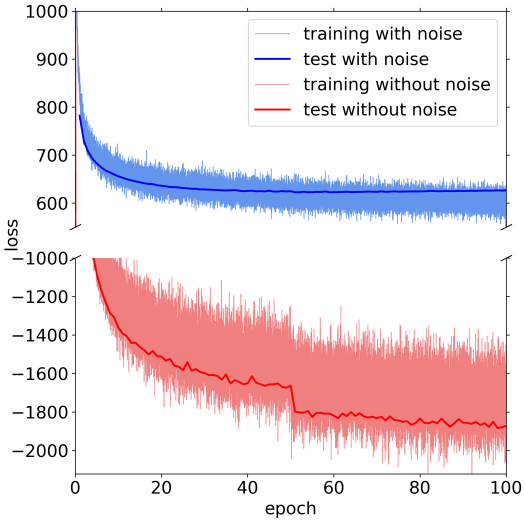


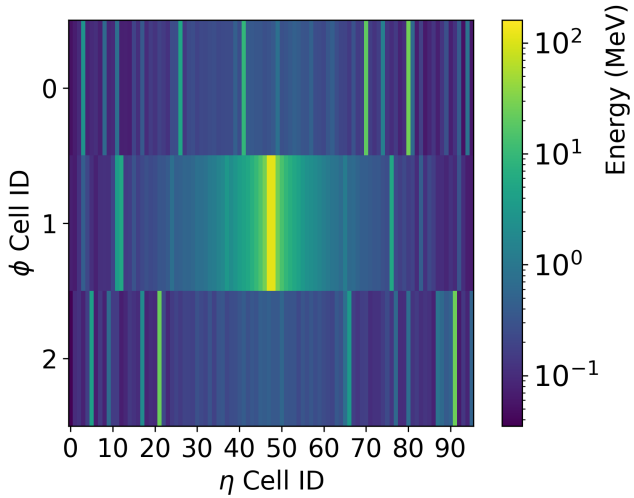
$$C = \frac{a_2\alpha^2 + a_1\alpha + a_0}{b_2\alpha^2 + b_1\alpha + b_0}$$

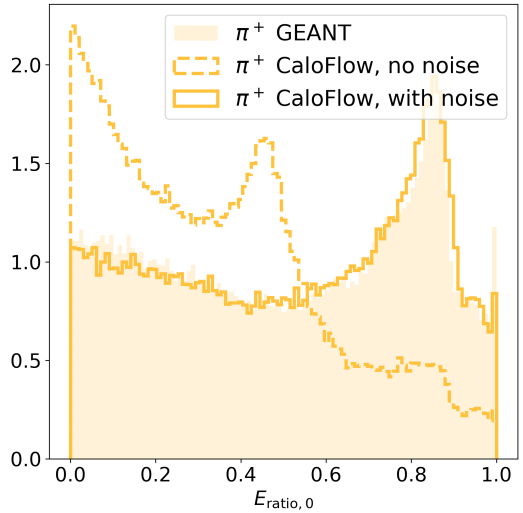
- still rather easy
- more flexible

The NN predicts the bin widths, heights, and derivatives that go in $a_i \& b_i$.

Adding Noise is important for the sampling quality.







- The log-likelihood is less noisy, but smaller. Yet, the quality of the samples is much better!
- This is due to a "wider" mapping of space and less overfitting.

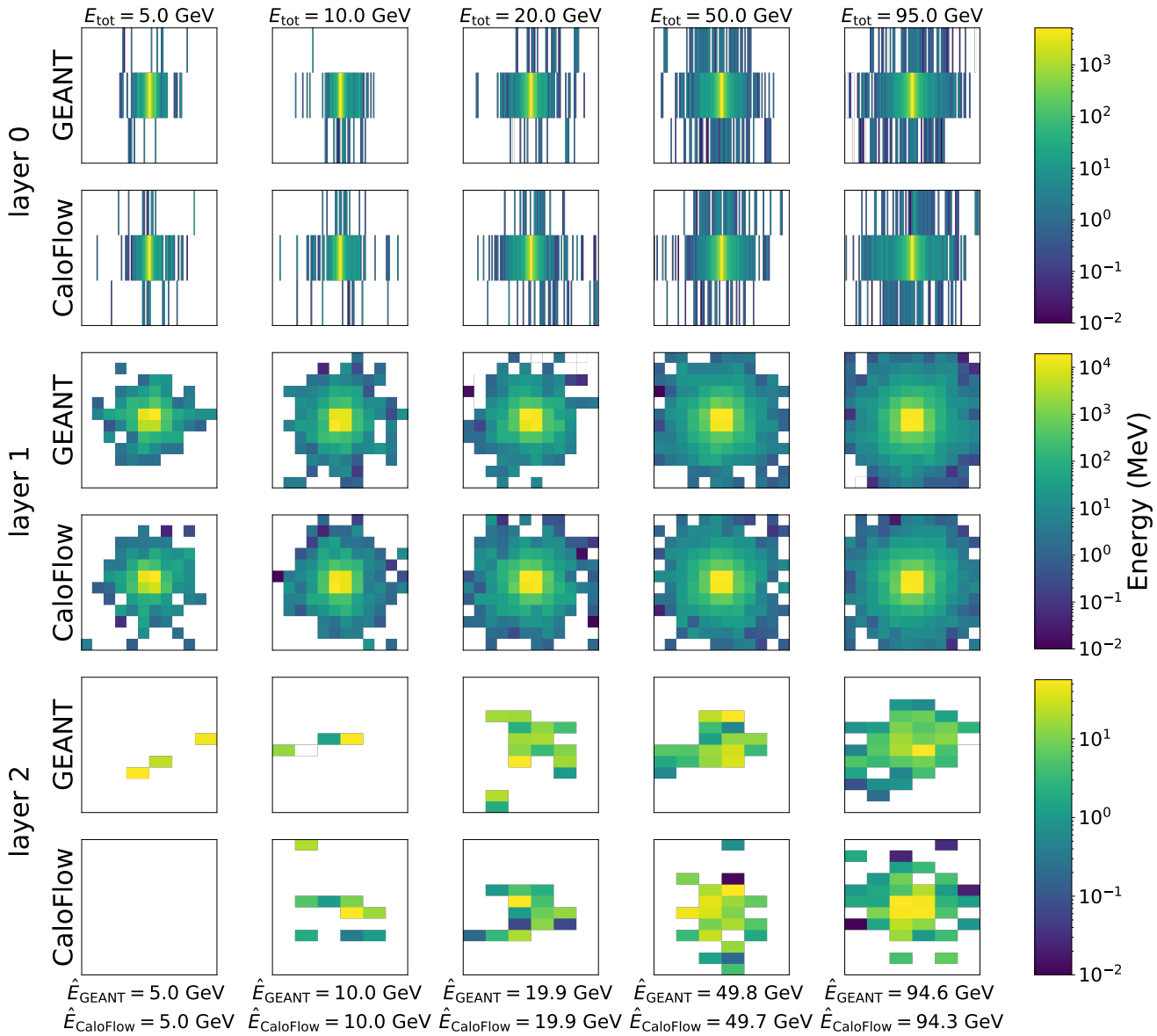
CaloFlow has moderate speed.

	CaloFlow*	CaloGAN*		Geant4†
training	22+82 min	210 min		0 min
generation batch size	time per shower	batch size req.	100k req.	
10	835 ms	455 ms	2.2 ms	1772 ms
100	96.1 ms	45.5 ms	0.3 ms	1772 ms
1000	41.4 ms	4.6 ms	0.08 ms	1772 ms
5000	36.2 ms	1.0 ms	0.07 ms	1772 ms
10000	36.2 ms	0.5 ms	0.07 ms	1772 ms

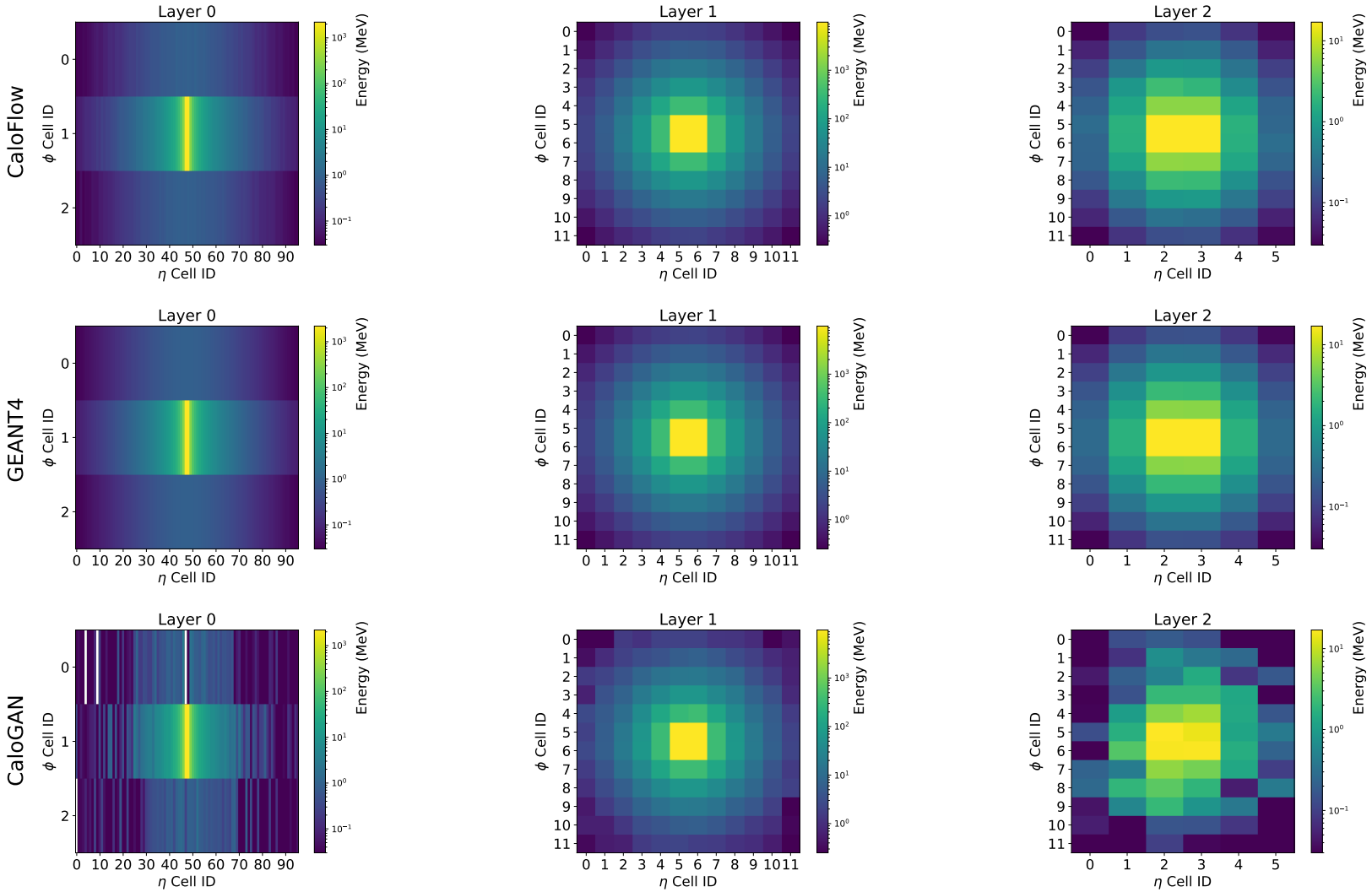
*: on our Titan V GPU

†: on the CPU of CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD; PRL]

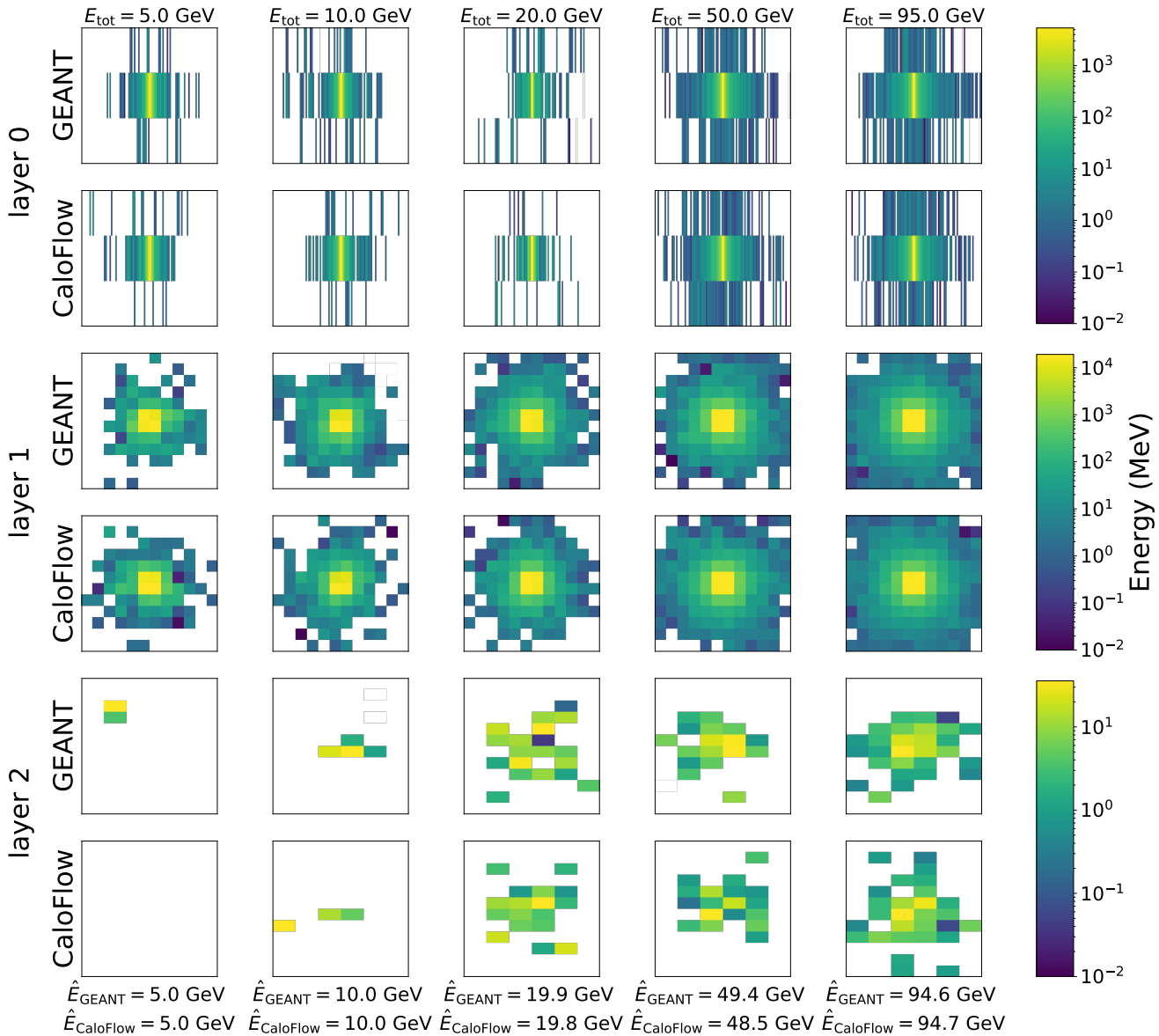
Nearest Neighbors: e^+



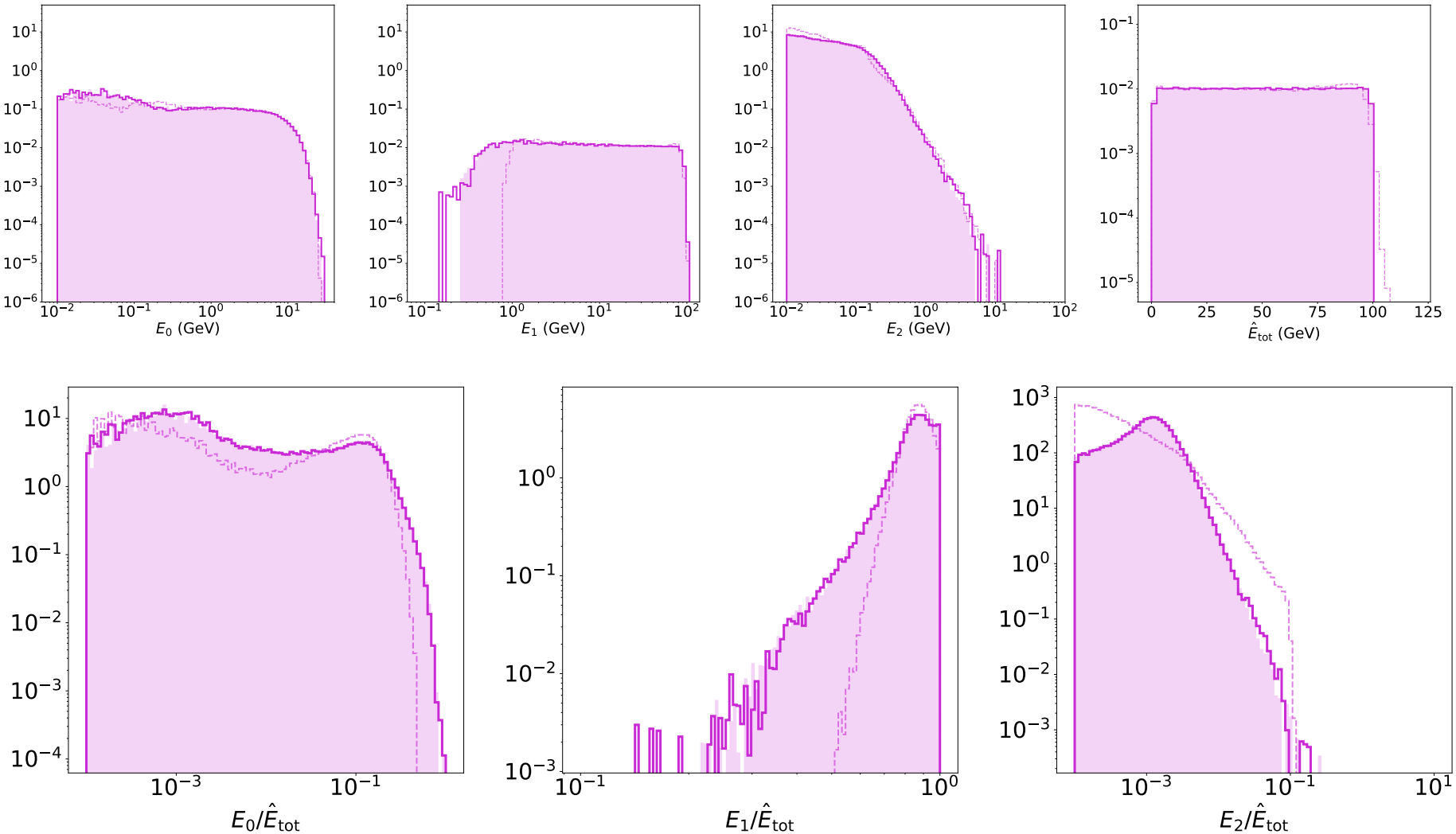
Comparing Shower Averages: γ



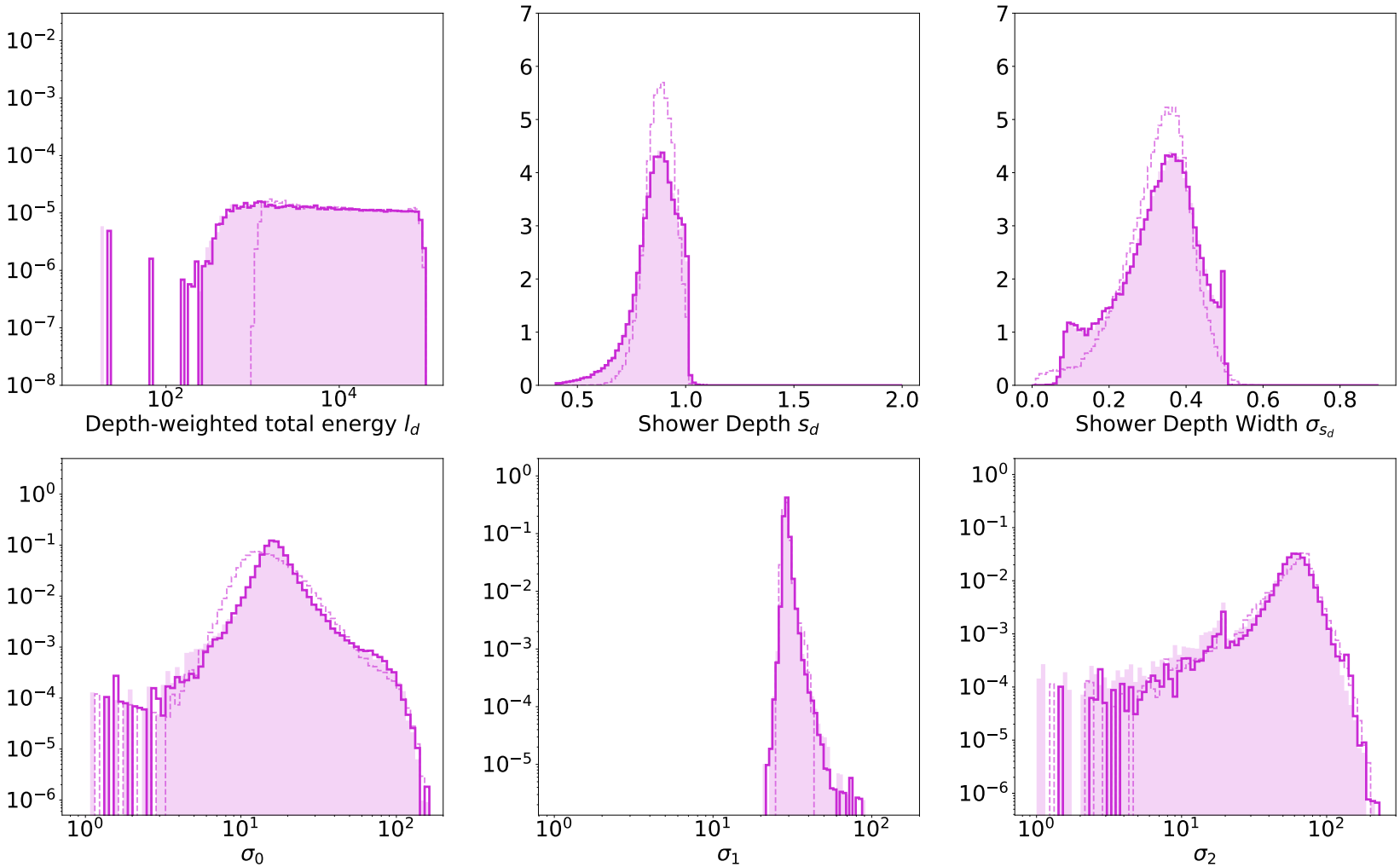
Nearest Neighbors: γ



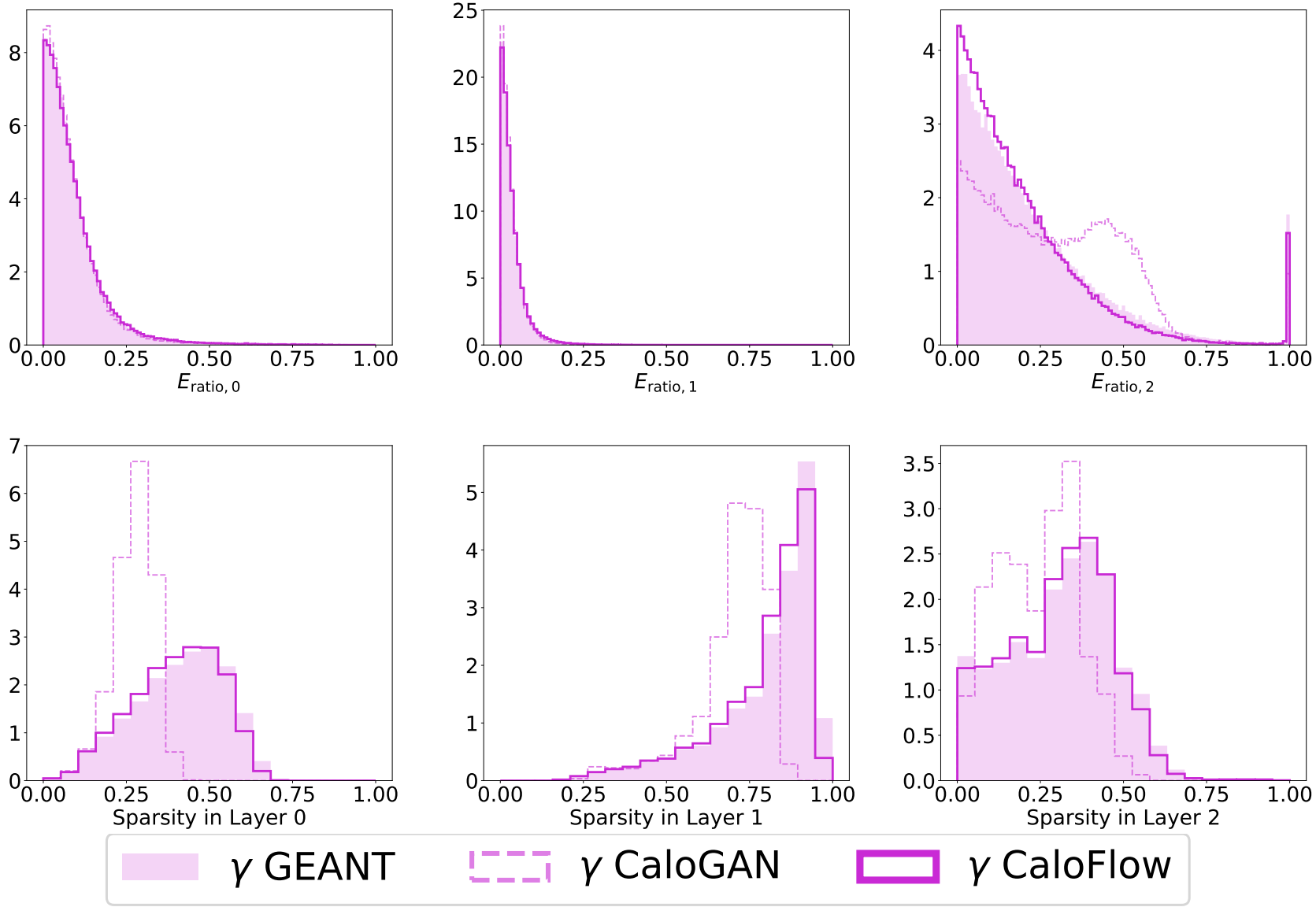
Flow I histograms: γ



Flow I+II histograms: γ



Flow II histograms: γ



Comparing Shower Averages: π^+