Breaking Simulation Bottlenecks with Normalizing Flows

— AI goes MAD, IFT —

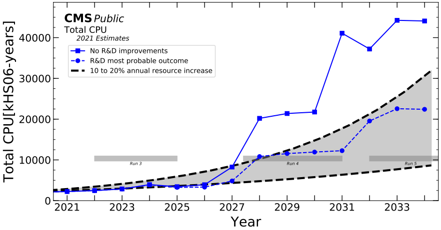
Claudius Krause

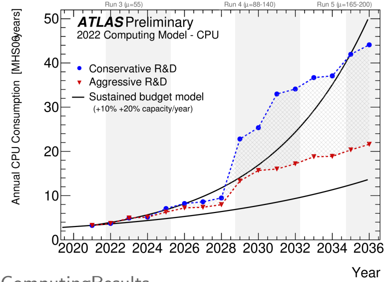
Rutgers, The State University of New Jersey

June 15, 2022

In collaboration with
Christina Gao, Stefan Höche, Joshua Isaacson, Holger Schulz
(2001.05486, ML:ST and 2001.10028, PRD)
and
David Shih (2106.05285 and 2110.11377)

Simulation bridges Theory and Experiment.

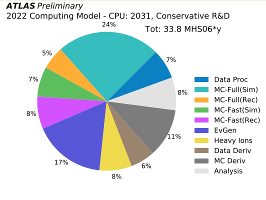




https://twiki.cern.ch/twiki/bin/view/CMSPublic/CMSOfflineComputingResults
https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/UPGRADE/CERN-LHCC-2022-005/

- At the start of LHC Run 4, the computational needs will likely exceed the available budget.
- A large fraction goes into simulation.

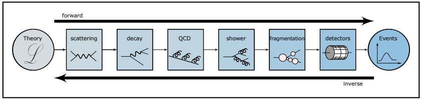
⇒ Simulating Data might become the Bottleneck.

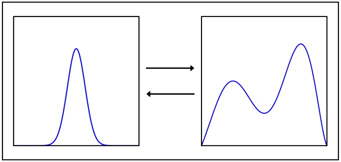


CERN-LHCC-2022-005

Breaking Simulation Bottlenecks with Normalizing Flows

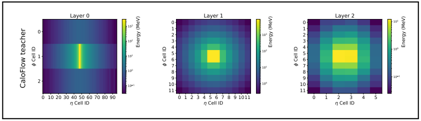
Part I: The Simulation Chain & its Bottlenecks



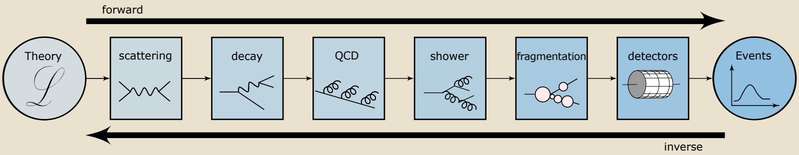


Part II: Normalizing Flows

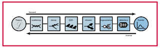
Part III: `i-flow` and CaloFlow



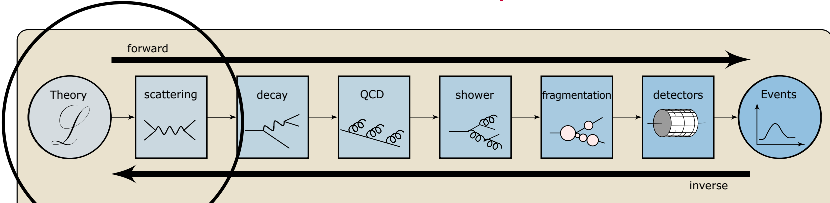
I: Full Simulations are expensive.



"Machine Learning and LHC Event Generation", Butter et al. [2203.07460]

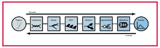


I: Full Simulations are expensive.

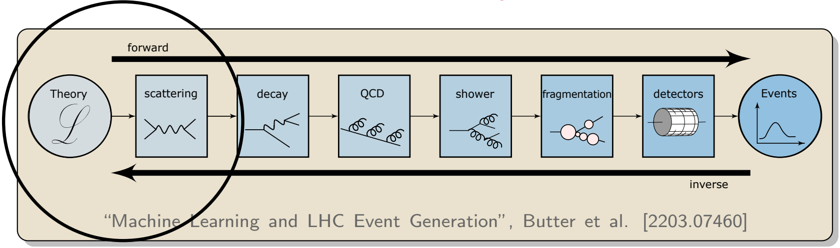


"Machine Learning and LHC Event Generation", Butter et al. [2203.07460]

⇒ Matrix elements are slow for many legs / loops.



I: Full Simulations are expensive.

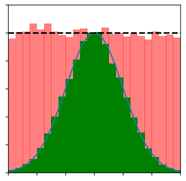


"Machine Learning and LHC Event Generation", Butter et al. [2203.07460]

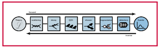
⇒ Matrix elements are slow for many legs / loops.

⇒ Unweighting efficiencies can be really small.

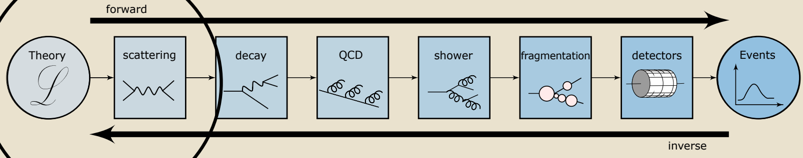
- Unweighting: we need to accept/reject each event with probability $\frac{f(x_i)}{\max f(x)}$. The kept events are unweighted and reproduce the shape of $f(x)$.



- The unweighting efficiency is the fraction of events that "survives" this procedure.



I: Full Simulations are expensive.



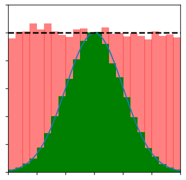
"Machine Learning and LHC Event Generation", Butter et al. [2203.07460]

$\Rightarrow$ Matrix elements are slow for many legs / loops.

$\Rightarrow$ Unweighting efficiencies can be really small.

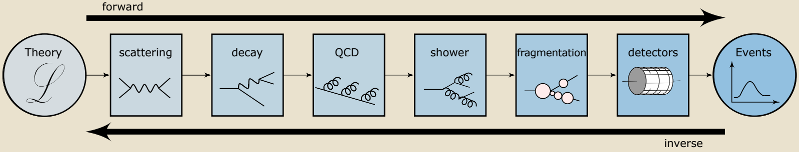
$\Rightarrow$ **need samples** $\sim \frac{\mathbf{d}\sigma}{\sigma}$**!**

- Unweighting: we need to accept/reject each event with probability $\frac{f(x_i)}{\max f(x)}$. The kept events are unweighted and reproduce the shape of $f(x)$.

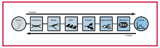


- The unweighting efficiency is the fraction of events that "survives" this procedure.

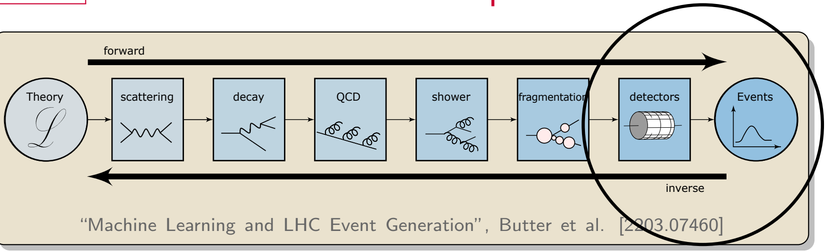
I: Full Simulations are expensive.



"Machine Learning and LHC Event Generation", Butter et al. [2203.07460]



I: Full Simulations are expensive.

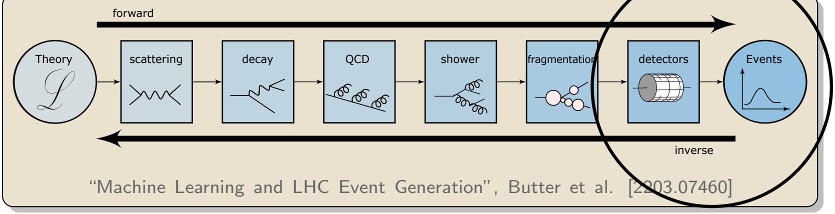


"Machine Learning and LHC Event Generation", Butter et al. [2203.07460]

⇒ Detector Simulation model stochastic interactions of particles with matter

- Flagship code GEANT4 is very slow.

I: Full Simulations are expensive.

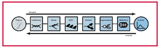


"Machine Learning and LHC Event Generation", Butter et al. [2203.07460]

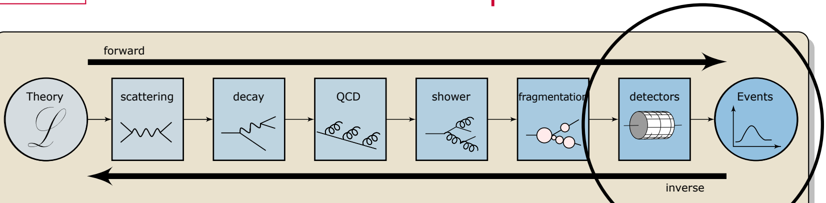
⇒ Detector Simulation model stochastic interactions of particles with matter

- Flagship code GEANT4 is very slow.

⇒ **need samples** $\sim \mathbf{p}(\text{shower}|\mathbf{E}_{\text{incident}})$!



I: Full Simulations are expensive.



"Machine Learning and LHC Event Generation", Butter et al. [2203.07460]

⇒ Detector Simulation model stochastic interactions of particles with matter

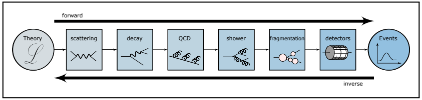
- Flagship code GEANT4 is very slow.

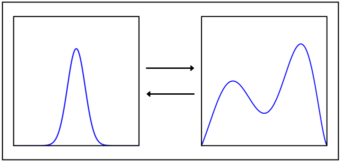
⇒ **need samples** $\sim$ **p**(shower|**E**$_{\text{incident}}$)!

⇒ Use Normalizing Flows to sample!

Breaking Simulation Bottlenecks with Normalizing Flows

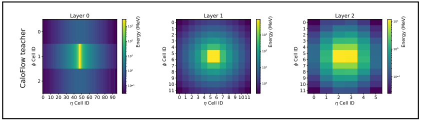
Part I: The Simulation Chain & its Bottlenecks

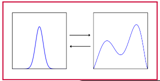




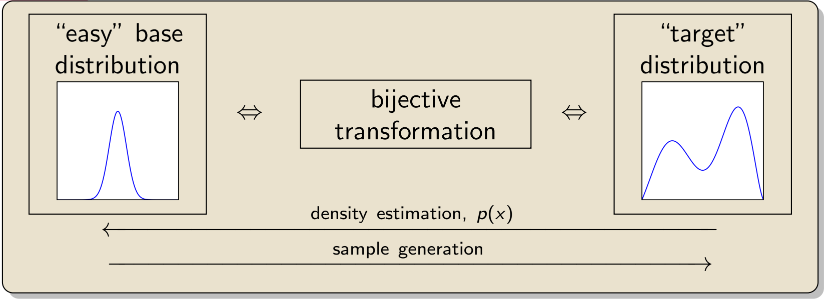
Part II: Normalizing Flows

Part III: `i-flow` and CaloFlow





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



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

- Normalizing Flows learn the parameters of a series of easy transformations.
- Each transformation has an analytic Jacobian and inverse.
- Require a triangular Jacobian for faster evaluation.

II: The Bijector is a chain of “easy” transformations.

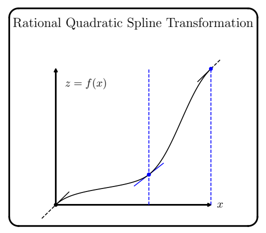
Each transformation

- must be invertible and have analytical Jacobian
- 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 Splines:

Durkan et al. [arXiv:1906.04032]

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

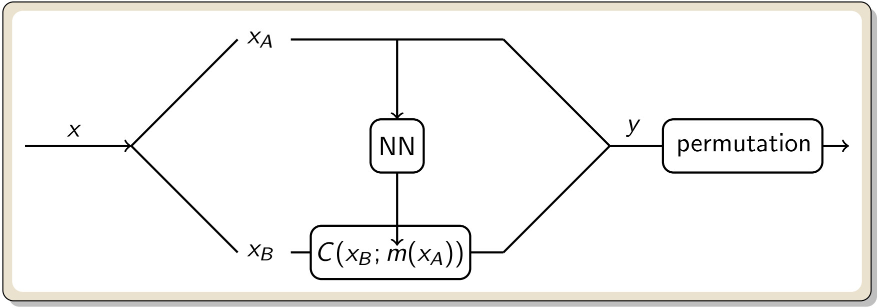


$$C = \frac{a_2\alpha^2 + a_1\alpha + a_0}{b_2\alpha^2 + b_1\alpha + b_0}$$

- numerically easy
- expressive

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

II: Triangular Jacobians 1: Bipartite Blocks (aka Coupling Layers)



forward:

$$y_A = x_A$$

$$y_{B,i} = C(x_{B,i}; m(x_A))$$

inverse:

$$x_A = y_A$$

$$x_{B,i} = C^{-1}(y_{B,i}; m(x_A))$$

The C are numerically cheap, invertible, and separable in $x_{B,i}$.

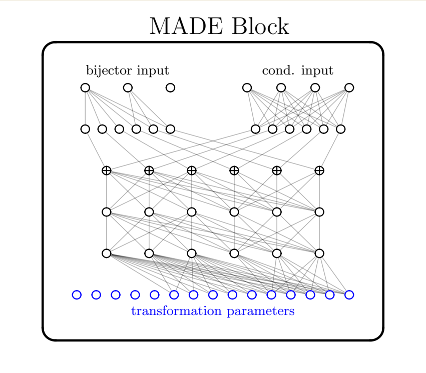
Jacobian:

$$\left|\frac{\partial y}{\partial x}\right| = \begin{vmatrix} 1 & \frac{\partial C}{\partial x_A} \\ 0 & \frac{\partial C}{\partial x_B} \end{vmatrix} = \prod_i \frac{\partial C(x_{B,i}; m(x_A))}{\partial x_{B,i}}$$

$$\Rightarrow \mathcal{O}(n)$$

Dinh et al. [arXiv:1410.8516]

II: Triangular Jacobians 2: Autoregressive Blocks (aka MADE Blocks)



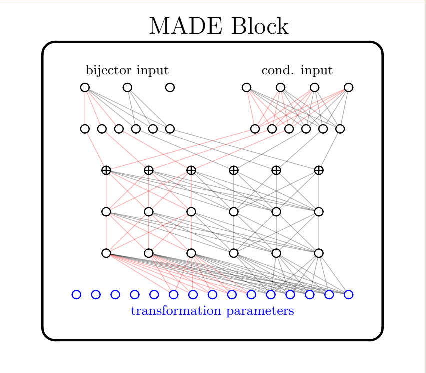
Implementation via masking:

- 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.

II: Triangular Jacobians 2: Autoregressive Blocks (aka MADE Blocks)



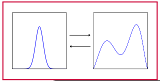
Implementation via masking:

- a single "forward" pass gives the full output of all $p(x_i|x_{i-1}\dots x_1)$.
  $\Rightarrow$ very fast

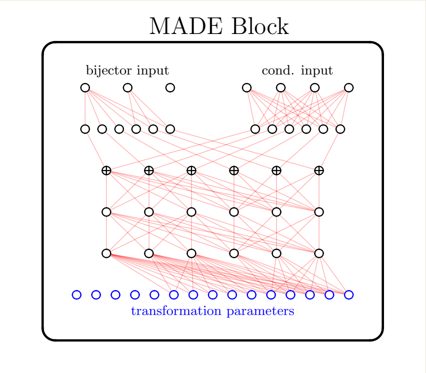
- the "inverse" needs to loop through all dimensions and gets a single $p(x_i|x_{i-1}\dots x_1)$ each time.
  $\Rightarrow$ very slow

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.



II: Triangular Jacobians 2: Autoregressive Blocks (aka MADE Blocks)



Implementation via masking:

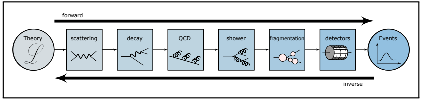
- a single "forward" pass gives the full output of all $p(x_i|x_{i-1}\dots x_1)$.
  $\Rightarrow$ very fast
- the "inverse" needs to loop through all dimensions and gets a single $p(x_i|x_{i-1}\dots x_1)$ each time.
  $\Rightarrow$ very slow

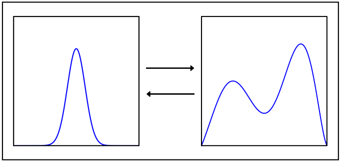
Germain/Gregor/Murray/Larochelle [arXiv:1502.03509]

- 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.

Breaking Simulation Bottlenecks with Normalizing Flows

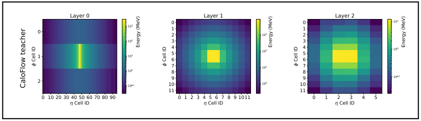
Part I: The Simulation Chain & its Bottlenecks

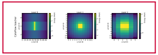




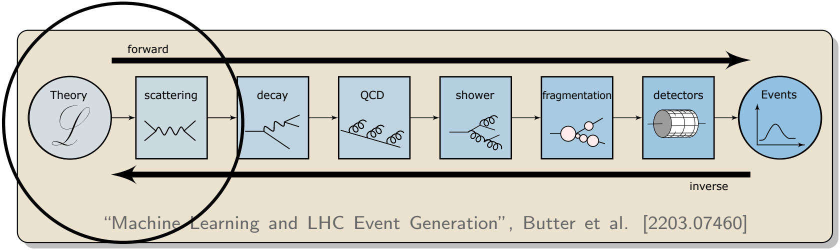
Part II: Normalizing Flows

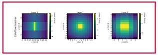
Part III: `i-flow` and CaloFlow





III: i-flow and CaloFlow





III: Numerical Integration is at the Core of Event Generation.

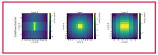
$$I = \int_0^1 f(\vec{x})\, d\vec{x} \quad \xrightarrow{\text{MC}} \quad \frac{1}{N}\sum_i f(\vec{x}_i) \qquad \vec{x}_i \ldots \text{uniform}, \quad \sigma_{\text{MC}}(I) \sim \frac{1}{\sqrt{N}}$$

$$= \int_0^1 \frac{f(\vec{x})}{q(\vec{x})}\, q(\vec{x}) d\vec{x} \quad \xrightarrow[\text{importance sampling}]{\text{MC}} \quad \frac{1}{N}\sum_i \frac{f(\vec{x}_i)}{q(\vec{x}_i)} \qquad \vec{x}_i \ldots q(\vec{x}),$$

In the limit $q(\vec{x}) \propto f(\vec{x})$, we get $\sigma_{\text{IS}}(I) = 0$

We therefore have to find a $q(\vec{x})$ that approximates the shape of $f(\vec{x})$.

$\Rightarrow$ Once found, we can use it for event generation,
i.e. sampling p_i, ϑ_i, and φ_i according to $d\sigma(p_i, \vartheta_i, \varphi_i)$



III: Numerical Integration is at the Core of Event Generation.

$$I = \int_0^1 f(\vec{x})\, d\vec{x} \quad \xrightarrow{\text{MC}} \quad \frac{1}{N}\sum_i f(\vec{x}_i) \qquad \vec{x}_i \ldots \text{uniform}, \quad \sigma_{\text{MC}}(I) \sim \frac{1}{\sqrt{N}}$$

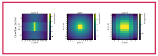
$$= \int_0^1 \frac{f(\vec{x})}{q(\vec{x})}\, q(\vec{x}) d\vec{x} \quad \xrightarrow[\text{importance sampling}]{\text{MC}} \quad \frac{1}{N}\sum_i \frac{f(\vec{x}_i)}{q(\vec{x}_i)} \qquad \vec{x}_i \ldots q(\vec{x}),$$

In the limit $q(\vec{x}) \propto f(\vec{x})$, we get $\sigma_{\text{IS}}(I) = 0$

We therefore have to find a $q(\vec{x})$ that approximates the shape of $f(\vec{x})$.

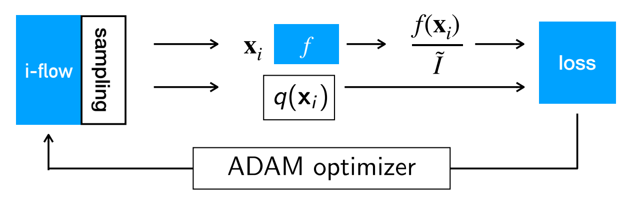
$\Rightarrow$ Once found, we can use it for event generation,
i.e. sampling p_i, ϑ_i, and φ_i according to $d\sigma(p_i, \vartheta_i, \varphi_i)$

We need both samples x and their probability $q(x)$.
$\Rightarrow$ We use a bipartite, coupling-layer-based Flow.



III: i-flow: Numerical Integration with Normalizing Flows.

How it works:



i-flow: C. Gao, J. Isaacson, CK [arXiv:2001.05486, ML:ST]
gitlab.com/i-flow/i-flow

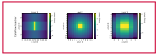
Statistical Divergences are used as loss functions:

- Kullback-Leibler (KL) divergence:

$$D_{KL} = \int p(x) \log \frac{p(x)}{q(x)} dx \approx \frac{1}{N} \sum \frac{p(x_i)}{q(x_i)} \log \frac{p(x_i)}{q(x_i)}, \qquad x_i \ldots q(x)$$

- Exponential divergence:

$$D_{exp} = \int p(x) \left(\log \frac{p(x)}{q(x)}\right)^2 dx \approx \frac{1}{N} \sum \frac{p(x_i)}{q(x_i)} \left(\log \frac{p(x_i)}{q(x_i)}\right)^2, \quad x_i \ldots q(x)$$



III: Sherpa needs a high-dimensional integrator.

Sherpa is a Monte Carlo event generator for the **S**imulation of **H**igh-**E**nergy **R**eactions of **PA**rticles. We use Sherpa to

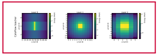
- compute the matrix element of the process.
- map the unit-hypercube of our integration domain to momenta and angles. To improve efficiency, Sherpa uses a recursive multichannel algorithm.

$$\Rightarrow n_{dim} = \underbrace{3n_{final} - 4}_{\text{kinematics}} + \underbrace{n_{final} - 1}_{\text{multichannel}}$$

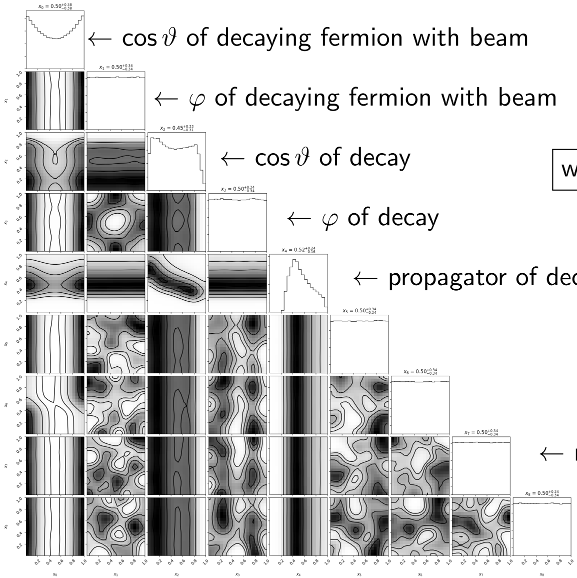
- However, the `COMIX++` ME-generator uses color-sampling, so we should also integrate over final state color configurations. While this improves the efficiency, it is not possible to handle group processes like $W + nj$ with a single flow.

$$\Rightarrow n_{dim} = 4n_{final} - 4 + 2n_{color}$$

https://sherpa.hepforge.org/



III: An easy example: $e^+e^- \to 3j$.



$\leftarrow$ $\cos\vartheta$ of decaying fermion with beam

$\leftarrow$ φ of decaying fermion with beam

$\leftarrow$ $\cos\vartheta$ of decay

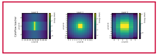
$\leftarrow$ φ of decay

$\leftarrow$ propagator of decaying fermion

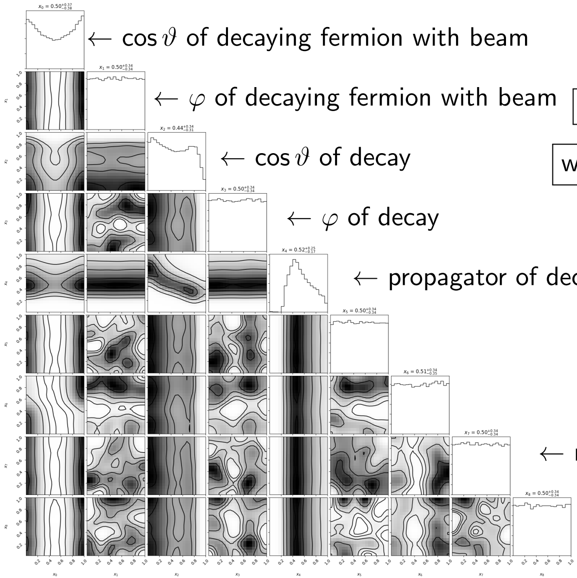
$\leftarrow$ multichannel

Target distribution

without learning color



III: An easy example: $e^+e^- \to 3j$.



$\leftarrow$ $\cos\vartheta$ of decaying fermion with beam

$\leftarrow$ φ of decaying fermion with beam

$\leftarrow$ $\cos\vartheta$ of decay

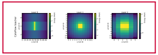
$\leftarrow$ φ of decay

$\leftarrow$ propagator of decaying fermion

$\leftarrow$ multichannel

Learned distribution

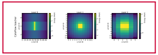
without learning color



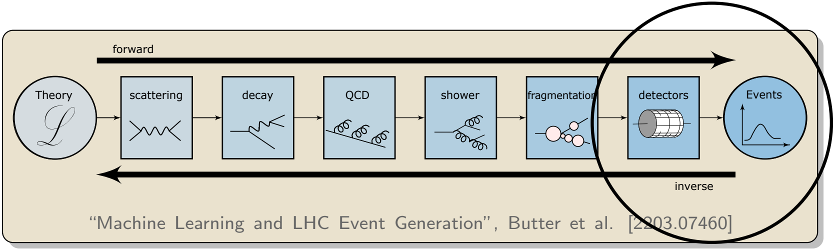
III: High Multiplicities are still difficult to learn.

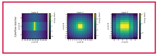
unweighting efficiency $\langle w \rangle / w_{\max}$		LO QCD $n=0$	$n=1$	$n=2$	$n=3$
$W^+ + n$ jets	Sherpa	$2.8 \cdot 10^{-1}$	$3.8 \cdot 10^{-2}$	$7.5 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$
	i-flow	$6.1 \cdot 10^{-1}$	$1.2 \cdot 10^{-1}$	$1.0 \cdot 10^{-2}$	$1.8 \cdot 10^{-3}$
	Gain	2.2	3.3	1.4	1.2
$W^- + n$ jets	Sherpa	$2.9 \cdot 10^{-1}$	$4.0 \cdot 10^{-2}$	$7.7 \cdot 10^{-3}$	$2.0 \cdot 10^{-3}$
	i-flow	$7.0 \cdot 10^{-1}$	$1.5 \cdot 10^{-1}$	$1.1 \cdot 10^{-2}$	$2.2 \cdot 10^{-3}$
	Gain	2.4	3.3	1.4	1.1
$Z + n$ jets	Sherpa	$3.1 \cdot 10^{-1}$	$3.6 \cdot 10^{-2}$	$1.5 \cdot 10^{-2}$	$4.7 \cdot 10^{-3}$
	i-flow	$3.8 \cdot 10^{-1}$	$1.0 \cdot 10^{-1}$	$1.4 \cdot 10^{-2}$	$2.4 \cdot 10^{-3}$
	Gain	1.2	2.9	0.91	0.51

C. Gao, S. Höche, J. Isaacson, CK, H. Schulz [arXiv:2001.10028, PRD]



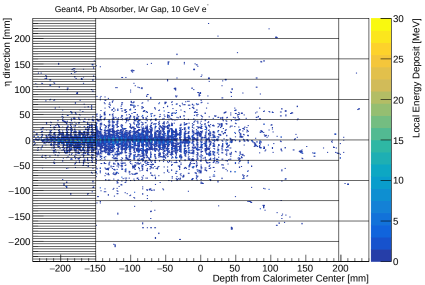
III: i-flow and CaloFlow

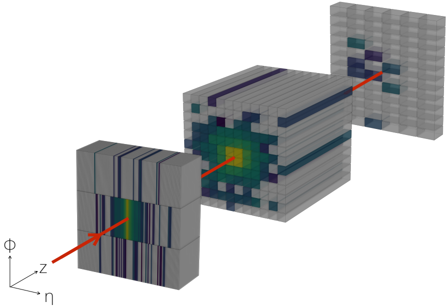




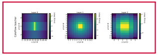
III: We use the same calorimeter geometry as CaloGAN.

- 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



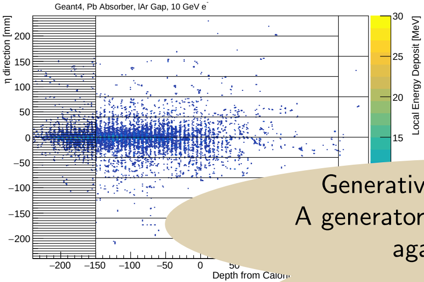


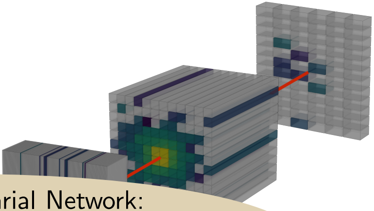
CaloGAN: Paganini, de Oliveira, Nachman [1705.02355, PRL; 1712.10321, PRD]



III: We use the same calorimeter geometry as CaloGAN.

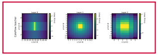
- We consider a simplified version of the ATLAS ECal: flat alternating layers of lead and LAr
- They form three instrumented layers of dimension 3×96, 12×12, and 12×6





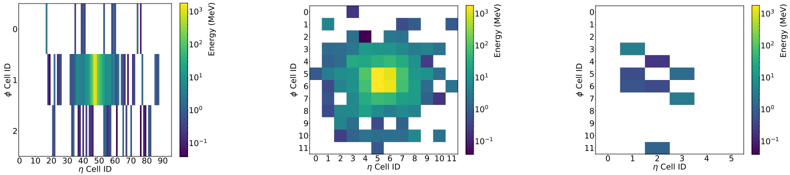
Generative Adversarial Network:
A generator and a critic play a game against each other.

CaloGAN: Paganini, de Oliveira, Nachman [1705.02355, PRL; 1712.10321, PRD]

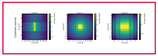


III: We use the same calorimeter geometry as CaloGAN.

- The GEANT4 configuration of CaloGAN is available at https://github.com/hep-lbdl/CaloGAN
- We produce our own dataset: available at [DOI: 10.5281/zenodo.5904188]
- 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 [1705.02355, PRL; 1712.10321, PRD]



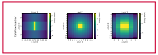
III: CaloFlow uses a 2-step approach to learn $p(\mathcal{I}|E_{\text{inc}})$.

Flow I

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

Flow II

- learns $p_2(\hat{\vec{\mathcal{I}}}|E_0, E_1, E_2, E_{\text{inc}})$ 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]
    i.e. matching IAF parameters to frozen MAF
  - Fast in sampling ($\approx 500\times$ faster than CaloFlow v1)



III: 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 $p_{\text{GEANT4}}(x) = p_{\text{generated}}(x)$

⇒ This captures the full 504-dim. space.

? But why wasn't this used before?

⇒ Previous deep generative models were separable to almost 100%!

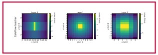
DCTRGAN: Diefenbacher et al. [2009.03796, JINST]

III: CaloFlow passes the "ultimate metric" test.

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		DNN based classifier		
		Geant4 vs. CaloGAN	Geant4 vs. (teacher) CaloFlow v1	Geant4 vs. (student) CaloFlow v2
e^+	unnorm.	1.000(0)	0.859(10)	0.786(7)
	norm.	1.000(0)	0.870(2)	0.824(4)
γ	unnorm.	1.000(0)	0.756(48)	0.758(14)
	norm.	1.000(0)	0.796(2)	0.760(3)
π^+	unnorm.	1.000(0)	0.649(3)	0.729(2)
	norm.	1.000(0)	0.755(3)	0.807(1)

AUC ($\in [0.5, 1]$): Area Under the ROC Curve, smaller is better, i.e. more confused

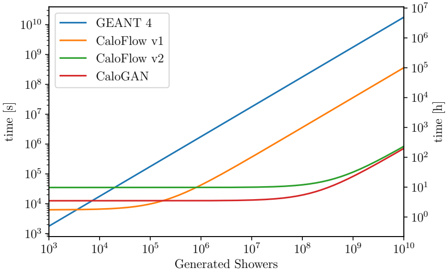


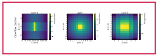
III: Sampling Speed: The Student beats the Teacher!

	CaloFlow* teacher	CaloFlow* student	CaloGAN*		Geant4†
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

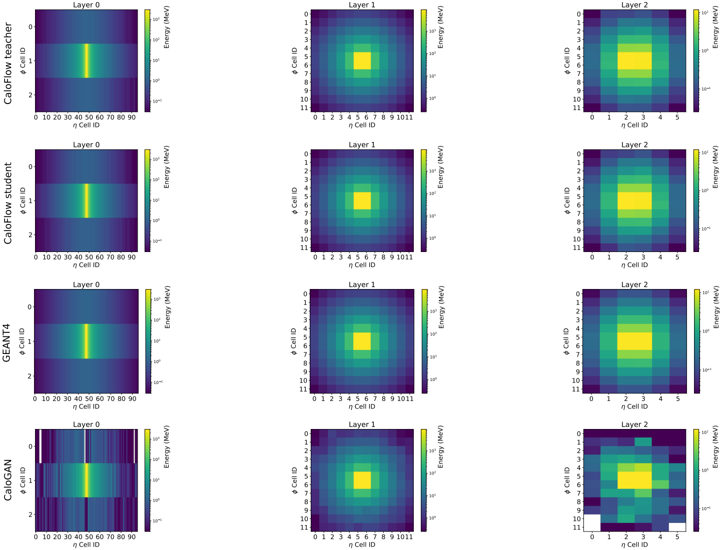
*: on our Titan V GPU

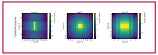
†: on the CPU of CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD]



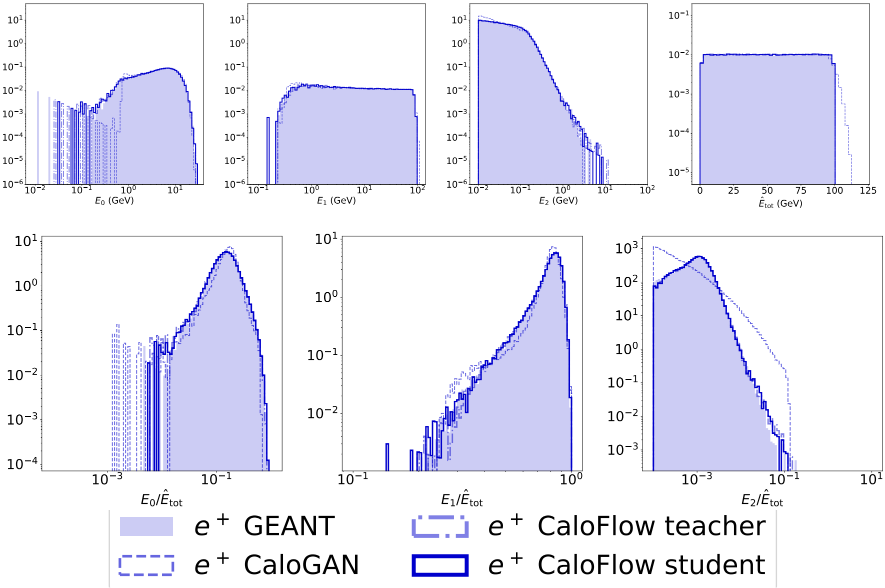


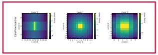
III: CaloFlow: Comparing Shower Averages: e^+



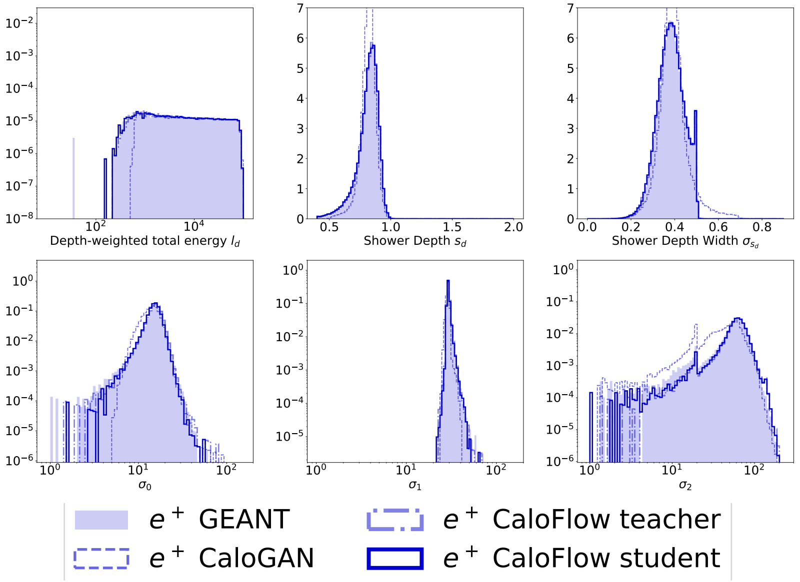


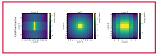
III: CaloFlow: Flow I histograms: e^+



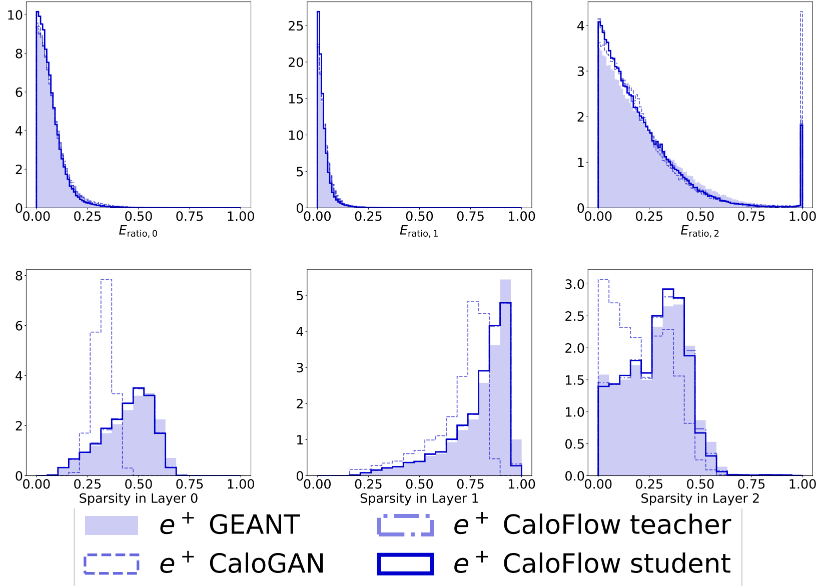


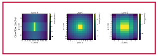
III: CaloFlow: Flow I+II histograms: e^+



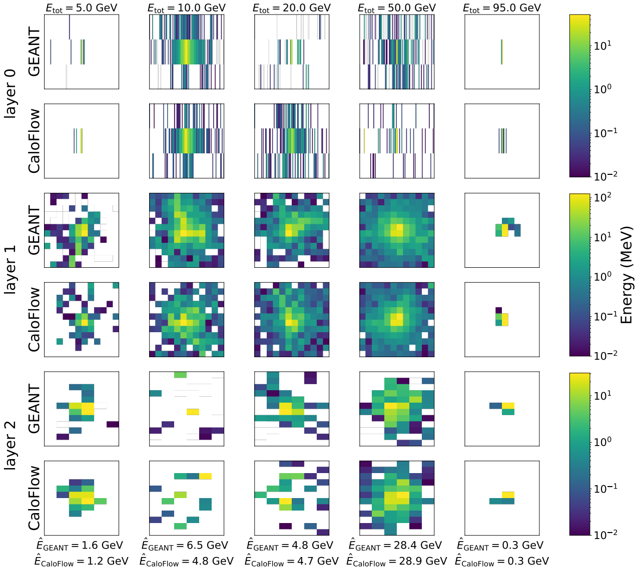


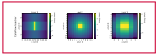
III: CaloFlow: Flow II histograms: e^+



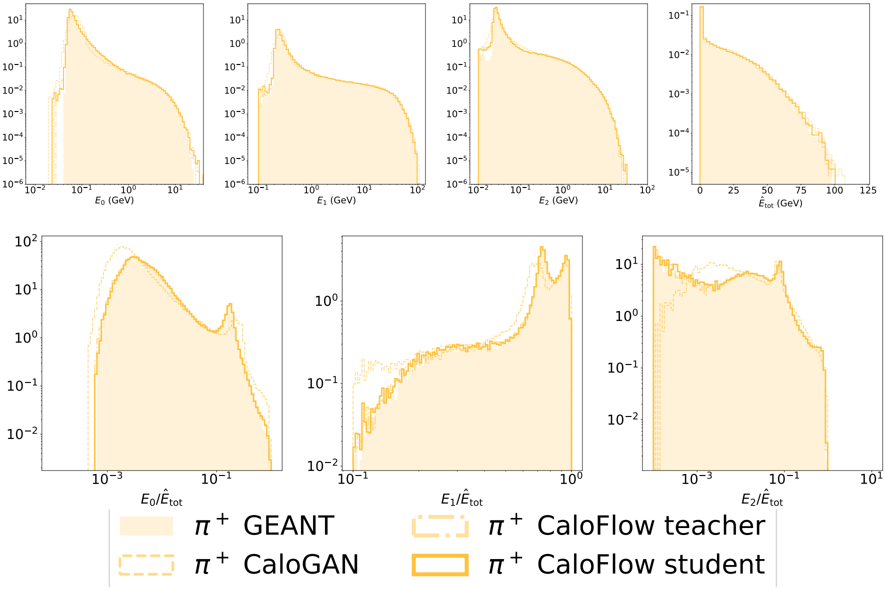


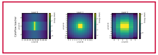
III: CaloFlow: Nearest Neighbors: π^+ (student)



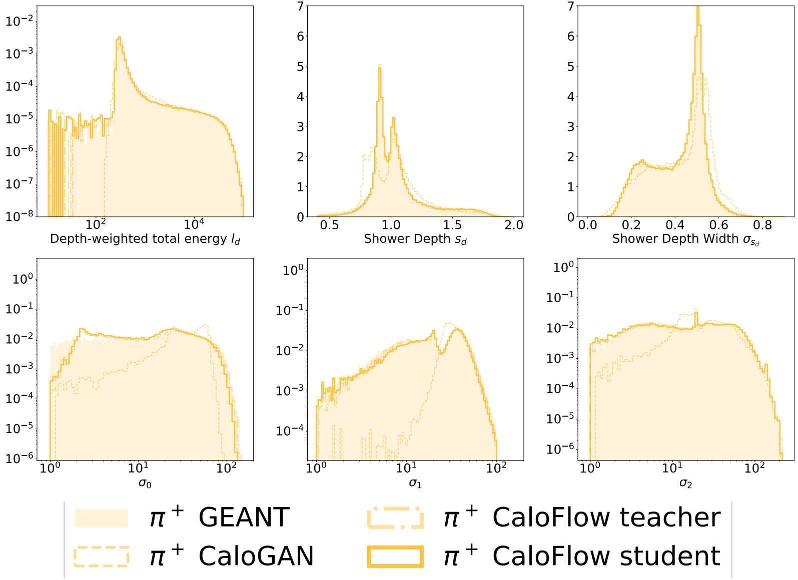


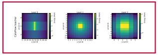
III: CaloFlow: Flow I histograms: π^+



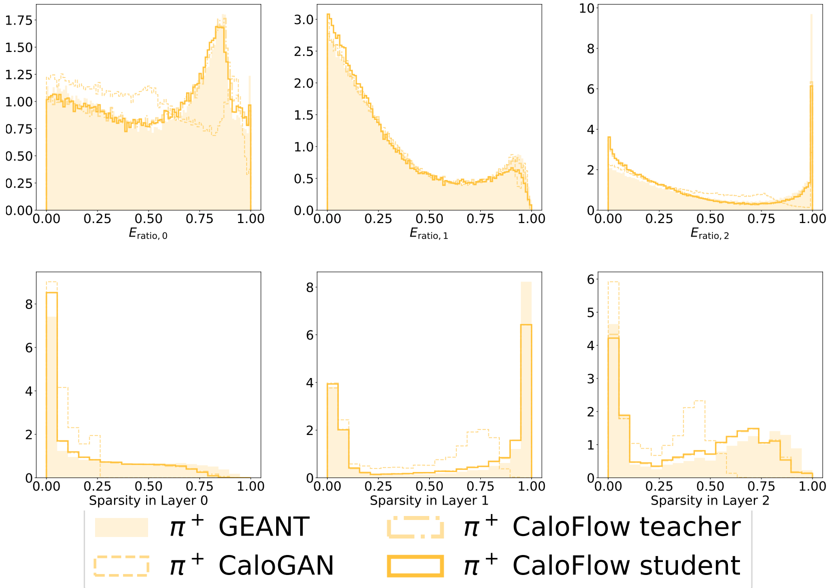


III: CaloFlow: Flow I+II histograms: π^+





III: CaloFlow: Flow II histograms: π^+



A little Advertisement — CaloChallenge 2022

Welcome to the home of the Fast Calorimeter Simulation Challenge 2022!

Homepage for the Fast Calorimeter Simulation Challenge 2022

View on GitHub

Welcome to the home of the Fast Calorimeter Simulation Challenge 2022!

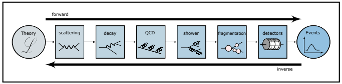
This is the homepage for the Fast Calorimeter Simulation Data Challenge. The purpose of this challenge is to spur the development and benchmarking of fast and high-fidelity calorimeter shower generation. Currently, generating calorimeter showers of elementary particles (electrons, photons, pions, ...) using GEANT4 is a major computational bottleneck at the LHC, and it is forecast to overwhelm the computing budget of the LHC in the near future. Therefore there is an urgent need to

Michele Faucci Giannelli, Gregor Kasieczka, Claudius Krause, Ben Nachman, Dalila Salamani, David Shih, and Anna Zaborowska

⇒ https://calochallenge.github.io/homepage/

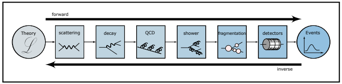
Breaking Simulation Bottlenecks with Normalizing Flows

- Simulations are a crucial bridge between Theory and Experiment!
- They might limit the analyses we can do at the LHC.

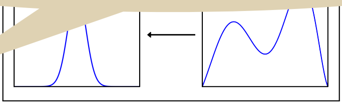


Breaking Simulation Bottlenecks with Normalizing Flows

- Simulations are a crucial bridge between Theory and Experiment!
- They might limit the analyses we can do at the LHC.



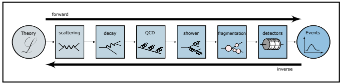
- I introduced Normalizing few of their realizations.
- They are Density Estimators and Generative Model.



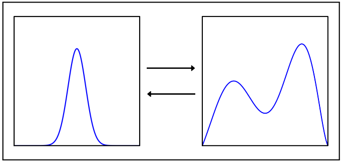
Other HEP applications:
Anomaly Detection, Lattice QCD, ...

Breaking Simulation Bottlenecks with Normalizing Flows

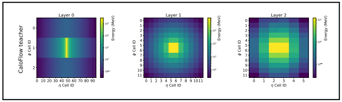
- Simulations are a crucial bridge between Theory and Experiment!
- They might limit the analyses we can do at the LHC.



- I introduced Normalizing Flows and a few of their realizations.
- They are Density Estimators and Generative Model.

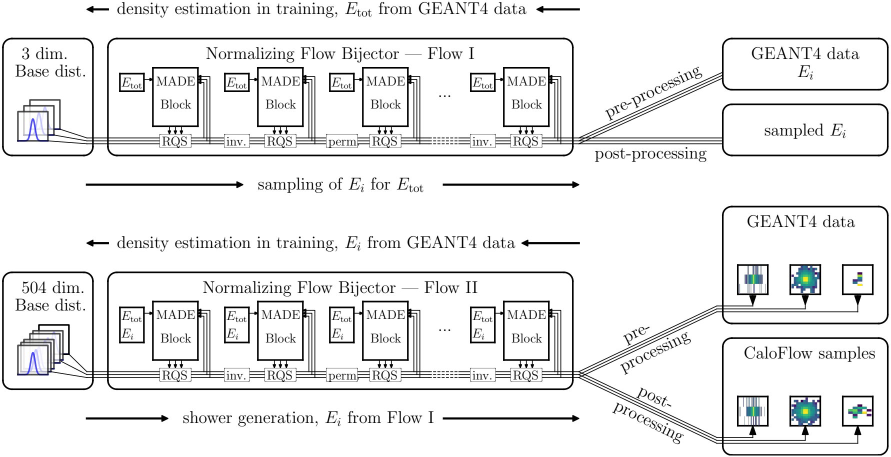


- `i-flow` improves the unweighting efficiency in event generation.
- CaloFlow provides a fast and faithful detector simulation.



Backup

CaloFlow uses a 2-step approach.

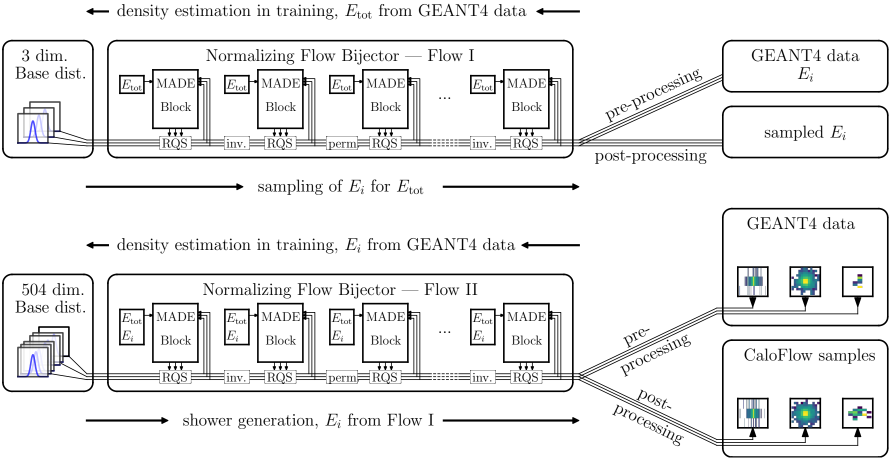


Data processing Flow I

"←" map E_i to $[0, 1]$ "→" invert logit

"←" work in logit space "→" map back to E_i

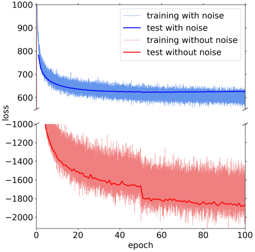
CaloFlow uses a 2-step approach.

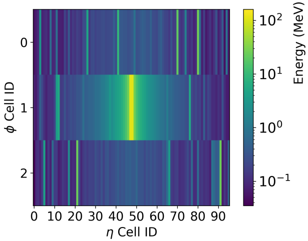


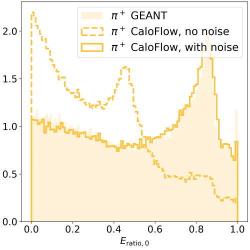
Data processing Flow II

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

Adding Noise is important for the sampling quality.

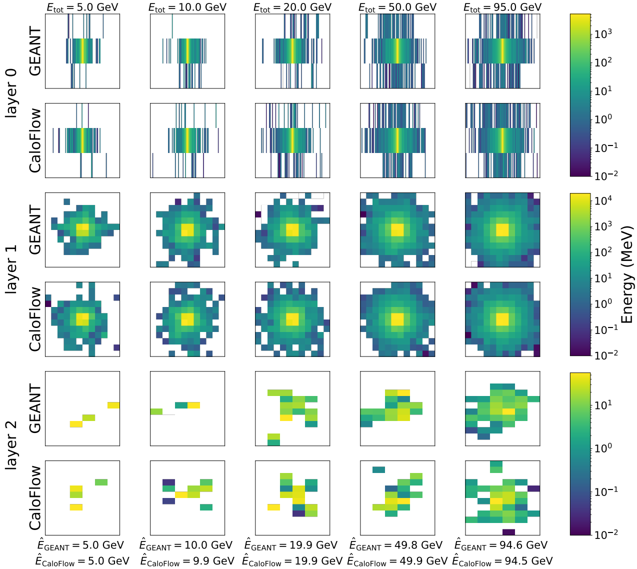




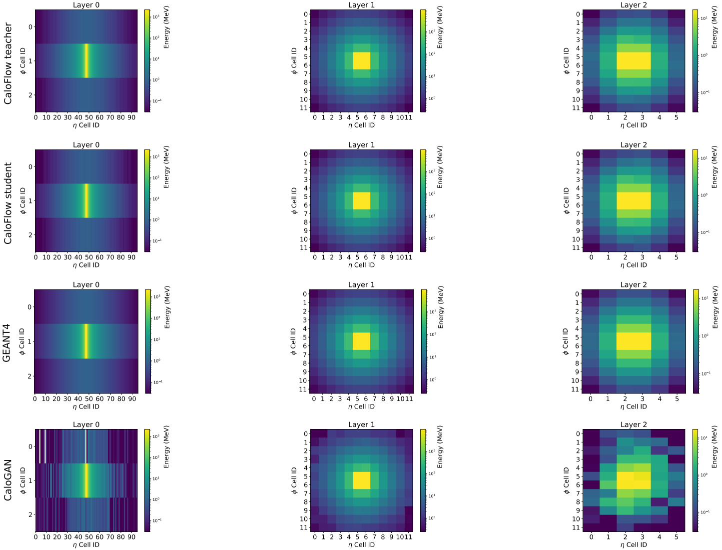


- The log-likelihood is less noisy, but smaller. Yet, the quality of the samples is much better!
- This is due to a "wider" mapping of space and less overfitting.

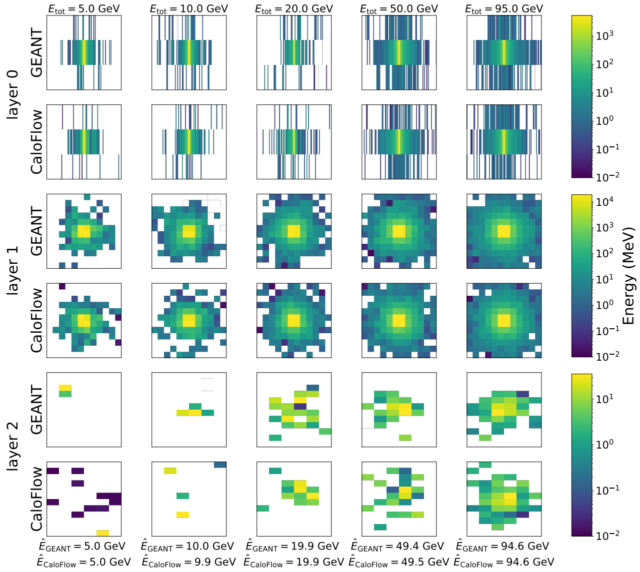
Nearest Neighbors: e^+ (student)



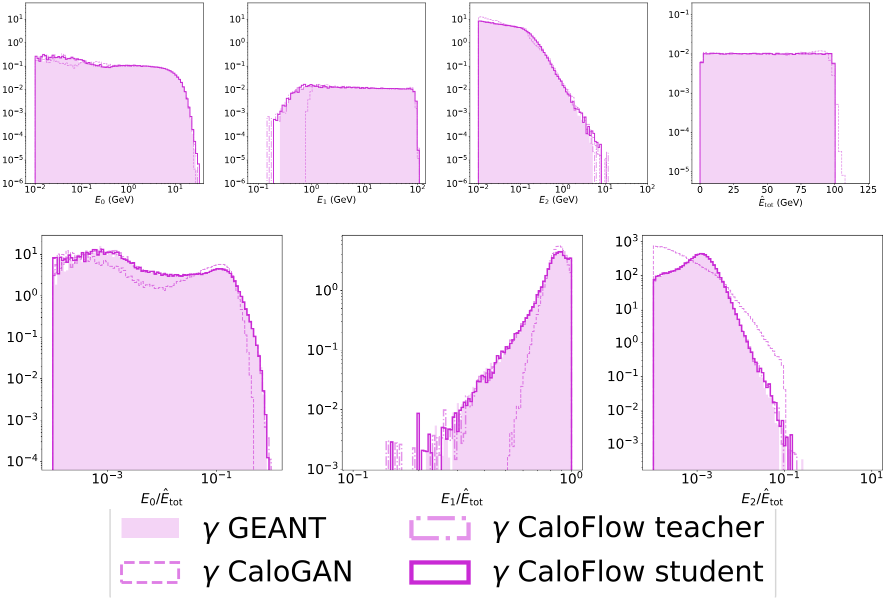
Comparing Shower Averages: γ



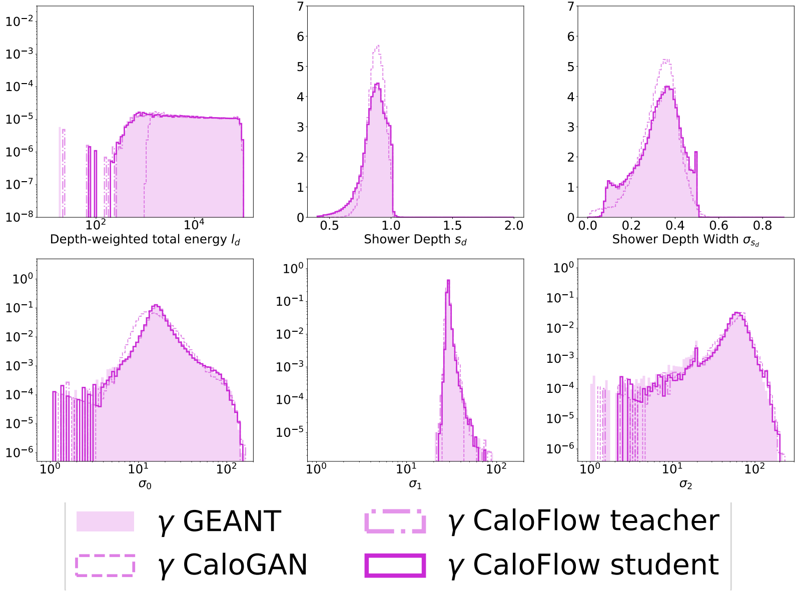
Nearest Neighbors: γ (student)



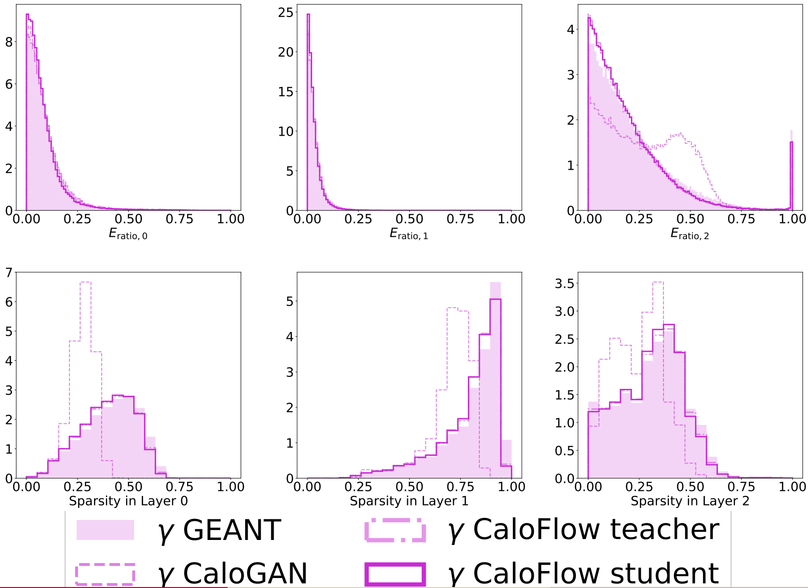
Flow I histograms: γ



Flow I+II histograms: γ



Flow II histograms: γ



Comparing Shower Averages: π^+