A Classifier as an Ultimate Metric: The CaloFlow Example

— ML4Sim Meeting —

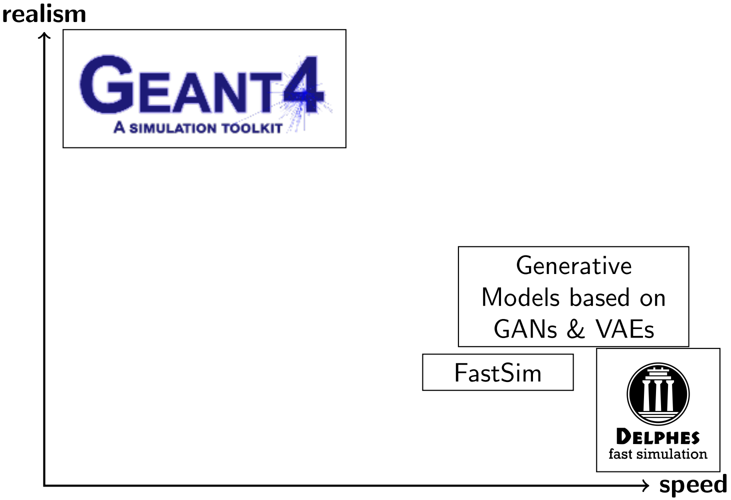
Claudius Krause

Rutgers, The State University of New Jersey

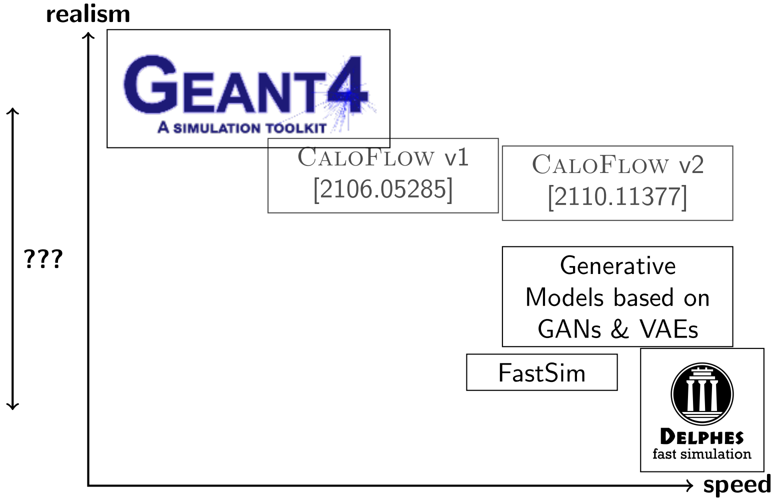
October 28, 2021

In collaboration with David Shih
arXiv: 2106.05285 and 2110.11377

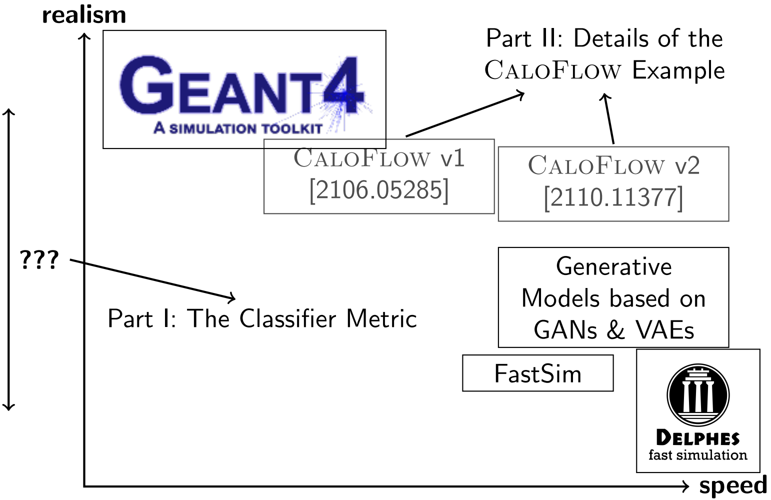
Detector Simulation needs to be fast and faithful



Detector Simulation needs to be fast and faithful

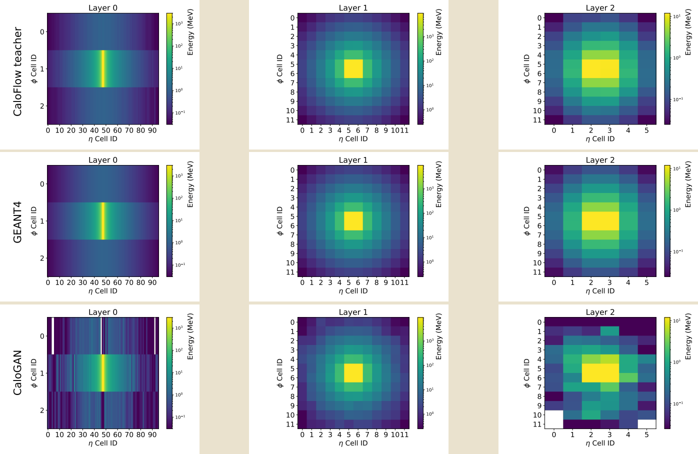


A Classifier as an Ultimate Metric: The CaloFlow Example



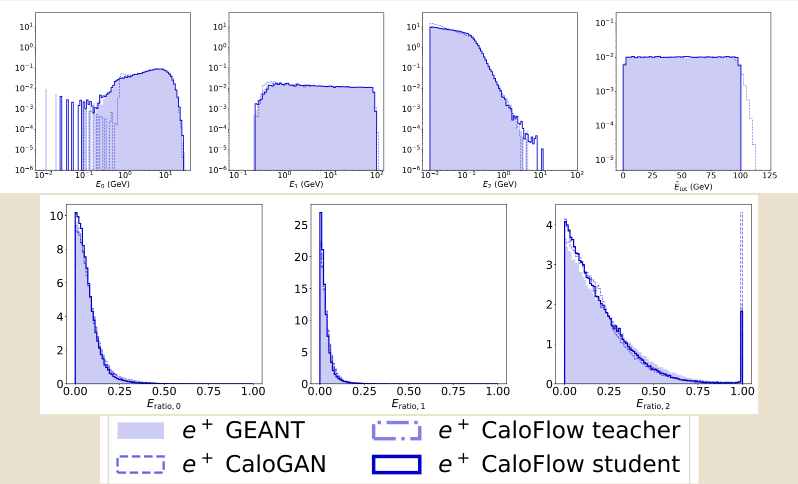
There are many options for comparisons.

We can look at average images or histograms.
⇒ Always a projection to a subspace.
⇒ How can we quantify the results?



There are many options for comparisons.

We can look at average images or histograms.
⇒ Always a projection to a subspace.
⇒ How can we quantify the results?



There are many options for comparisons.

We can look at average images or histograms.
$\Rightarrow$ Always a projection to a subspace.
$\Rightarrow$ How can we quantify the results?

NFs learn $p(x)$ explicitly! We can use the LL to compare them.
$\Rightarrow$ That doesn't work for GEANT4 / GAN / VAE samples.

Particle	CALOFLOW v1 NLL	CALOFLOW v2 NLL
e^+	142.159	146.393
γ	194.064	197.347
π^+	637.265	639.678

There are many options for comparisons.

We can look at average images or histograms.
$\Rightarrow$ Always a projection to a subspace.
$\Rightarrow$ How can we quantify the results?

NFs learn $p(x)$ explicitly! We can use the LL to compare them.
$\Rightarrow$ That doesn't work for GEANT4 / GAN / VAE samples.

Compare performance in downstream tasks.
$\Rightarrow$ Too many and too time-consuming.

There are many options for comparisons.

We can look at average images or histograms.
⇒ Always a projection to a subspace.
⇒ How can we quantify the results?

NFs learn $p(x)$ explicitly! We can use the LL to compare them.
⇒ That doesn't work for GEANT4 / GAN / VAE samples.

Compare performance in downstream tasks.
⇒ Too many and too time-consuming.

We can compare the performance of classifying e^+ vs π^+ etc. within GEANT4 to within the generative model.
⇒ Doesn't tell us much.

e^+ vs. π^+

		Test on GEANT4	Test on CALOGAN
Train on	GEANT4	99.6% ± 0.1%	96.5% ± 1.1%
	CALOGAN	98.2% ± 0.9%	99.9% ± 0.2%

e^+ vs. γ

		Test on GEANT4	Test on CALOGAN
Train on	GEANT4	66.1% ± 1.2%	70.6% ± 2.6%
	CALOGAN	54.3% ± 0.8%	100.0% ± 0.0%

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

There are many options for comparisons.

We can look at average images or histograms.
$\Rightarrow$ Always a projection to a subspace.
$\Rightarrow$ How can we quantify the results?

NFs learn $p(x)$ explicitly! We can use the LL to compare them.
$\Rightarrow$ That doesn't work for GEANT4 / GAN / VAE samples.

Compare performance in downstream tasks.
$\Rightarrow$ Too many and too time-consuming.

We can compare the performance of classifying e^+ vs π^+ etc. within GEANT4 to within the generative model.
$\Rightarrow$ Doesn't tell us much.

We can compare $p_{\text{GEANT4}}(x)$ and $p_{\text{generated}}(x)$ via their ratio.
$\Rightarrow$ The "Ultimate Metric" based on a classifier!

A Classifier provides the "ultimate metric".

According to the Neyman-Pearson Lemma we have:

- The likelihood ratio is the most powerful test statistic to distinguish the two samples.
- A powerful classifier trained to distinguish the samples should therefore learn (something monotonically related to) this.
- If this classifier is confused, we conclude $p_{\mathrm{GEANT4}}(x) = p_{\mathrm{generated}}(x)$

$\Rightarrow$ This captures the full 504-dim. space.

? But why wasn't this used before?

$\Rightarrow$ Previous deep generative models were separable to almost 100%!

DCTRGAN: Diefenbacher et al. [2009.03796, JINST]

There are a few caveats / points for discussion.

- A NN-based classifier is only an approximation to the NP-classifier.

? Is it powerful enough? Is it well calibrated?

$\Rightarrow$ try different architectures / hyperparameters / pre-processing

? Is having $p_{\text{GEANT4}}(x) = p_{\text{generated}}(x)$ an overkill?

$\Rightarrow$ Train on high-level features instead?

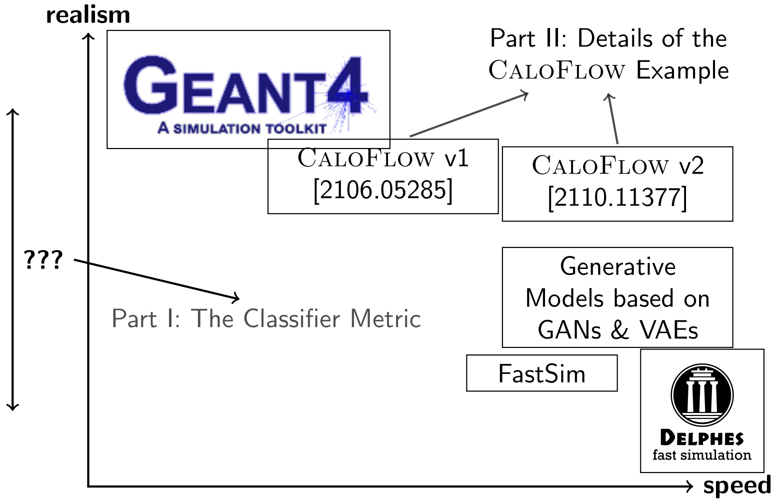
? What if the result is 100% ?

? Are the "tells" physically relevant? ($\Rightarrow$ high-level features?)

? How can we compare two "100%" models with each other?

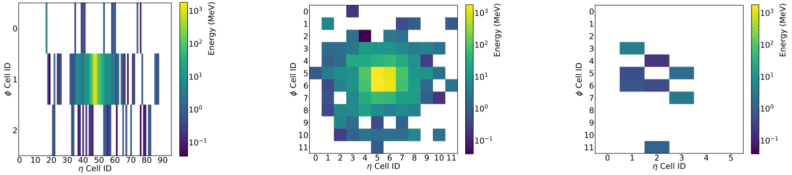
- It's time-consuming to train a classifier.

A Classifier as an Ultimate Metric: The CaloFlow Example



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
- The Geant4 configuration of CaloGAN is available at https://github.com/hep-lbdl/CaloGAN
- Showers of e^+, γ, and π^+ (100k each, centered, 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]

Normalizing Flows learn a change-of-coordinates efficiently.

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

$\xleftarrow{\text{density estimation, } p(x)}$

$\xrightarrow{\text{sample generation}}$

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.

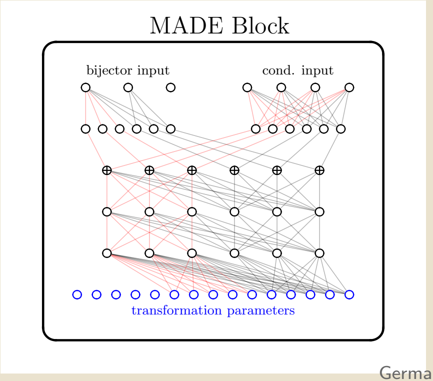
- Each transformation has an analytic Jacobian and inverse.

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

- An autoregressive architecture ensures a triangular Jacobian.

$\Rightarrow$ Can be obtained by masking a DNN.

Masking Ensures 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]

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

CaloFlow uses a 2-step approach.

Flow I

- learns $p_1(E_0, E_1, E_2|E_{\text{tot}})$
- is a MAF that is optimized using the LL.

Flow II

- learns $p_2(\vec{\mathcal{I}}|E_0, E_1, E_2, E_{\text{tot}})$ of normalized showers
- in CaloFlow v1 (2106.05285 — called "teacher"):
  - MAF trained with LL
  - Slow in sampling ($\approx 500\times$ slower than CaloGAN)
- in CaloFlow v2 (2110.11377 — called "student"):
  - IAF trained with Probability Density Distillation from teacher (LL prohibitive) van den Oord et al. [1711.10433]
  - Fast in sampling ($\approx 500\times$ faster than CaloFlow v1)

A Classifier provides the "ultimate metric".

According to the Neyman-Pearson Lemma we have:
$p_{\text{GEANT4}}(x) = p_{\text{generated}}(x)$ if a classifier cannot distinguish data from generated samples.

AUC / JSD		DNN		
		GEANT4 vs. CALOGAN	GEANT4 vs. CALOFLOW v1 (teacher)	GEANT4 vs. CALOFLOW v2 (student)
e^+	unnorm.	1.000(0) / 0.993(1)	0.847(8) / 0.345(12)	0.785(7) / 0.200(10)
	norm.	1.000(0) / 0.997(0)	0.869(2) / 0.376(4)	0.824(5) / 0.255(8)
γ	unnorm.	1.000(0) / 0.996(1)	0.660(6) / 0.067(4)	0.761(14) / 0.167(18)
	norm.	1.000(0) / 0.994(1)	0.794(4) / 0.213(7)	0.761(4) / 0.159(6)
π^+	unnorm.	1.000(0) / 0.988(1)	0.632(2) / 0.048(1)	0.729(2) / 0.144(3)
	norm.	1.000(0) / 0.997(0)	0.751(4) / 0.148(4)	0.807(2) / 0.231(4)

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

A Classifier as an Ultimate Metric: The CaloFlow Example

- We propose a classifier as the "ultimate metric".
- This captures the full shower and correlations.
- It can be applied to any (deep) generative model.

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

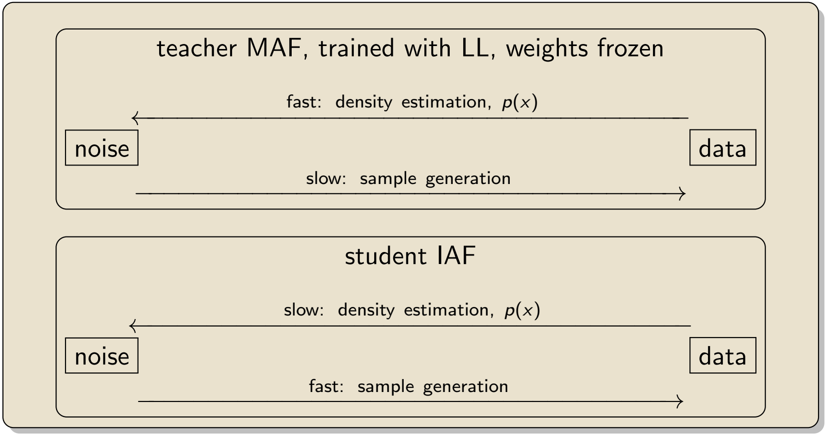
⇒ First time application of Normalizing Flows!

- The results look impressive.

⇒ CaloFlow v2 is as fast as CaloGAN (0.08ms / shower) and outperforms CaloGAN in the "Ultimate Metric".

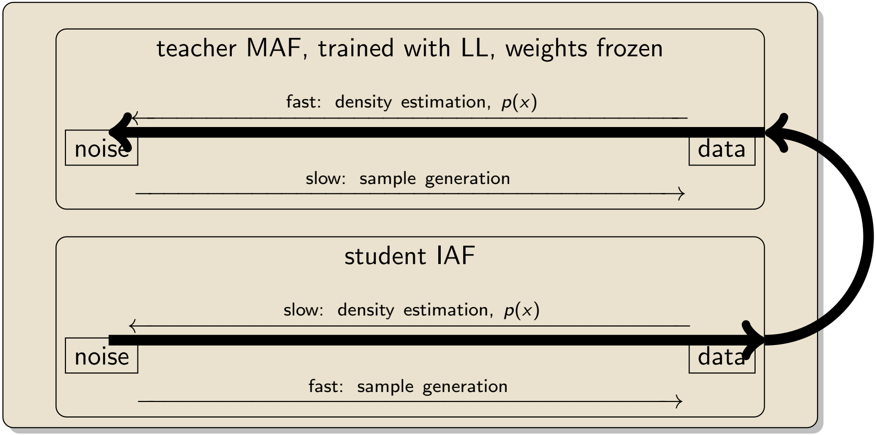
Backup

Probability Density Distillation passes the information from the teacher to the student



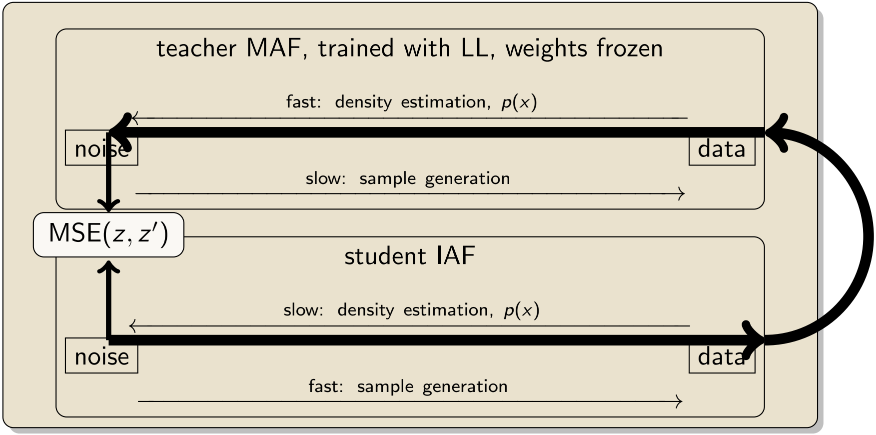
$$\begin{aligned}\text{Loss} = \text{MSE}(z, z') &+ \text{MSE}(x, x') + \text{MSE}(z_i, z_i') \\ &+ \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')\end{aligned}$$

Probability Density Distillation passes the information from the teacher to the student



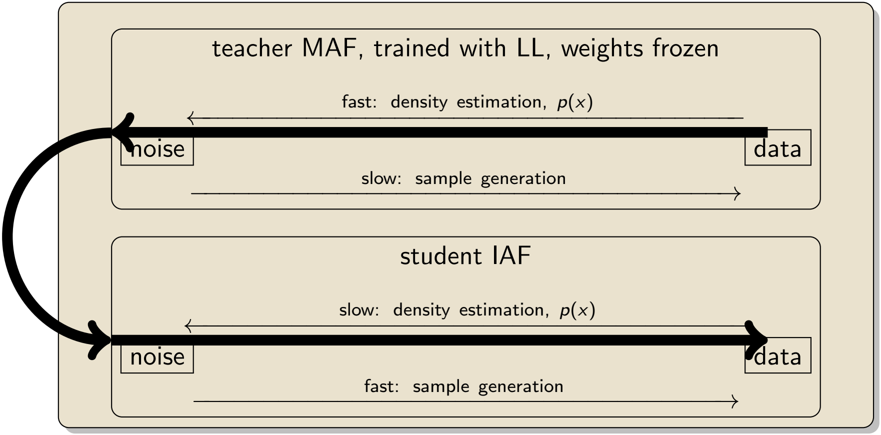
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

Probability Density Distillation passes the information from the teacher to the student



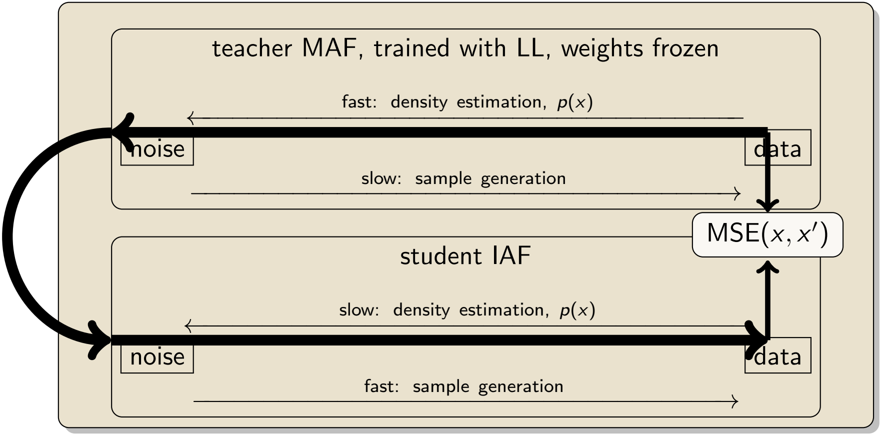
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

Probability Density Distillation passes the information from the teacher to the student



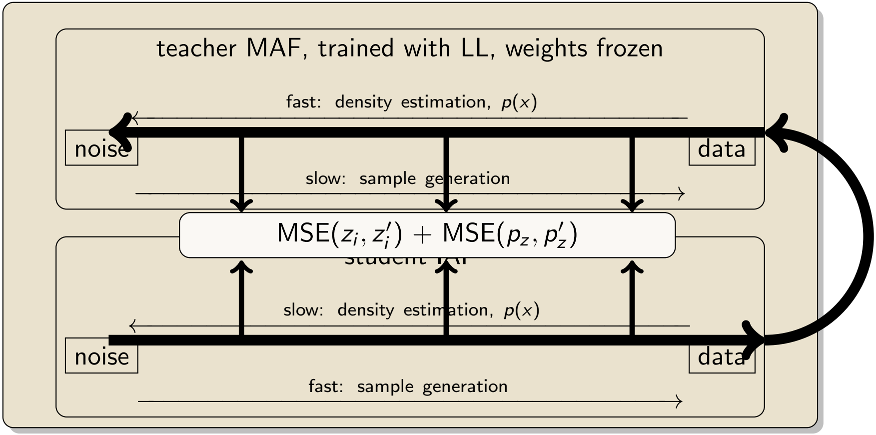
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

Probability Density Distillation passes the information from the teacher to the student



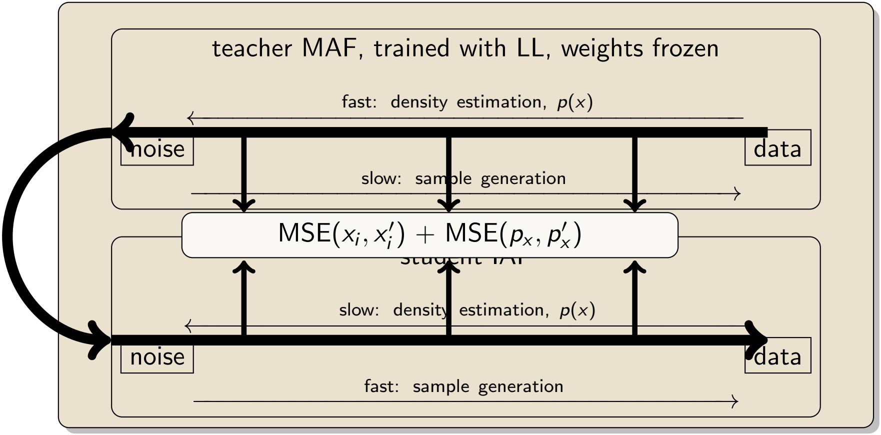
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

Probability Density Distillation passes the information from the teacher to the student



$$\begin{aligned}\text{Loss} = \mathrm{MSE}(z, z') &+ \mathrm{MSE}(x, x') + \mathrm{MSE}(z_i, z_i') \\ &+ \mathrm{MSE}(x_i, x_i') + \mathrm{MSE}(p_z, p_z') + \mathrm{MSE}(p_x, p_x')\end{aligned}$$

Probability Density Distillation passes the information from the teacher to the student



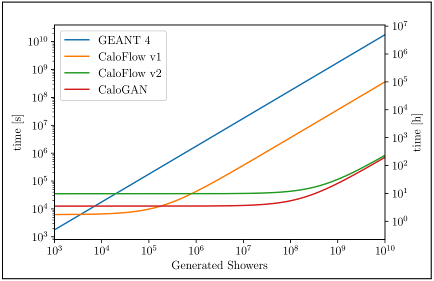
$$\begin{aligned}\text{Loss} = \mathrm{MSE}(z, z') \; &+ \mathrm{MSE}(x, x') + \mathrm{MSE}(z_i, z_i') \\ &+ \mathrm{MSE}(x_i, x_i') + \mathrm{MSE}(p_z, p_z') \; + \mathrm{MSE}(p_x, p_x')\end{aligned}$$

Sampling Speed: The Student beats the Teacher!

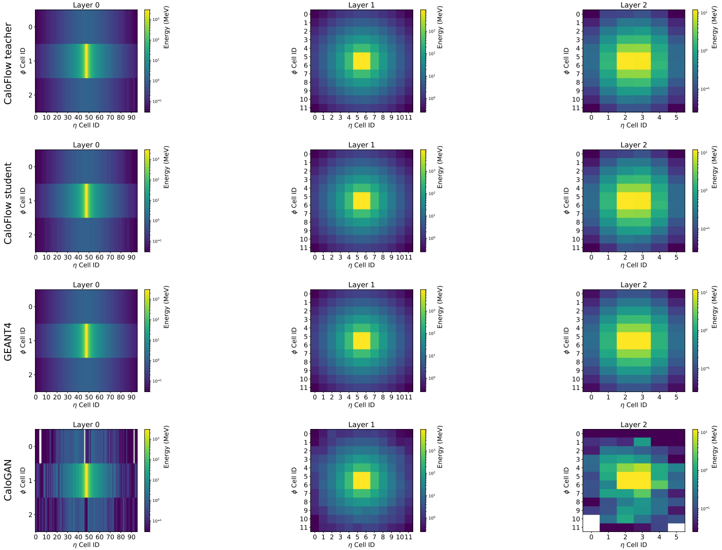
	CaloFlow*		CaloGAN*		Geant4†
	v1 (teacher)	v2 (student)			
training	22+82 min	+ 480 min	210 min		0 min
generation batch size	time per shower				
			batch size req.	100k req.	
10	835 ms	5.81 ms	455 ms	2.2 ms	1772 ms
100	96.1 ms	0.60 ms	45.5 ms	0.3 ms	1772 ms
1000	41.4 ms	0.12 ms	4.6 ms	0.08 ms	1772 ms
10000	36.2 ms	**0.08 ms**	0.5 ms	**0.07 ms**	1772 ms

*: on our Titan V GPU

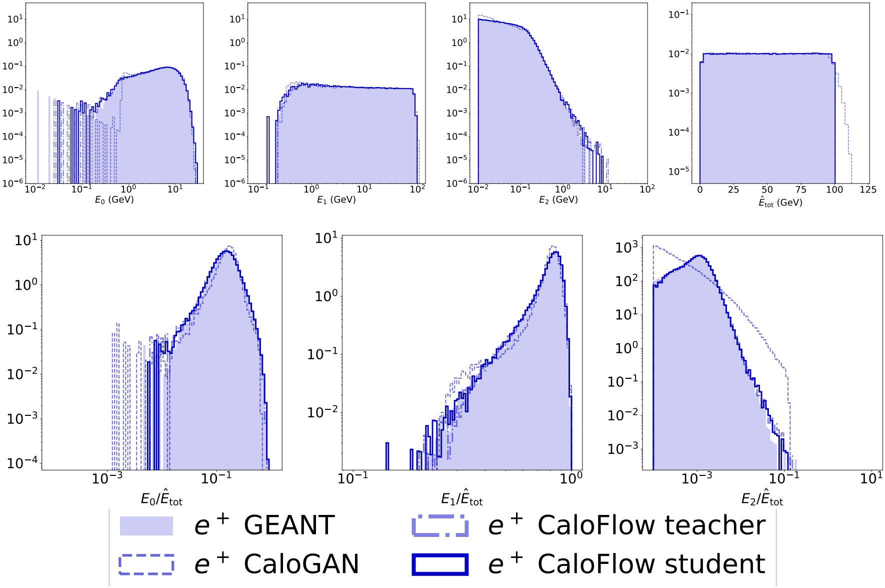
†: on the CPU of CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD]



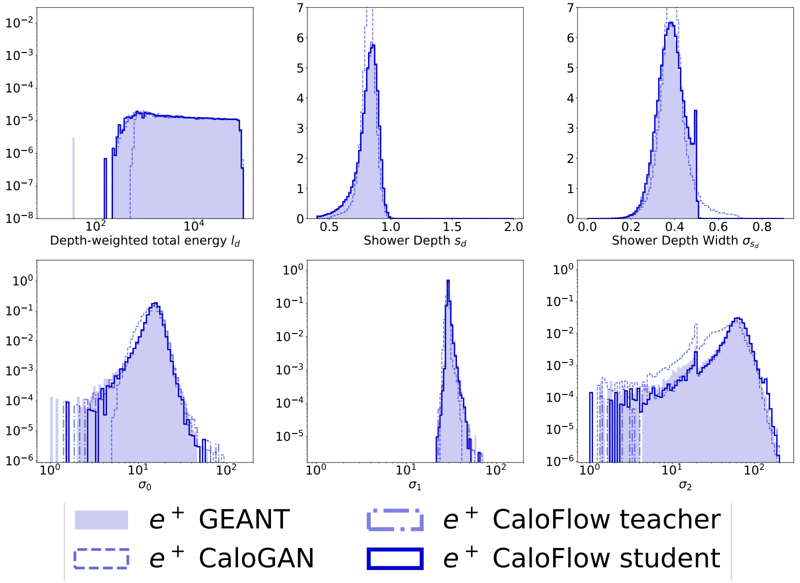
CaloFlow: Comparing Shower Averages: e^+



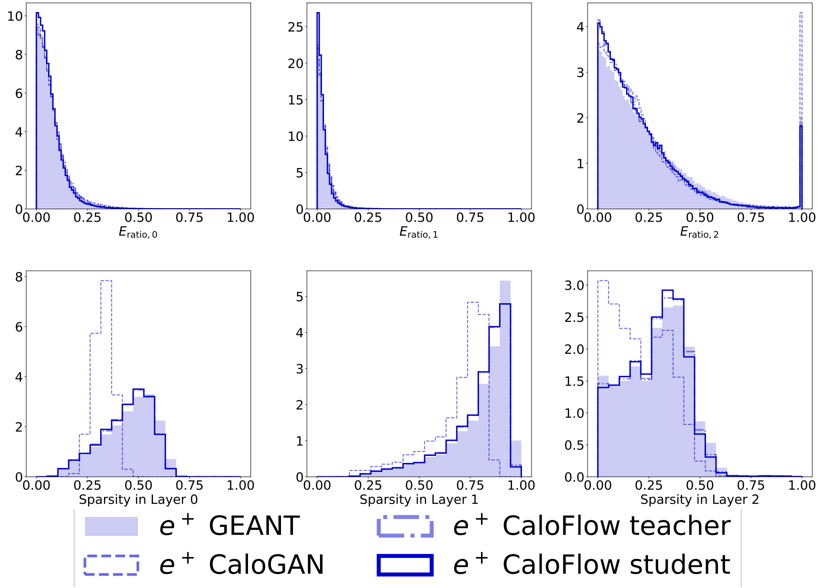
CaloFlow: Flow I histograms: e^+



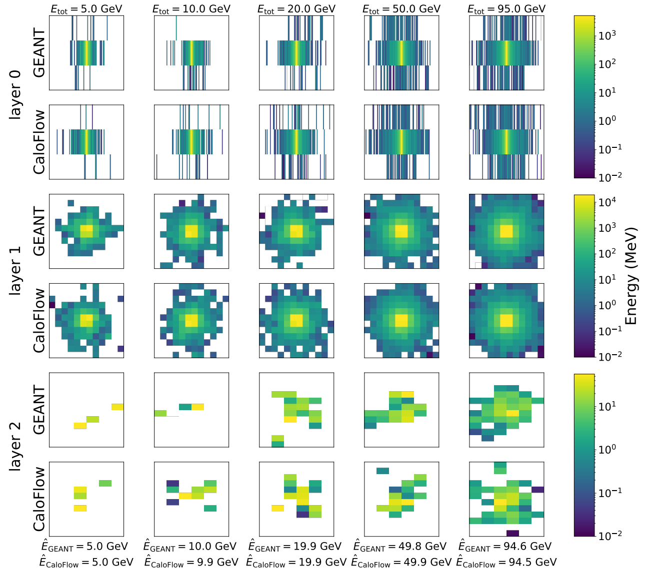
CaloFlow: Flow I+II histograms: e^+



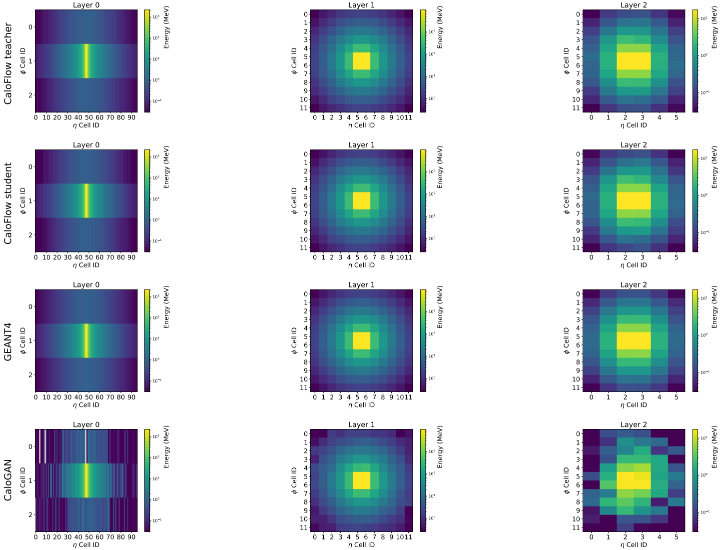
CaloFlow: Flow II histograms: e^+



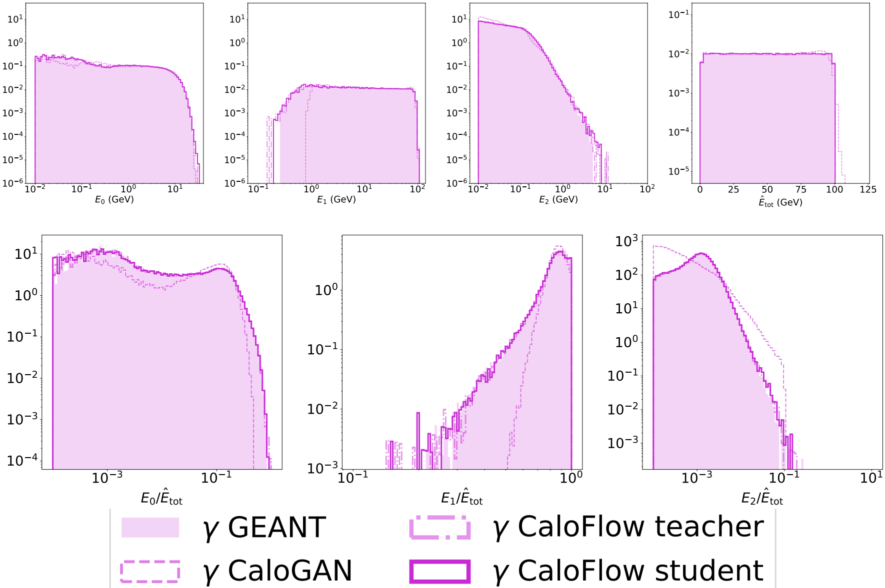
CaloFlow: Nearest Neighbors: e^+ (student)



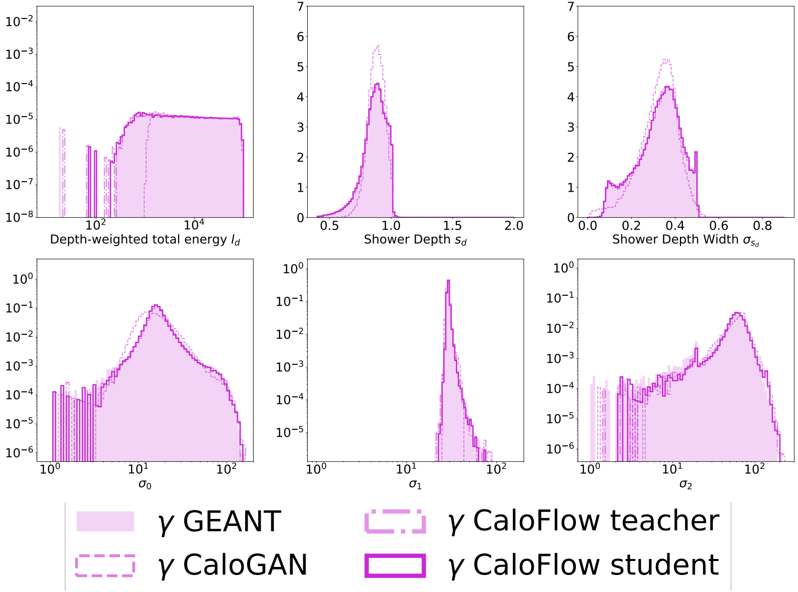
CaloFlow: Shower Averages: γ



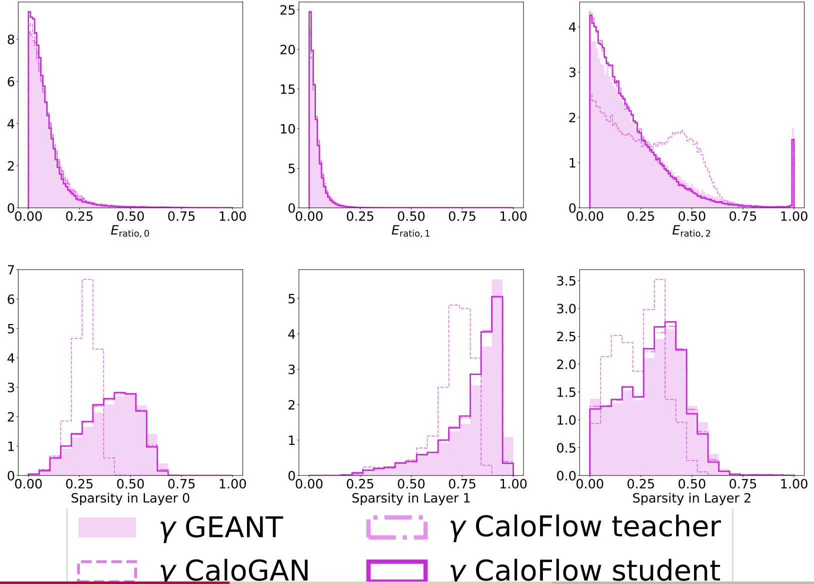
CaloFlow: Flow I histograms: γ



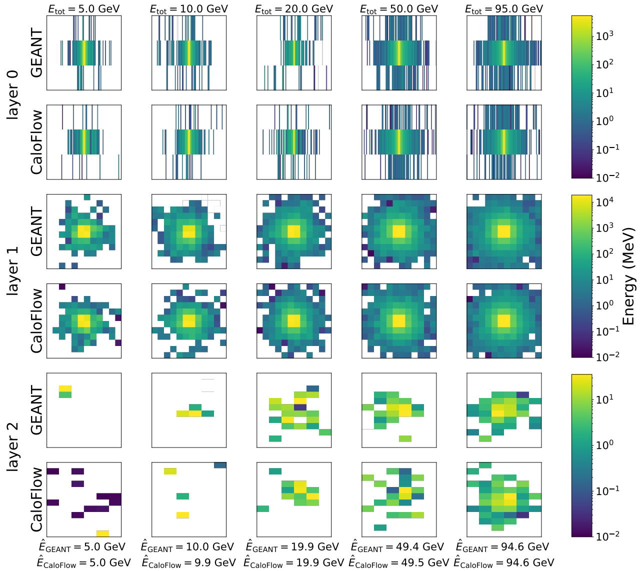
CaloFlow: Flow I+II histograms: γ



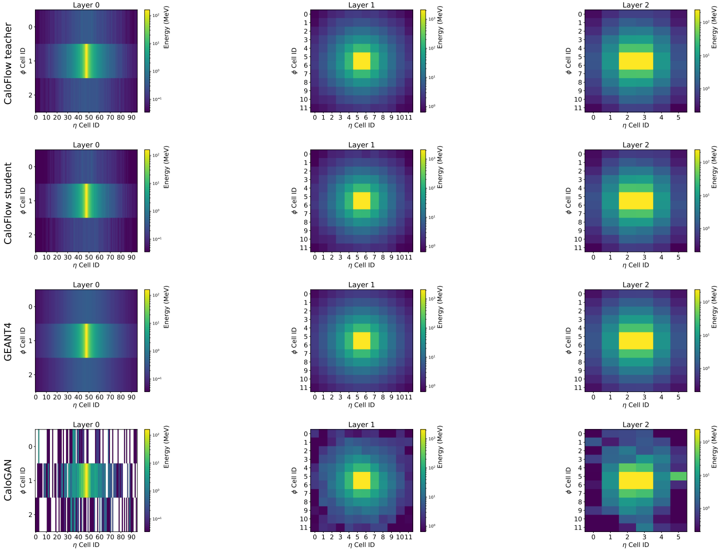
CaloFlow: Flow II histograms: γ



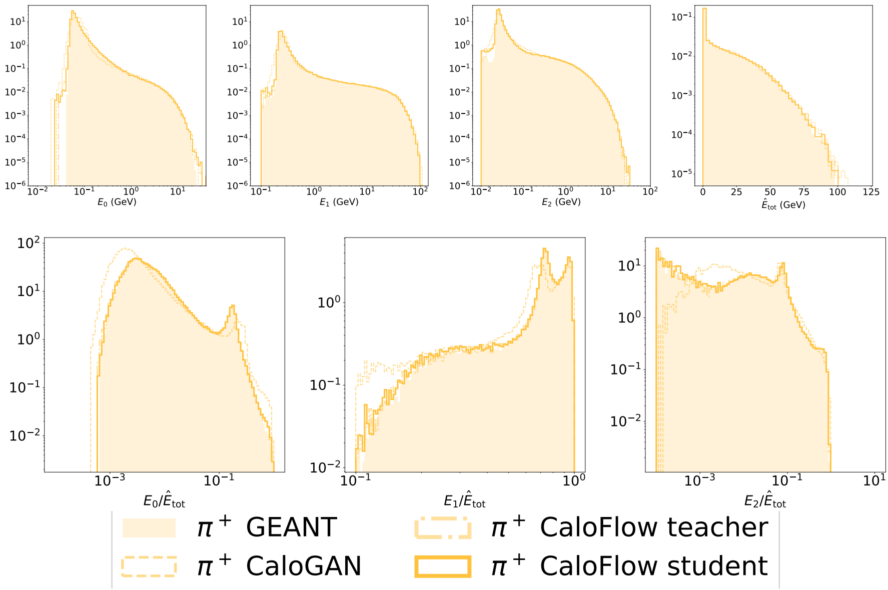
CaloFlow: Nearest Neighbors: γ (student)



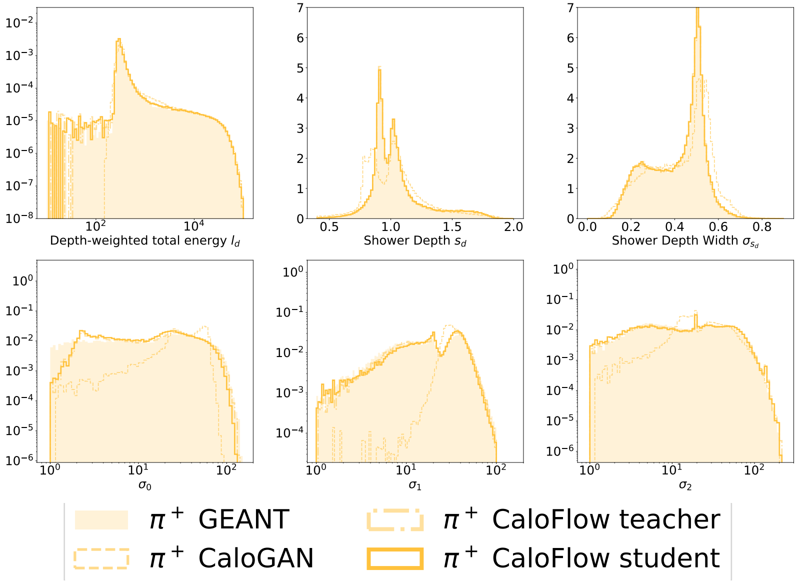
CaloFlow: Shower Averages: π^+



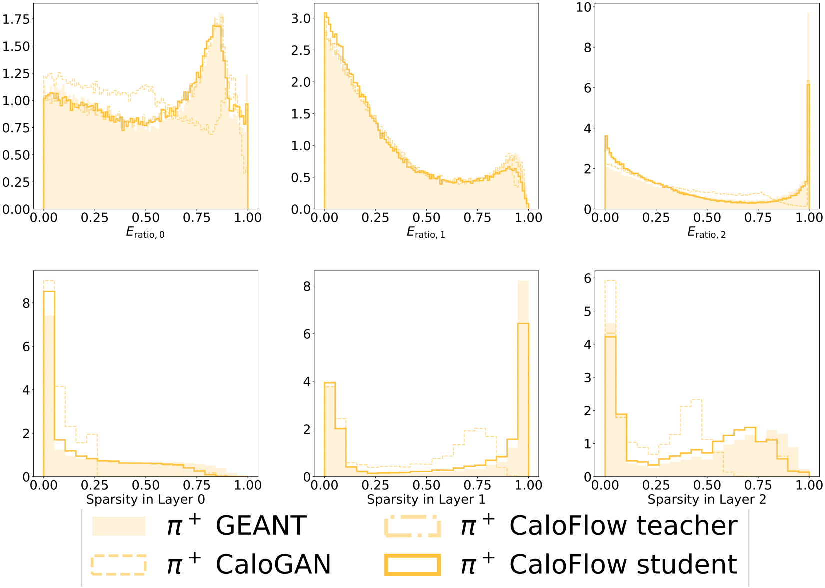
CaloFlow: Flow I histograms: π^+



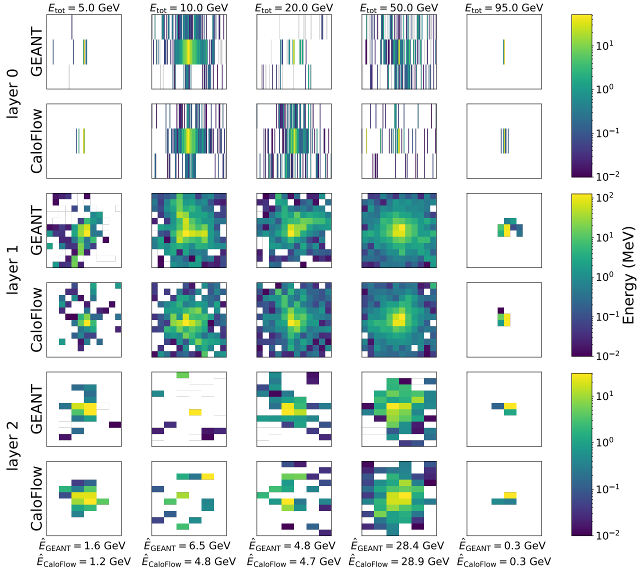
CaloFlow: Flow I+II histograms: π^+



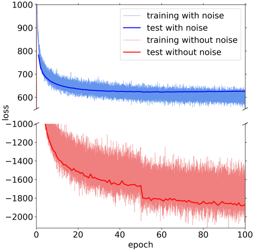
CaloFlow: Flow II histograms: π^+

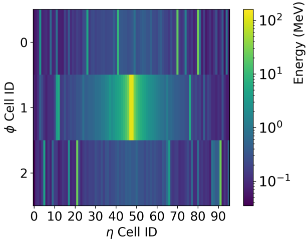


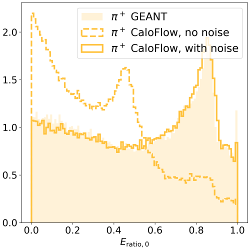
CaloFlow: Nearest Neighbors: π^+ (student)



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.