CaloFlow: Fast and Accurate Generation of Calorimeter Showers with Normalizing Flows

— HSF Detector Simulation WG —

Claudius Krause

Rutgers, The State University of New Jersey

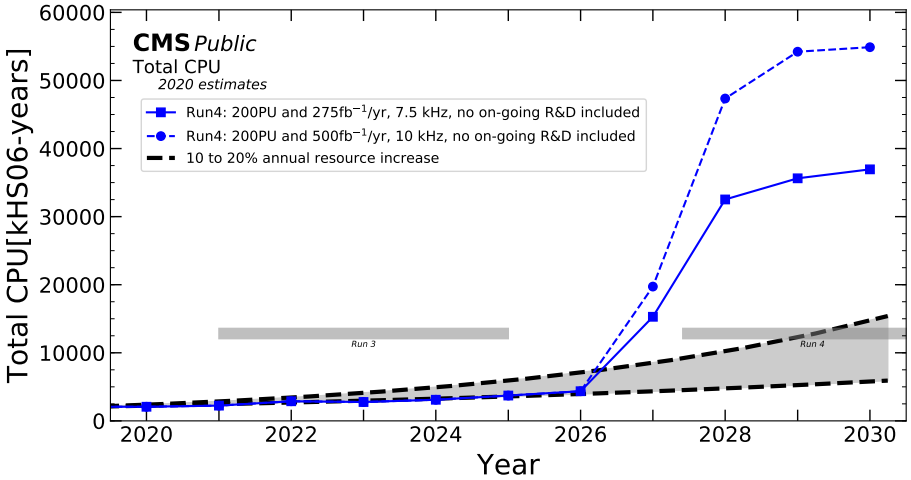
November 8, 2021

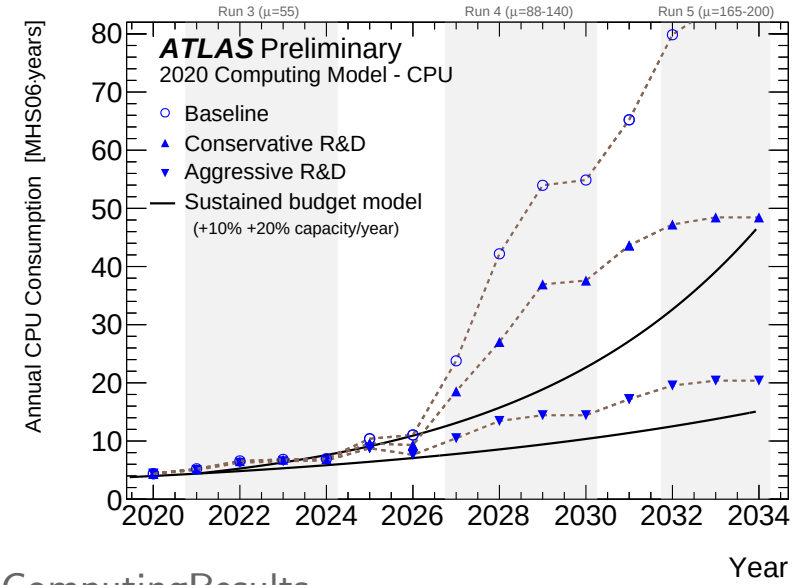
RUTGERS

UNIVERSITY | NEW BRUNSWICK

In collaboration with David Shih
arXiv: 2106.05285 and 2110.11377

Deep Generative Models will be crucial for the LHC.





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.
- A large fraction goes into simulation.

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

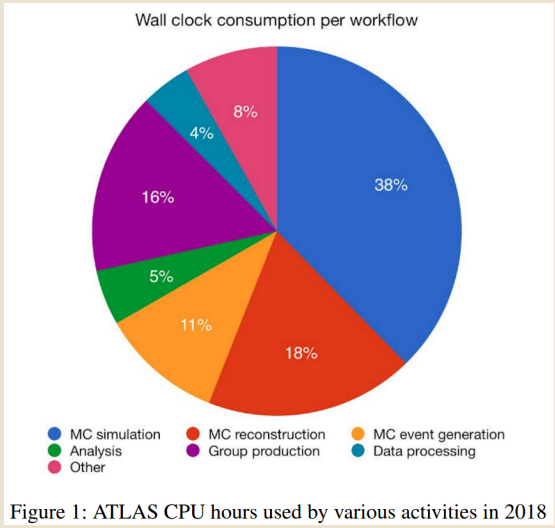
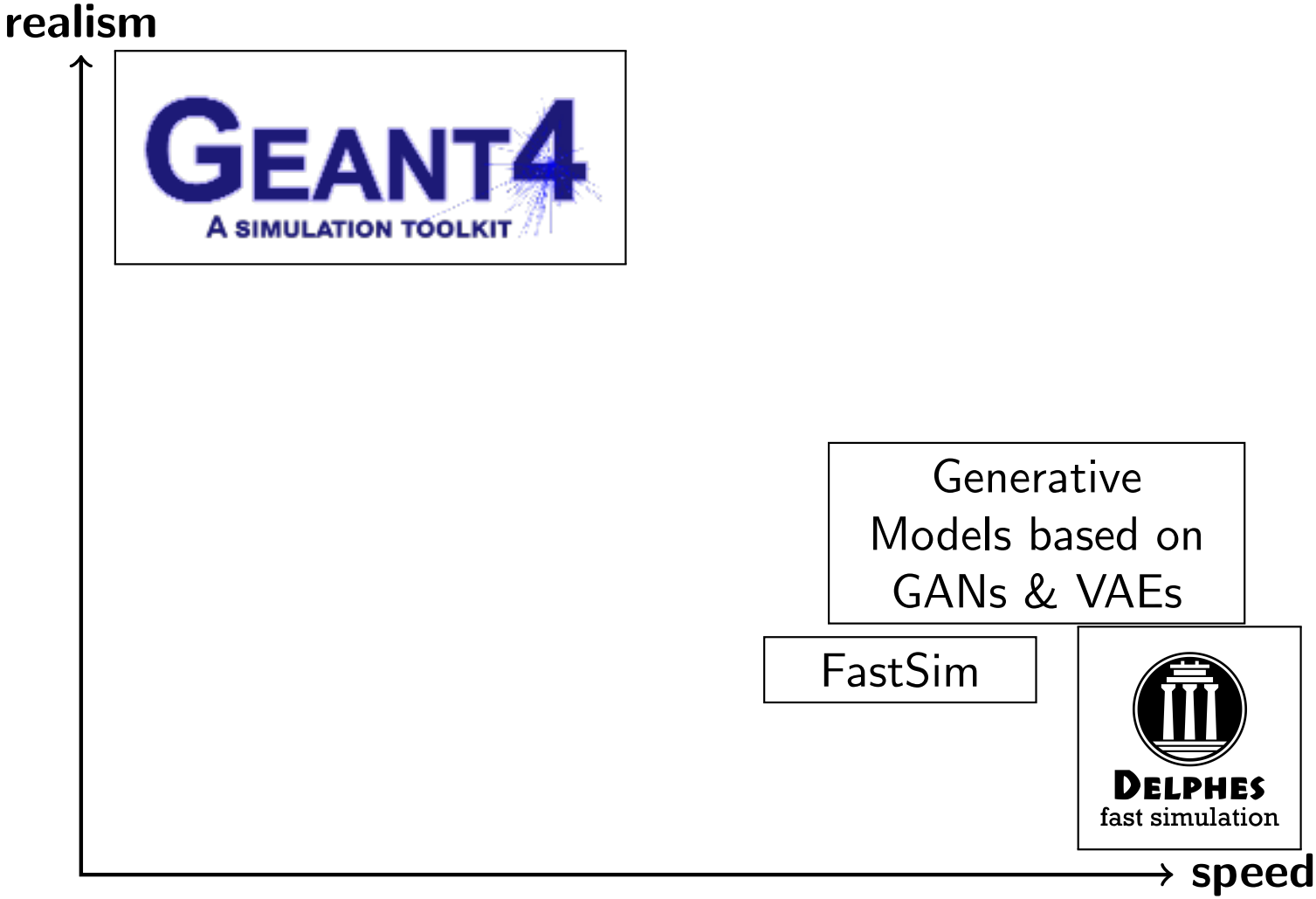
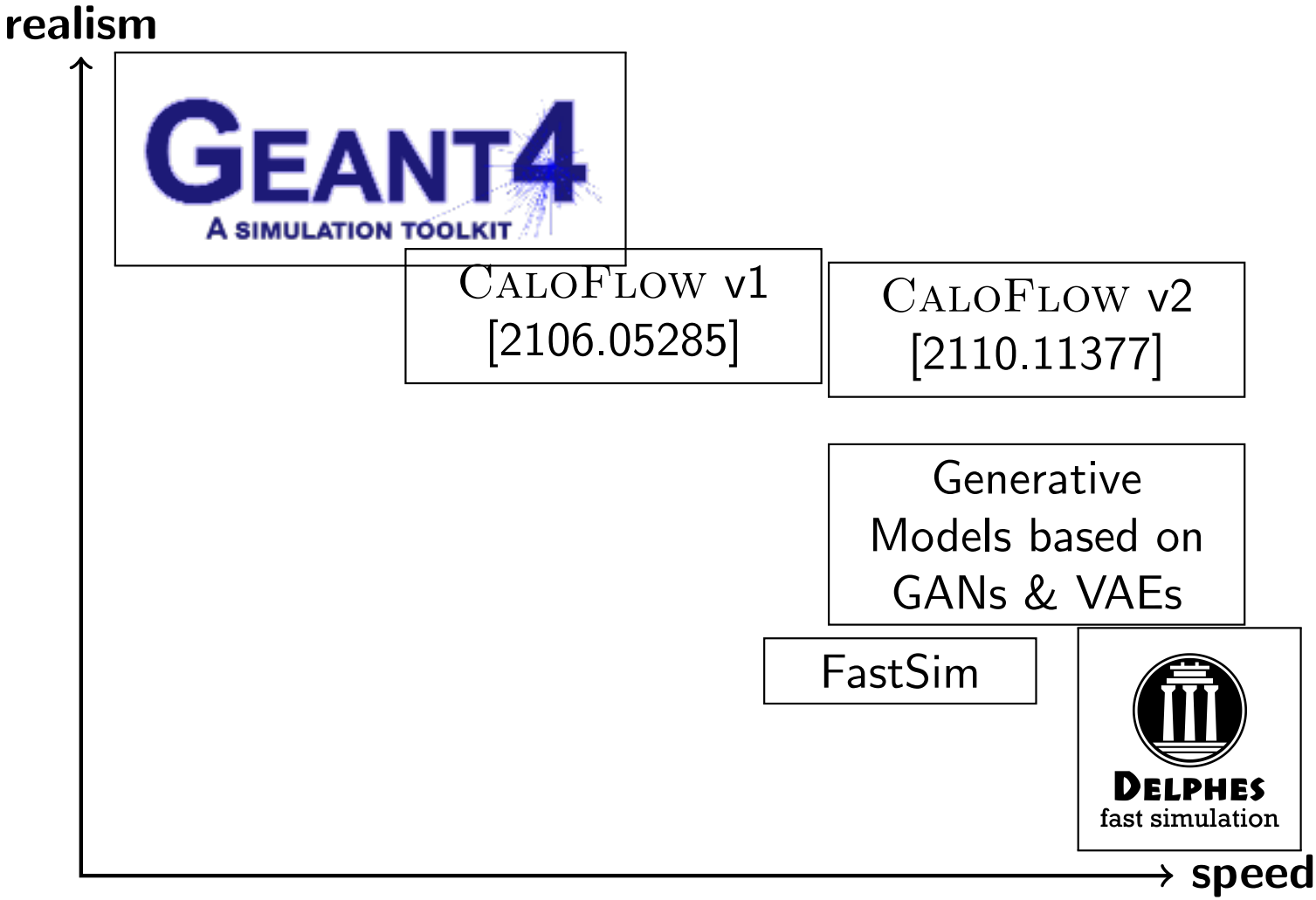


Figure 1: ATLAS CPU hours used by various activities in 2018

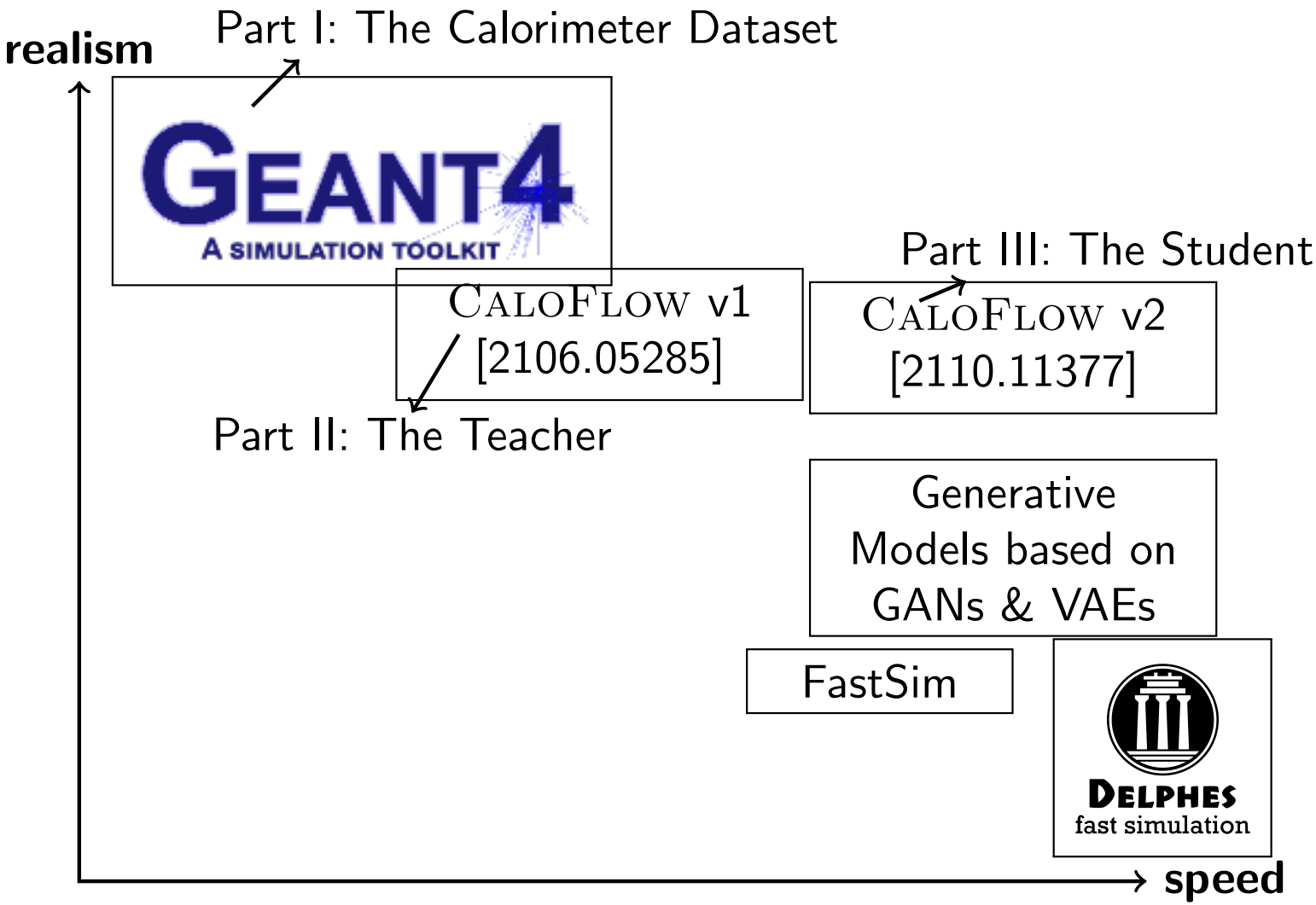
Detector Simulation needs to be fast and faithful



Detector Simulation needs to be fast and faithful

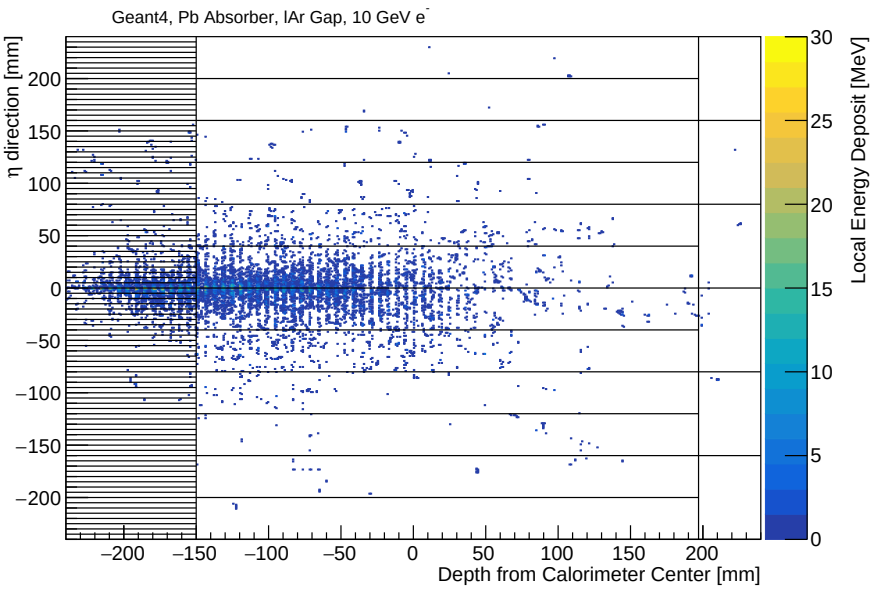


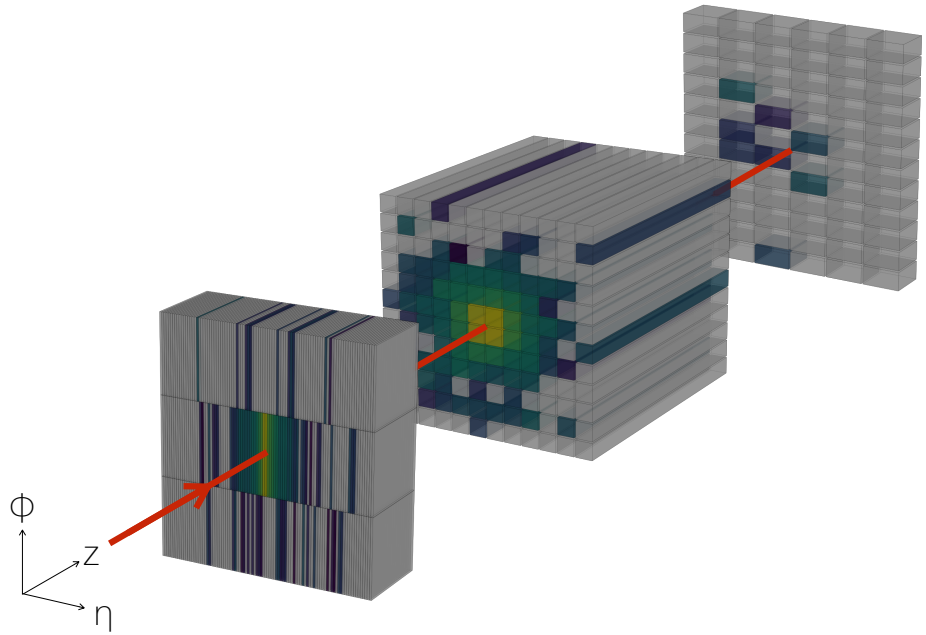
Detector Simulation needs to be fast and faithful



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

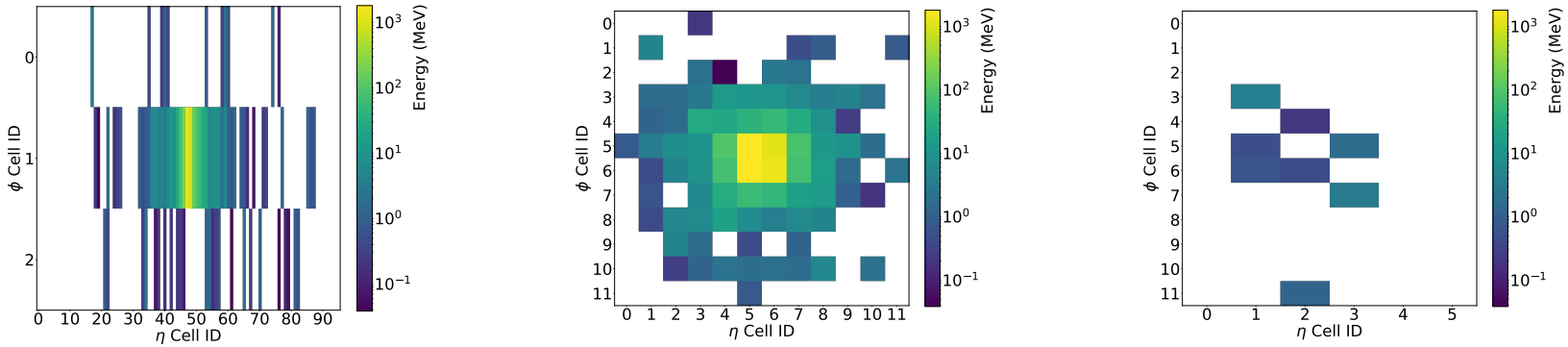




CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD]

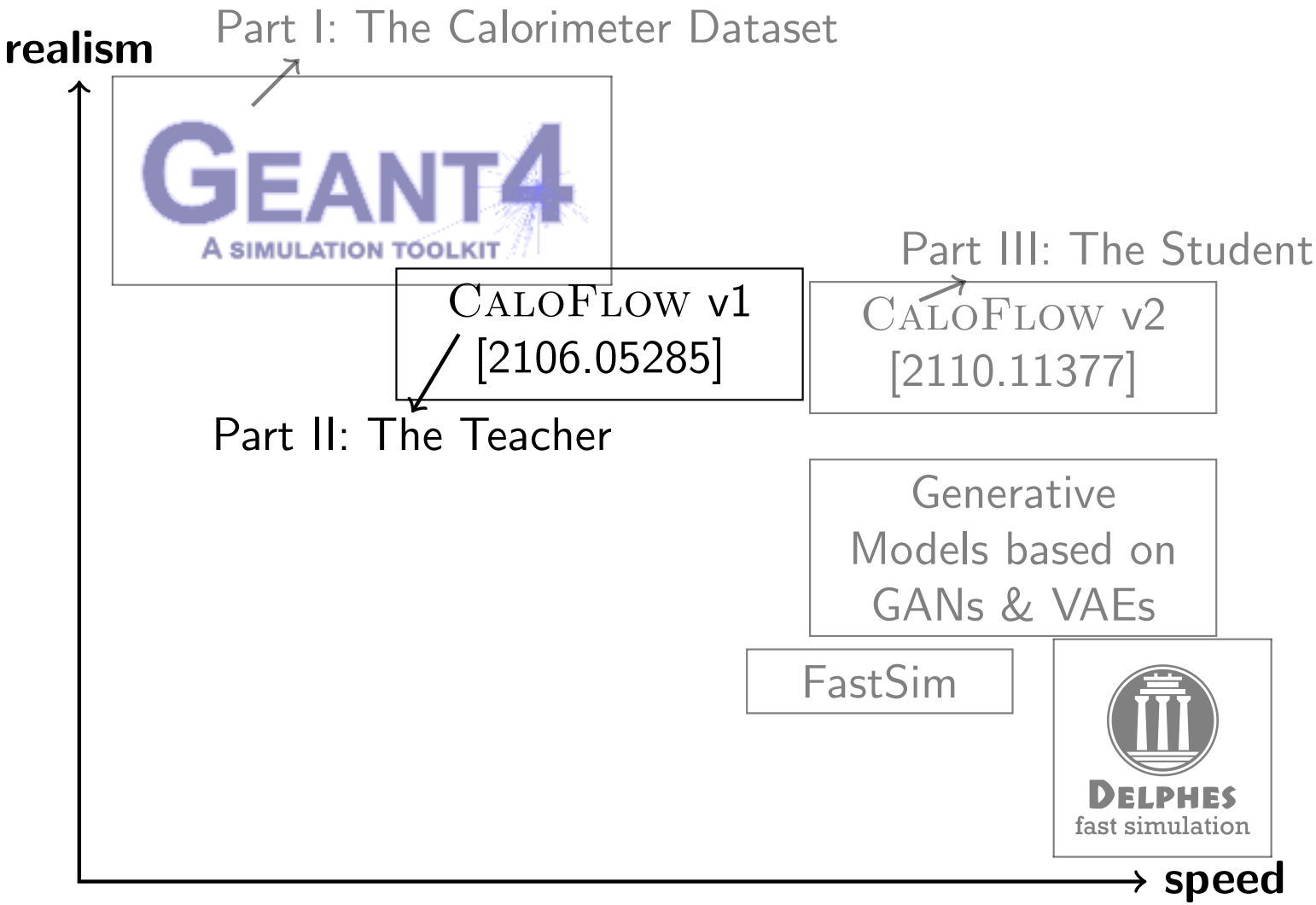
We use the same calorimeter geometry as CaloGAN

- 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:

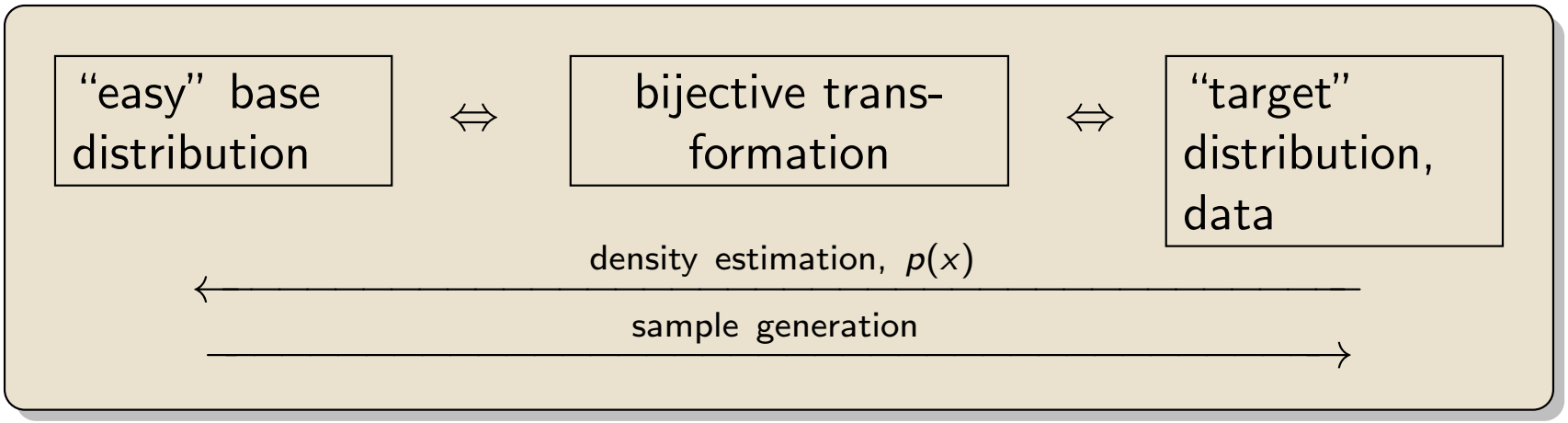


CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD]

Normalizing Flows for Calorimeter Showers: When the Student beats the Teacher



Normalizing Flows learn a change-of-coordinates efficiently.



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.

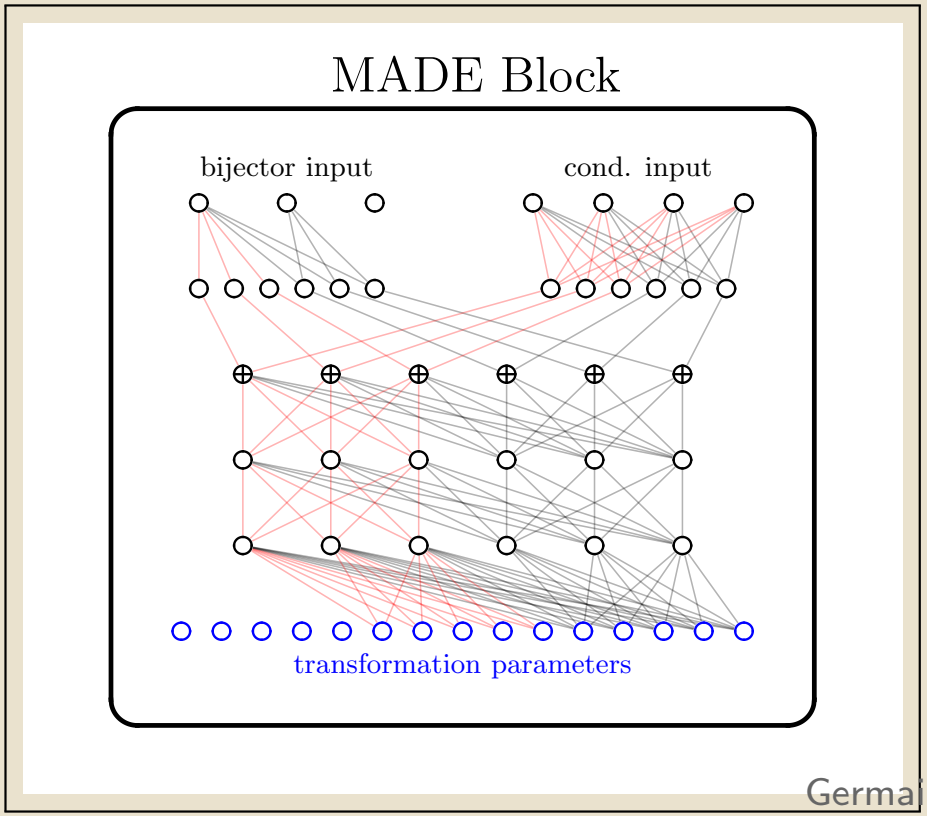
- Each transformation has an analytic Jacobian and inverse.

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

- An autoregressive architecture ensures a triangular Jacobian.

⇒ 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.

Challenges of Deep Generative Models

General challenges of deep generative models and how we avoid them:

? By which metric can we monitor the quality of the generator?

$\Rightarrow$ NFs learn $p(x)$ explicitly! $\Rightarrow$ select model based on LL.

? How can we ensure energy conservation?

$\Rightarrow$ We learn 2 NFs: $p_1(E_0, E_1, E_2|E_{\text{tot}})$ & $p_2(\vec{\mathcal{I}}|E_0, E_1, E_2, E_{\text{tot}})$

? How do we learn sparse data and "sharp edges"?

$\Rightarrow$ Learning in logit space and applying a threshold after generation.

? Will faster sampling time pay off the training time?

$\Rightarrow$ The student will do the trick! (see part III)

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:
  - MAF trained with LL
  - Slow in sampling ($\approx 500\times$ slower than CaloGAN)
  - Impressive quality!

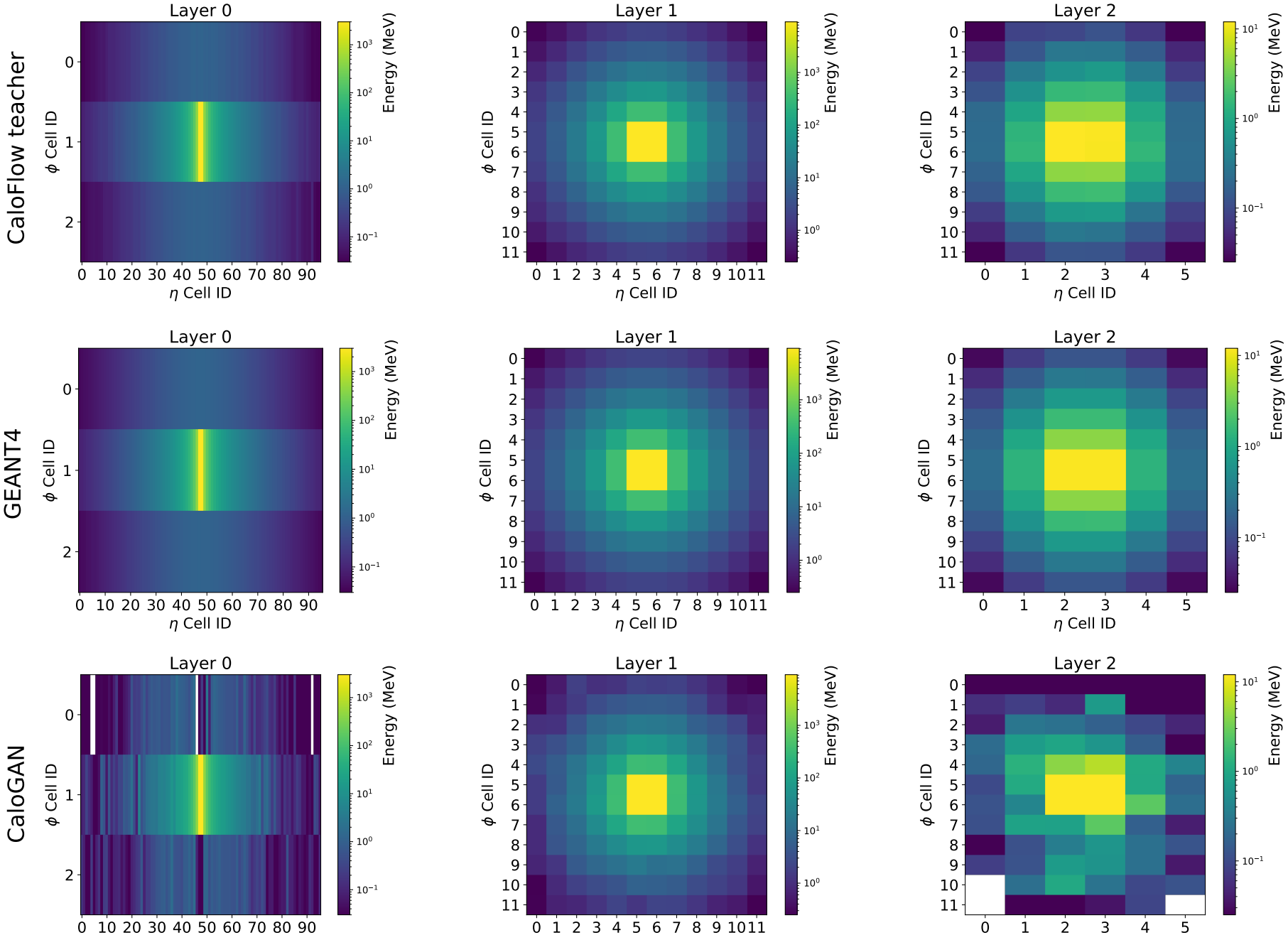
A classifier is the "ultimate metric".

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

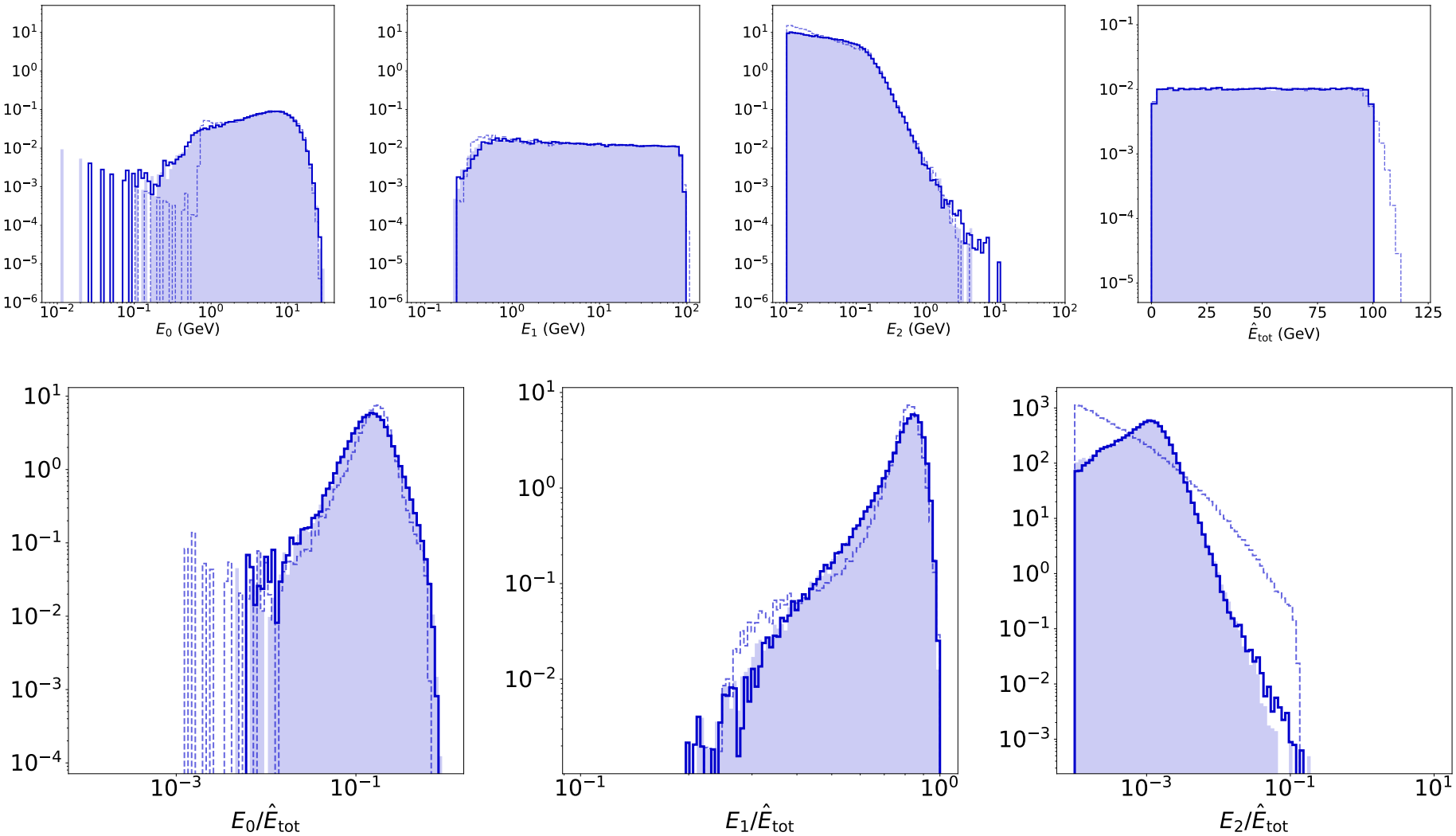
AUC / JSD		DNN		
		GEANT4 vs. CALOGAN	GEANT4 vs. CALOFLOW v1	GEANT4 vs. CALOFLOW v2
e^+	unnorm.	1.000(0) / 0.993(1)	0.847(8) / 0.345(12)	
	norm.	1.000(0) / 0.997(0)	0.869(2) / 0.376(4)	
γ	unnorm.	1.000(0) / 0.996(1)	0.660(6) / 0.067(4)	
	norm.	1.000(0) / 0.994(1)	0.794(4) / 0.213(7)	
π^+	unnorm.	1.000(0) / 0.988(1)	0.632(2) / 0.048(1)	
	norm.	1.000(0) / 0.997(0)	0.751(4) / 0.148(4)	

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

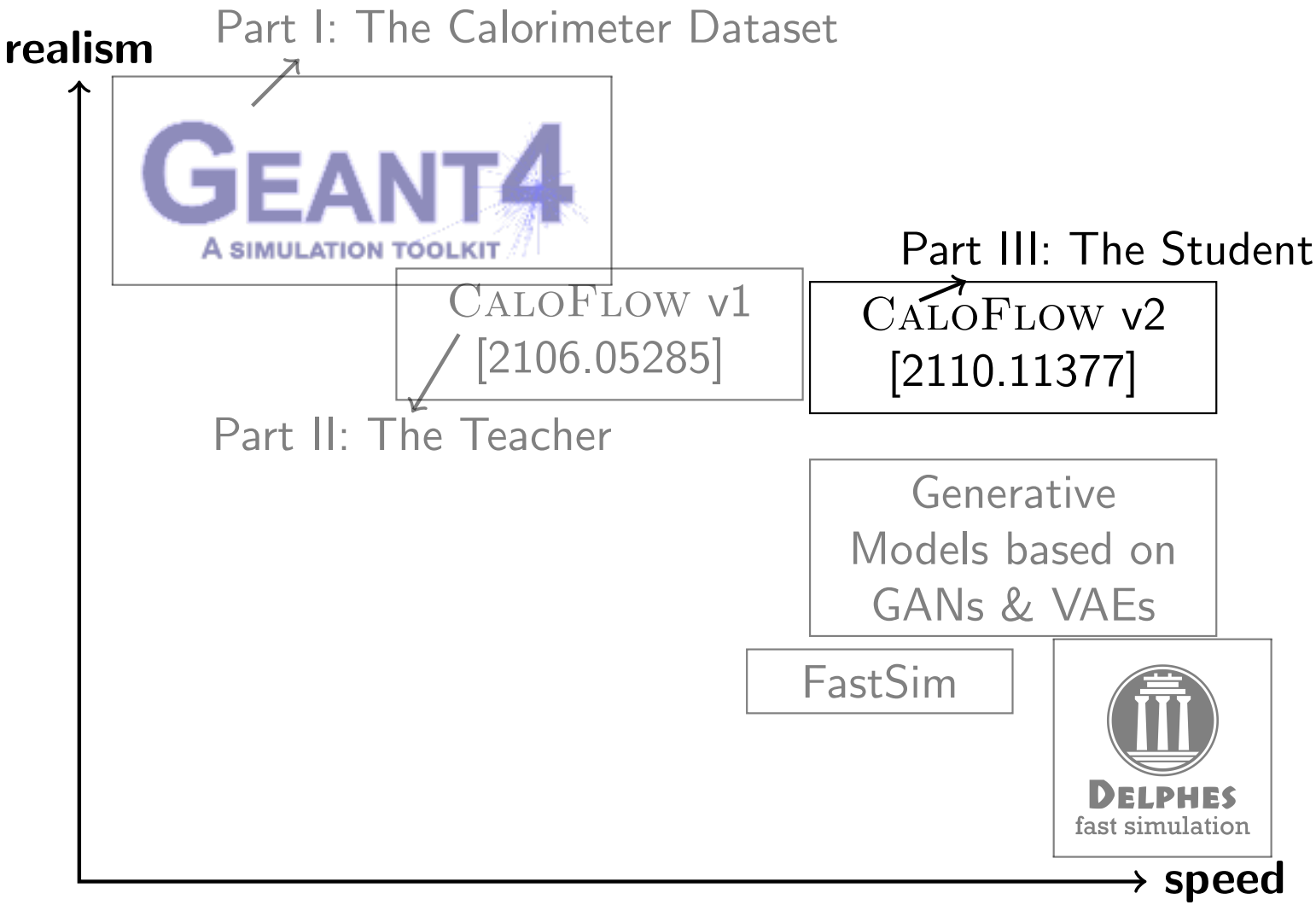
Comparing Shower Averages: e^+



CaloFlow v1 histograms: e^+



Normalizing Flows for Calorimeter Showers: When the Student beats the Teacher



CaloFlow uses a 2-step approach.

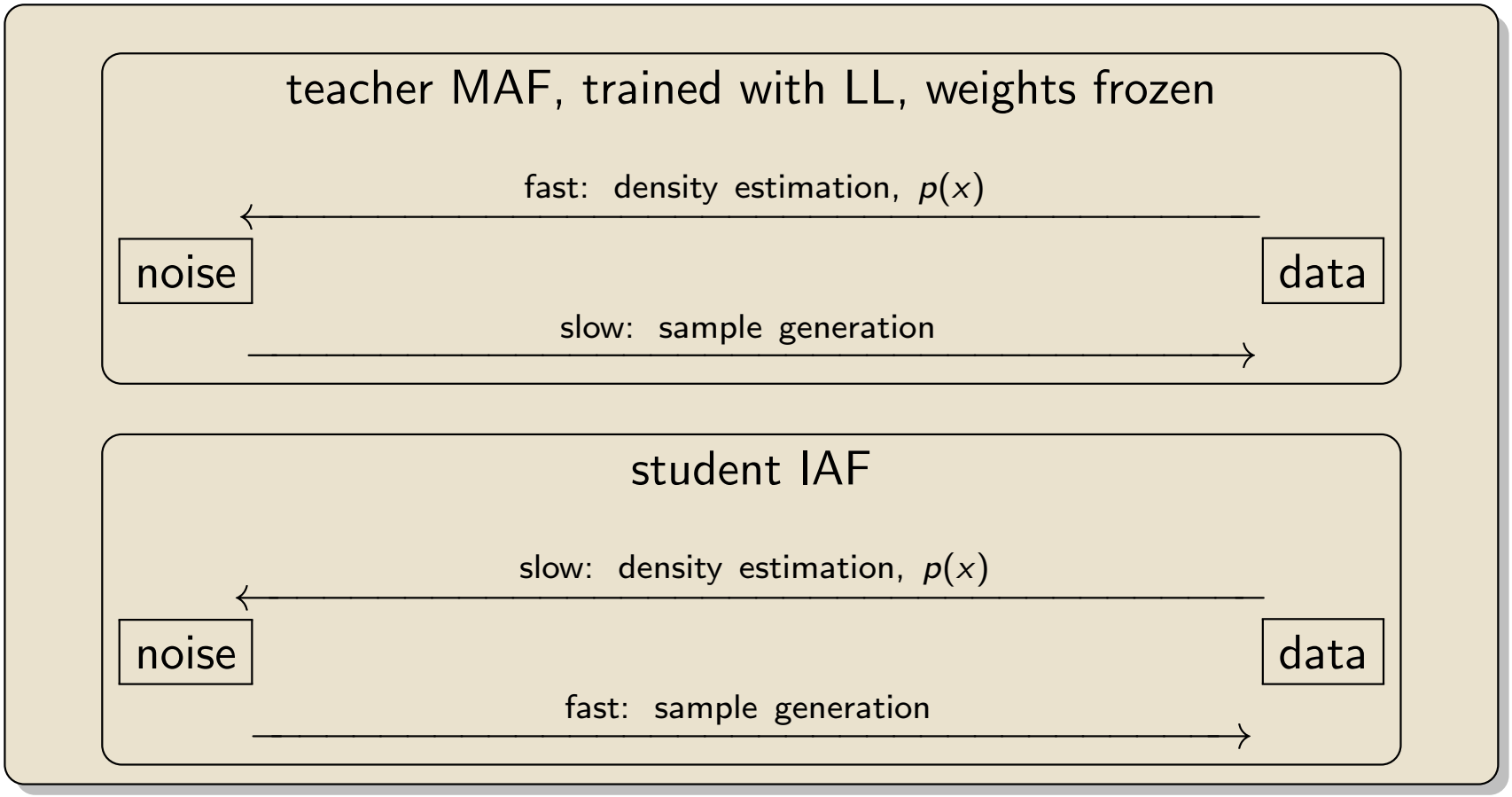
Flow I

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

Flow II

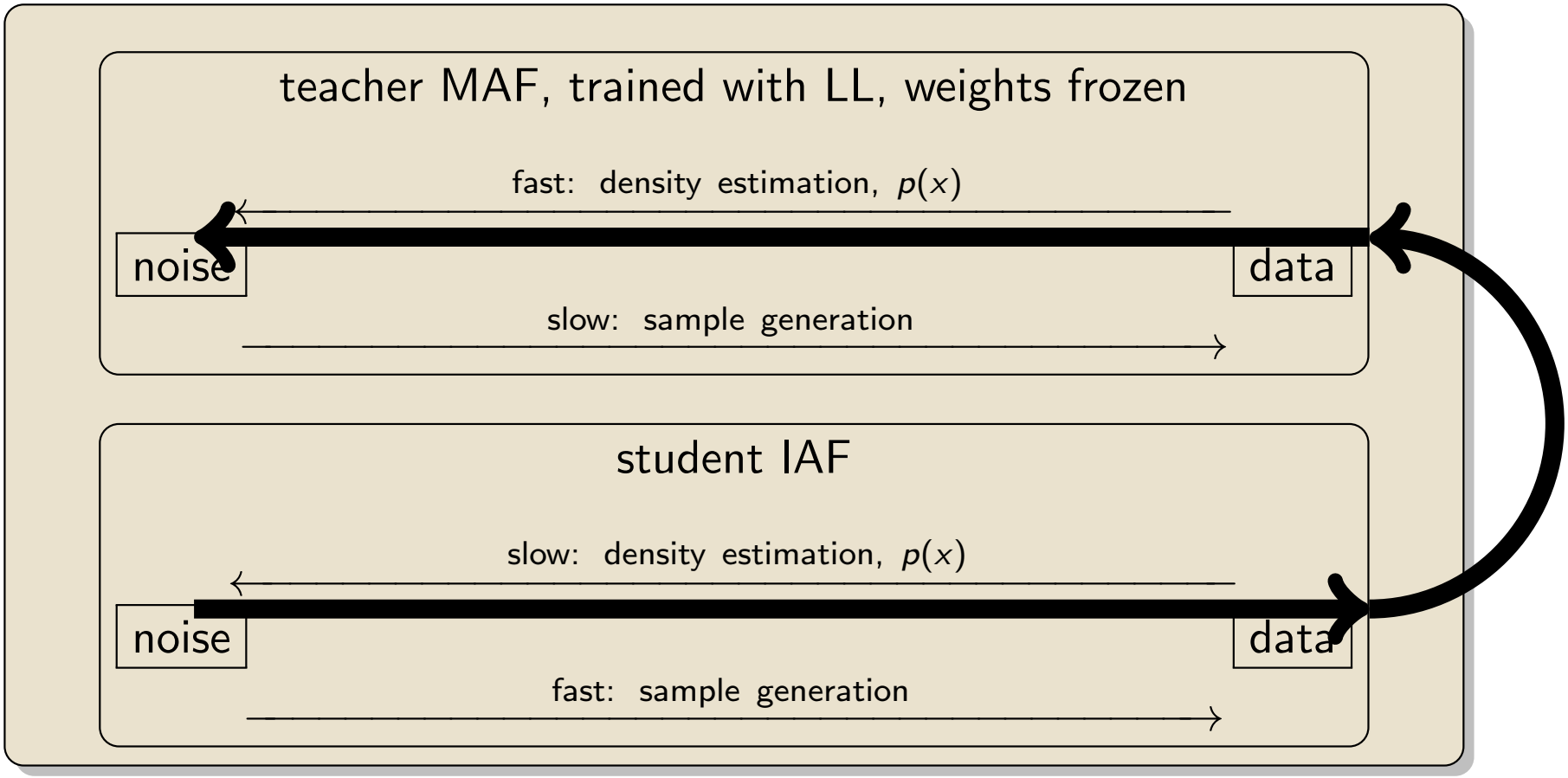
- learns $p_2(\vec{\mathcal{I}} | E_0, E_1, E_2, E_{\text{tot}})$ of normalized showers
- in CaloFlow v1 (2106.05285 — called "teacher"):
  - MAF trained with LL
  - Slow in sampling ($\approx 500\times$ slower than CaloGAN)
- in CaloFlow v2 (2110.11377 — called "student"):
  - IAF trained with Probability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

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



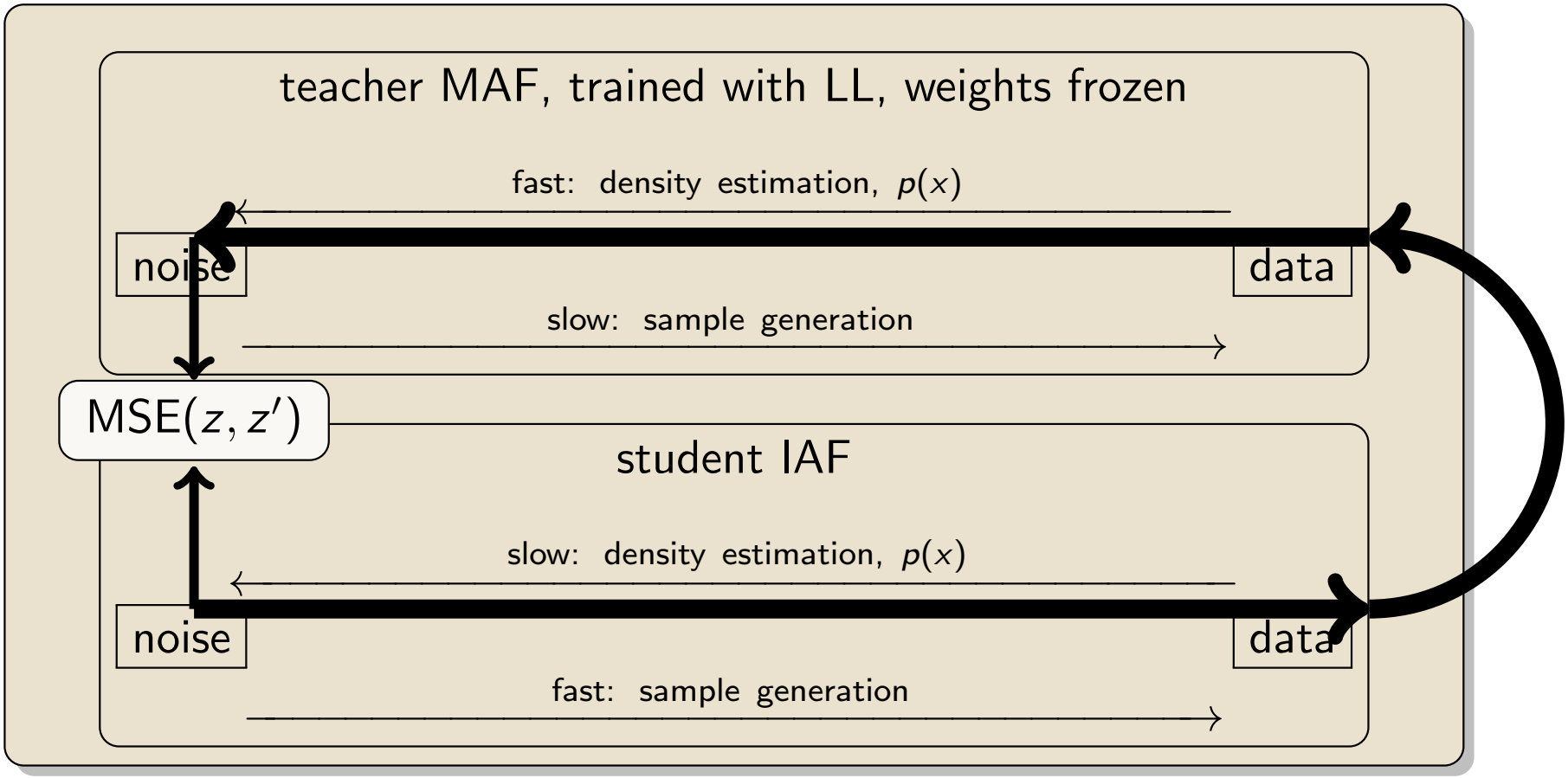
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

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



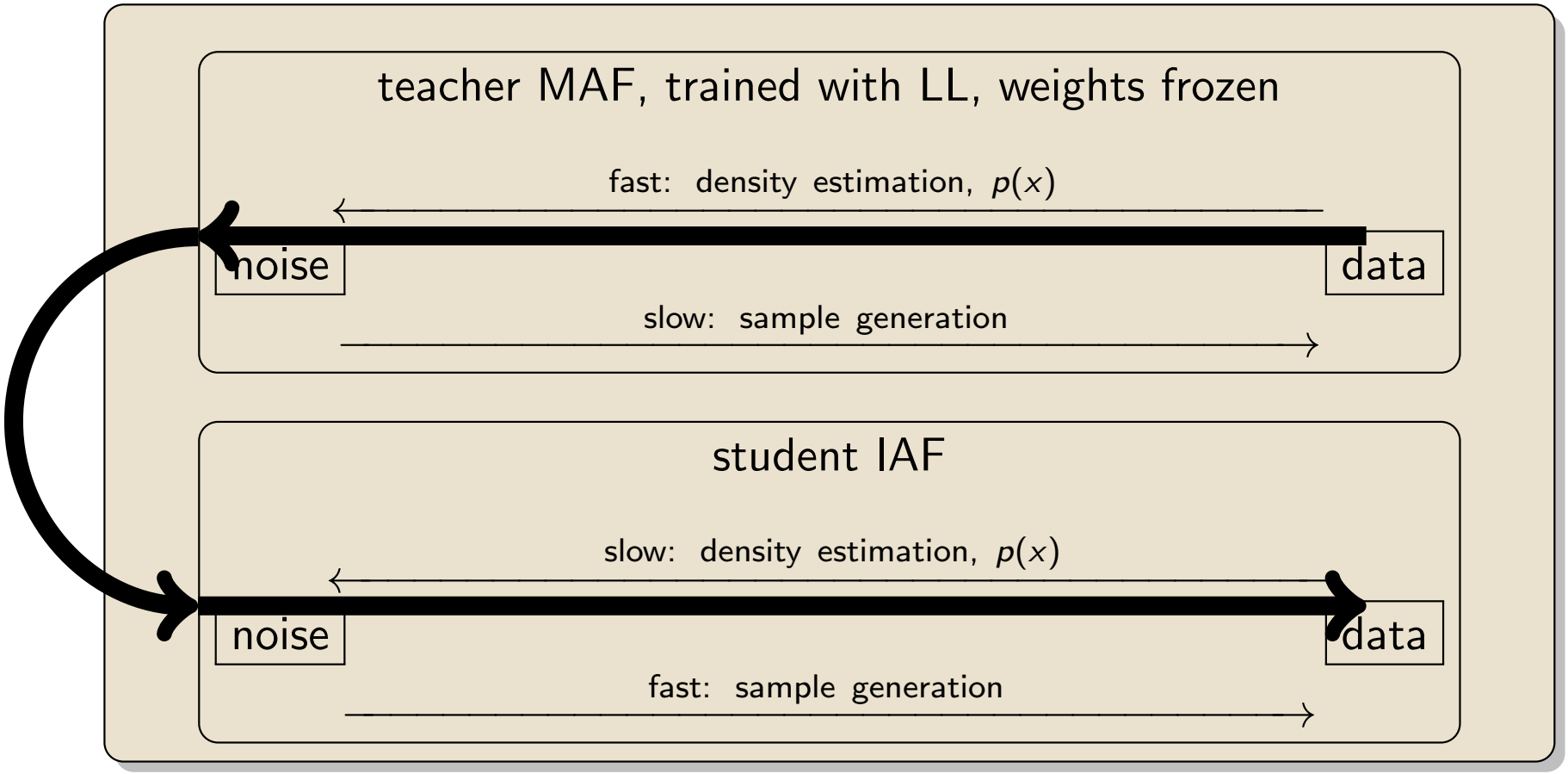
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

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



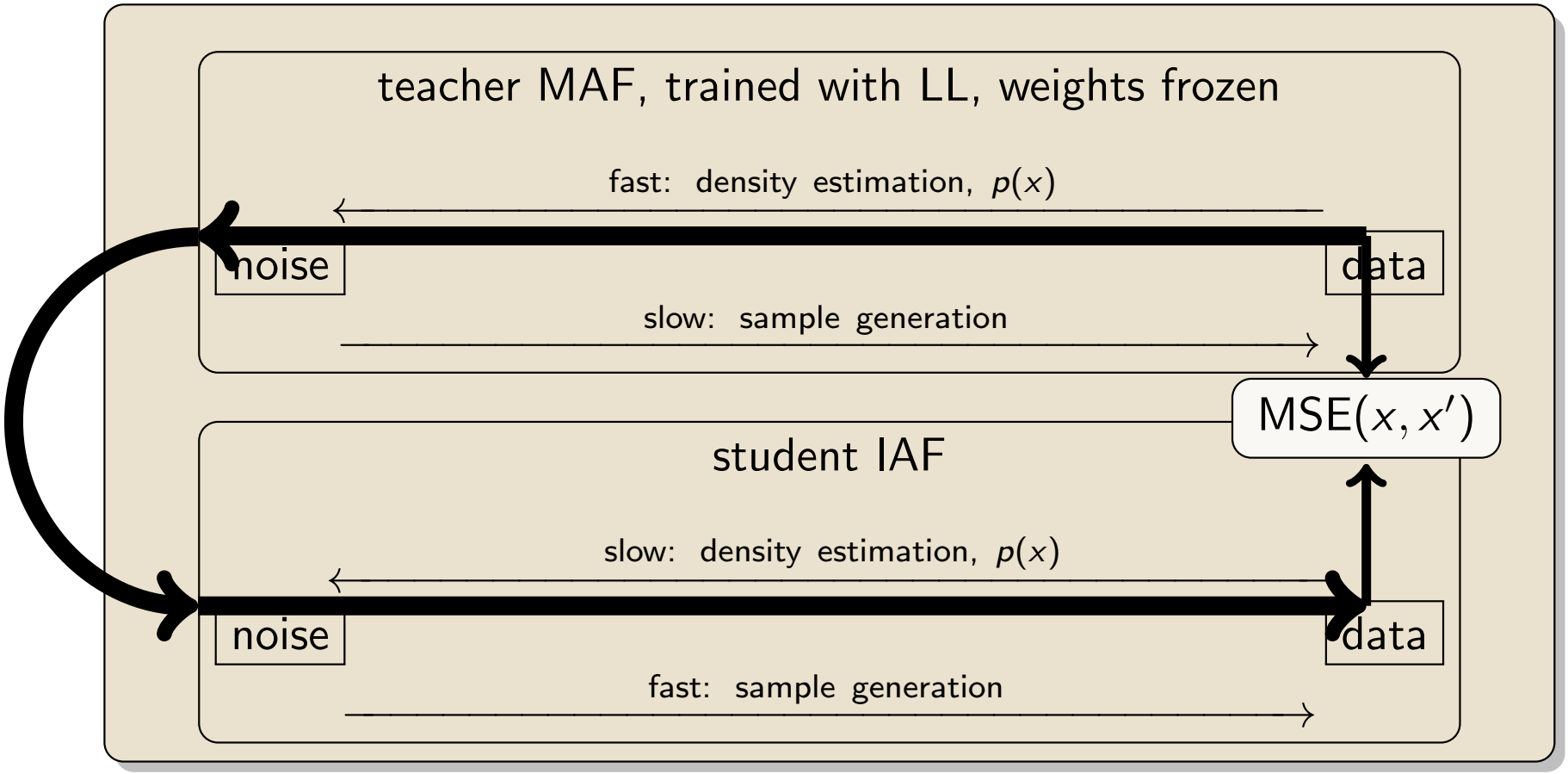
$$\text{Loss} = \mathrm{MSE}(z, z') + \mathrm{MSE}(x, x') + \mathrm{MSE}(z_i, z_i') + \mathrm{MSE}(x_i, x_i') + \mathrm{MSE}(p_z, p_z') + \mathrm{MSE}(p_x, p_x')$$

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



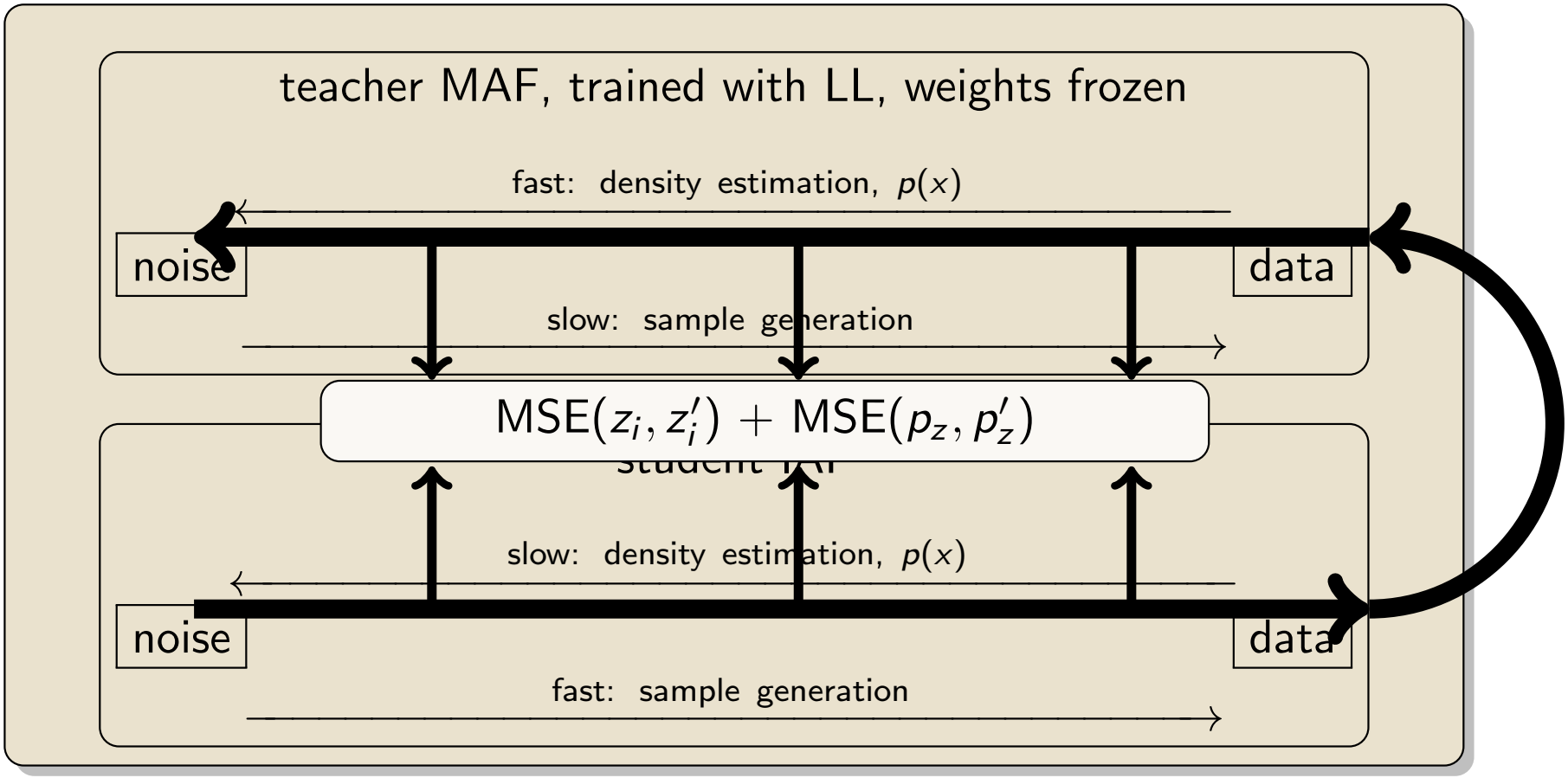
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

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



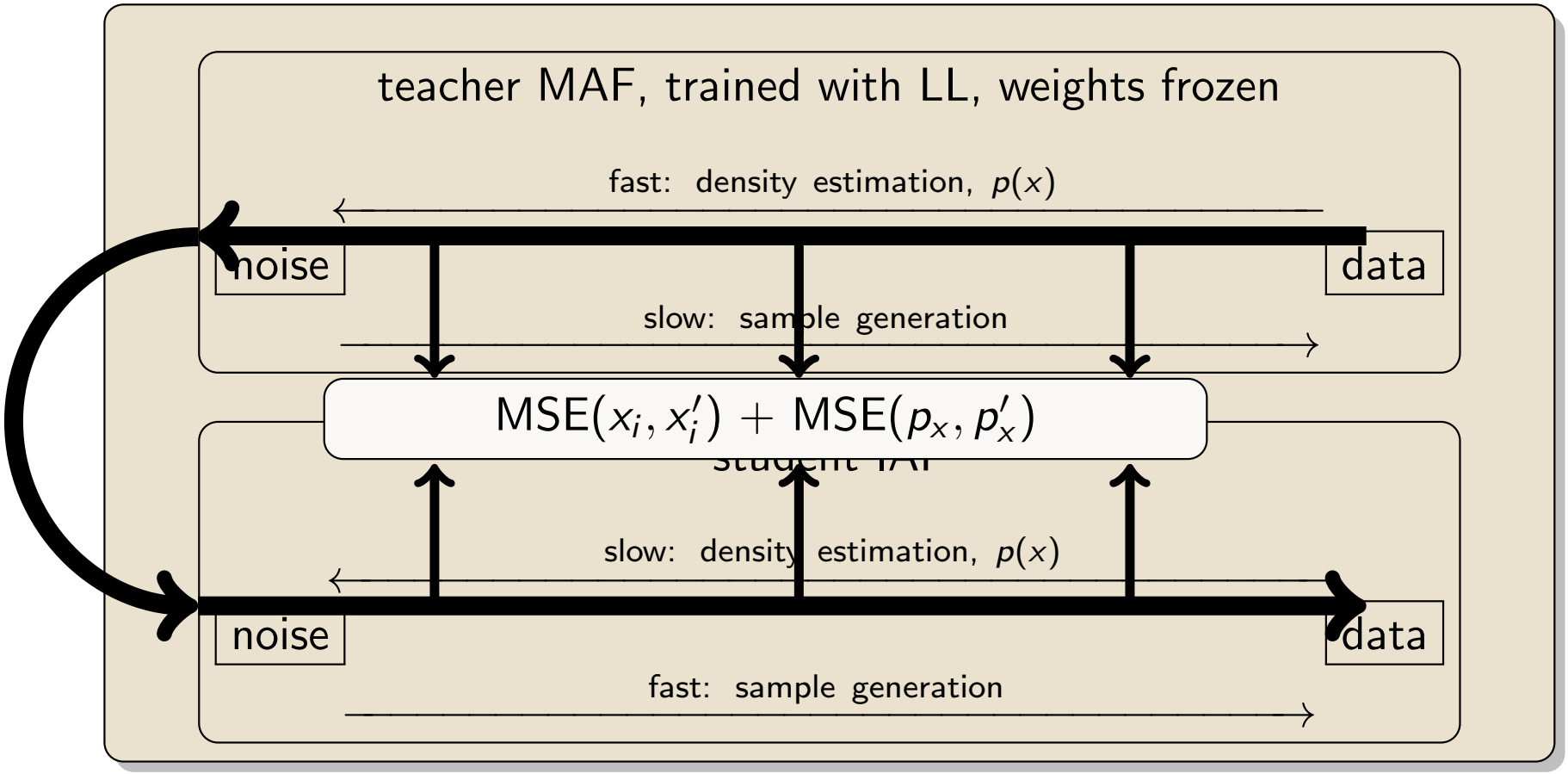
$$\text{Loss} = \mathrm{MSE}(z, z') + \mathrm{MSE}(x, x') + \mathrm{MSE}(z_i, z_i') + \mathrm{MSE}(x_i, x_i') + \mathrm{MSE}(p_z, p_z') + \mathrm{MSE}(p_x, p_x')$$

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



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

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



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

The Student also passes the "ultimate metric" test.

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		
		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)

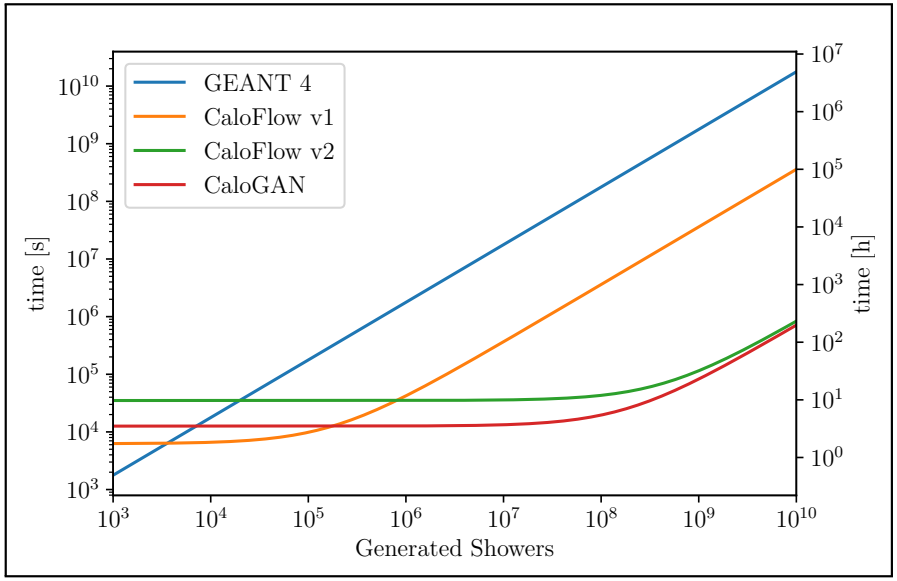
AUC ($\in [0.5, 1]$): Area Under the ROC Curve
JSD ($\in [0, 1]$): Jensen-Shannon divergence based on the binary cross entropy

Sampling Speed: The Student beats the Teacher!

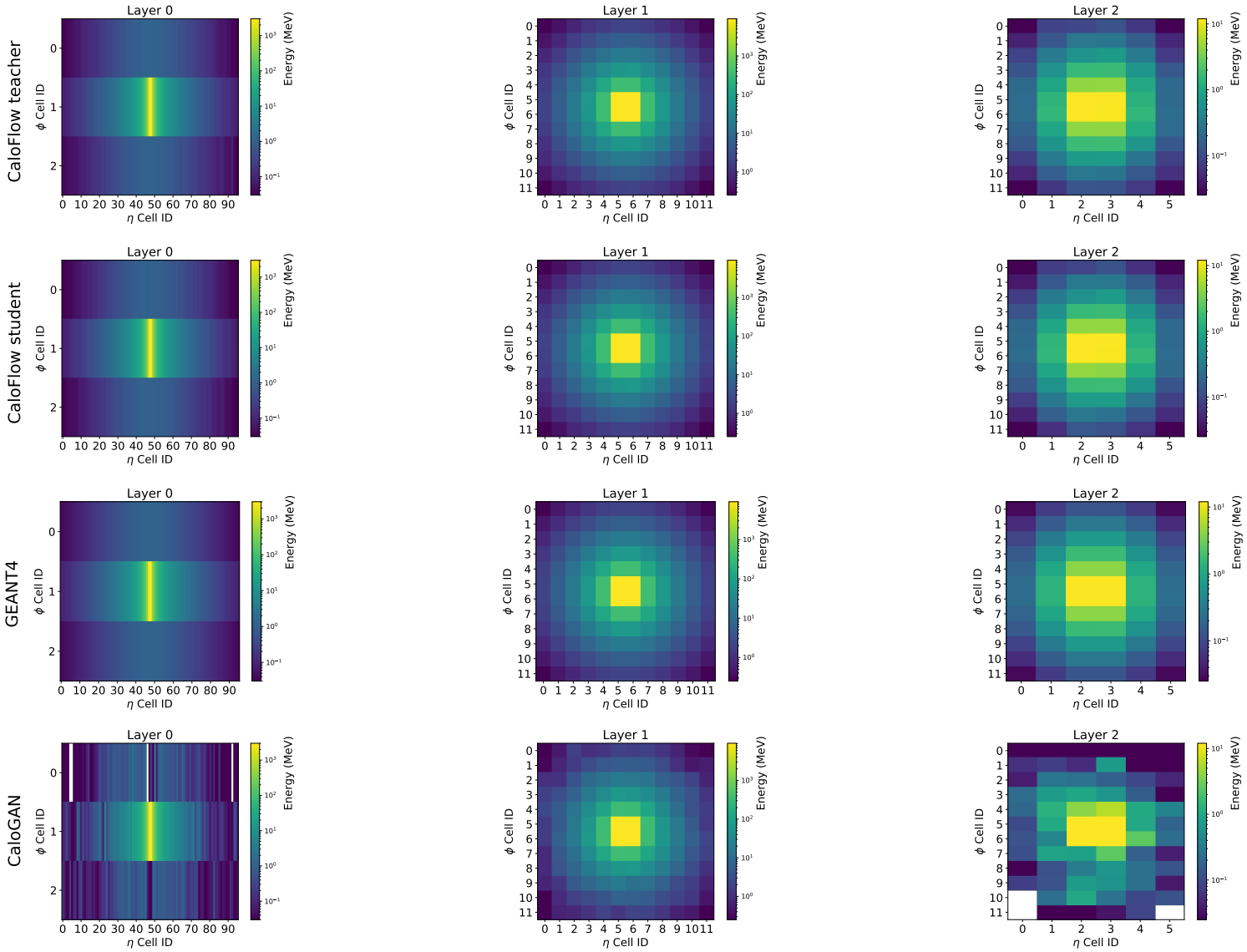
	CaloFlow*		CaloGAN*		Geant4†
	teacher	student			
training	22+82 min	+ 480 min	210 min		0 min
generation batch size	time per shower		batch size req.	100k req.	
10	835 ms	5.81 ms	455 ms	2.2 ms	1772 ms
100	96.1 ms	0.60 ms	45.5 ms	0.3 ms	1772 ms
1000	41.4 ms	0.12 ms	4.6 ms	0.08 ms	1772 ms
10000	36.2 ms	**0.08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

*: on our Titan V GPU

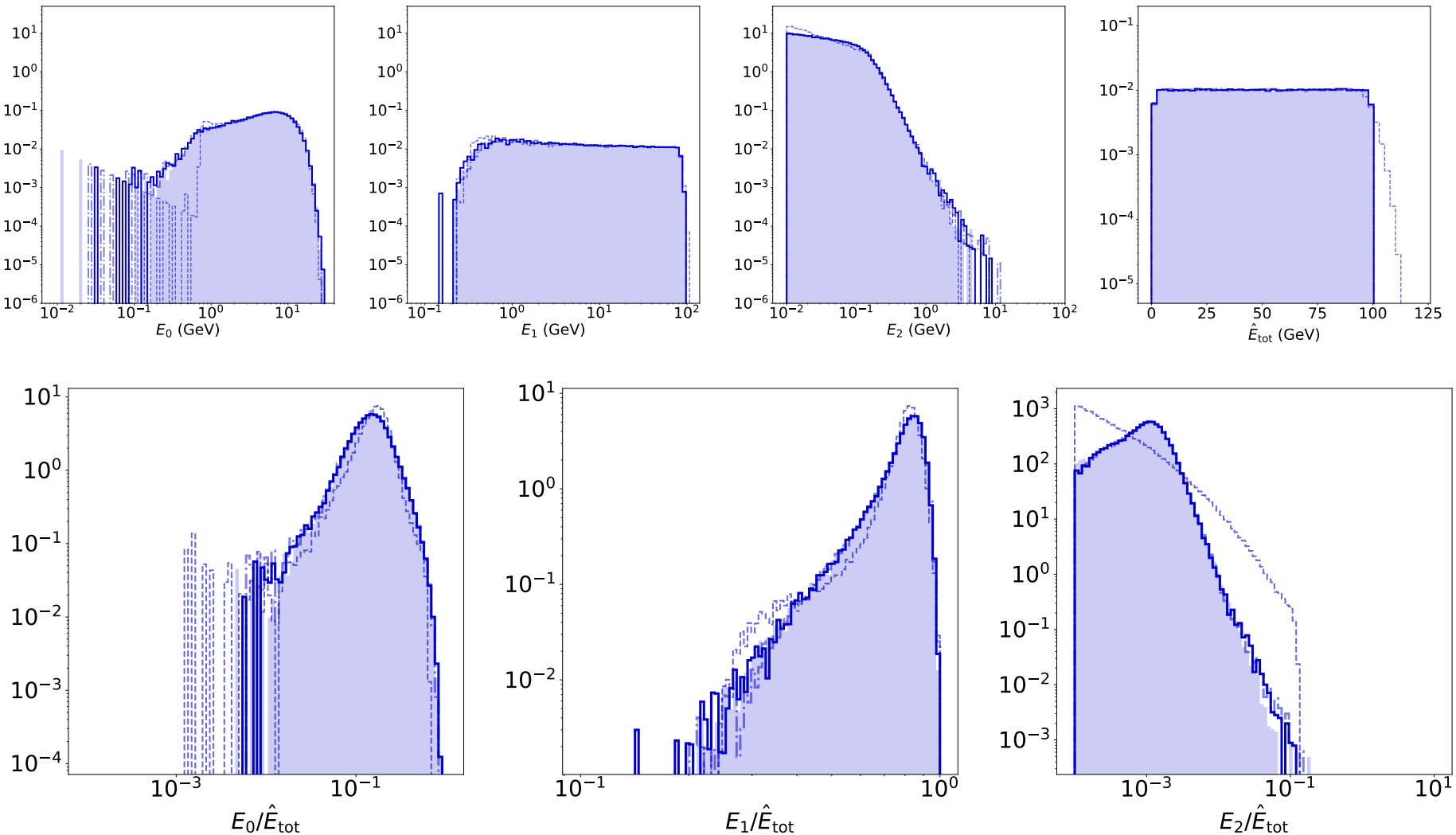
†: on the CPU of CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD]



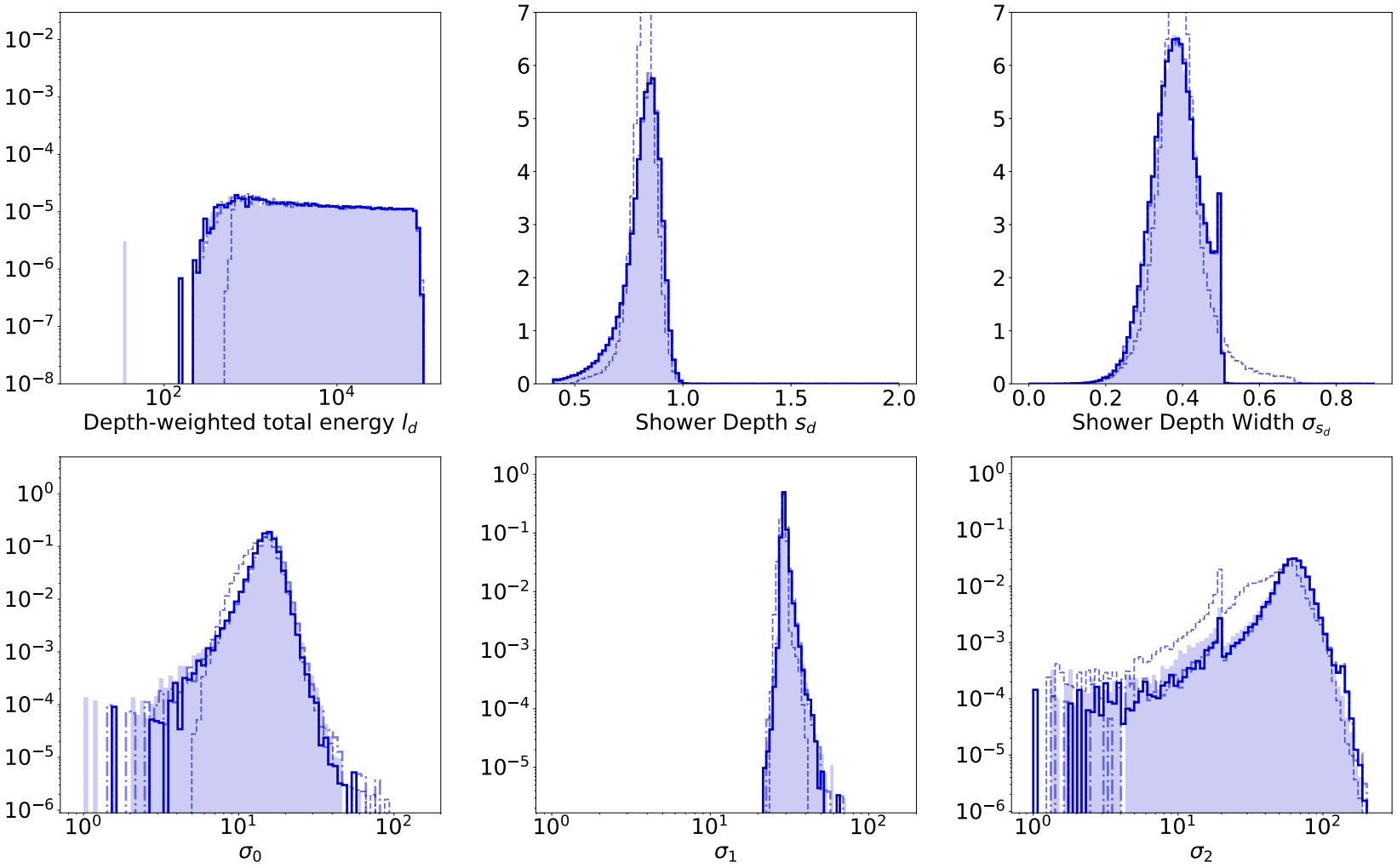
CaloFlow: Comparing Shower Averages: e^+



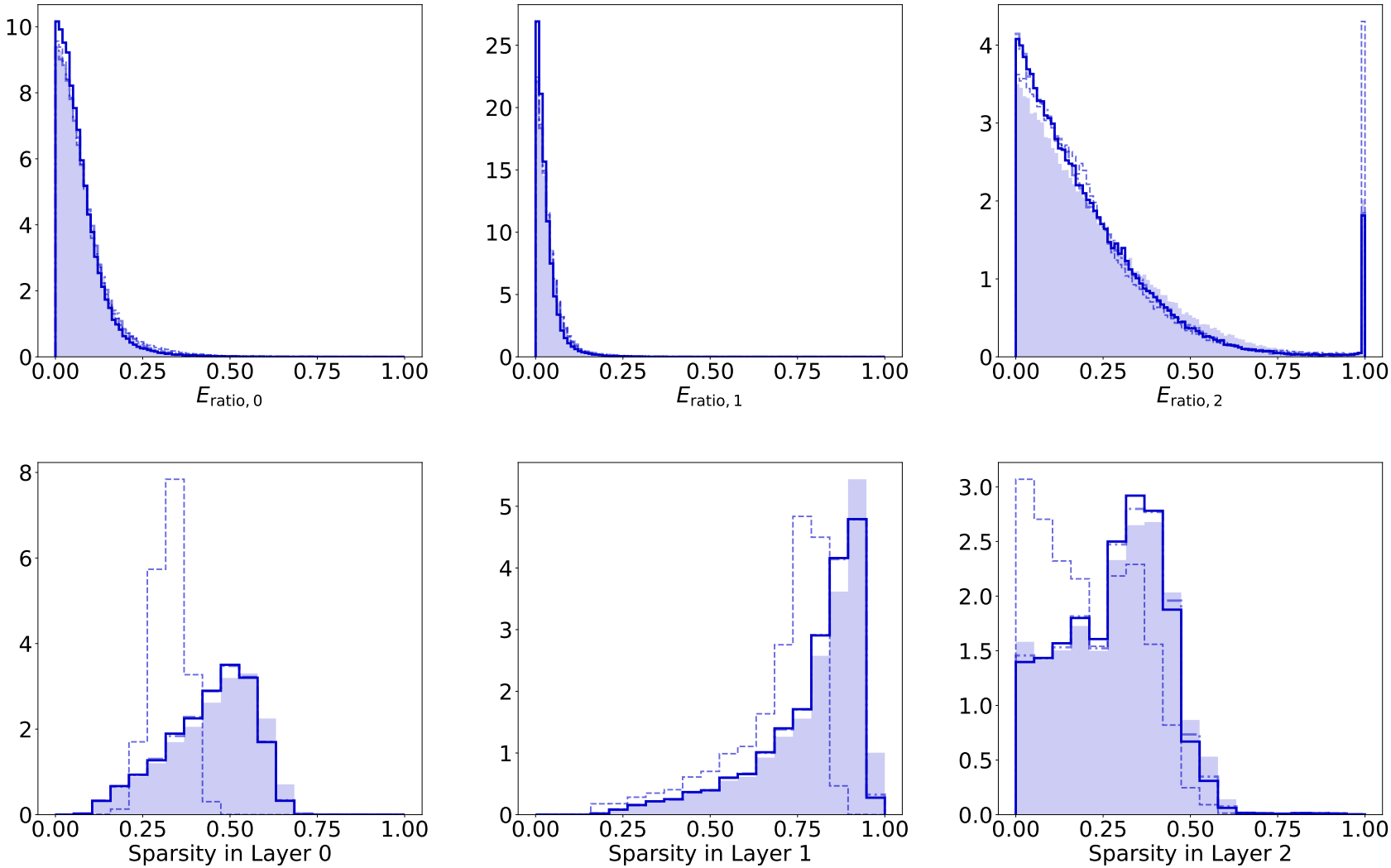
CaloFlow: Flow I histograms: e^+



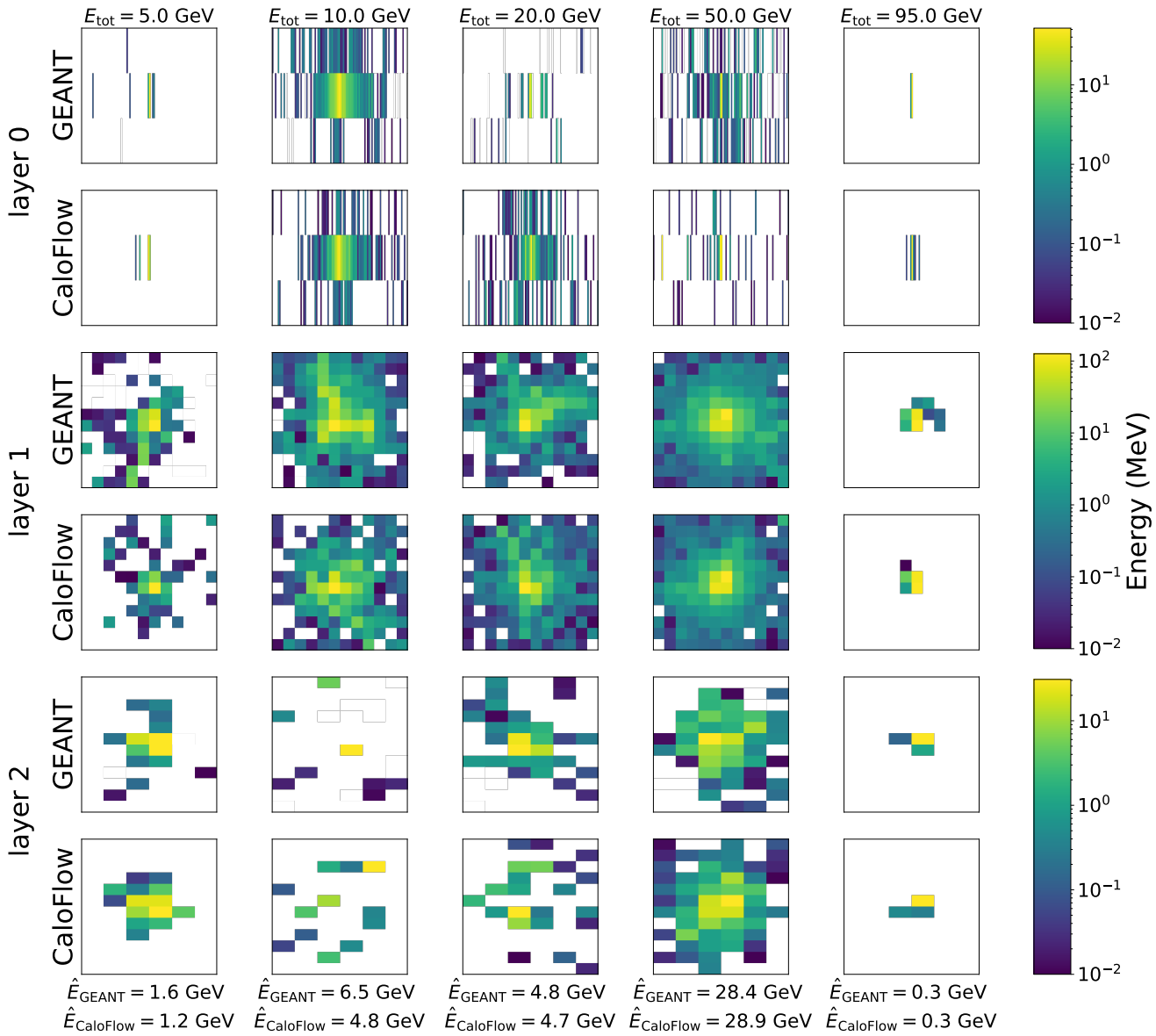
CaloFlow: Flow I+II histograms: e^+



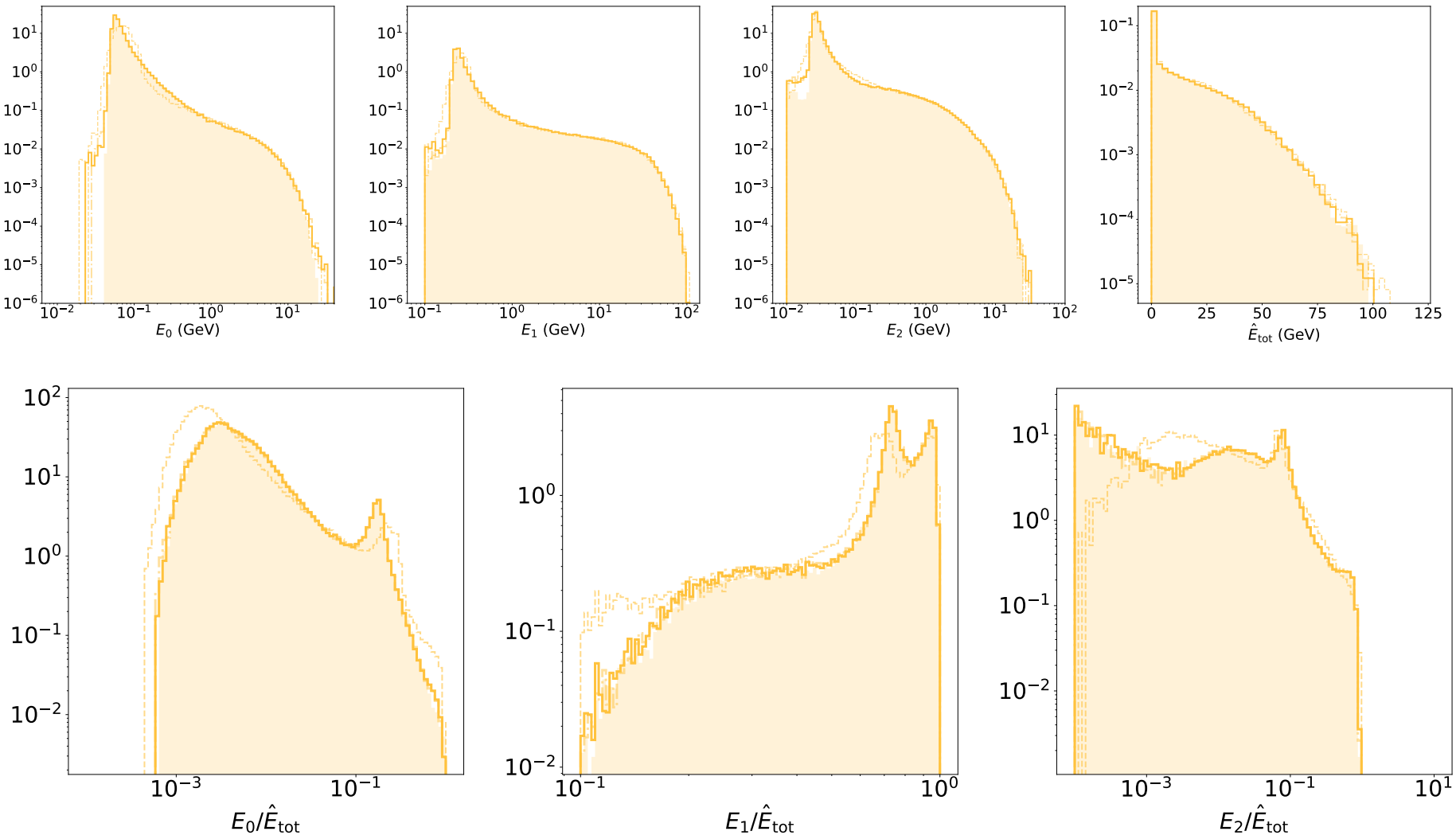
CaloFlow: Flow II histograms: e^+



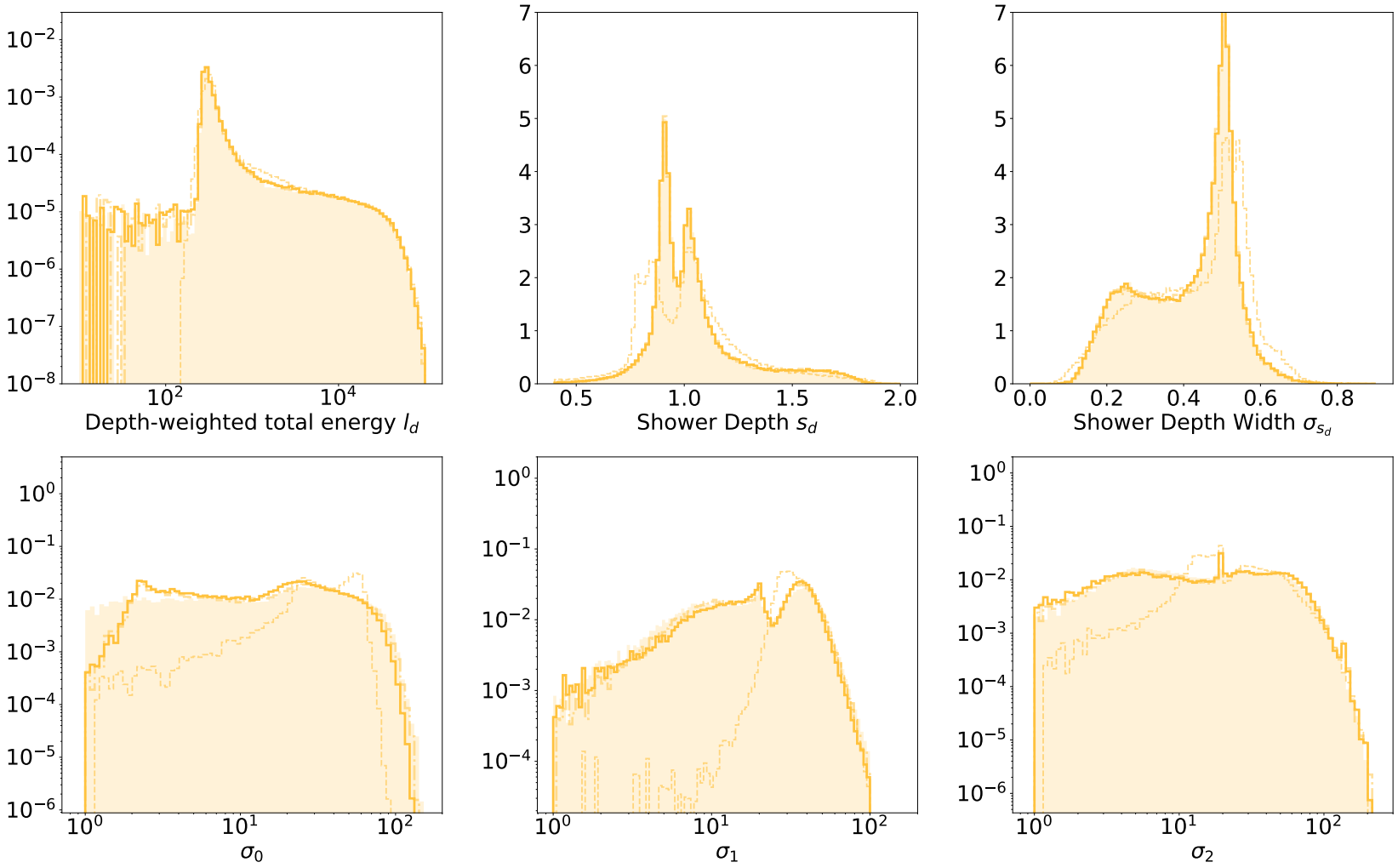
CaloFlow: Nearest Neighbors: π^+ (student)



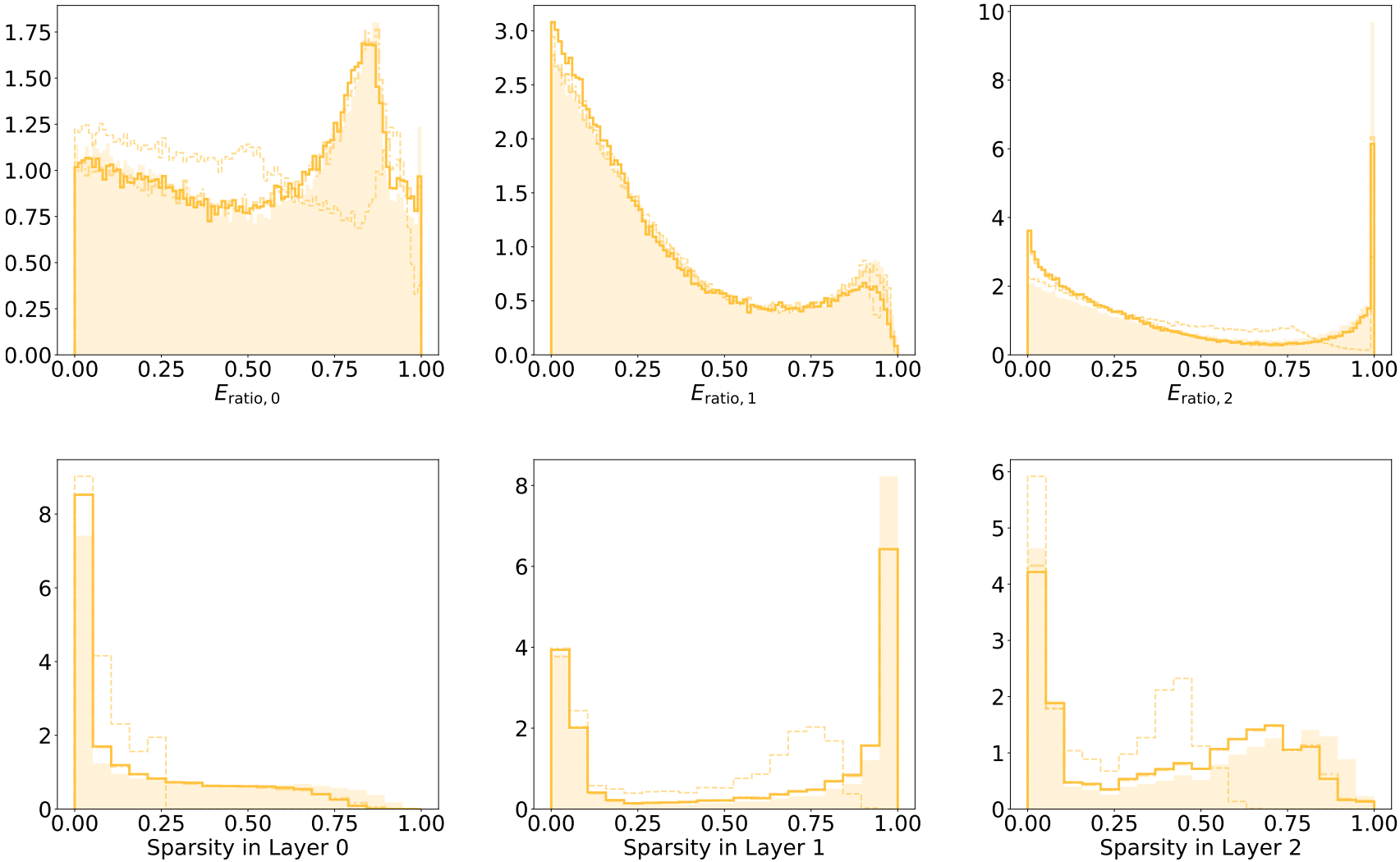
CaloFlow: Flow I histograms: π^+



CaloFlow: Flow I+II histograms: π^+



CaloFlow: Flow II histograms: π^+



CaloFlow: Fast and Accurate Generation of Calorimeter Showers with Normalizing Flows

- We use the same calorimeter and GEANT4 setup as the original CaloGAN.
- These are 504-dim. showers of e^+, γ, and π^+

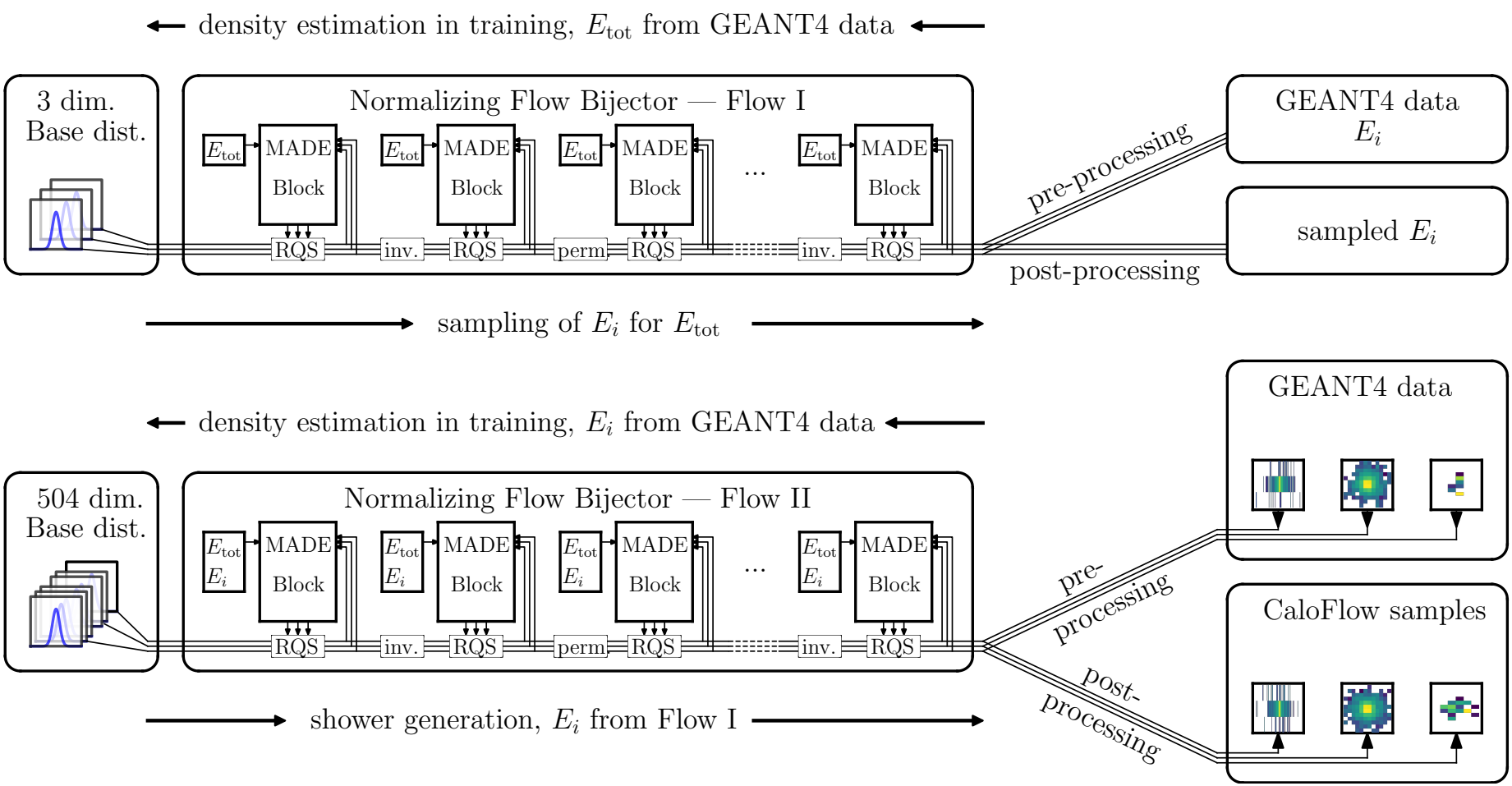
$\Rightarrow$ First time application of Normalizing Flows!

- We use a 2-step setup to ensure energy conservation.
- Having $\log p(x)$ allows stable training and straightforward model selection for the MAF.
- The "ultimate test" based on a classifier, and histograms show impressive results.

- Probability Distillation allows us to train an IAF and gives a speed-up of $\mathcal{O}(500)$!
- The samples still pass the "ultimate test".

Backup

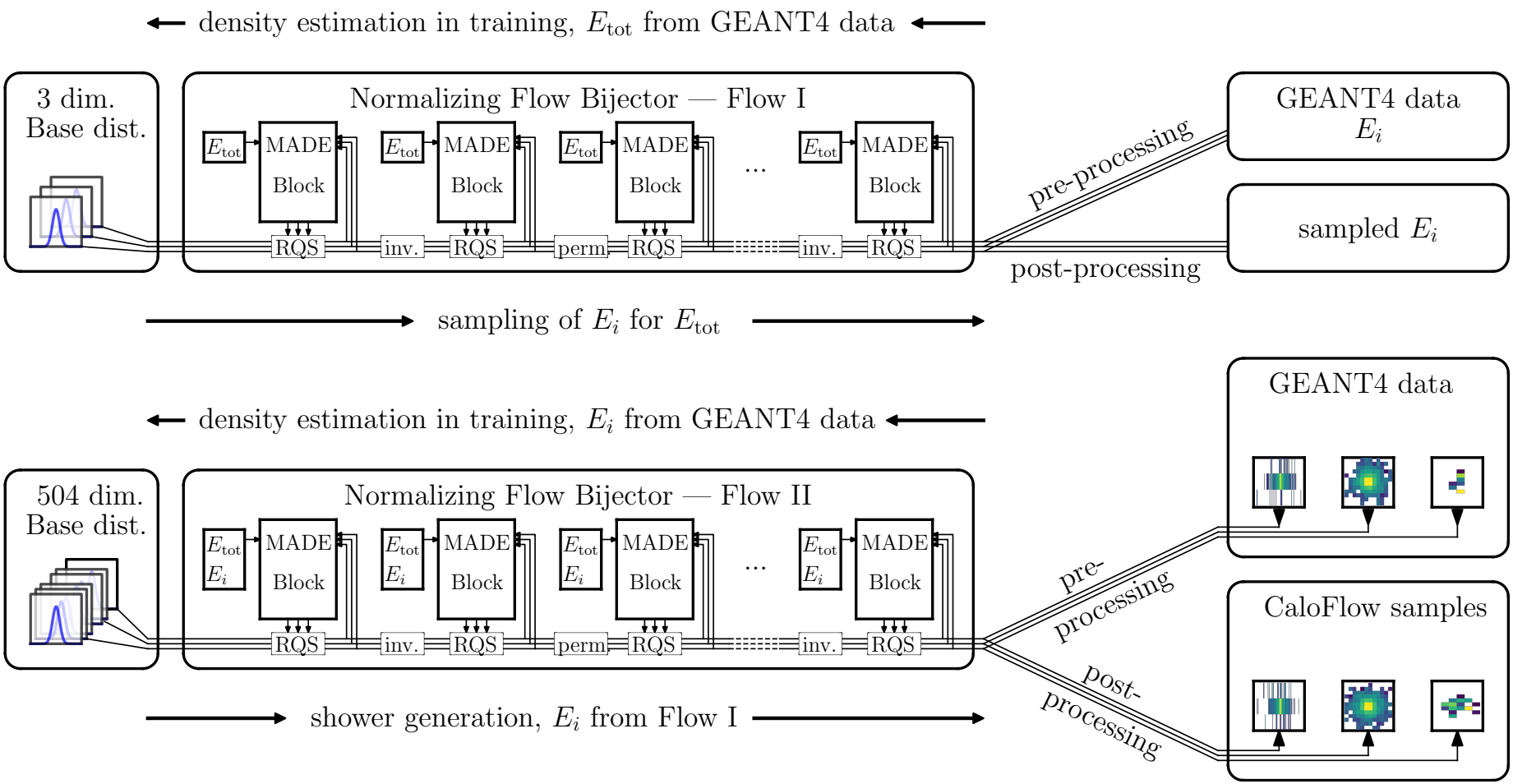
CaloFlow uses a 2-step approach.



Data processing Flow I

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

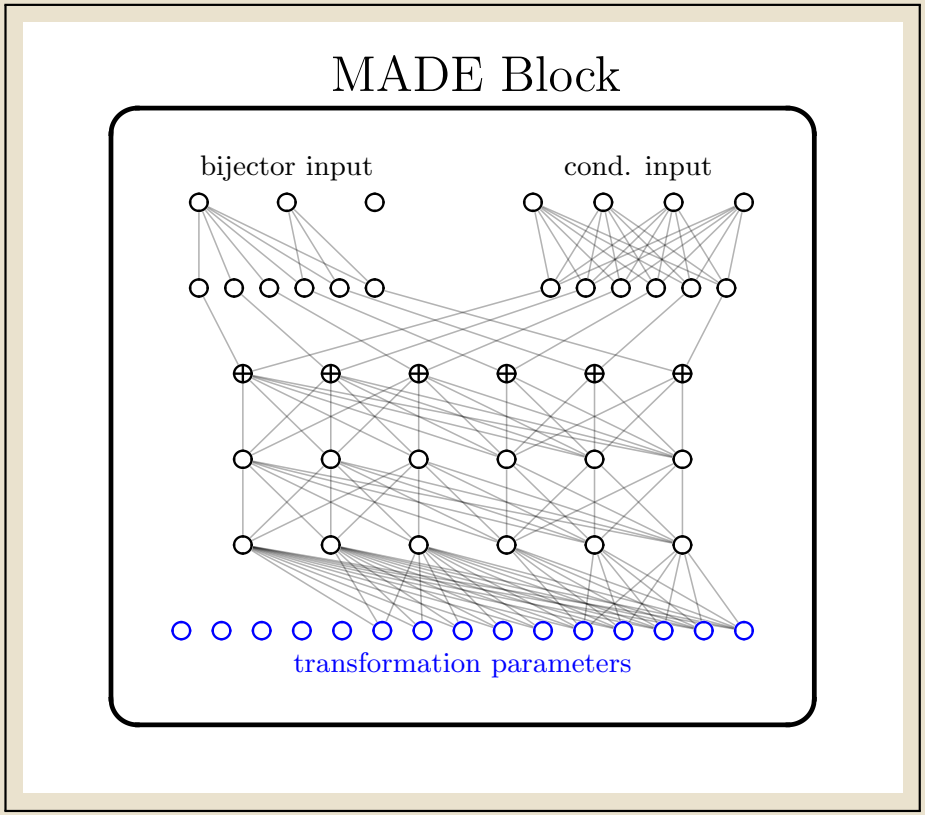
CaloFlow uses a 2-step approach.



Data processing Flow II

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

Ensuring the Autoregressive Property.



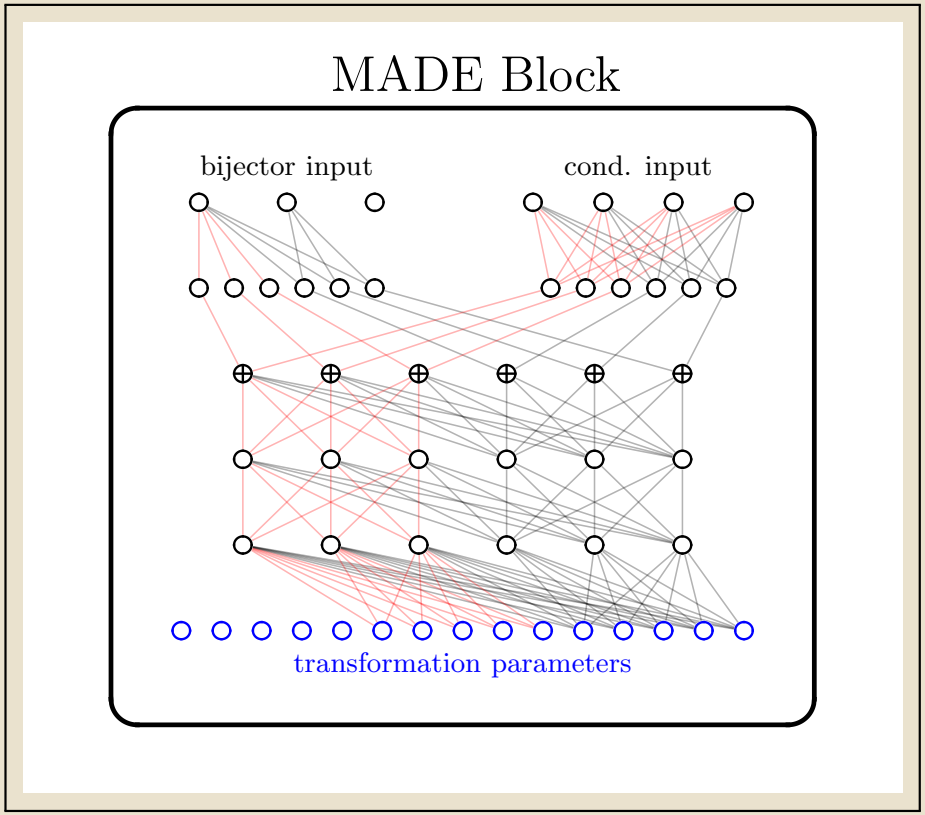
Implementation via masking:

- 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.



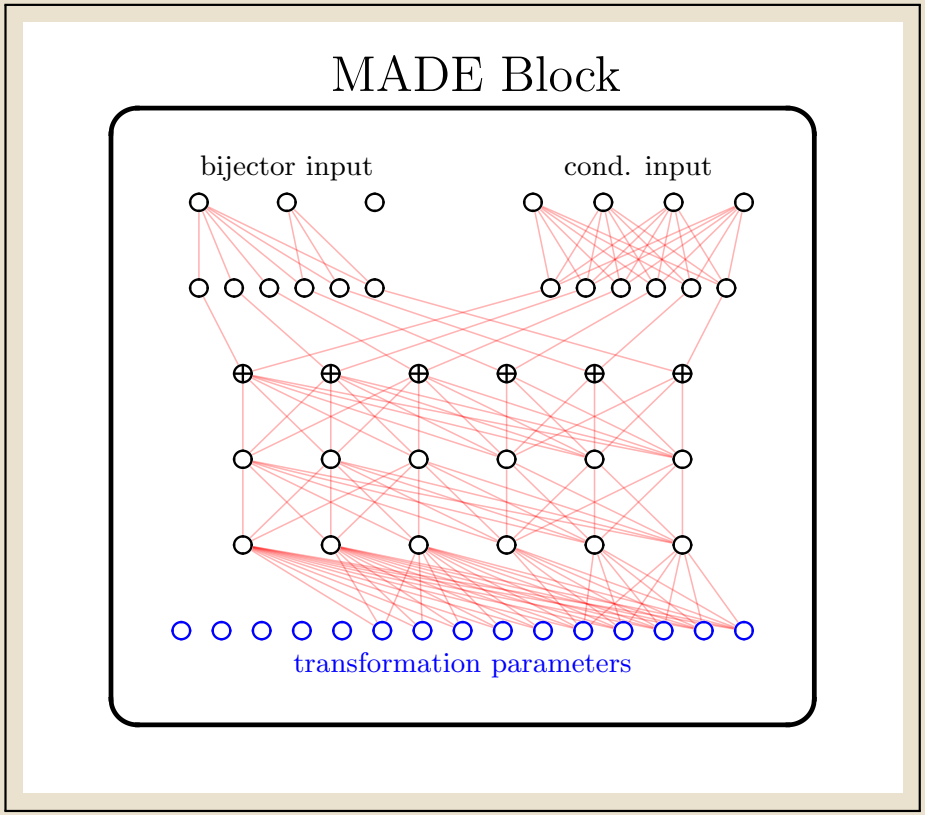
Implementation via masking:

- 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.

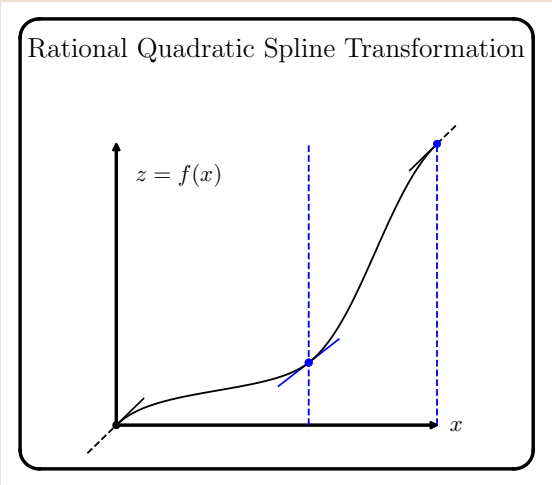
The Coupling Function defines the bijection.

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]

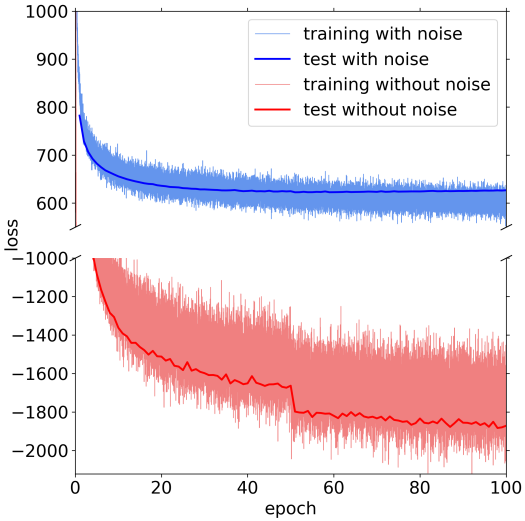


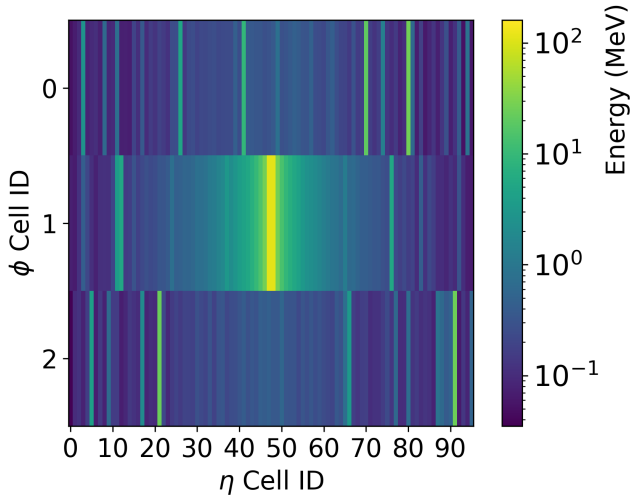
$$C = \frac{a_2\alpha^2 + a_1\alpha + a_0}{b_2\alpha^2 + b_1\alpha + b_0}$$

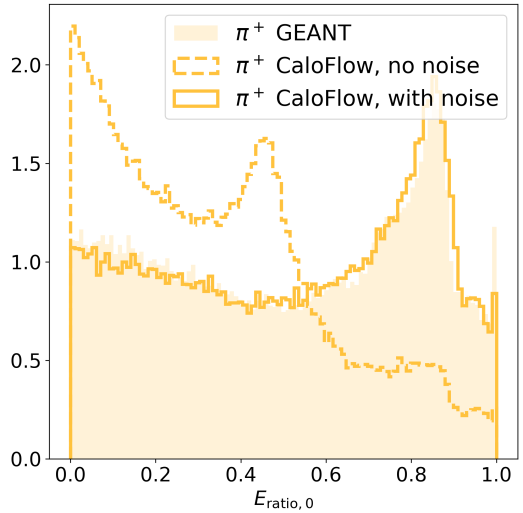
- still rather easy
- more flexible

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

Adding Noise is important for the sampling quality.

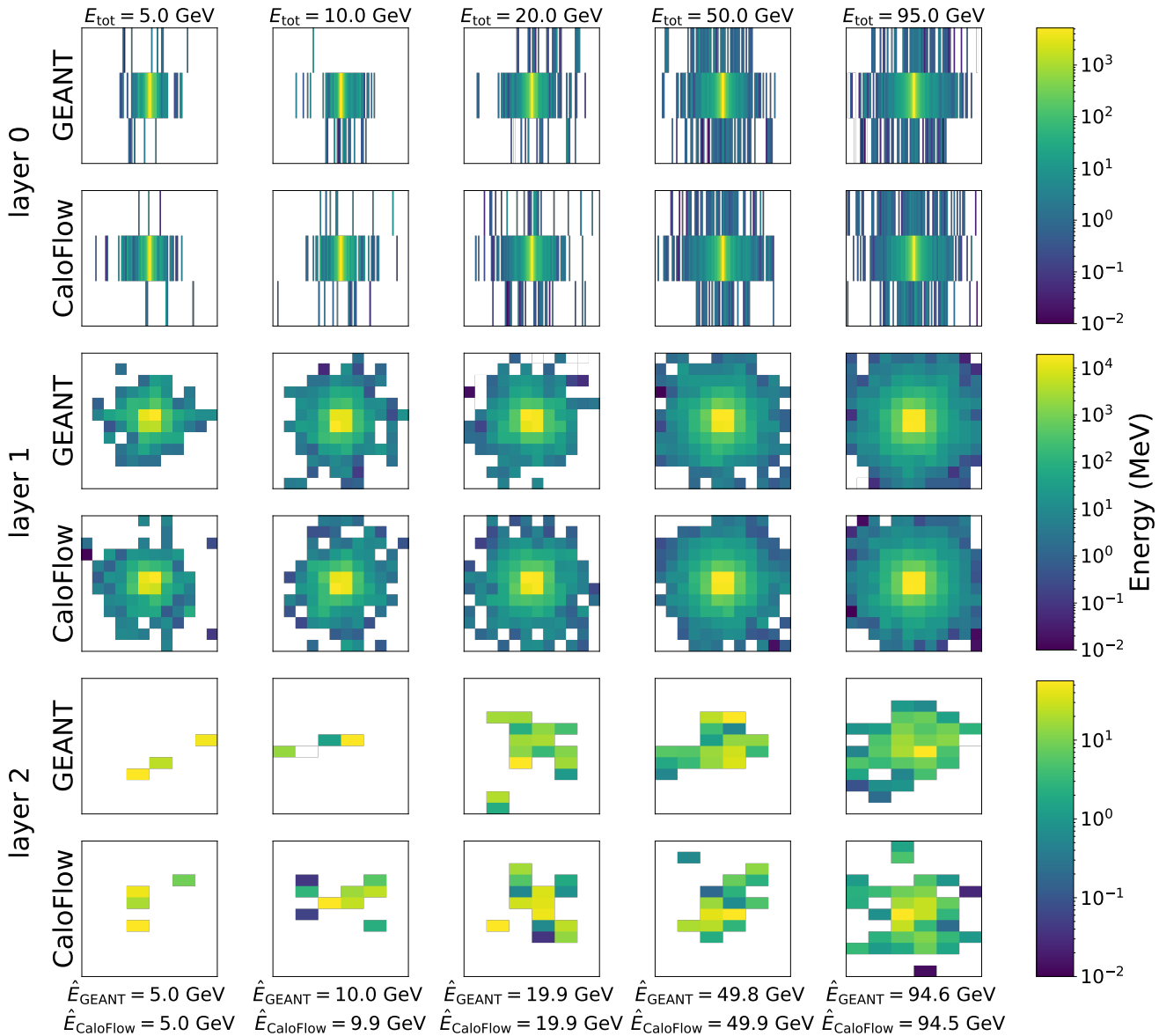




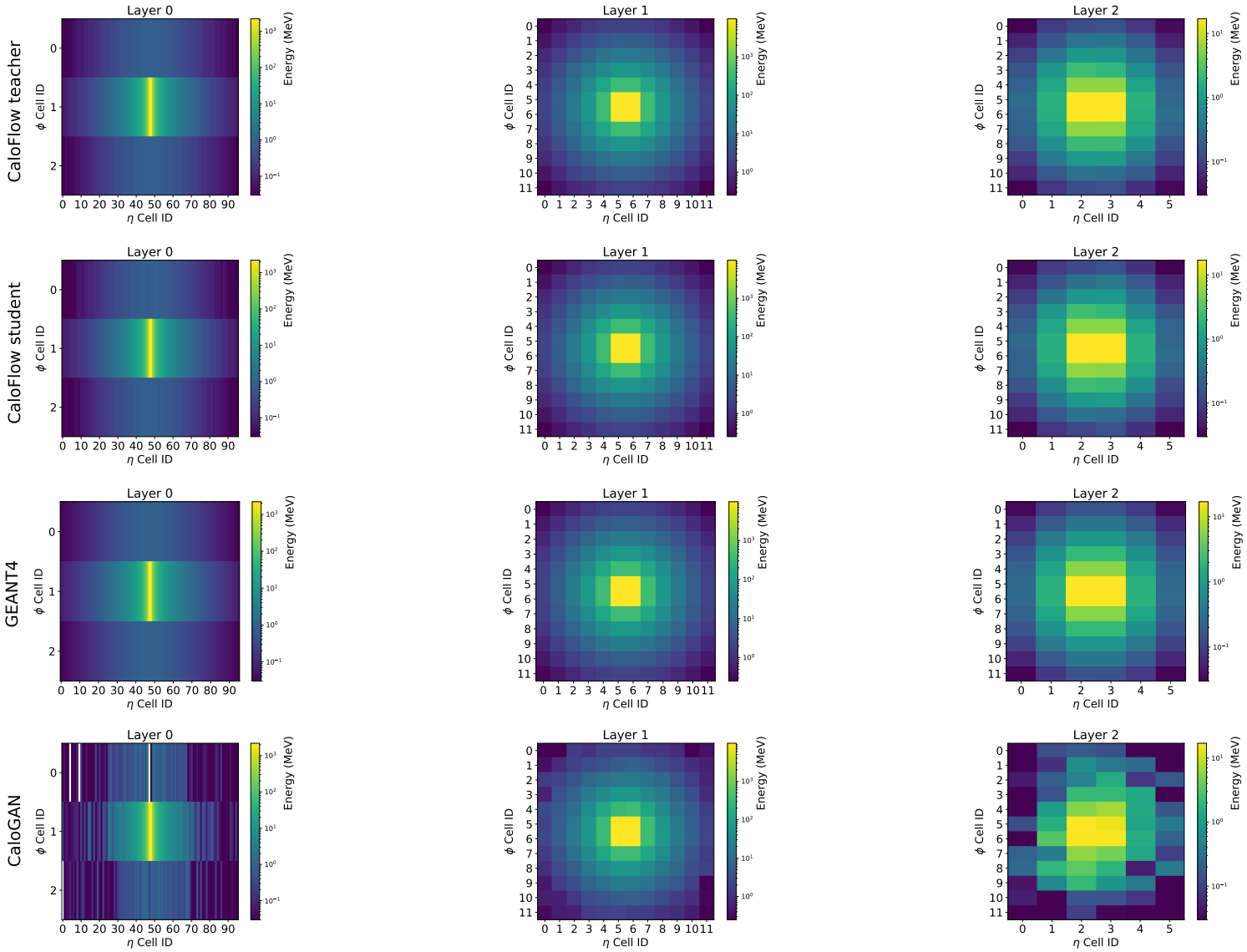


- The log-likelihood is less noisy, but smaller. Yet, the quality of the samples is much better!
- This is due to a "wider" mapping of space and less overfitting.

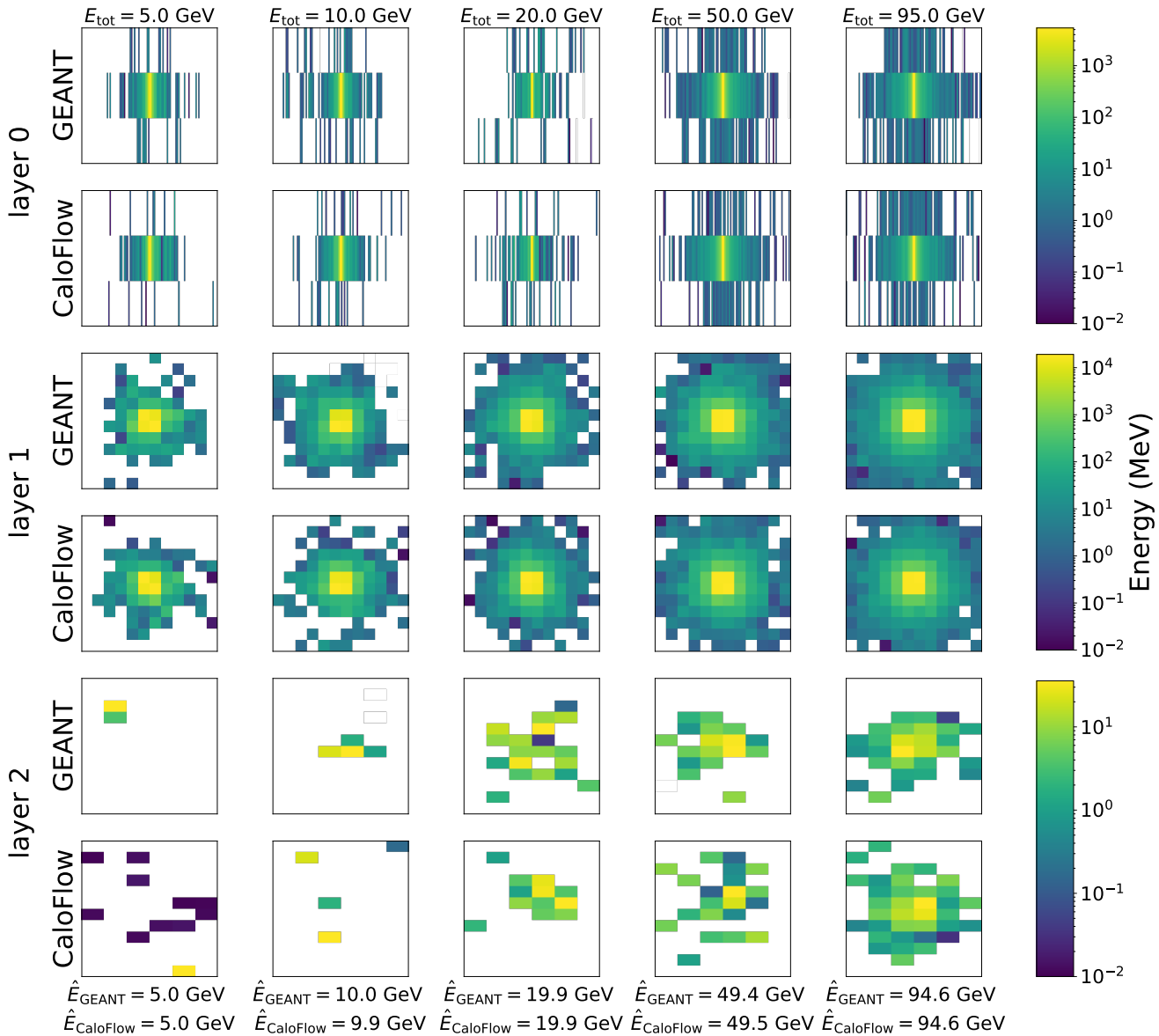
Nearest Neighbors: e^+ (student)



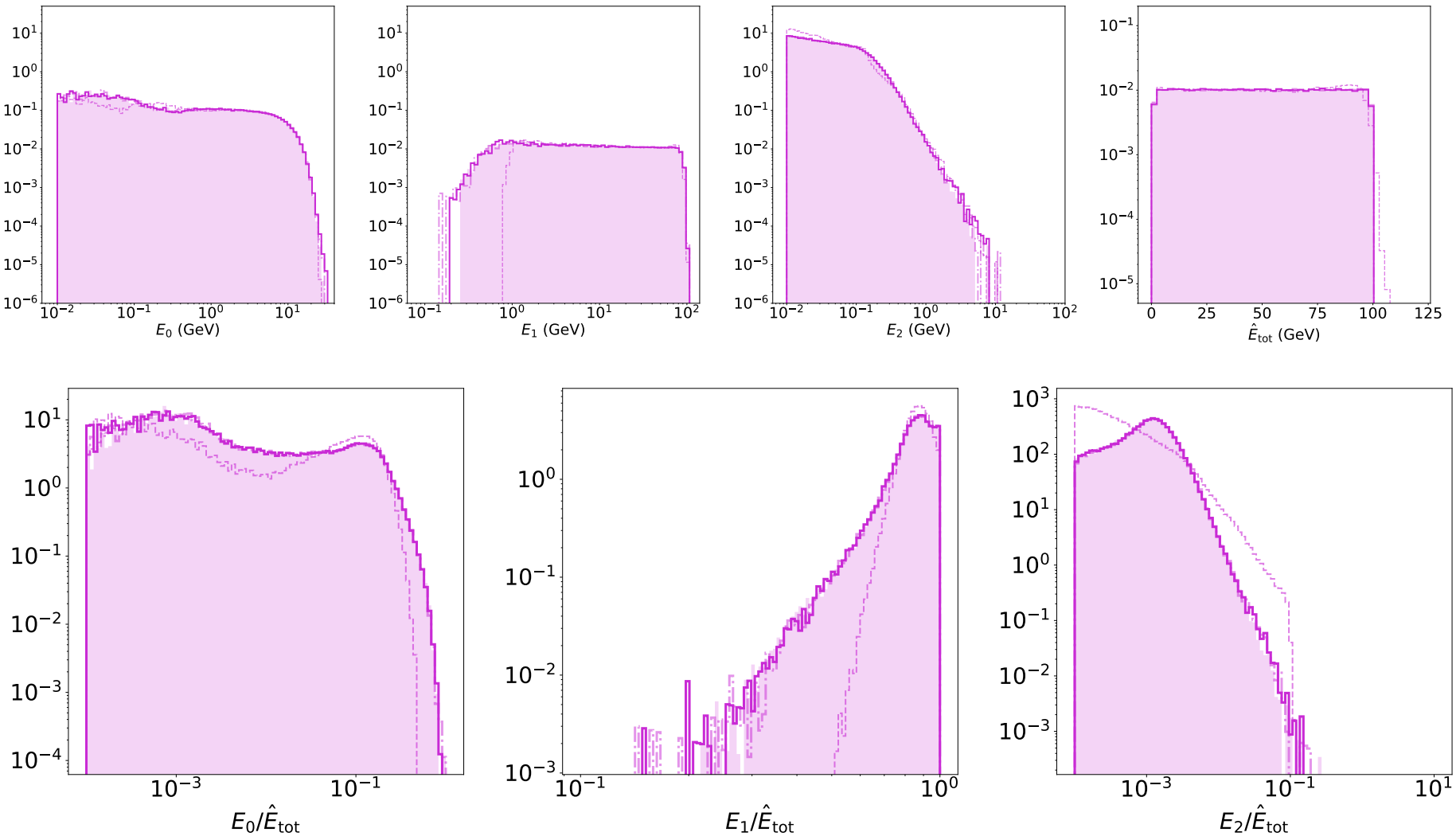
Comparing Shower Averages: γ



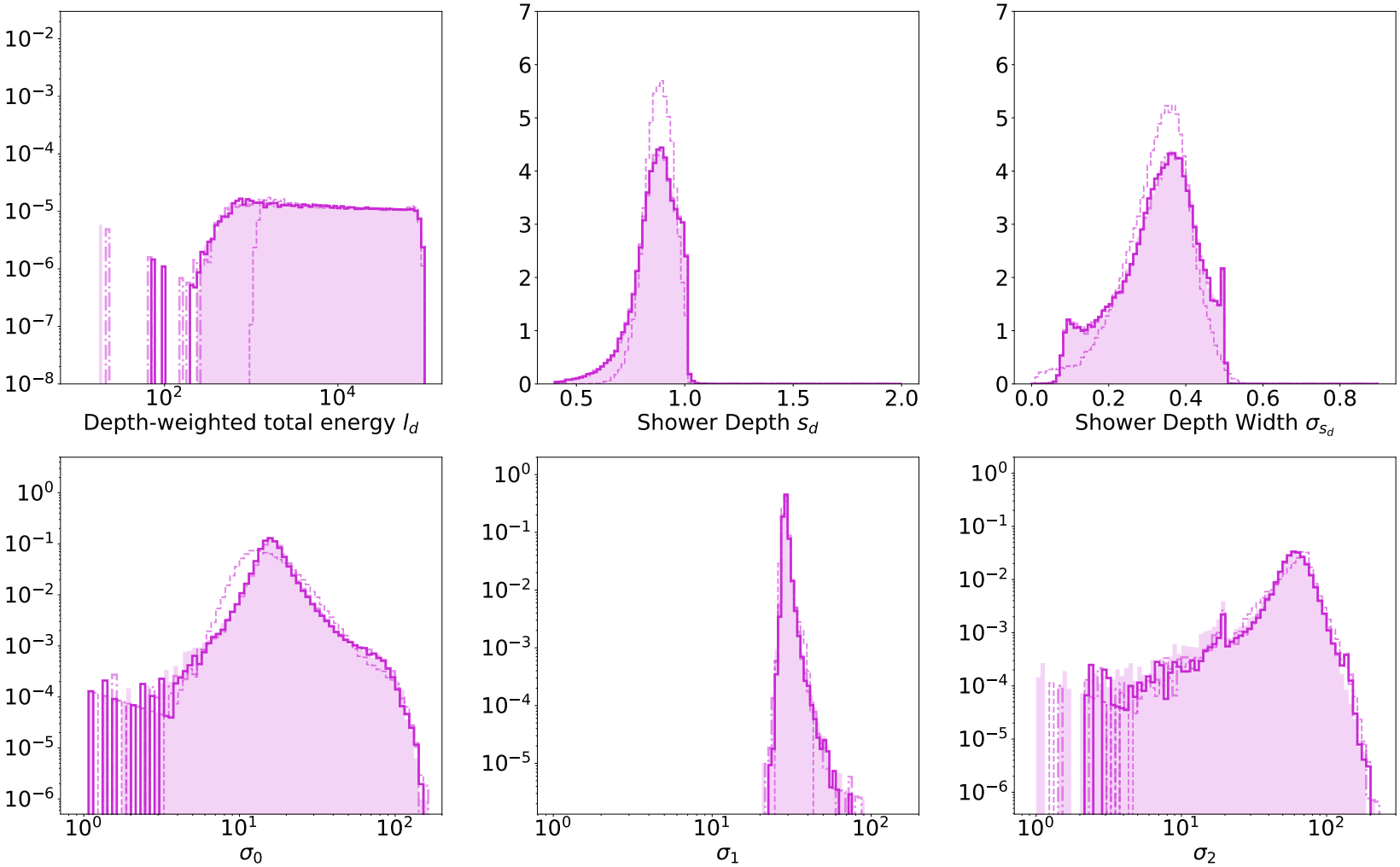
Nearest Neighbors: γ (student)



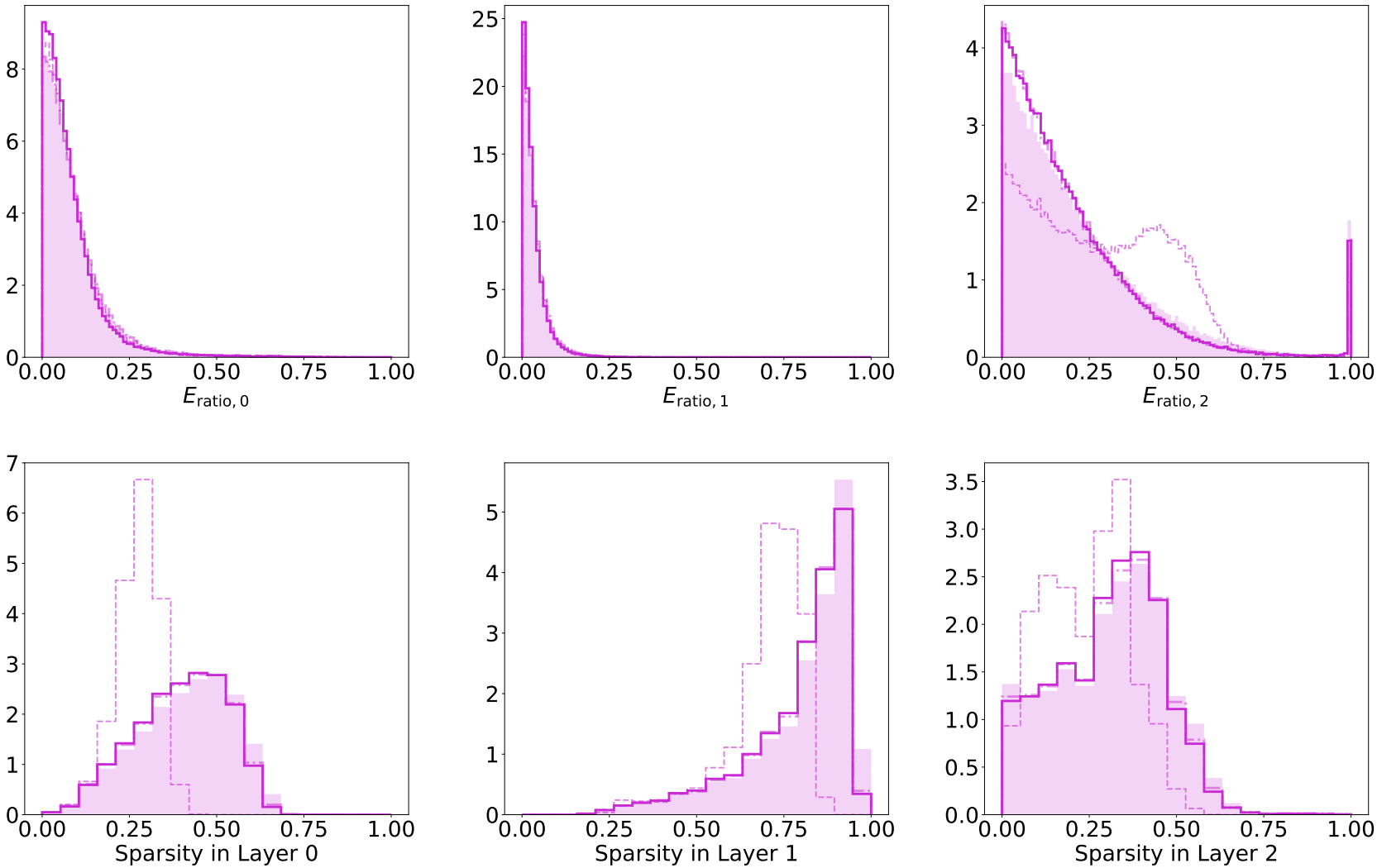
Flow I histograms: γ



Flow I+II histograms: γ



Flow II histograms: γ



Comparing Shower Averages: π^+