Fast and Faithful (Calorimeter) Simulations with Normalizing Flows

— AI and Quantum Information Applications in Fundamental Physics, Konjiam, South Korea —

Claudius Krause

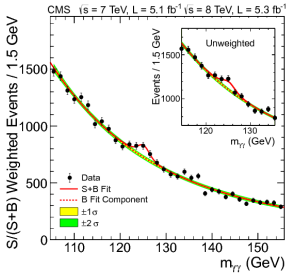
Institute for Theoretical Physics, University of Heidelberg
(and Rutgers, The State University of New Jersey)

February 14, 2023

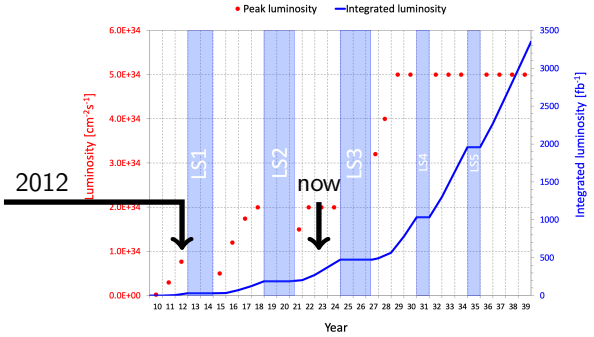
Based on
[2106.05285, 2110.11377, 2210.14245, 2212.06172] and lots [in preparation]

In collaboration with
David Shih, Ian Pang, Matt Buckley, Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Imahn Shekhzadeh, Florian Ernst, Luigi Favaro, Tilman Plehn, Theo Heimel, Ramon Winterhalder, Anja Butter, Joshua Isaacson, Fabio Maltoni, Olivier Mattelaer

We will have a lot more data in the near future.



CMS Collaboration [arXiv:1207.7235, Phys.Lett.B]

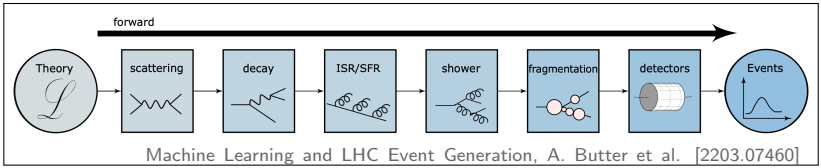


https://lhc-commissioning.web.cern.ch/schedule/HL-LHC-plots.htm

- We will have 20–25× more data.

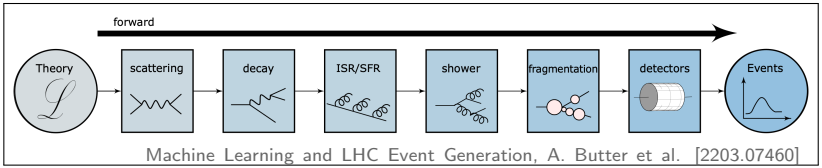
⇒ We want to understand every aspect of it based on 1st principles (and find New Physics)!

Simulation bridges Theory and Experiment.

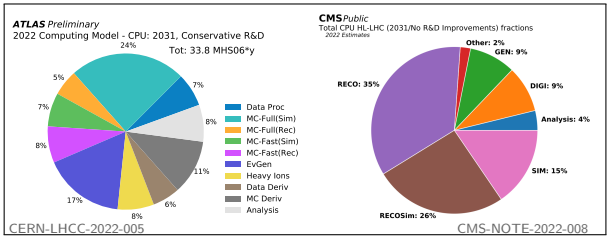


Machine Learning and LHC Event Generation, A. Butter et al. [2203.07460]

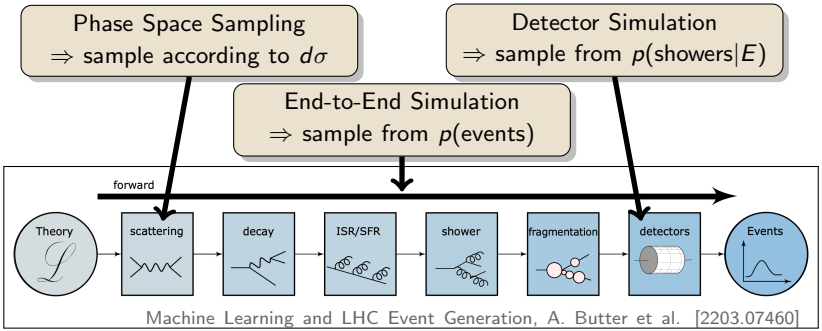
Simulation bridges Theory and Experiment.

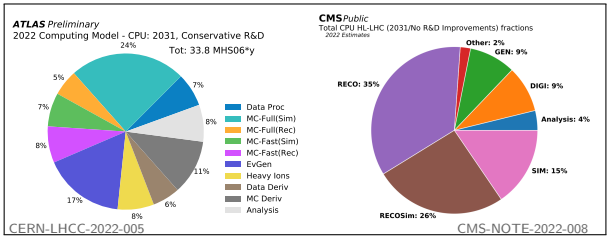


Machine Learning and LHC Event Generation, A. Butter et al. [2203.07460]



Simulation bridges Theory and Experiment.





Deep Generative Models are Random Number Generators.

We have a distribution $p(x)$ and want to sample ("generate") new elements that follow it.

given: $\{x_i\}$ want: $x \sim p(x)$

- or -

given: $f(x)$ want: $x \sim f(x)/\int f(x)$

Deep Generative Models are Random Number Generators.

We have a distribution $p(x)$ and want to sample ("generate") new elements that follow it.

given: $\{x_i\}$ want: $x \sim p(x)$

- or -

given: $f(x)$ want: $x \sim f(x) / \int f(x)$

- Deep Generative Models have seen a lot of progress in recent years.
- They can generate text, speech, images, ...
- In HEP, we have seen GANs, VAEs, Normalizing Flows, Diffusion Models, and their derivates.

"Calorimeter Simulation" via midjourney.com

The first models in use were GANs and VAEs.

Variational Autoencoder (VAE): Information is encoded and decoded through a bottleneck.

- Trade-off between generalizability and quality (β-term in the loss)
- Physics is not always compressible.
- Precision not competitive to other generative models.

The first models in use were GANs and VAEs.

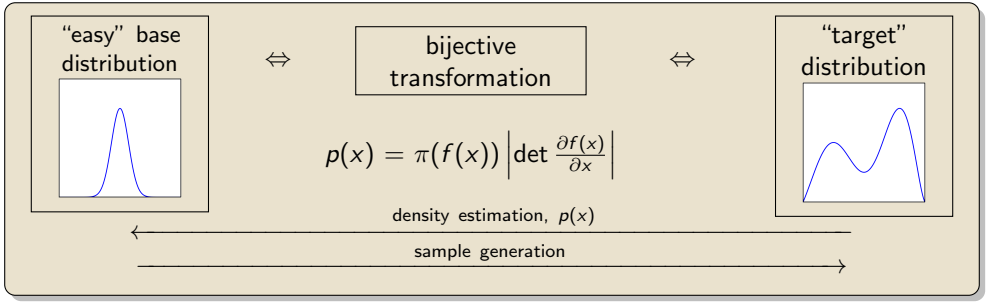
Variational Autoencoder (VAE): Information is encoded and decoded through a bottleneck.

- Trade-off between generalizability and quality (β-term in the loss)
- Physics is not always compressible.
- Precision not competitive to other generative models.

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

- Delicate optimization on a saddlepoint.
- Suffer from mode-collapse and other artefacts.
- Model selection is difficult.
- Hard to get the distributions to agree at the %-level.

Normalizing Flows learn a coordinate transformation.



Having access to the log-likelihood (LL) allows several training options:

⇒ Based on samples: via maximizing LL(samples).

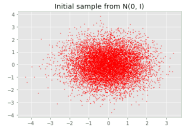
⇒ Based on target function $f(x)$: via matching $p(x)$ to $f(x)$.

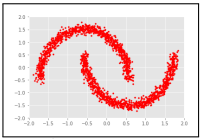
How do Normalizing Flows tame Jacobians?

- NFs learn the parameters θ of a series of easy transformations.
  Dinh et al. [arXiv:1410.8516], Rezende/Mohamed [arXiv:1505.05770]
- Each transformation is 1d & has an analytic Jacobian and inverse.
  $\Rightarrow$ We use Rational Quadratic Splines
  Durkan et al. [arXiv:1906.04032], Gregory/Delbourgo [IMA J. of Num. An., '82]
- Require a triangular Jacobian for faster evaluation.
  $\Rightarrow$ The parameters θ depend only on a subset of all other coordinates.

How do Normalizing Flows tame Jacobians?

- NFs learn the parameters θ of a series of easy transformations.
  Dinh et al. [arXiv:1410.8516], Rezende/Mohamed [arXiv:1505.05770]
- Each transformation is 1d & has an analytic Jacobian and inverse.
  $\Rightarrow$ We use Rational Quadratic Splines
  Durkan et al. [arXiv:1906.04032], Gregory/Delbourgo [IMA J. of Num. An., '82]
- Require a triangular Jacobian for faster evaluation.
  $\Rightarrow$ The parameters θ depend only on a subset of all other coordinates.

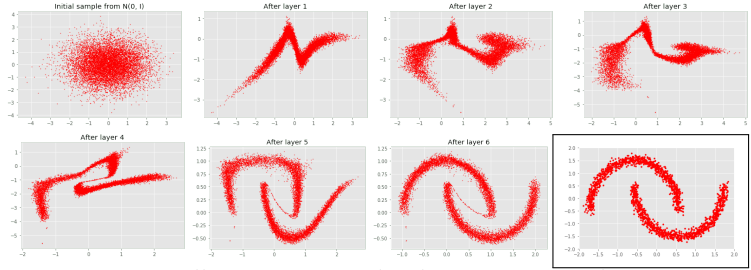




https://engineering.papercup.com/posts/normalizing-flows-part-2/

How do Normalizing Flows tame Jacobians?

- NFs learn the parameters θ of a series of easy transformations.
  Dinh et al. [arXiv:1410.8516], Rezende/Mohamed [arXiv:1505.05770]
- Each transformation is 1d & has an analytic Jacobian and inverse.
  $\Rightarrow$ We use Rational Quadratic Splines
  Durkan et al. [arXiv:1906.04032], Gregory/Delbourgo [IMA J. of Num. An., '82]
- Require a triangular Jacobian for faster evaluation.
  $\Rightarrow$ The parameters θ depend only on a subset of all other coordinates.



https://engineering.papercup.com/posts/normalizing-flows-part-2/

The Bijector is a chain of "easy" transformations.

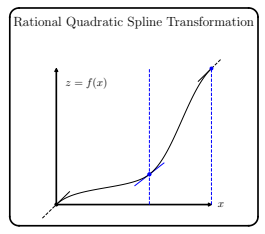
Each transformation

- 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]

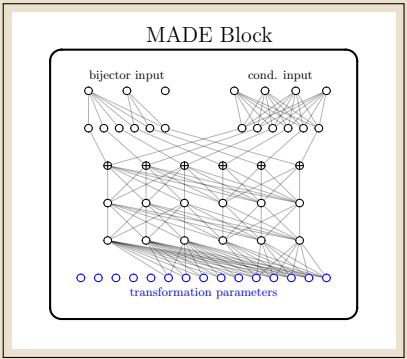


$$C = \frac{a_2\alpha^2 + a_1\alpha + a_0}{b_2\alpha^2 + b_1\alpha + b_0}$$

- numerically easy
- expressive

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

Triangular Jacobians 1: with Autoregressive Blocks



$$\boxed{\theta_{x_i}(x_{j<i})}$$

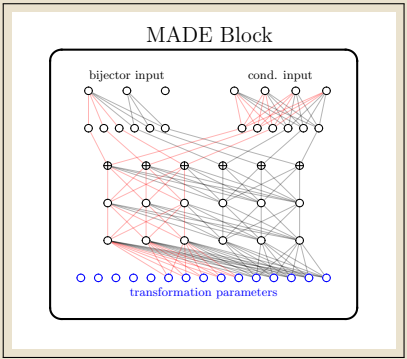
Implementation via masking:

- a single "forward" pass gives all $\theta_{x_i}(x_{i-1} \dots x_1)$.
  $\Rightarrow$ very fast
- its "inverse" needs to loop through all dimensions.
  $\Rightarrow$ very slow

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

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

Triangular Jacobians 1: with Autoregressive Blocks



$$\boxed{\theta_{x_i}(x_{j<i})}$$

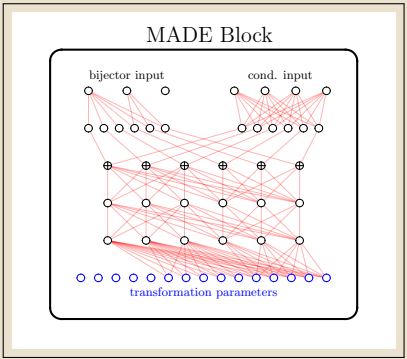
Implementation via masking:

- a single "forward" pass gives all $\theta_{x_i}(x_{i-1} \dots x_1)$.
  $\Rightarrow$ very fast
- its "inverse" needs to loop through all dimensions.
  $\Rightarrow$ very slow

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

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

Triangular Jacobians 1: with Autoregressive Blocks



$$\boxed{\theta_{x_i}(x_{j<i})}$$

Implementation via masking:

- a single "forward" pass gives all $\theta_{x_i}(x_{i-1} \dots x_1)$.
  $\Rightarrow$ very fast
- its "inverse" needs to loop through all dimensions.
  $\Rightarrow$ very slow

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

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

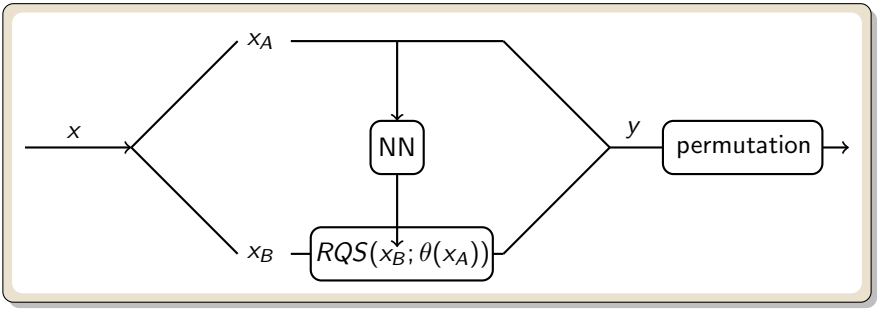
Triangular Jacobians 2: with Bipartite Blocks

$$\theta_{x\in A}(x \in B) \qquad \& \qquad \theta_{x\in B}(x \in A)$$

- Coordinates are split in 2 sets, transforming each other.

  \+ Forward and inverse pass are equally fast. – Said to be not as expressive.

Dinh et al. [arXiv:1410.8516]



Normalizing Flows are great!

Pros and Cons of Normalizing Flows:

+ LL optimaztion is more stable than saddlepoint optimization of GANs.
+ Do not suffer from mode-collapse.
+ Model selection is straightforward with LL(val-set).
+ Flows are versatile (train for one thing, use for another).
+ Empirically: better at learning distributions to the %-level

– They scale bad with the dimensionality of the problem.
– Some architectures might be slow.
– There are topological constraints.
– Sparse data is hard to learn.

Evaluating Generative Models is a hard task.

We want to know if $p_{\text{generated}} = p_{\text{training data}}$ $(= p_{\text{truth}}?)$

- If we have access to $p(x)$, we can compute f-divergences.
  ↳ Example: KL-divergence $\int dx\, p(x) \log \frac{p(x)}{q(x)}$
- Alternatively, we could use Integral Probability Metrics.
  ↳ Example: Wasserstein distance
- In Computer Science, one uses the Frechét Inception Distance.
- In HEP, we usually look at histograms.

See also: Kansal et al. [arXiv:2211.10295]

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

$\Rightarrow$ This captures the full phase space incl. correlations.

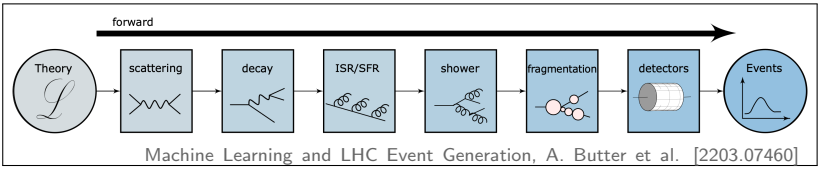
$\Rightarrow$ However, it is sufficient, but not neccessary.

? But why wasn't this used before?

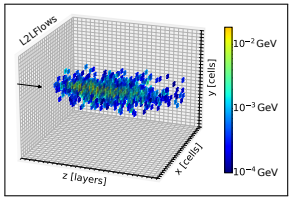
$\Rightarrow$ Previous deep generative models were separable to almost 100%!

DCTRGAN: Diefenbacher et al. [2009.03796, JINST]

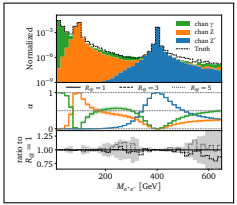
Fast and Faithful (Calorimeter) Simulations with Normalizing Flows.



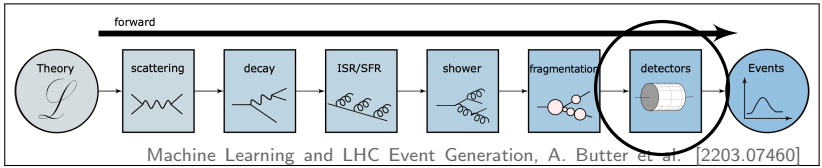
1: Calorimeter Simulation



2: Phase Space Integration

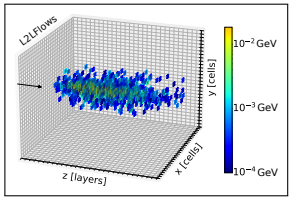


Fast and Faithful (Calorimeter) Simulations with Normalizing Flows.

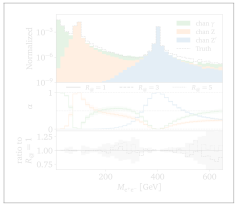


Machine Learning and LHC Event Generation, A. Butter et al. [2203.07460]

1: Calorimeter Simulation

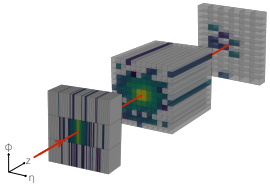


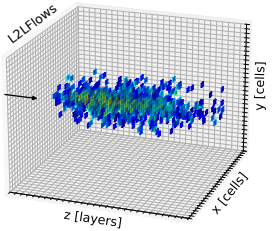
2: Phase Space Integration



Different Datasets come with different challenges

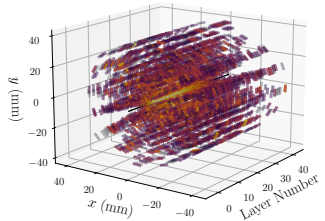
Part I: The CaloGAN Dataset





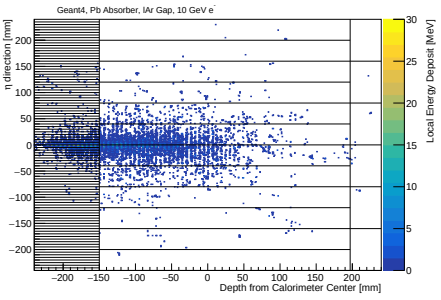
Part II: The ILD Dataset

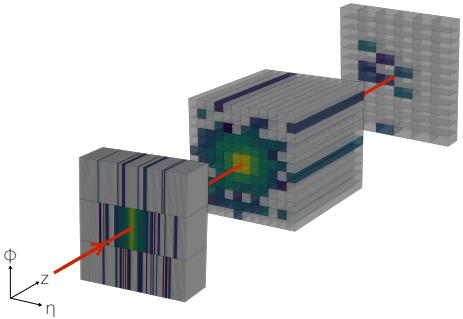
Part III: The CaloChallenge 2022



I: We use the same calorimeter geometry as CaloGAN

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

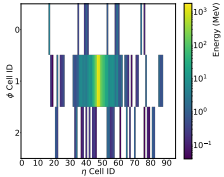


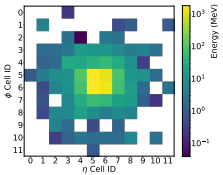


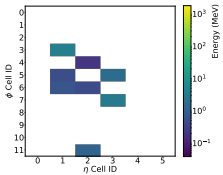
CaloGAN: Paganini, de Oliveira, Nachman [1705.02355, PRL; 1712.10321, PRD]

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: available at [DOI: 10.5281/zenodo.5904188]
- Showers of e^+, γ, and π^+ (100k each)
- All are centered and perpendicular
- E_{inc} 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]

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

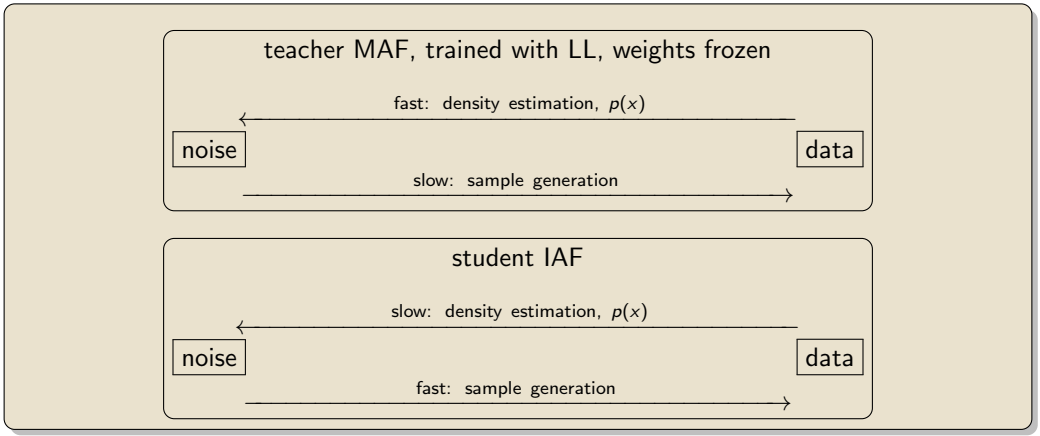
Flow I

- learns $p_1(E_0, E_1, E_2|E_{\text{inc}})$
- is a Masked Autoregressive Flow, optimized using the log-likelihood.

Flow II

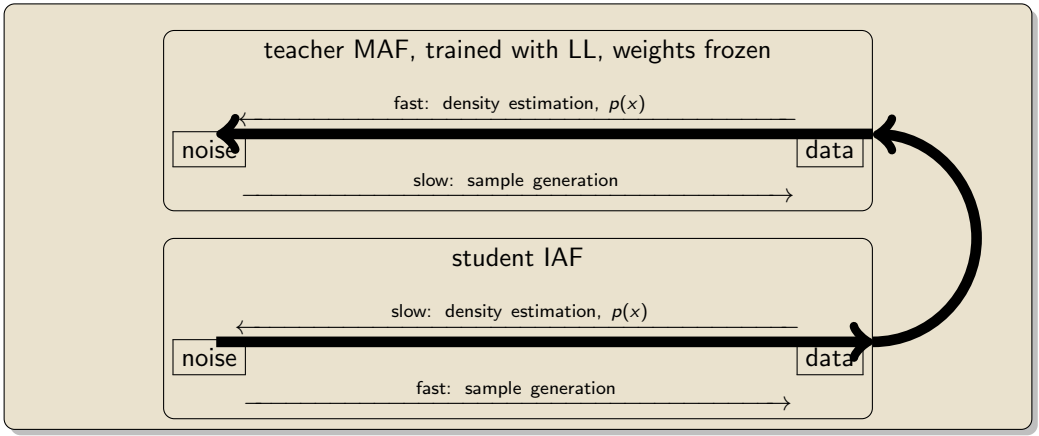
- learns $p_2(\hat{\vec{\mathcal{I}}}|E_0, E_1, E_2, E_{\text{inc}})$ of normalized showers
- in CaloFlow v1 (2106.05285 — called "teacher"):
  - Masked Autoregressive Flow trained with log-likelihood
  - Slow in sampling ($\approx 500\times$ slower than CaloGAN)
- in CaloFlow v2 (2110.11377 — called "student"):
  - Inverse Autoregressive Flow trained with Probability Density Distillation from teacher (log-likelihood prohibitive), i.e. matching IAF parameters to frozen MAF van den Oord et al. [1711.10433]
  - Fast in sampling ($\approx 500\times$ faster than CaloFlow v1)

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



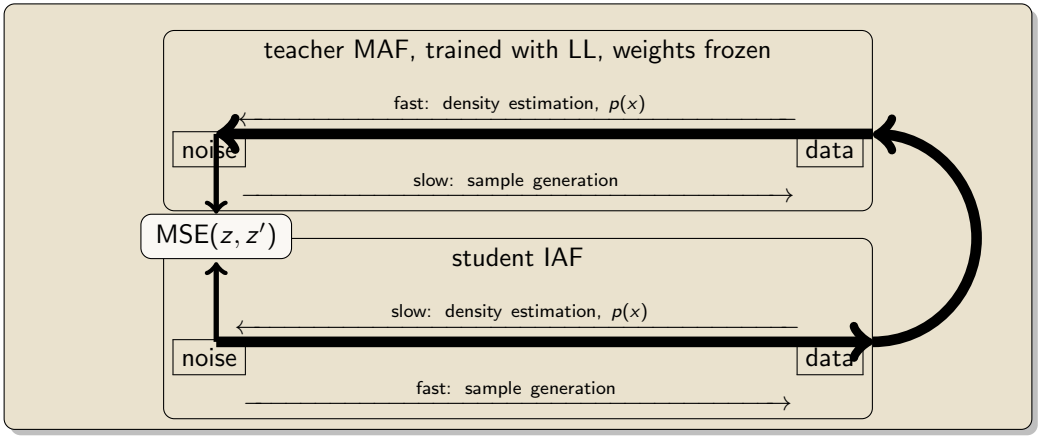
$$\begin{aligned}\text{Loss} = \text{MSE}(z, z') &+ \text{MSE}(x, x') + \text{MSE}(z_i, z_i') \\ &+ \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')\end{aligned}$$

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



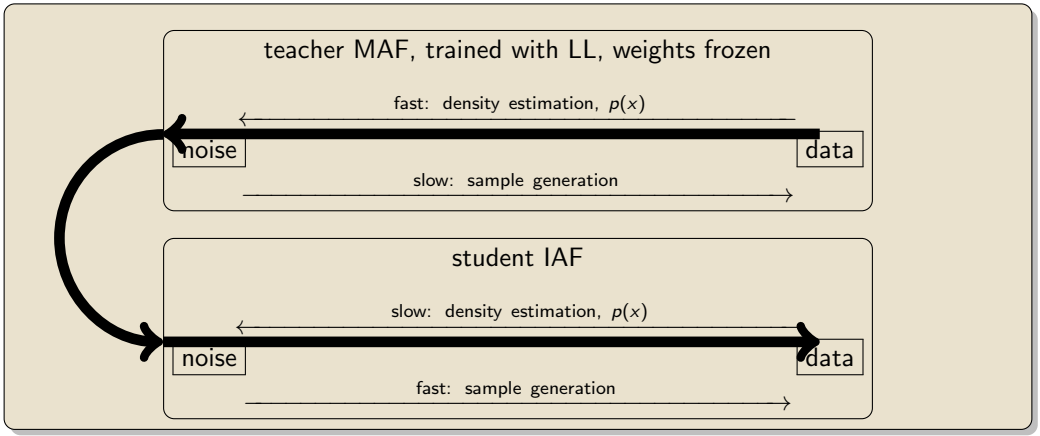
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

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



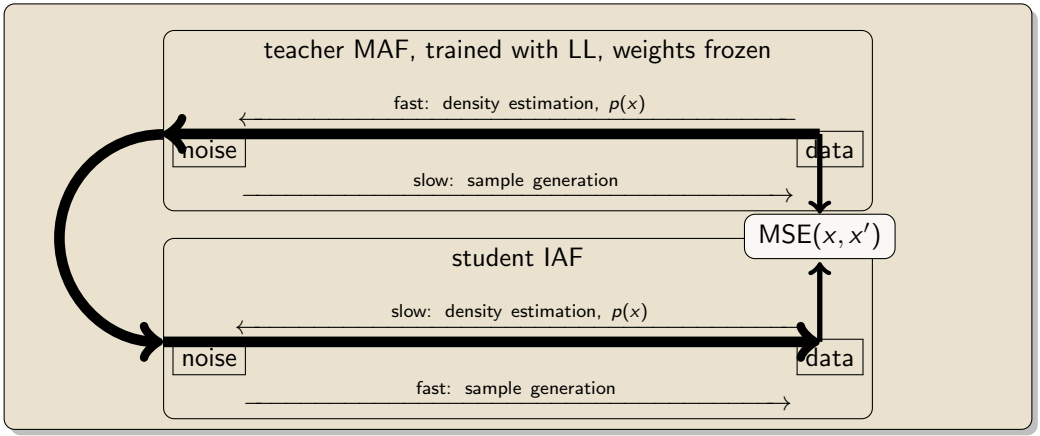
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

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



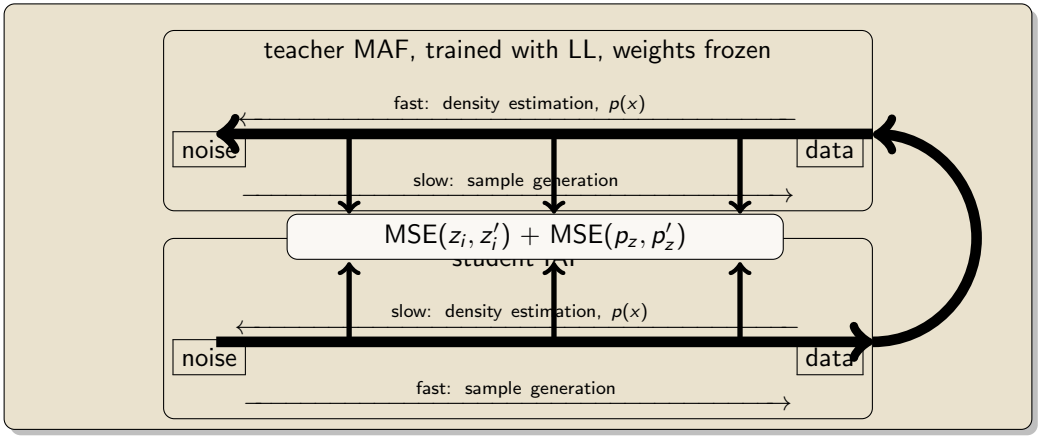
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

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



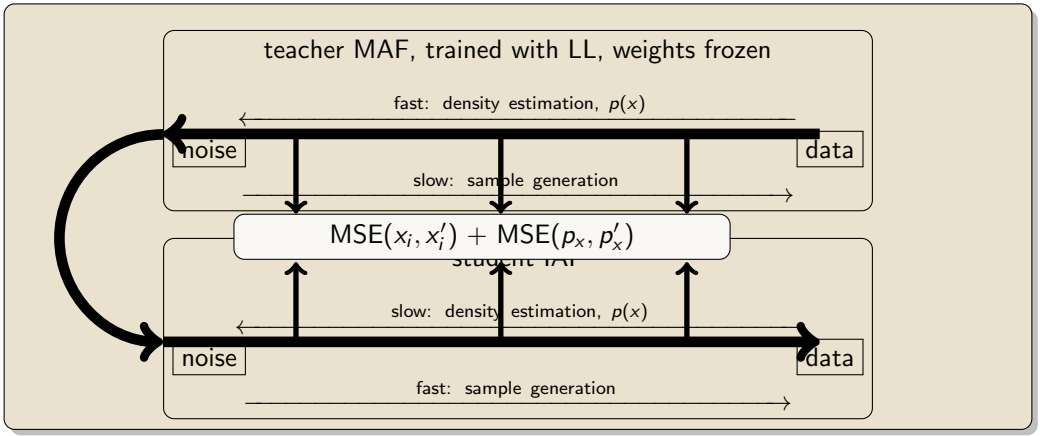
$$\text{Loss} = \text{MSE}(z, z') + \text{MSE}(x, x') + \text{MSE}(z_i, z_i') + \text{MSE}(x_i, x_i') + \text{MSE}(p_z, p_z') + \text{MSE}(p_x, p_x')$$

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



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

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



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

I: 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.

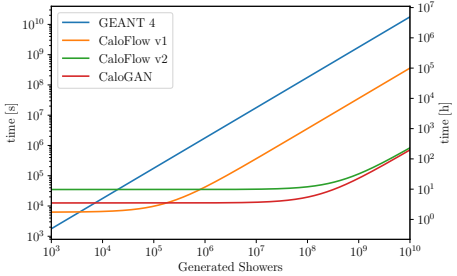
AUC		GEANT4 vs. CaloGAN	GEANT4 vs. (teacher) CaloFlow v1	GEANT4 vs. (student) CaloFlow v2
e^+	unnorm.	1.000(0)	0.859(10)	0.786(7)
	norm.	1.000(0)	0.870(2)	0.824(4)
	high-level	1.000(0)	0.795(1)	0.762(3)
γ	unnorm.	1.000(0)	0.756(48)	0.758(14)
	norm.	1.000(0)	0.796(2)	0.760(3)
	high-level	1.000(0)	0.727(2)	0.739(2)
π^+	unnorm.	1.000(0)	0.649(3)	0.729(2)
	norm.	1.000(0)	0.755(3)	0.807(1)
	high-level	1.000(0)	0.888(1)	0.893(2)

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

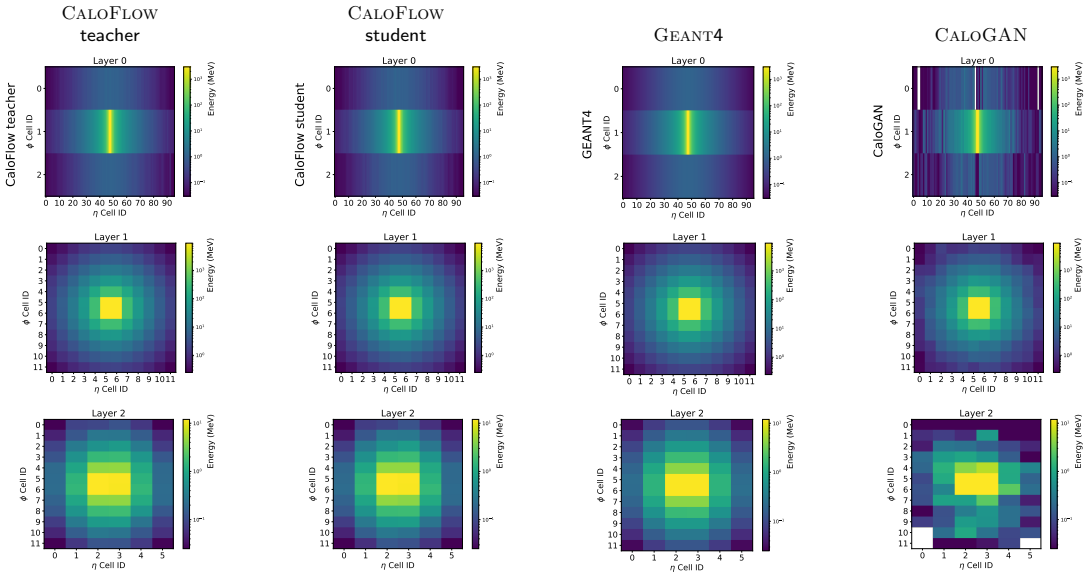
I: Sampling Speed: The Student beats the Teacher!

	CaloFlow*		CaloGAN*	Geant4†
	teacher	student		
training	22+82 min	+ 480 min	210 min	0 min
generation time per shower	36.2 ms	**0.08 ms**	**0.07 ms**	1772 ms

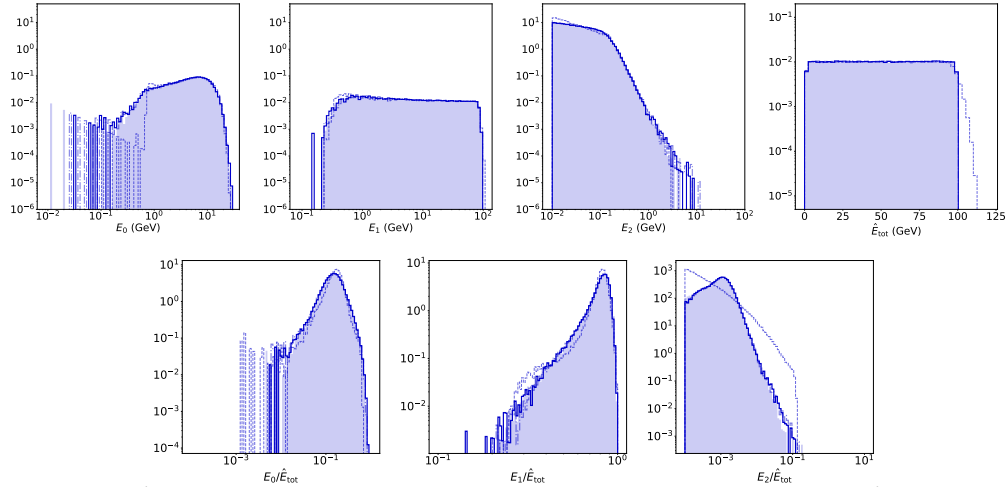
*: on our Titan V GPU, †: on the CPU of CaloGAN: Paganini, de Oliveira, Nachman [1712.10321, PRD]



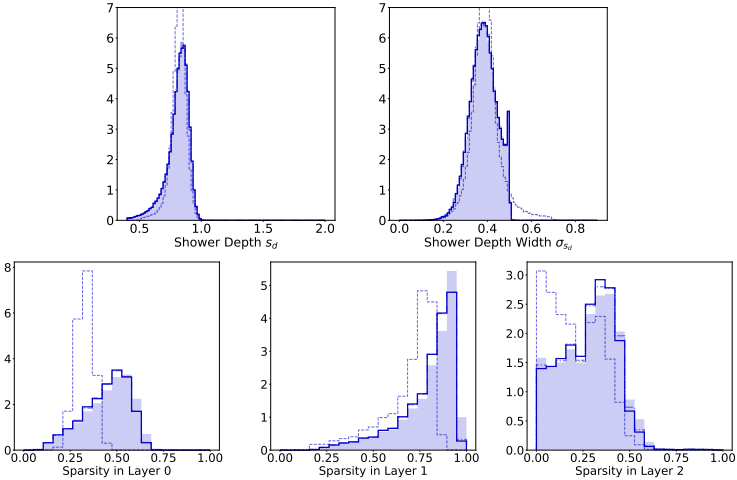
I: CaloFlow: Comparing Shower Averages: e^+



I: CaloFlow: histograms: e^+



I: CaloFlow: histograms: e^+



I: What about other architectures?

Work in progress with Florian Ernst, Luigi Favaro, Tilman Plehn, David Shih:

$\Rightarrow$ We compare the performance of a completely different architecture on the same dataset.

1. Replace the autoregressive flow with a coupling layer based one.
2. Do it in one step: learn $p(\hat{\mathcal{I}}_0, \hat{\mathcal{I}}_1, \hat{\mathcal{I}}_2, E_0, E_1, E_2 | E_{\text{inc}})$ (507-dim.)
3. Make it Bayesian to
   - see where the model is uncertain
   - have it regularized
   - have some kind of dropout

Bellagente, Haußmann, Luchmann, Plehn [2104.04543, SciPost]

I: What about other architectures?

Work in progress with Florian Ernst, Luigi Favaro, Tilman Plehn, David Shih:

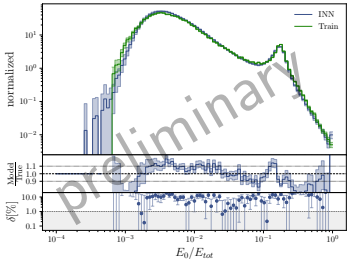
$\Rightarrow$ We compare the performance of a completely different architecture on the same dataset.

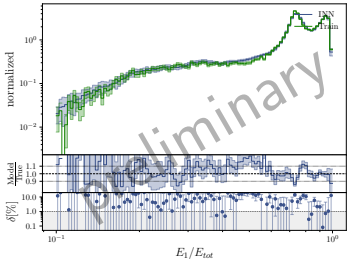
1. Replace the autoregressive flow with a coupling layer based one.
2. Do it in one step: learn $p(\hat{\mathcal{I}}_0, \hat{\mathcal{I}}_1, \hat{\mathcal{I}}_2, E_0, E_1, E_2 | E_{\text{inc}})$ (507-dim.)
3. Make it Bayesian to
   - see where the model is uncertain
   - have it regularized
   - have some kind of dropout

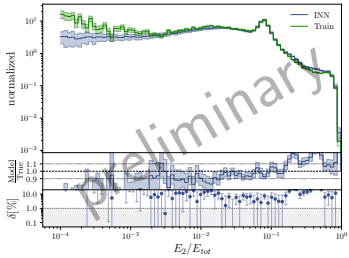
Replace NN weights θ with Gaussians and learn their mean and width.

Bellagente, Haußmann, Luchmann, Plehn [2104.04543, SciPost]

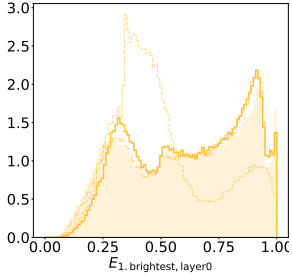
I: Tails are more uncertain.

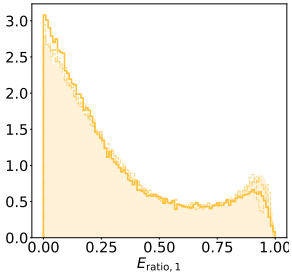


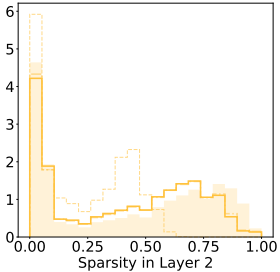


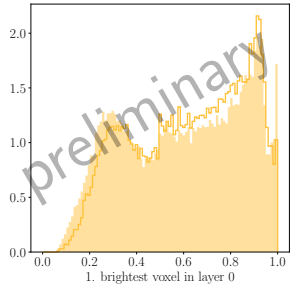


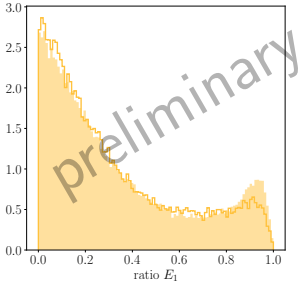
I: Comparing preliminary π^+ histograms

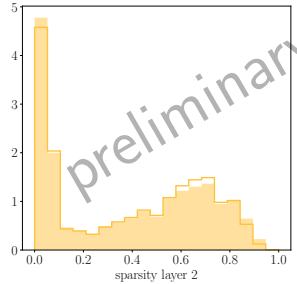






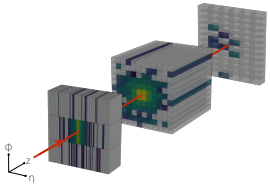


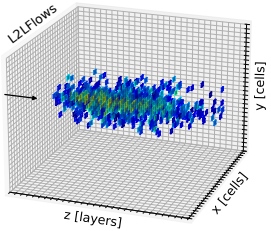




Different Datasets come with different challenges

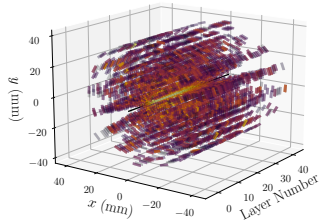
Part I: The CaloGAN Dataset





Part II: The ILD Dataset

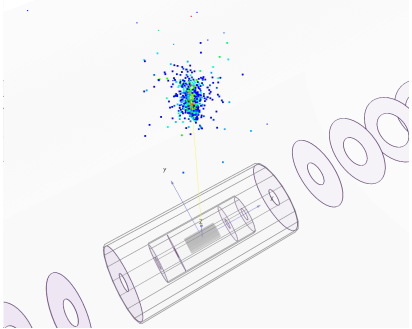
Part III: The CaloChallenge 2022

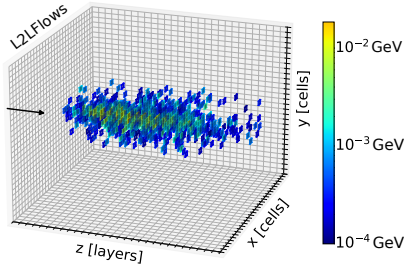


II: How does it scale to more voxels?

Work in Progress with Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Imahn Shekhzadeh, David Shih:

- We consider a the prototype of the ILD: alternating layers of tungsten and silicon
- We consider photon showers with $E_{\text{inc}} \in [10, 100]$ GeV
- We discretize the data to $30 \times 30 \times 30$ voxel and focus on the central $10 \times 10 \times 30$





E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, K. Krüger [2005.05334, Comput.Softw.Big Sci.]

II: We also use a 2-step approach to learn $p(\vec{\mathcal{I}}|E_{\text{inc}})$, like CaloFlow.

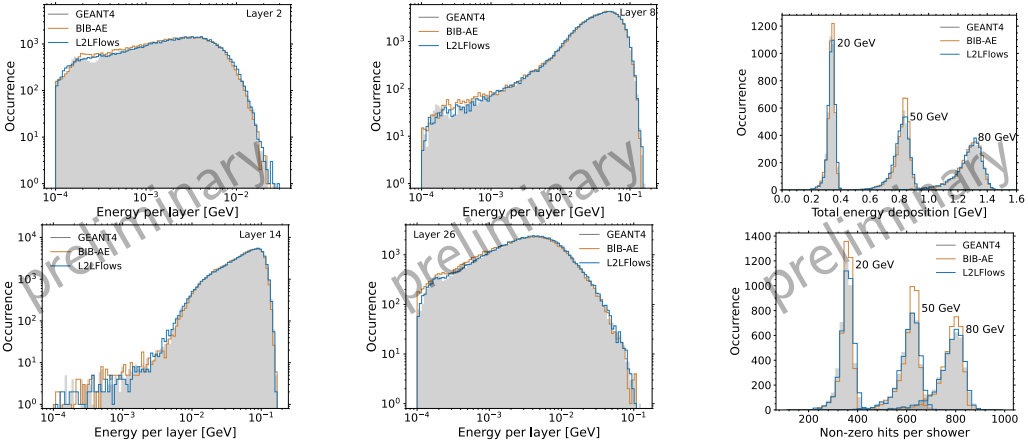
Step I: Energy Distribution Flow

- learns $p(E_0, E_1, E_2, \ldots, E_{29}|E_{\text{inc}})$
- is a Masked Autoregressive Flow, optimized using the log-likelihood.

Step II: Causal Flows

- use one Flow for each calorimeter layer (30 in total)
- learn $p_i(\mathcal{I}_i|\mathcal{I}_{i-n_{\text{cond}}}, \ldots, \mathcal{I}_{i-1}, E_0, \ldots, E_{29}, E_{\text{inc}})$ of normalized showers.
- use an embedding net to condense the information of $\mathcal{I}_{i-n_{\text{cond}}}, \ldots, \mathcal{I}_{i-1}$.
- are Masked Autoregressive Flows, optimized using the log-likelihood.
- can be parallelized on 30 machines in training.

II: L2LFlows: preliminary results

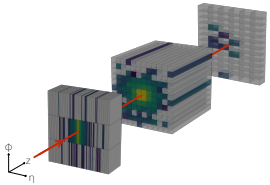


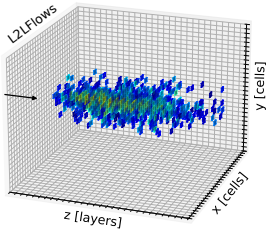
Classifier AUCs:

BIB-AE: 0.995(3) L2LFlows: 0.852(4)

Different Datasets come with different challenges

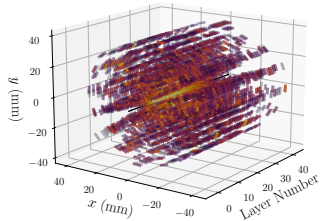
Part I: The CaloGAN Dataset





Part II: The ILD Dataset

Part III: The CaloChallenge 2022



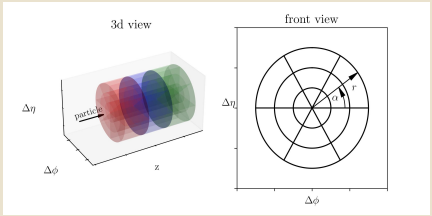
III: Going the next step: towards deployment in FastSimulation

⇒ need a survey of current approaches on a common dataset!

⇒ Fast Calorimeter Challenge 2022

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

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



- Dataset 1: AtlFast3 trainig data (γ: 368, π: 533 voxels)
  [2109.02551, Comput.Softw.Big Sci.]
- Dataset 2: simulated detector (e^-: 6480 voxels)
- Dataset 3: simulated detector (e^-: 40500 voxels)

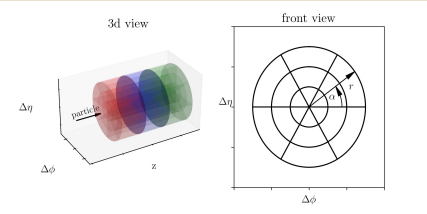
III: Going the next step: towards deployment in FastSimulation

⇒ need a survey of current approaches on a common dataset!

⇒ Fast Calorimeter Challenge 2022

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

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



- Dataset 1: AtlFast3 trainig data (γ: 368, π: 533 voxels)
  - ↳ Applied CaloFlow with Ian Pang and David Shih [2109.02551, Comput.Softw.Big Sci.]
- Dataset 2: simulated detector (e^-: 6480 voxels)
  - ↳ Work in progress with Matt Buckley, Ian Pang and David Shih
- Dataset 3: simulated detector (e^-: 40500 voxels)
  - ↳ Work in progress with Matt Buckley, Ian Pang and David Shih

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

Flow I

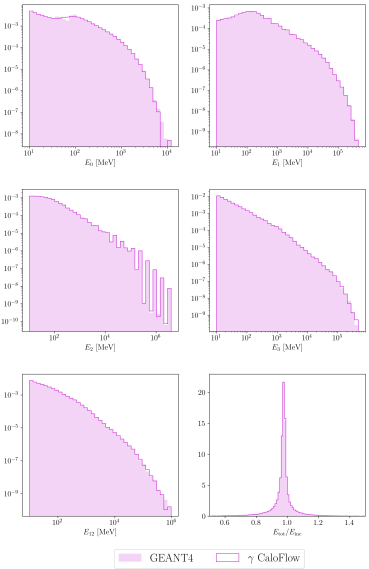
- learns $p_1(E_0, E_1, E_2, \ldots | E_{\text{inc}})$
- is a Masked Autoregressive Flow, optimized using the log-likelihood.

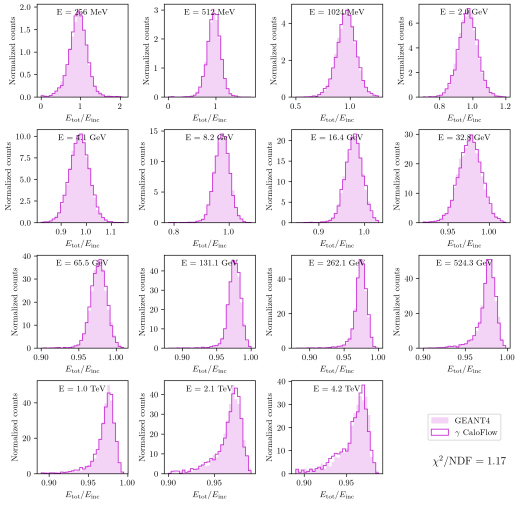
Flow II

- learns $p_2(\hat{\vec{\mathcal{I}}}|E_0, E_1, E_2, \ldots, E_{\text{inc}})$ of normalized showers
- in CaloFlow v1 (called "teacher"):
  - Masked Autoregressive Flow trained with log-likelihood
  - Slow in sampling
- in CaloFlow v2 (called "student"):
  - Inverse Autoregressive Flow trained with Probability Density Distillation from teacher (log-likelihood prohibitive), i.e. matching IAF parameters to frozen MAF van den Oord et al. [1711.10433]
  - Fast in sampling ($\approx 300 - 500\times$ faster than CaloFlow v1)

III: CaloFlow on CaloChallenge dataset 1: γ

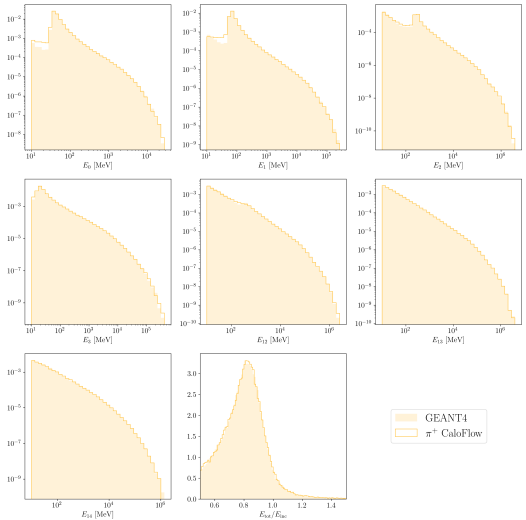
C.K., Ian Pang, David Shih [2210.14245]



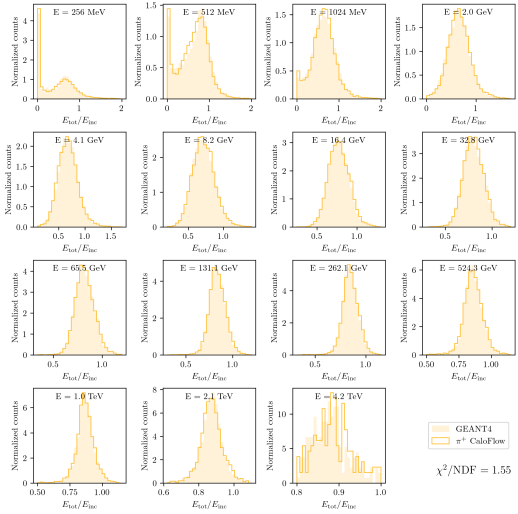


ATLAS χ^2/NDF = 13.5, AtlFast3 [2109.02551, Comput.Softw.Big Sci.]

III: CaloFlow on CaloChallenge dataset 1: π^+



C.K., Ian Pang, David Shih [2210.14245]



ATLAS $\chi^2/\text{NDF} = 12.7$, AtlFast3 [2109.02551, Comput.Softw.Big Sci.]

III: CaloFlow on CaloChallene dataset 1: numbers

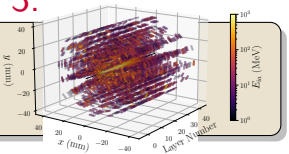
AUC		DNN based classifier	
		Geant4 vs. CaloFlow (teacher)	Geant4 vs. CaloFlow (student)
γ	low-level	0.701(3)	0.739(3)
	high-level	0.551(3)	0.556(3)
π^+	low-level	0.779(1)	0.854(2)
	high-level	0.698(2)	0.726(3)

Particle type	Batch size	Student generation time per event	
		GPU	CPU
γ	1	57.23 ms	115.88 ms
	10000	0.07 ms	-
π^+	1	74.09 ms	126.05 ms
	10000	0.09 ms	-

III: Moving on to datasets 2 and 3.

CaloChallenge datasets 2 and 3 are much bigger:

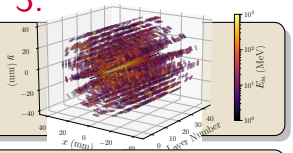
- Dataset 2: 144 voxels in 45 layers $\rightarrow$ 6480 total.
- Dataset 3: 900 voxels in 45 layers $\rightarrow$ 40500 total.



III: Moving on to datasets 2 and 3.

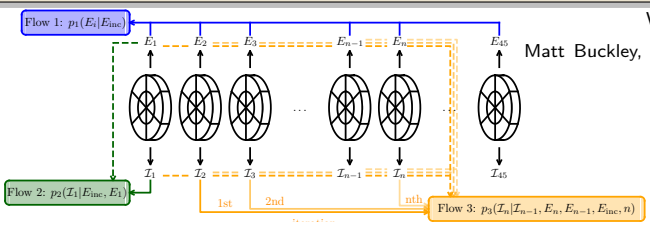
CaloChallenge datasets 2 and 3 are much bigger:

- Dataset 2: 144 voxels in 45 layers $\rightarrow$ 6480 total.
- Dataset 3: 900 voxels in 45 layers $\rightarrow$ 40500 total.



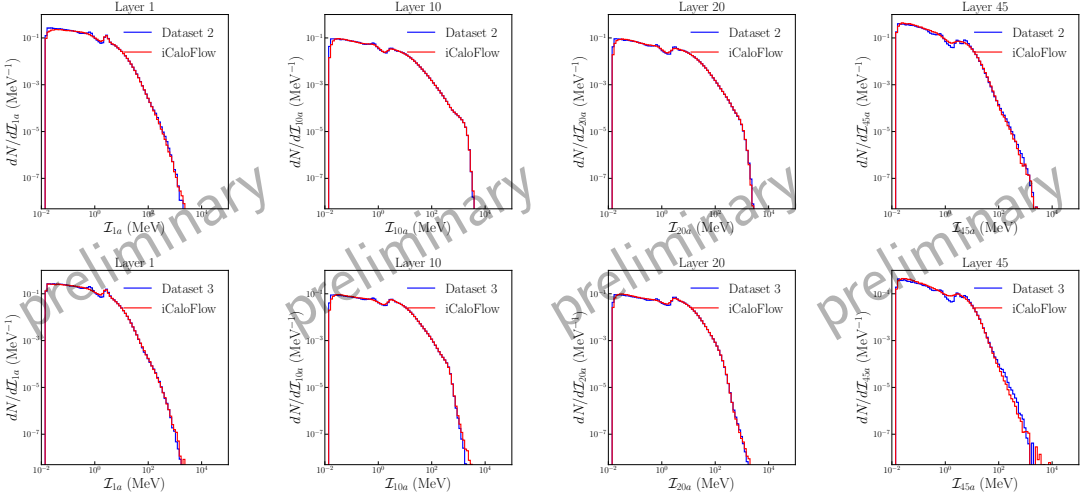
iCaloFlow: Split learning $p(\vec{\mathcal{I}}|E_{\text{inc}})$ into 3 steps, leveraging the regular detector geometry.

1. learns $p_1(E_1, E_2, E_3, \ldots, E_{45}|E_{\text{inc}})$ $\rightarrow$ how energy is distributed among layers.
2. learns $p_2(\mathcal{I}_1|E_1, E_{\text{inc}})$ $\rightarrow$ how the shower in the first layer looks like.
3. learns $p_3(\mathcal{I}_n|\mathcal{I}_{n-1}, n, E_n, E_{n-1}, E_{\text{inc}})$ $\rightarrow$ how the shower in layer n looks like, given layer $n-1$



Work in progress with Matt Buckley, Ian Pang, David Shih

III: iCaloFlow: results



Classifier AUCs:				
	dataset 2, low:	0.797(5)	dataset 2, high:	0.798(3)
	dataset 3, low:	0.911(3)	dataset 3, high:	0.941(1)

Conclusions Part 1: Calorimeter Simulations

⇒ Normalizing Flows are able to generate high-quality showers, outperforming other generative models.

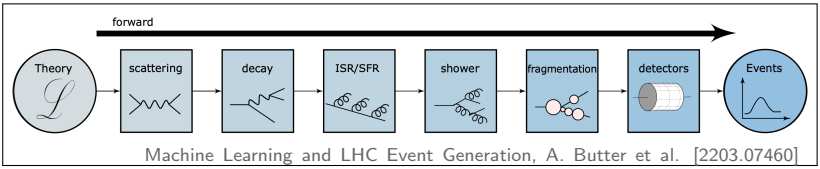
⇒ Training and model-selection is usually more stable.

⇒ The naive scaling to higher dimensions requires a lot of compute. But some assumptions on the underlying physics can help reduce the needed ressources.

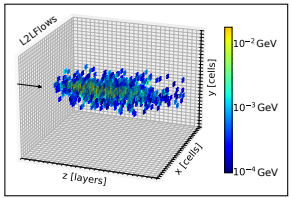
Open issues / points of discussion:

- Is learning at the voxel-level the best choice?
- How can we scale this to a full detector simulation?
- Can we run everything batch-wise?

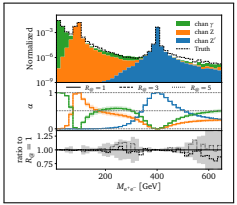
Fast and Faithful (Calorimeter) Simulations with Normalizing Flows.



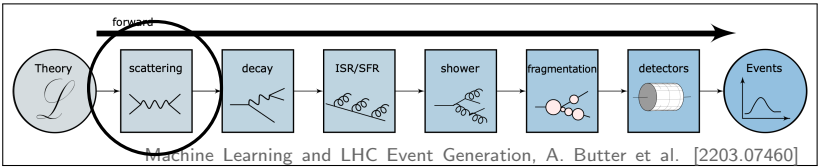
1: Calorimeter Simulation



2: Phase Space Integration

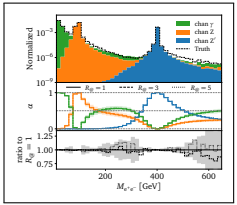


Fast and Faithful (Calorimeter) Simulations with Normalizing Flows.



1: Calorimeter Simulation

2: Phase Space Integration



Phase Space integration uses Importance Sampling.

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

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

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 $d\sigma(p_i, \vartheta_i, \varphi_i)$

Phase Space integration uses Importance Sampling.

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

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

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

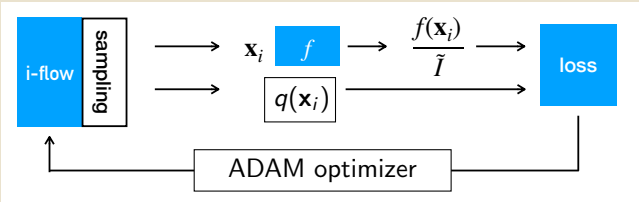
We therefore have to find a $q(\vec{x})$ that approximates the shape of $f(\vec{x})$.

$\Rightarrow$ 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.

i-flow: Numerical Integration with Normalizing Flows.

How it works:

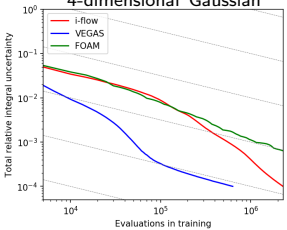


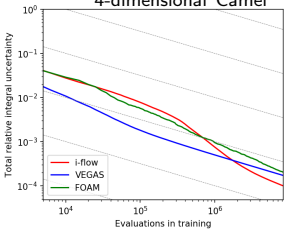
i-flow: C. Gao, J. Isaacson, CK [arXiv:2001.05486, ML:ST]
gitlab.com/i-flow/i-flow

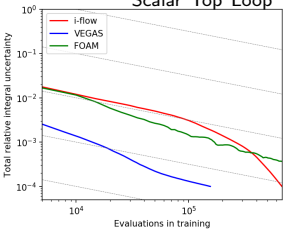
Statistical Divergences are used as loss functions:

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

i-flow adapts better, but needs more iterations.







Event Generators need a high-dimensional integrator.

We use Sherpa (**S**imulation of **H**igh-**E**nergy **R**eactions of **PA**rticles) 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.

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

Similar statements apply to MadGraph, too.

https://sherpa.hepforge.org/

Event Generators need a high-dimensional integrator.

We use Sherpa (**S**imulation of **H**igh-**E**nergy **R**eactions of **PA**rticles) 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.

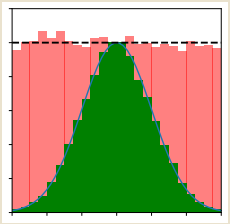
$$\Rightarrow n_{dim} = \underbrace{3n_{final} - 4}_{\text{kinematics}} + \underbrace{n_{final} - 1}_{\text{multichannel}}$$

Similar statements apply to MadGraph, too.

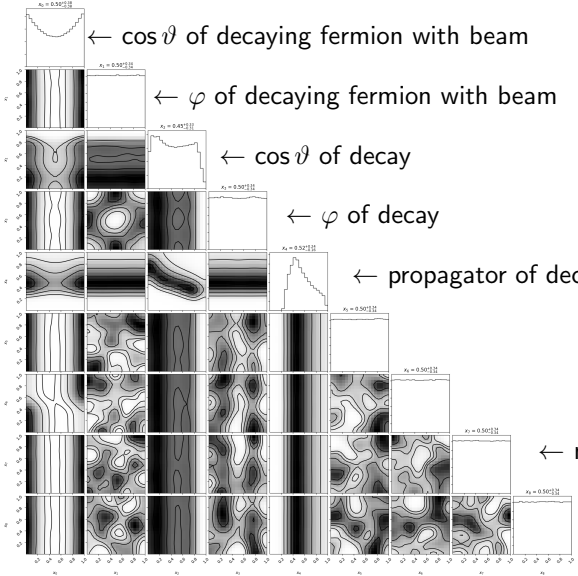
https://sherpa.hepforge.org/

Figure of merit: Unweighting efficiency

- 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.



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



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

$\leftarrow \varphi$ of decaying fermion with beam

Target distribution

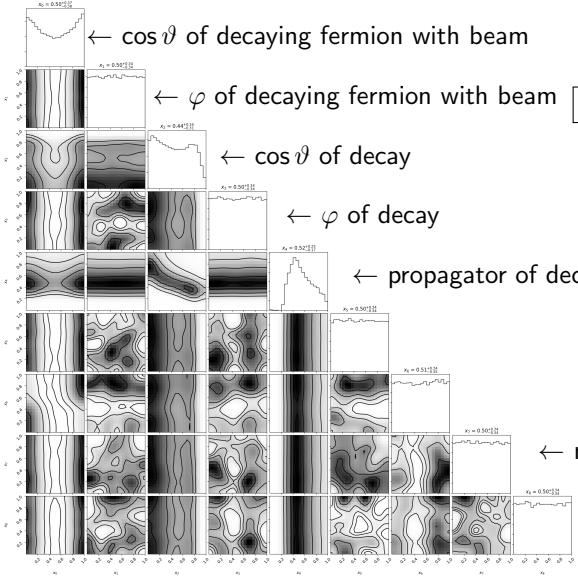
$\leftarrow \cos\vartheta$ of decay

$\leftarrow \varphi$ of decay

$\leftarrow$ propagator of decaying fermion

$\leftarrow$ multichannel

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



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

← φ of decaying fermion with beam

Learned distribution

← $\cos\vartheta$ of decay

← φ of decay

← propagator of decaying fermion

← multichannel

High Multiplicities are difficult to learn in this setup.

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]

High Multiplicities are difficult to learn in this setup.

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**

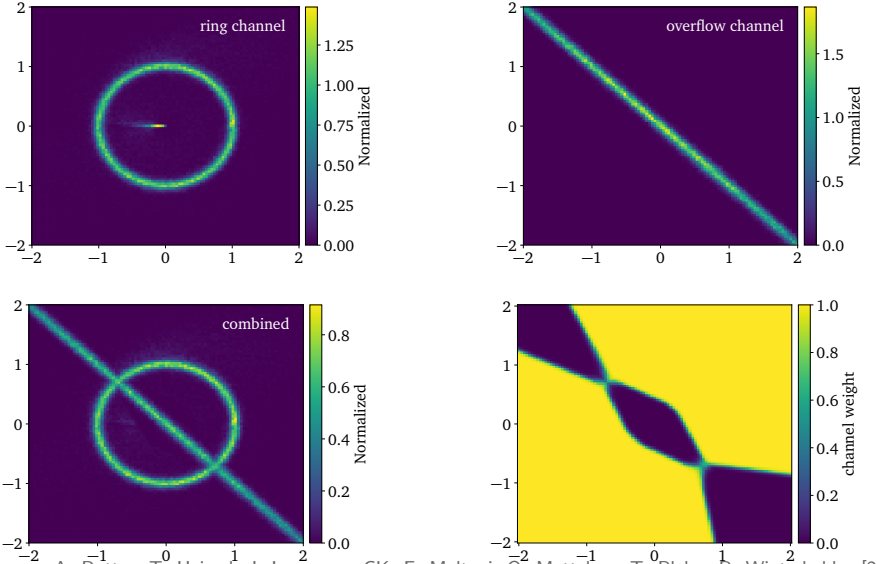
C. Gao, S. Höche, J. Isaacson, CK, H. Schulz [arXiv:2001.10028, PRD]

That cries for improvements:

$\Rightarrow$ MadNIS — Neural Multi-Channel Importance Sampling

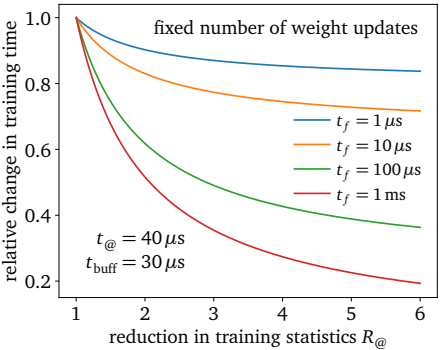
A. Butter, T. Heimel, J. Isaacson, CK, F. Maltoni, O. Mattelaer, T. Plehn, R. Winterhalder [2212.06172]

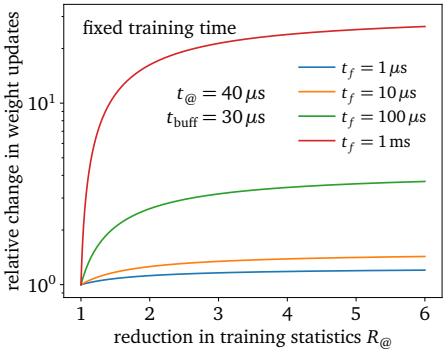
MadNIS learns the channel weights.



A. Butter, T. Heimel, J. Isaacson, CK, F. Maltoni, O. Mattelaer, T. Plehn, R. Winterhalder [2212.06172]

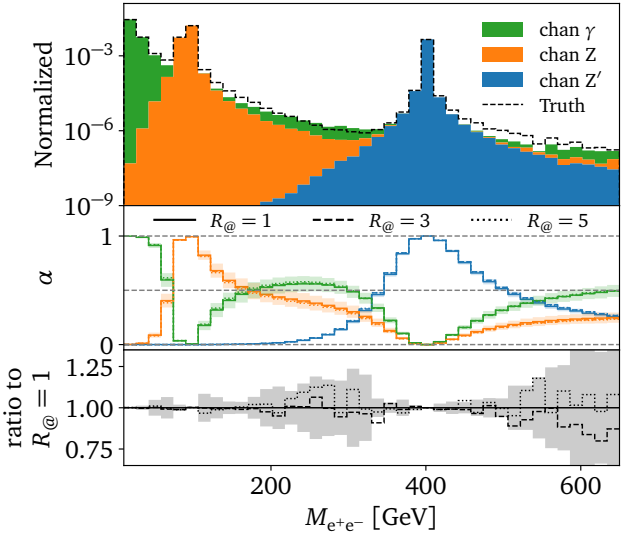
MadNIS re-uses expensive matrix elements





A. Butter, T. Heimel, J. Isaacson, CK, F. Maltoni, O. Mattelaer, T. Plehn, R. Winterhalder [2212.06172]

MadNIS: putting everything together, the Z' example



A. Butter, T. Heimel, J. Isaacson, CK, F. Maltoni, O. Mattelaer, T. Plehn, R. Winterhalder [2212.06172]

Conclusions Part 2: Phase Space Integration

$\Rightarrow$ Normalizing Flows are perfect for Importance Sampling.

$\Rightarrow$ They don't introduce a bias in the result, only increase the uncertainty if not converged.

$\Rightarrow$ They can be combined with other parts of MadGraph / Sherpa.

Open issues / points of discussion:

- How is the timing of the full chain?
- What is missing to be production-ready?