Overview of Machine Learning for Particle Physics

Benjamin Nachman

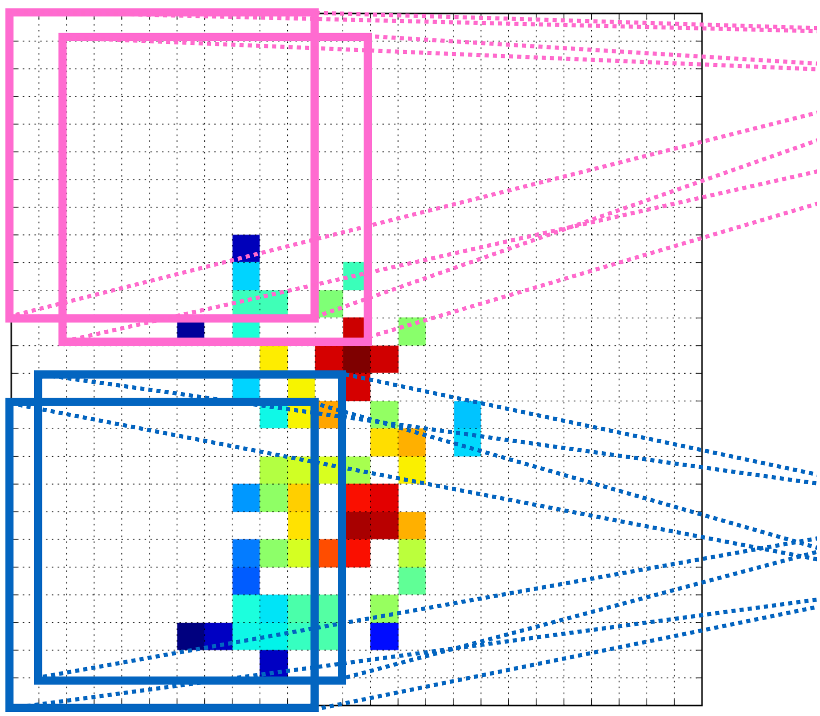
Lawrence Berkeley National Laboratory

bpnachman.com

bpnachman@lbl.gov

@bpnachman

bnachman



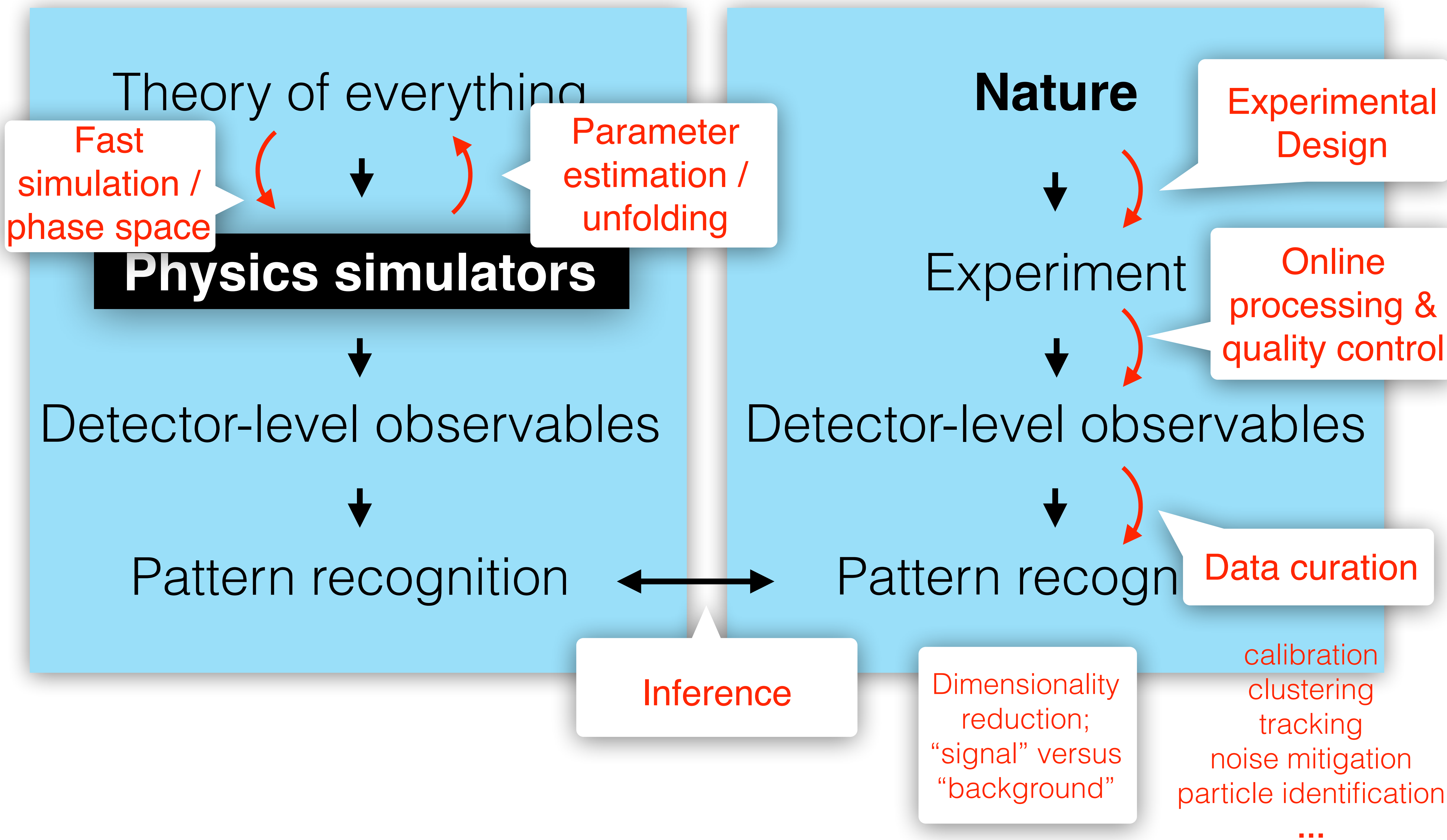
US ATLAS ML Training

July 26, 2023

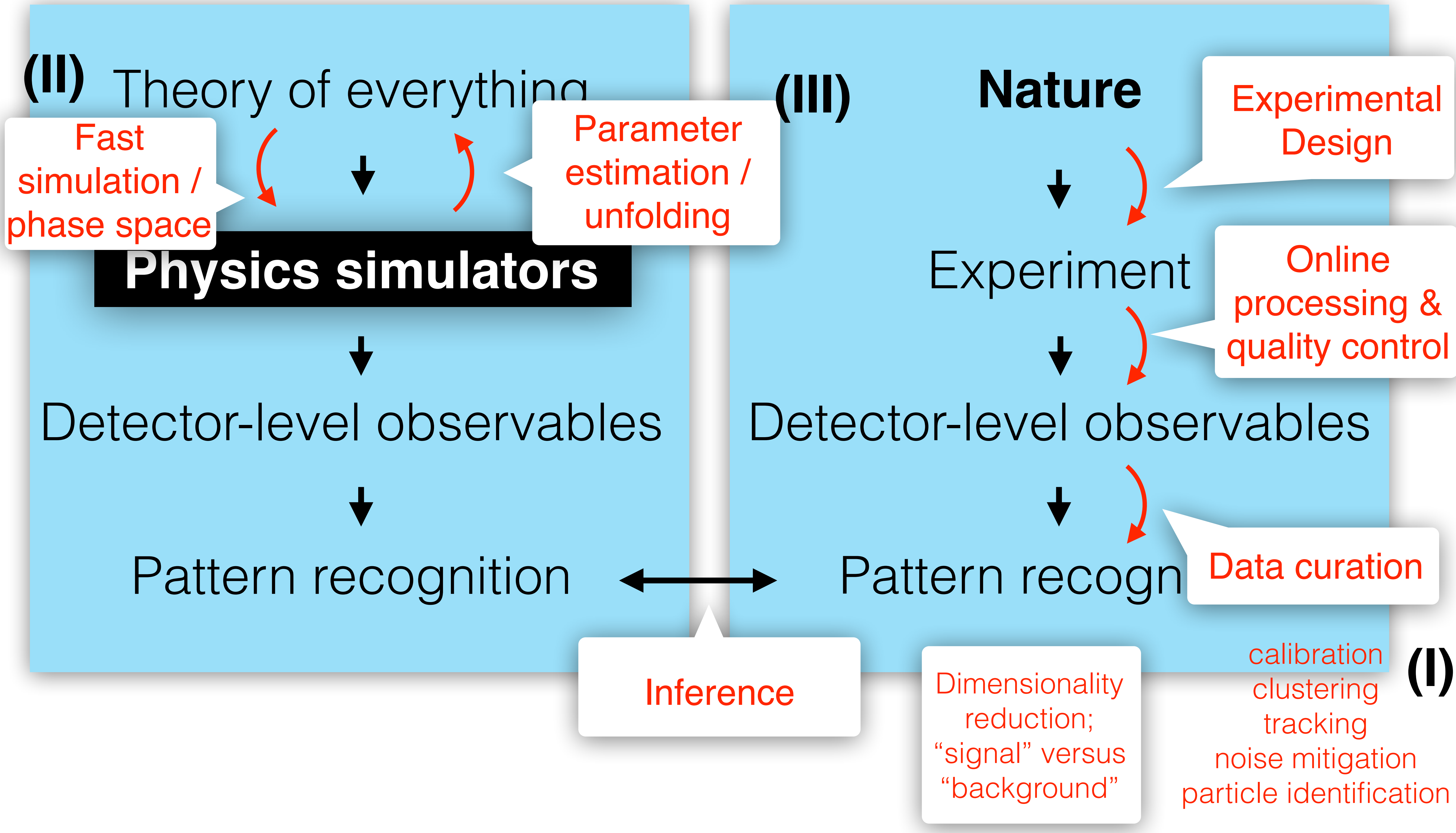
Particle Physics

Theory of everything	**Nature**
↓	↓
Physics simulators	Experiment
↓	↓
Detector-level observables	Detector-level observables
↓	↓
Pattern recognition ⟷	Pattern recognition

Particle Physics + Machine Learning



Particle Physics + Machine Learning



Particle Physics + Machine Learning

Theory of everything

↓

Physics simulators

↓

Detector-level observables

↓

Pattern recognition ⟷

Nature

↓

Experiment

↓

Detector-level observables

↓

Pattern recogn

Data curation

Dimensionality reduction; “signal” versus “background”

calibration
clustering
tracking
noise mitigation
particle identification
...

(I)

See 1709.04464 for image refs.

Step 1: how to represent our data

6

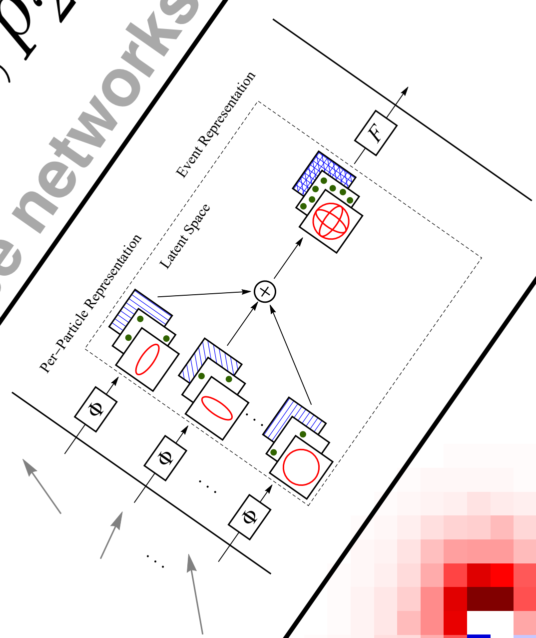
Fixed sets

$$J = \{p_1^\mu, p_2^\mu, \ldots, p_n^\mu\}$$

Dense networks

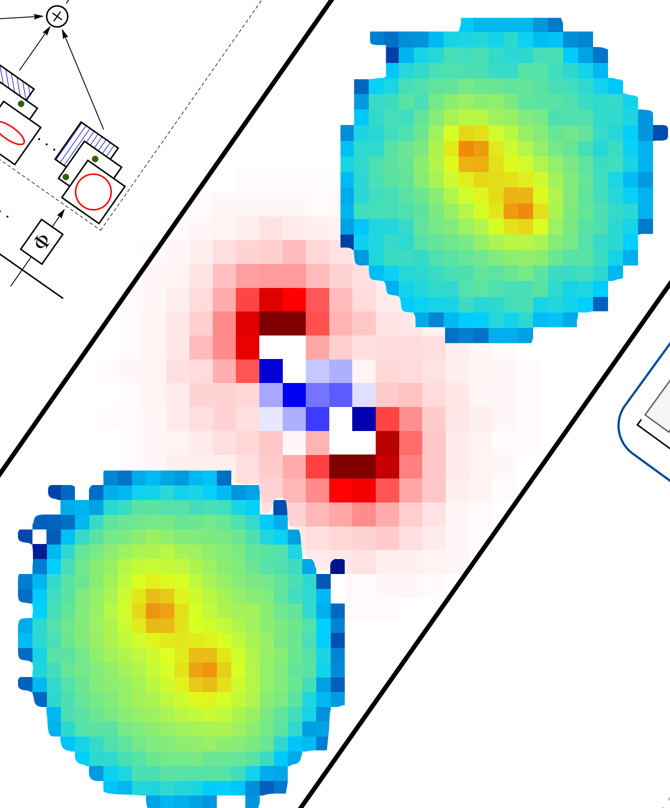
Variable sets

Deep sets



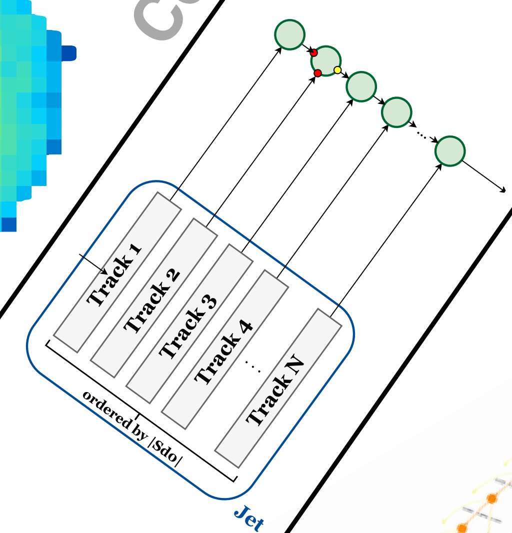
Images

Convolutional NNs (CNNs)



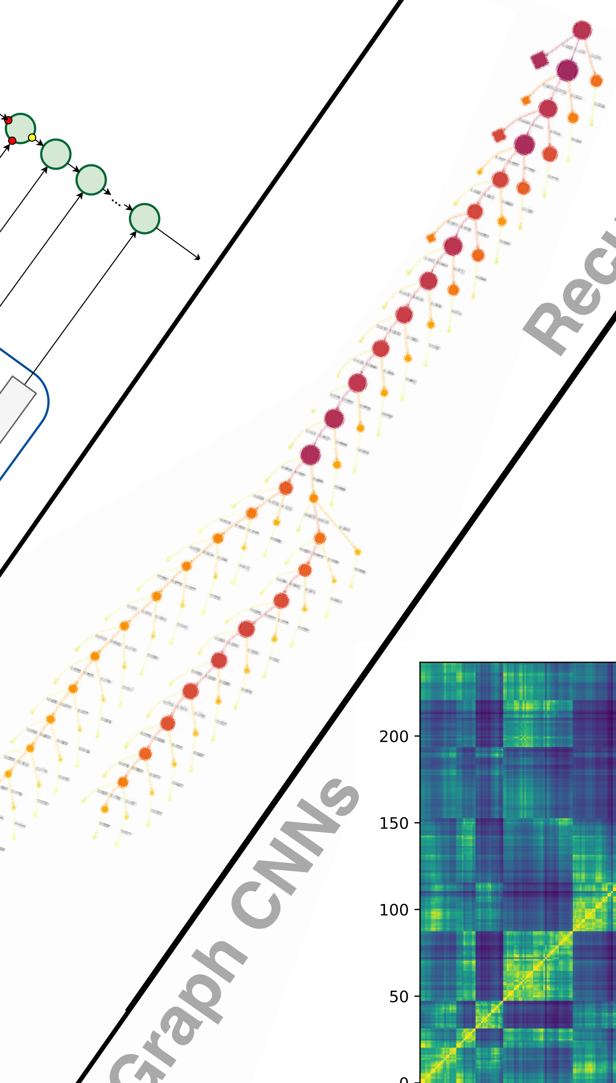
Sequences

Recurrent NNs



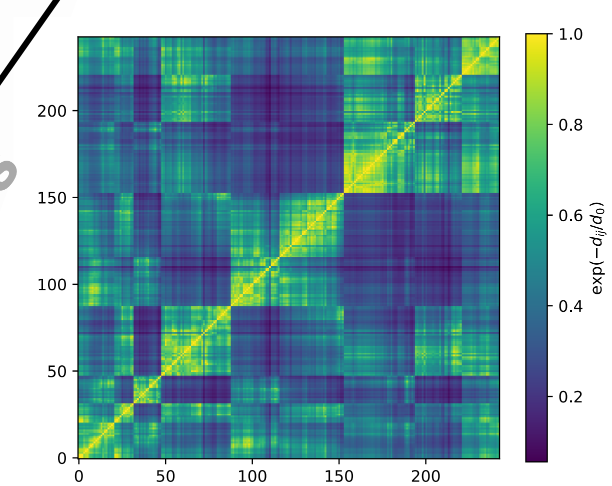
Trees

Recursive NNs



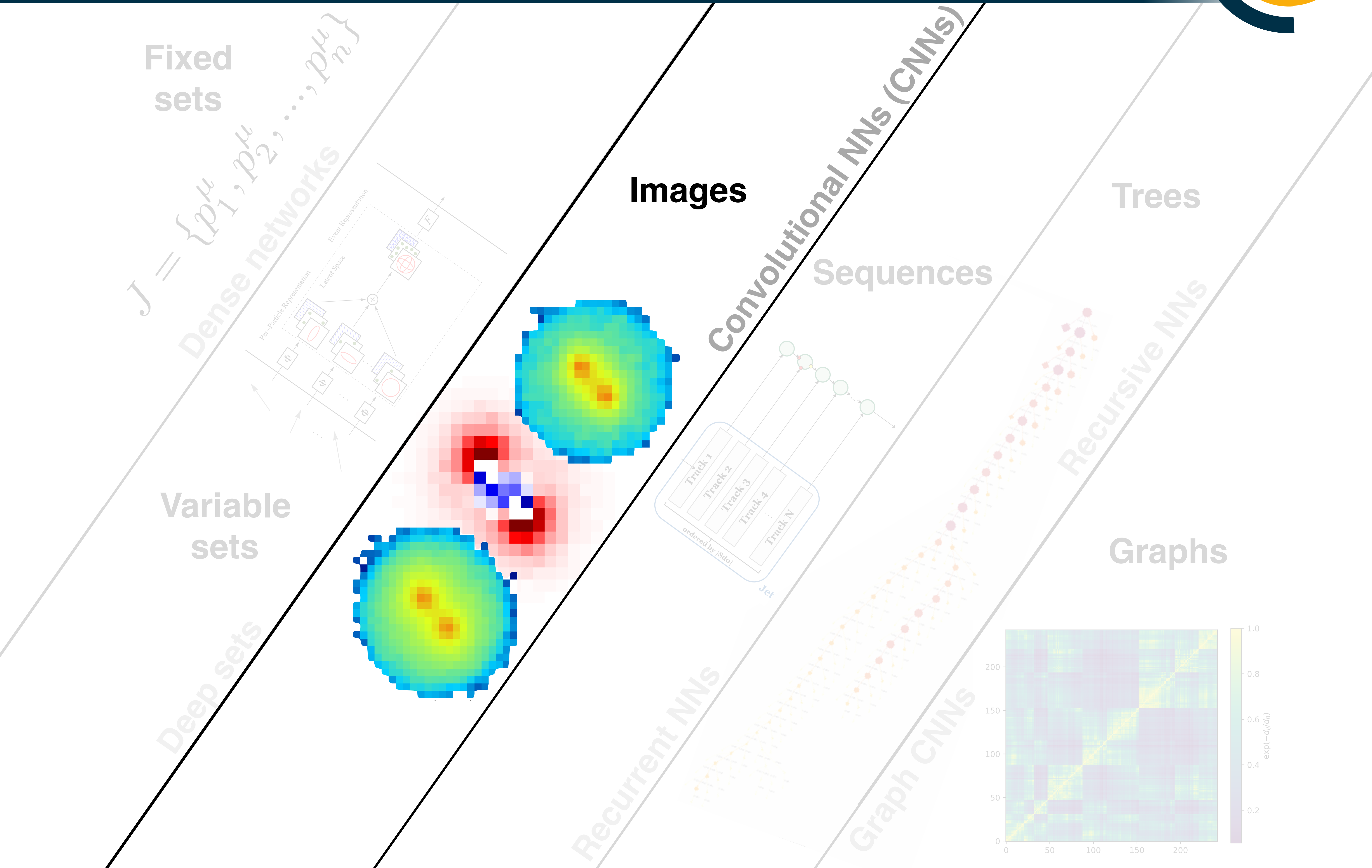
Graphs

Graph CNNs

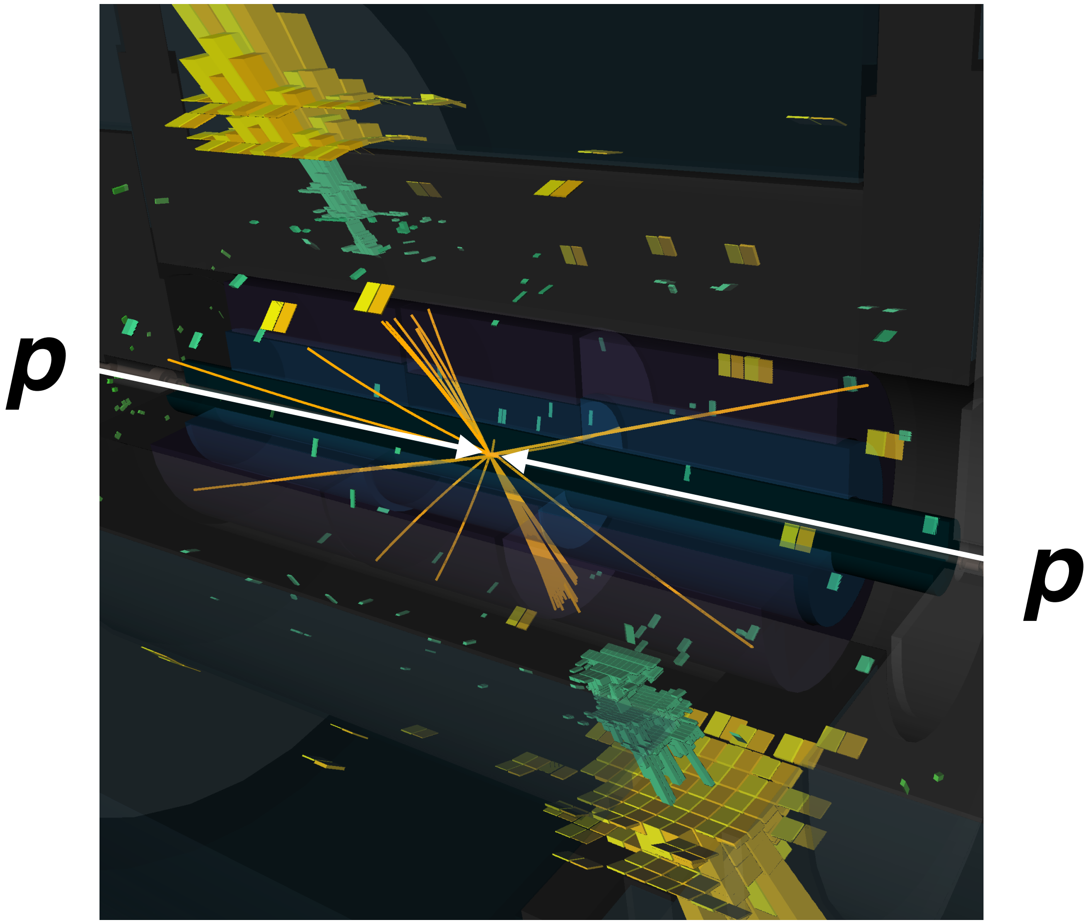


See 1709.04464 for image refs.

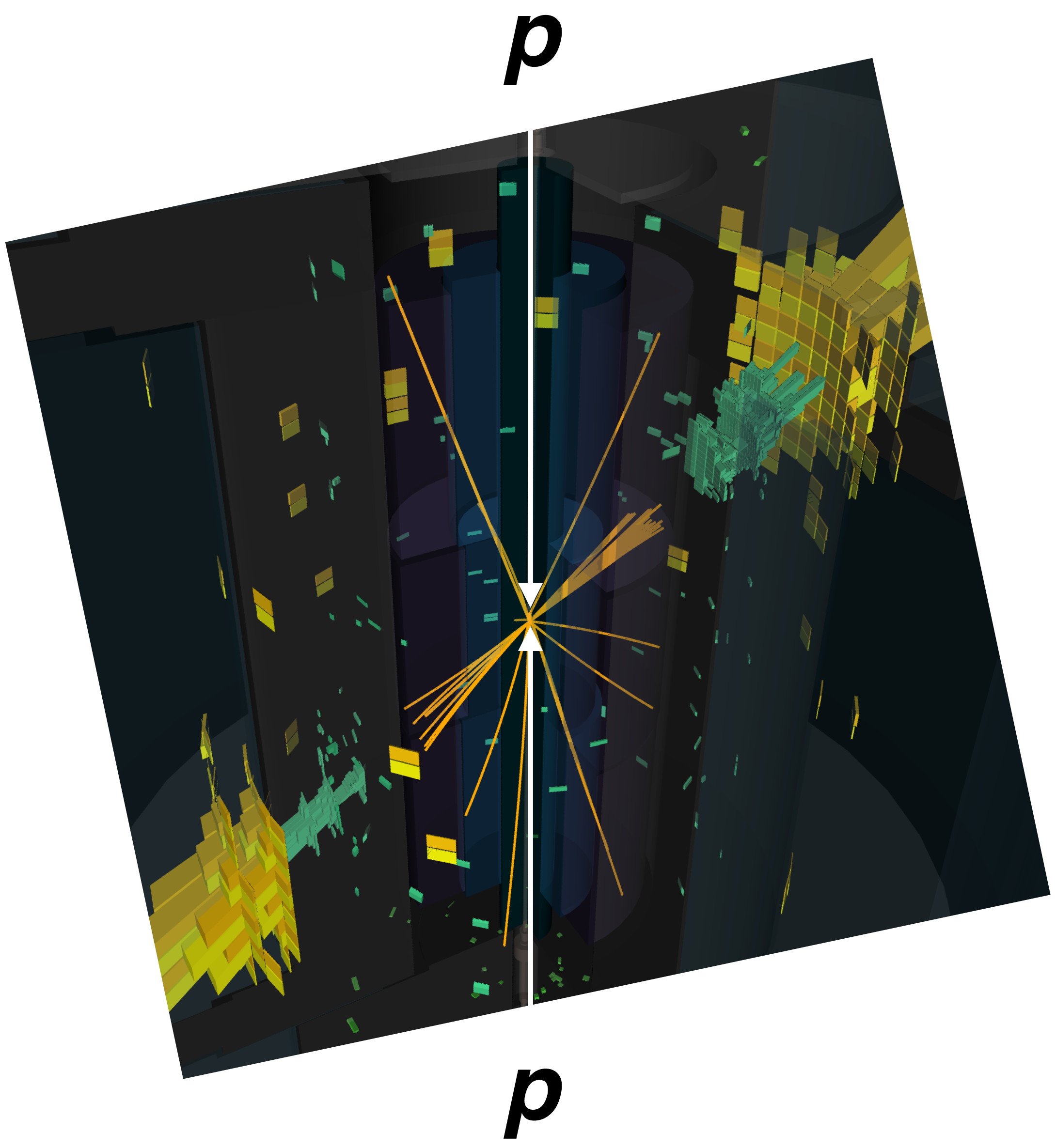
Step 1: how to represent our data



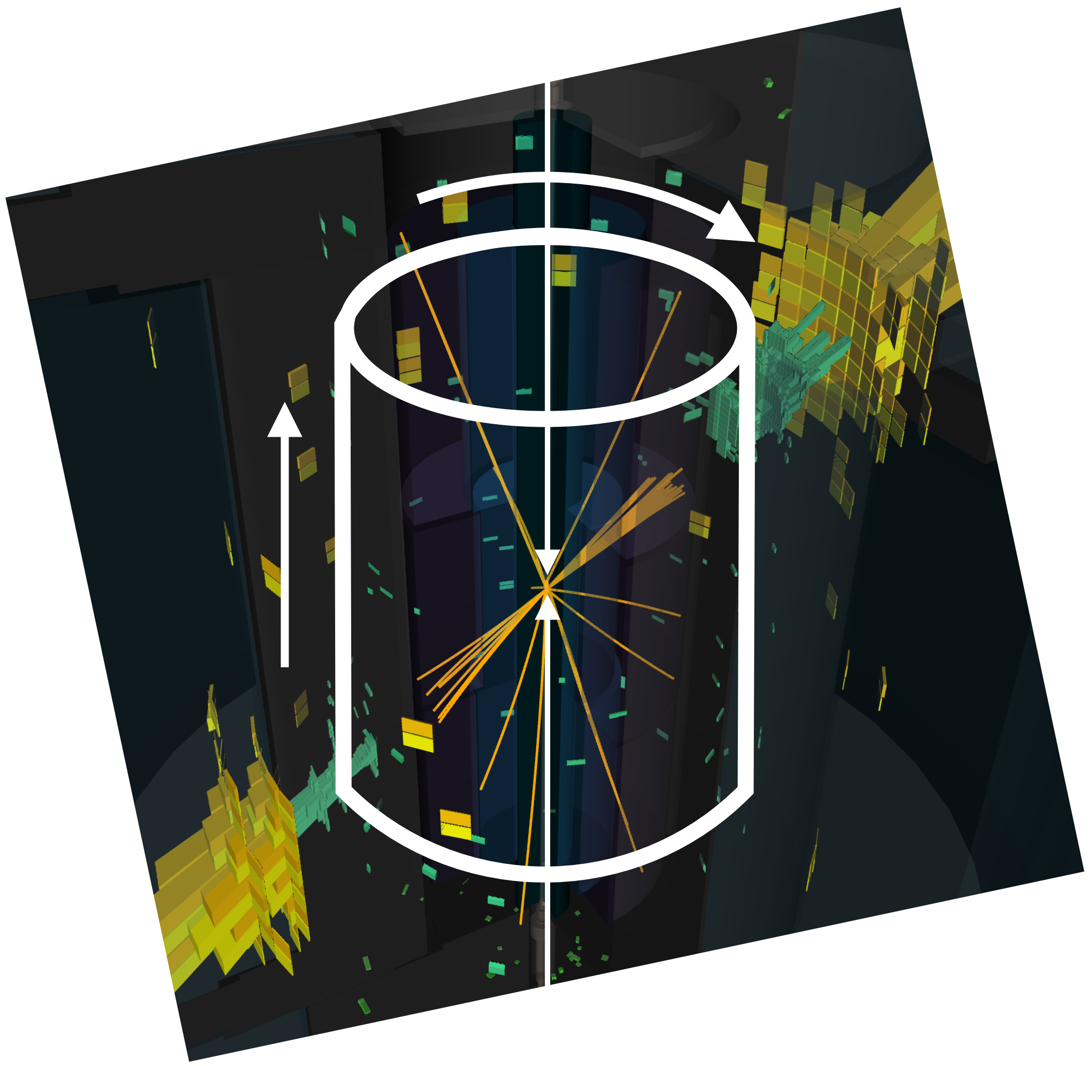
HEP data as an image



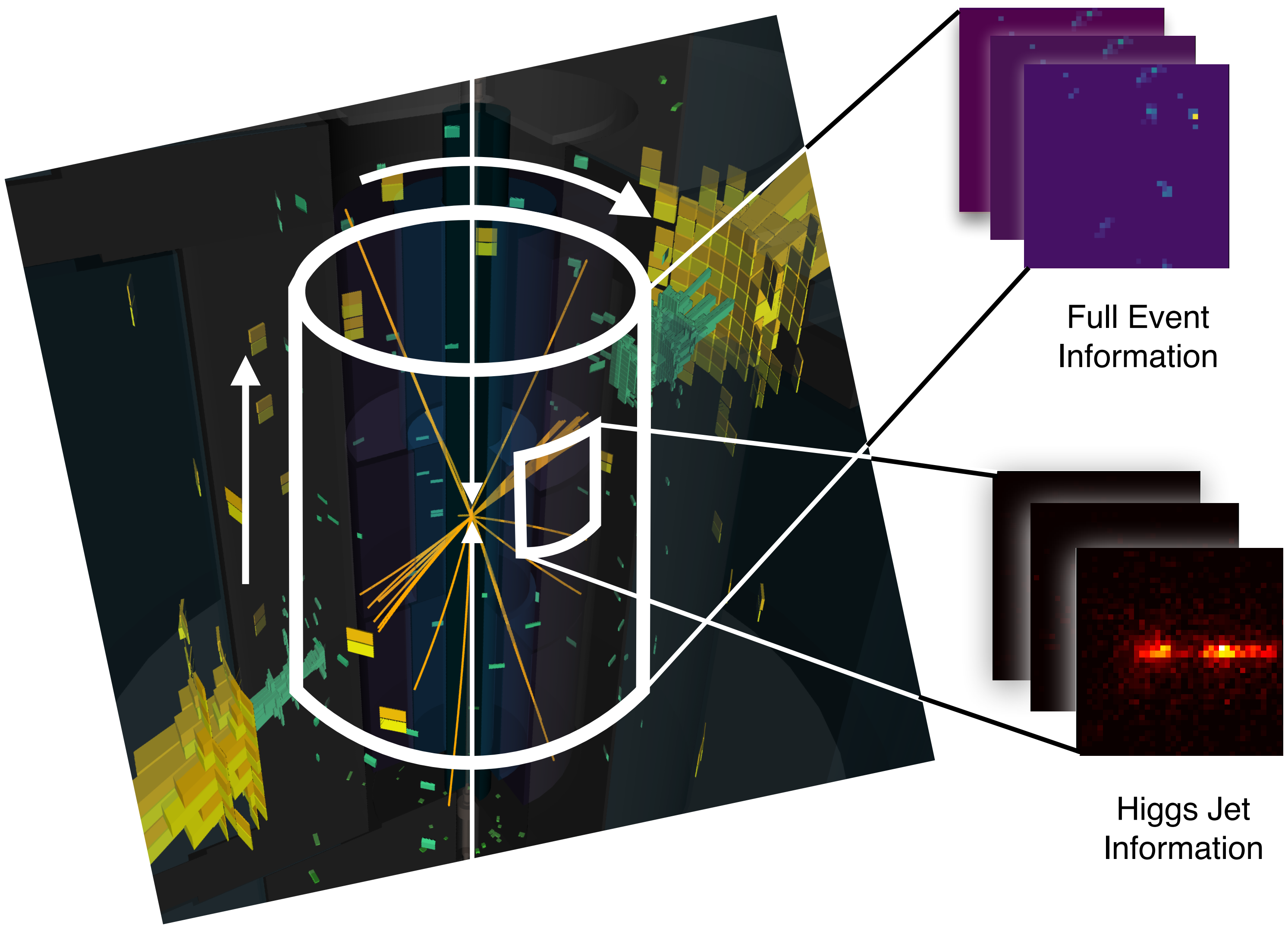
HEP data as an image



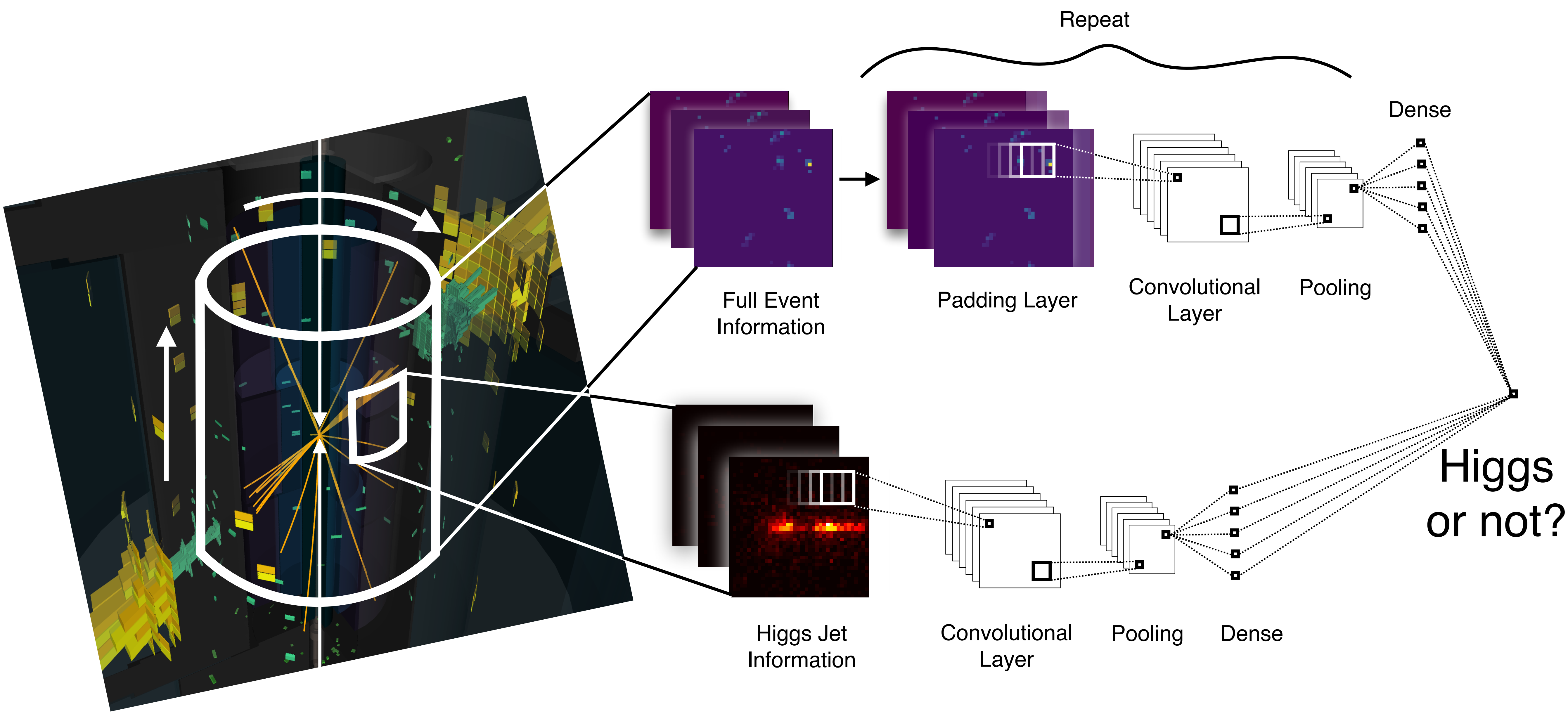
HEP data as an image



HEP data as an image



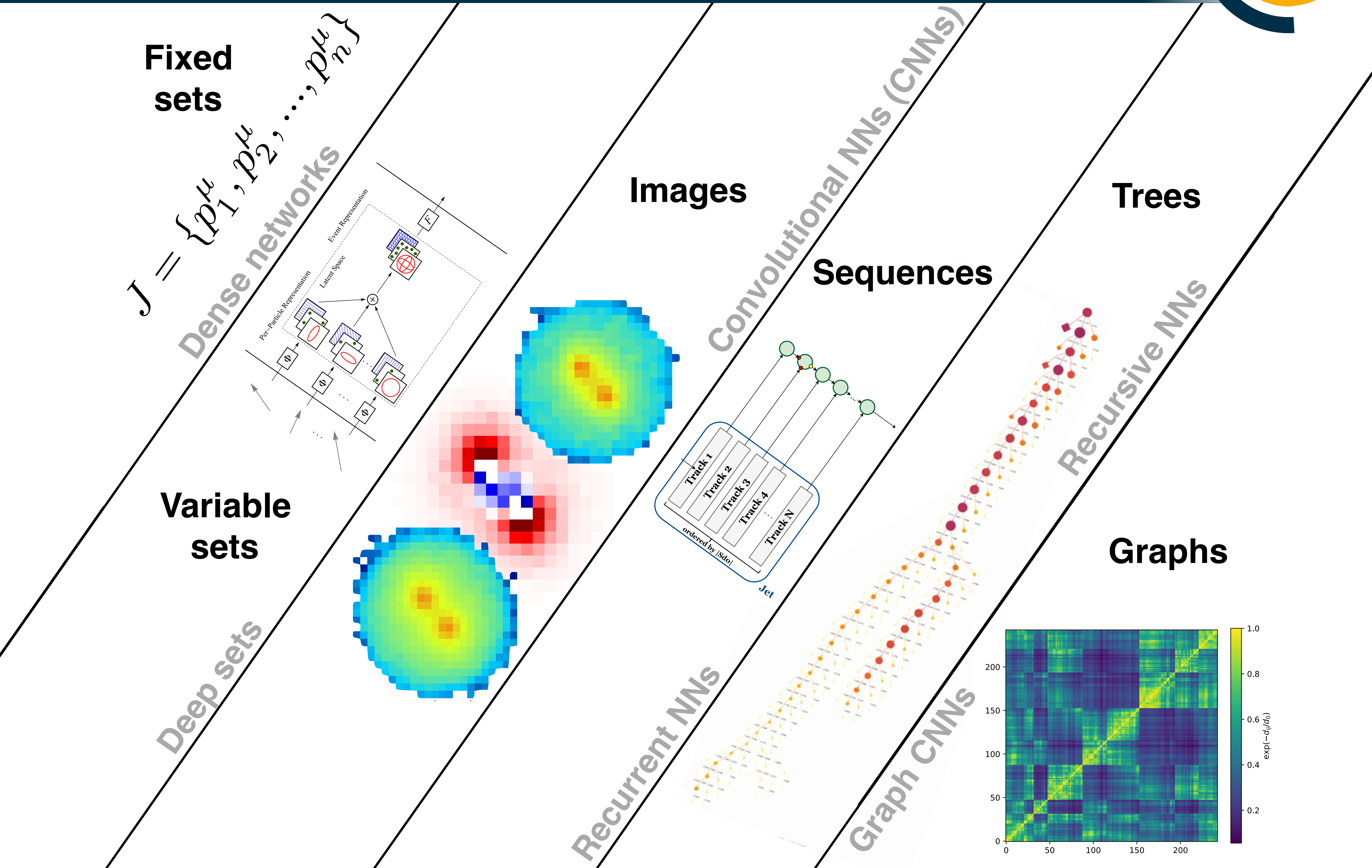
HEP data as an image



Can combine local and global information from jet images and “event” images.

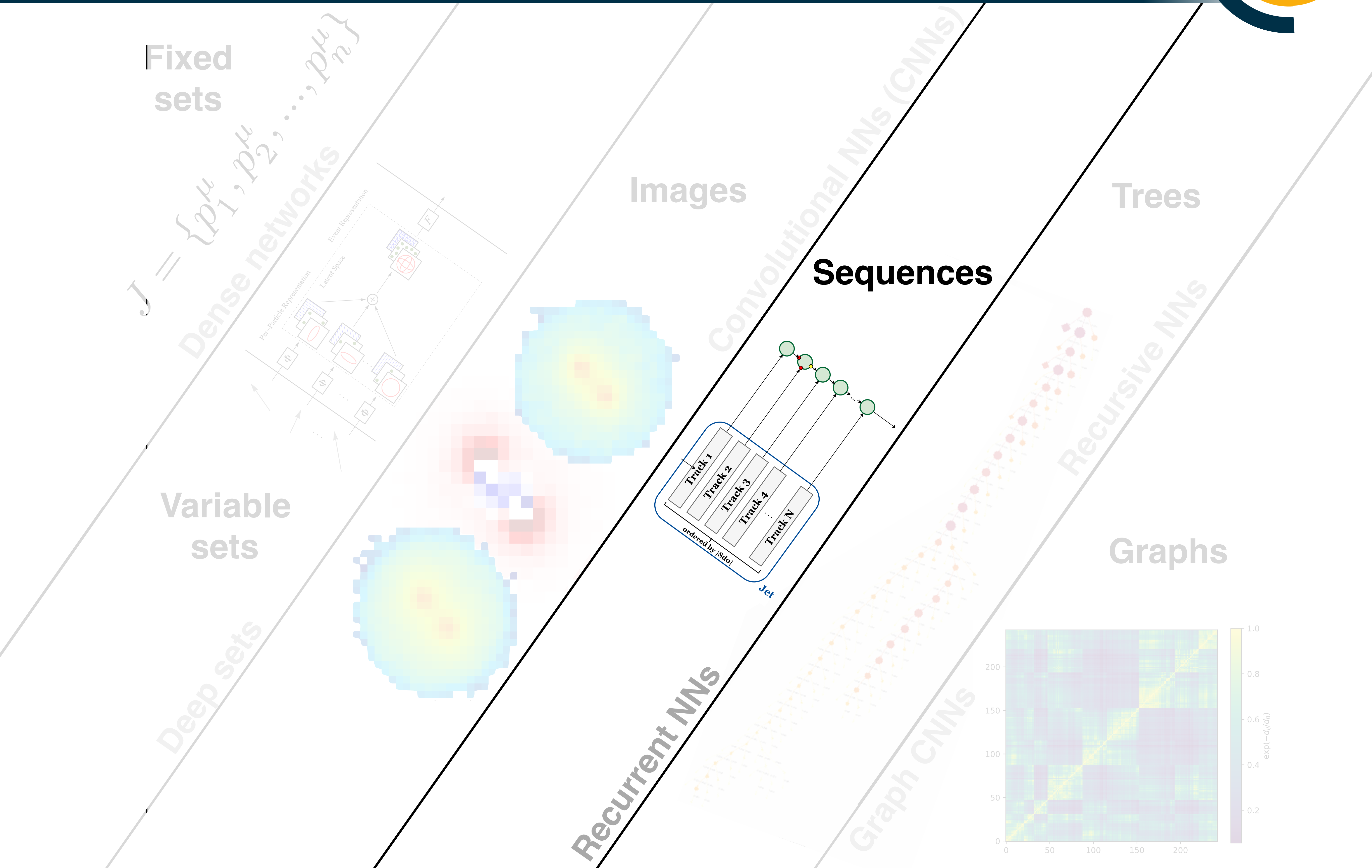
1807.10768

Step 1: how to represent our data



See 1709.04464 for image refs.

Step 1: how to represent our data



Sequence learning

One key challenge with images is that they have a fixed size.

In many contexts, this is ideal, because the data also have a fixed size. However, this is not always the case.

For example, events / jets have a variable number of particles.

One can represent these particles as a sequence in order to apply variable-length approaches that can access the full feature granularity.

Sequence learning with RNNs

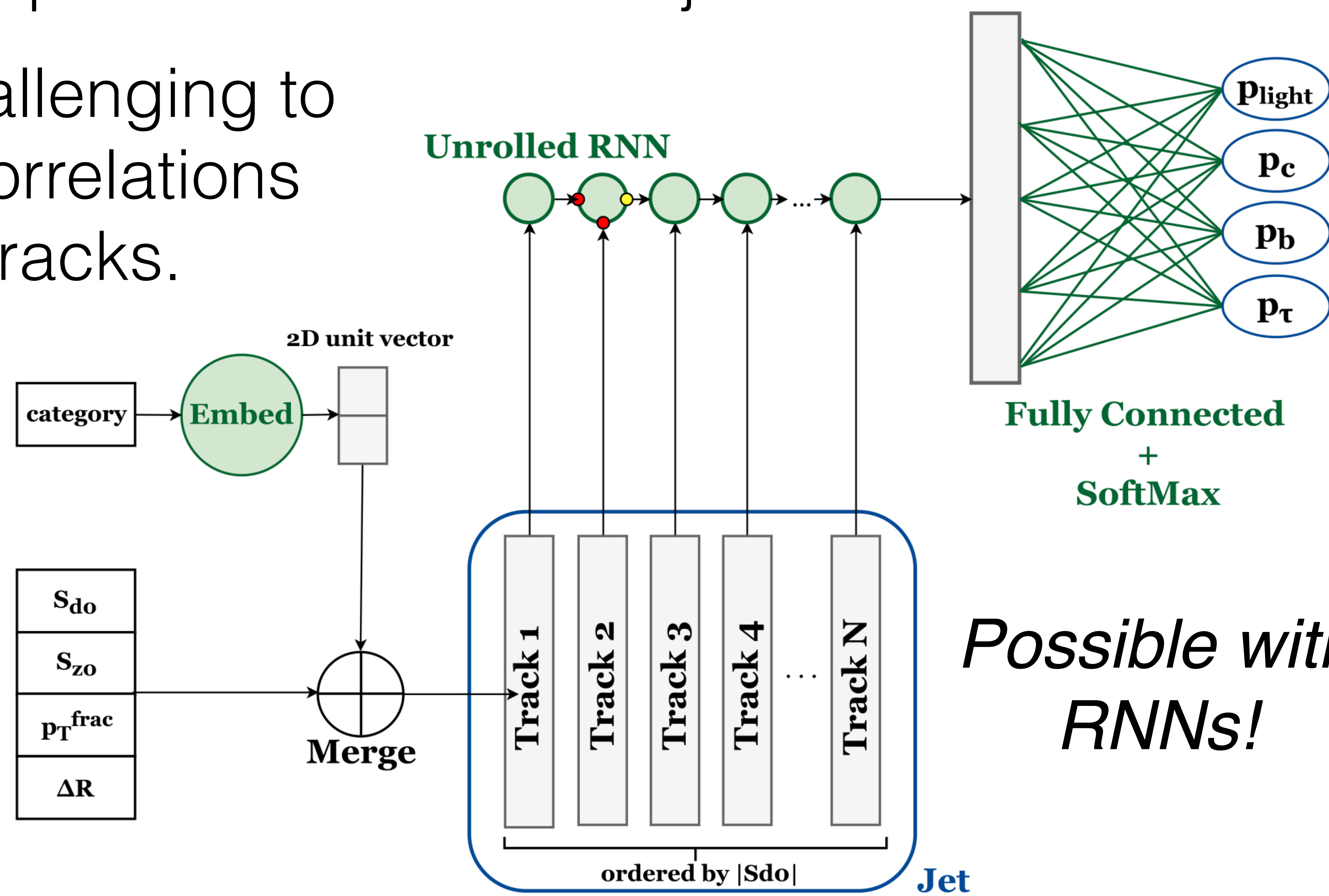
Flavor tagging (classify jets from b-quark or not) has a long history of ML. Use features of the charged-particle tracks inside jets.

In the past, challenging to incorporate correlations between tracks.

Sequence learning with RNNs

Flavor tagging (classify jets from b-quark or not) has a long history of ML. Use features of the charged-particle tracks inside jets.

In the past, challenging to incorporate correlations between tracks.



Possible with RNNs!

ATL-PHYS-PUB-2017-003

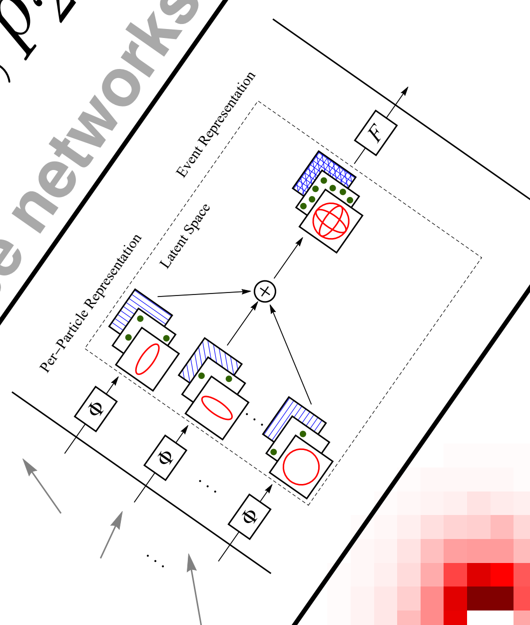
See 1709.04464 for image refs.

Step 1: how to represent our data

Fixed sets

$$J = \{p_1^\mu, p_2^\mu, \dots, p_n^\mu\}$$

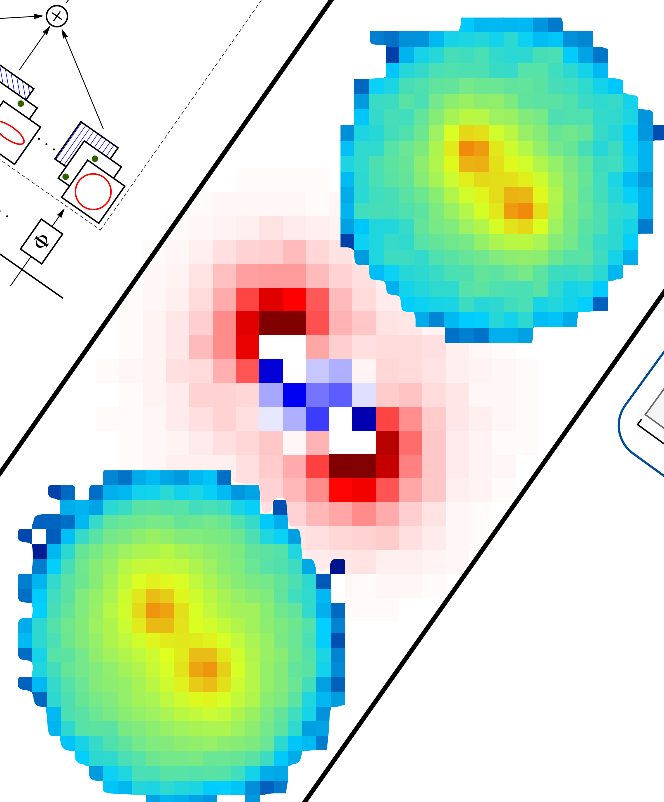
Dense networks



Variable sets

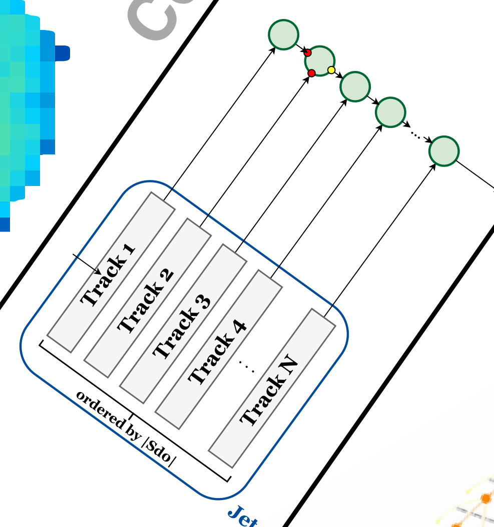
Deep sets

Images



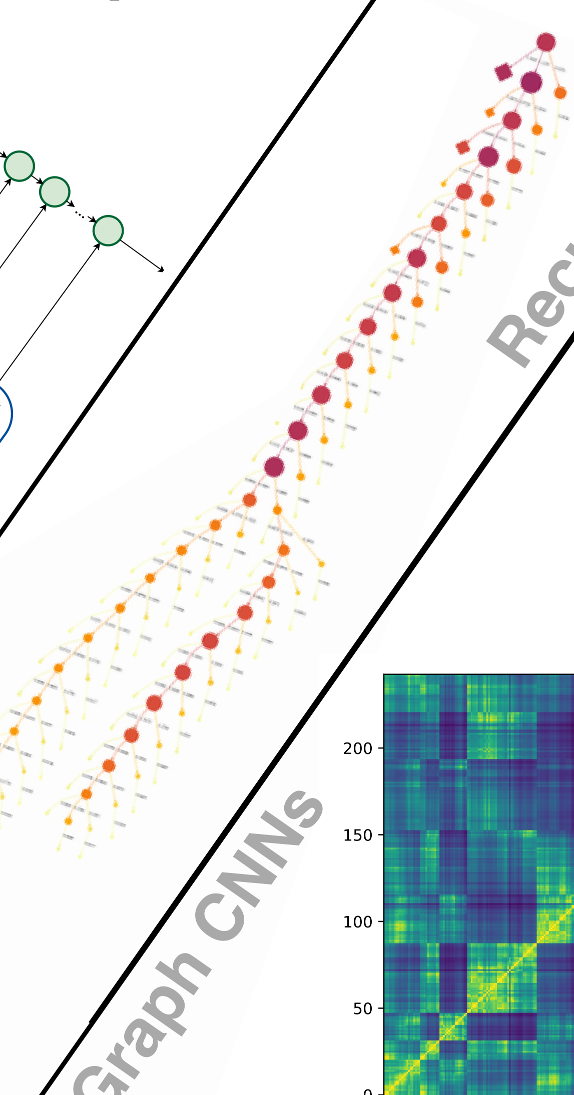
Convolutional NNs (CNNs)

Sequences



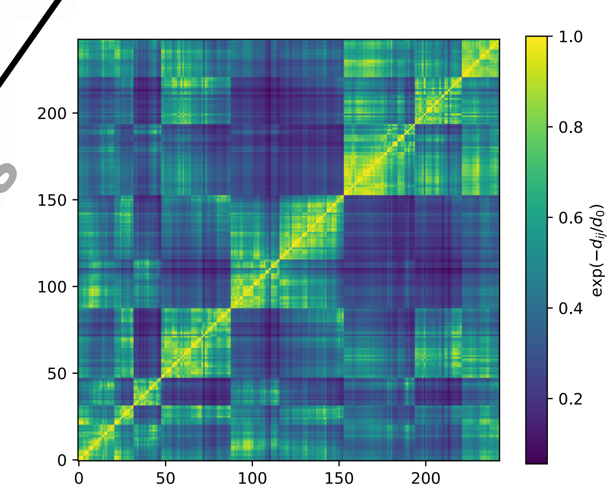
Recurrent NNs

Trees



Recursive NNs

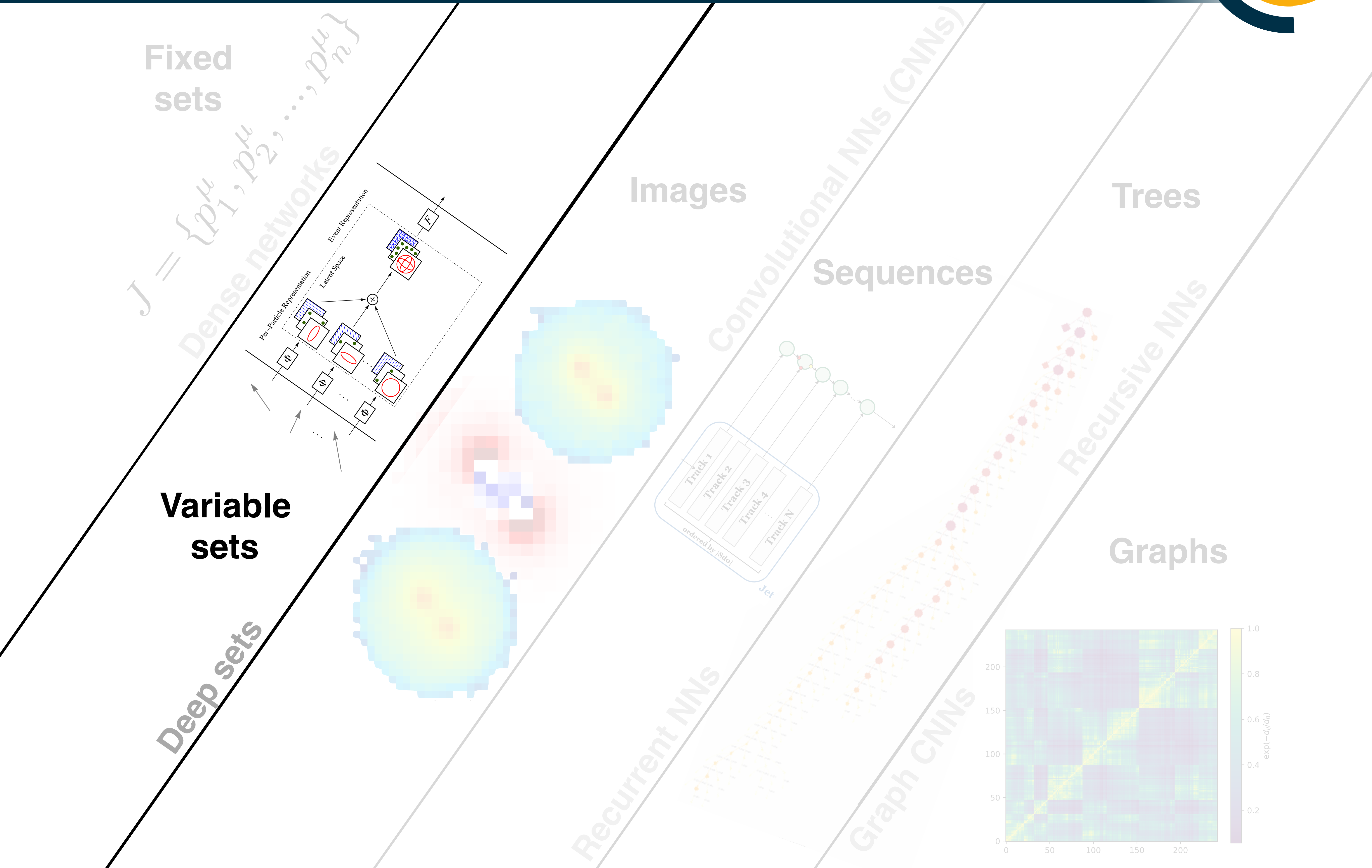
Graphs



Graph CNNs

See 1709.04464 for image refs.

Step 1: how to represent our data



Learning with sets

A challenge with sequence learning is that thanks to quantum mechanics, there is often no unique order.

A common scenario is that we have a variable-length **SET** of particles and we would like to learn from them directly.

Solution: set learning / point cloud approaches

Solution 1: Deep sets

Factorize the problem into two networks: one that embeds the set into a fixed-length latent space and one that acts on a permutation invariant operation on that latent space:

$$f(\{x_1, \ldots, x_M\}) = F\left(\sum_{i=1}^{M} \Phi(x_i)\right)$$

Due to the sum, this structure can operate on any length set and the order of the inputs doesn't matter.

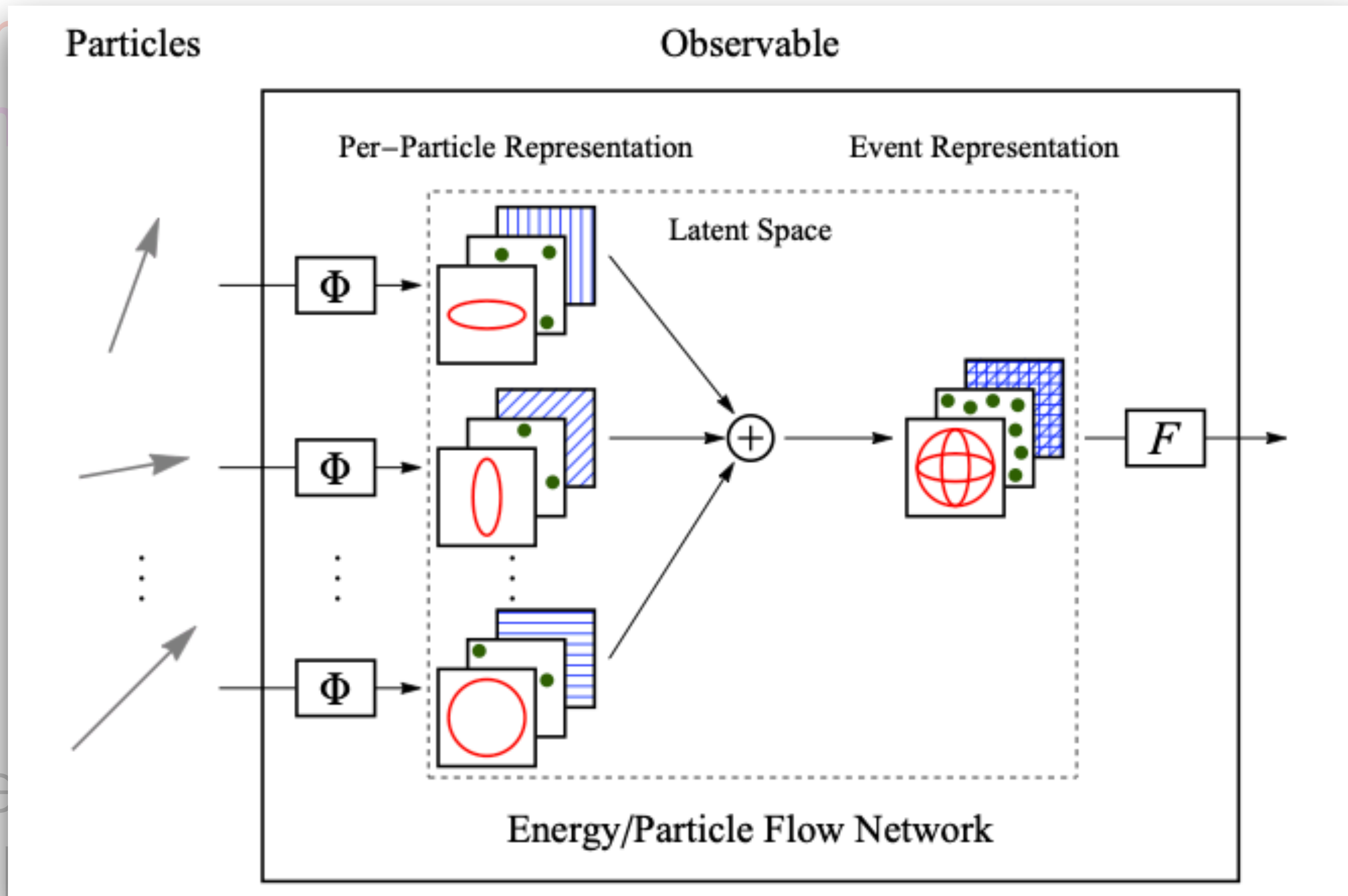
1703.06114, 1810.05165

Solution 1: Deep sets

Factorize the problem into two networks: one that embeds

the set i… …t acts on

a per… …space:

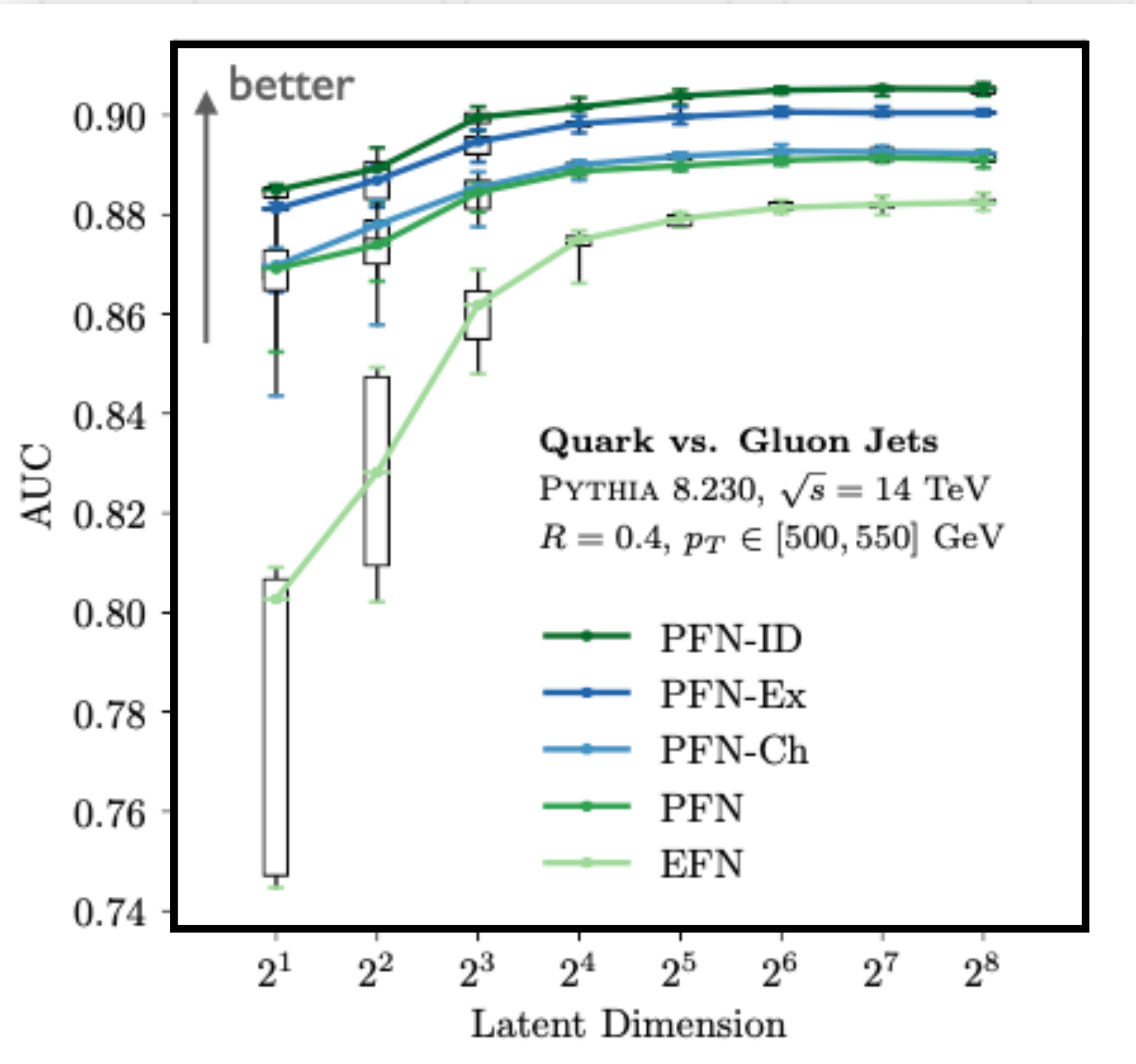


Due… …any

lengt… …atter.

1703.06114, 1810.05165

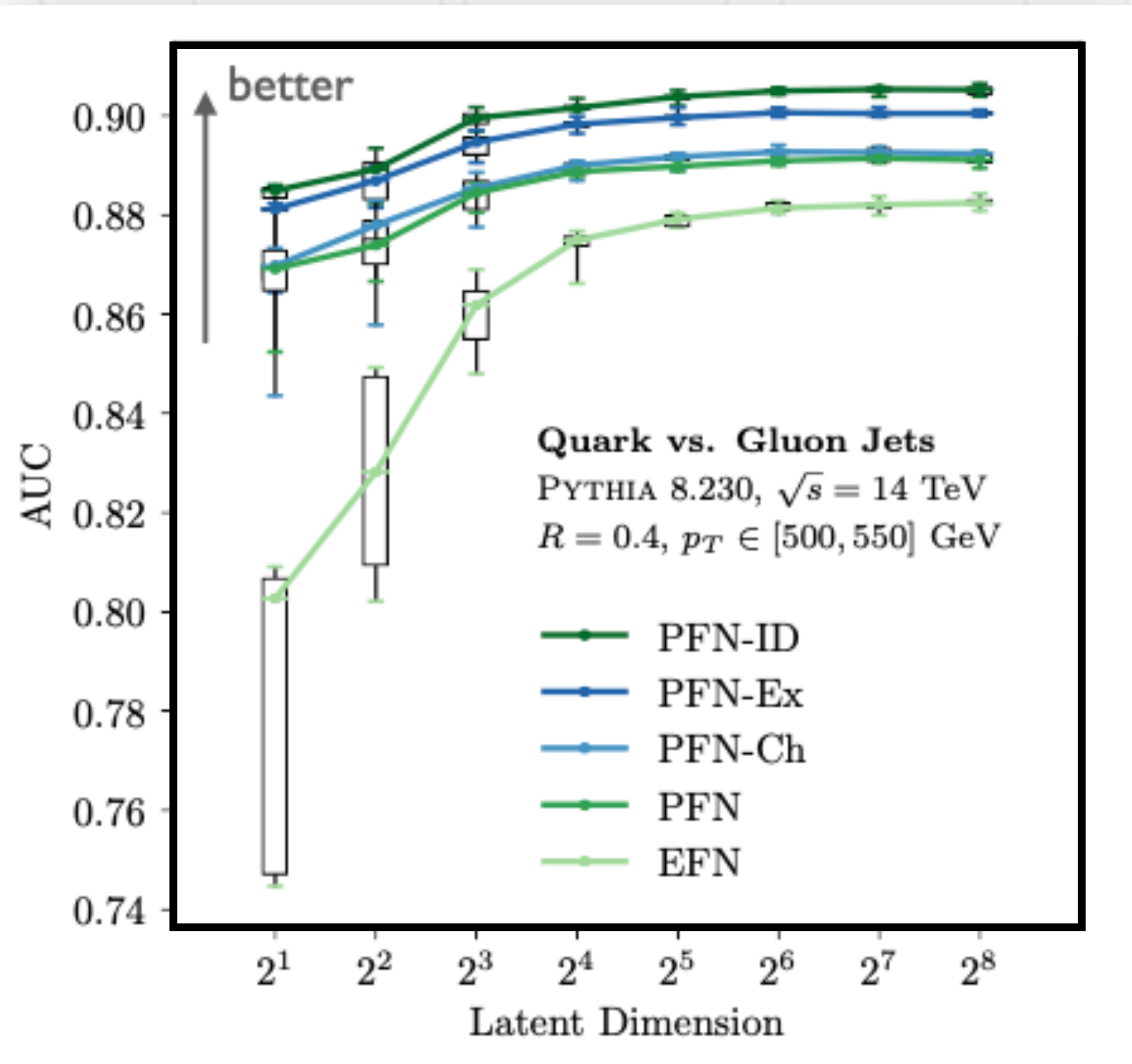
Solution 1: Deep sets

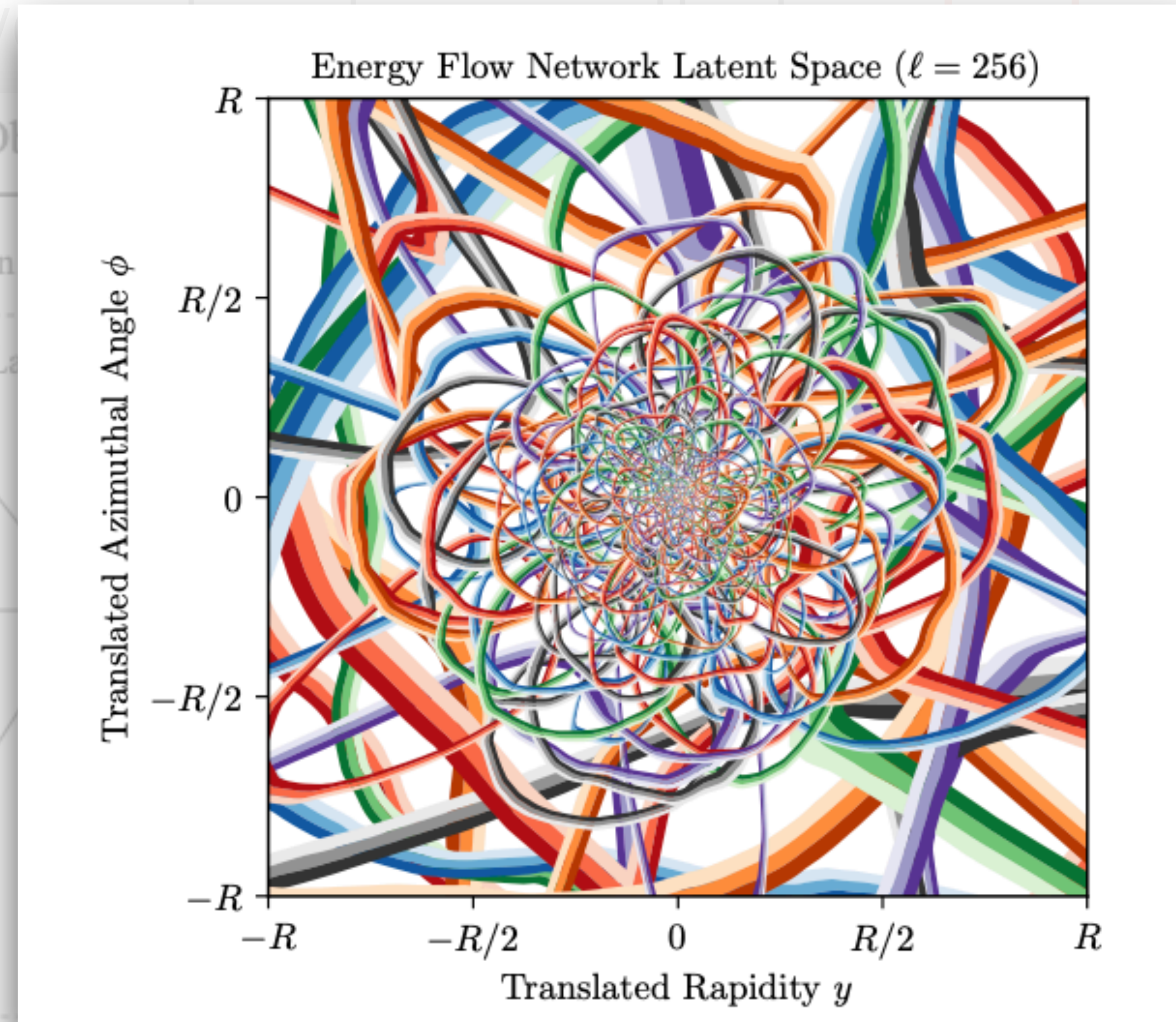


- Can readily incorporate per-particle features
- Can be made infrared and collinear safe (EFN) safe

1703.06114, 1810.05165

Solution 1: Deep sets

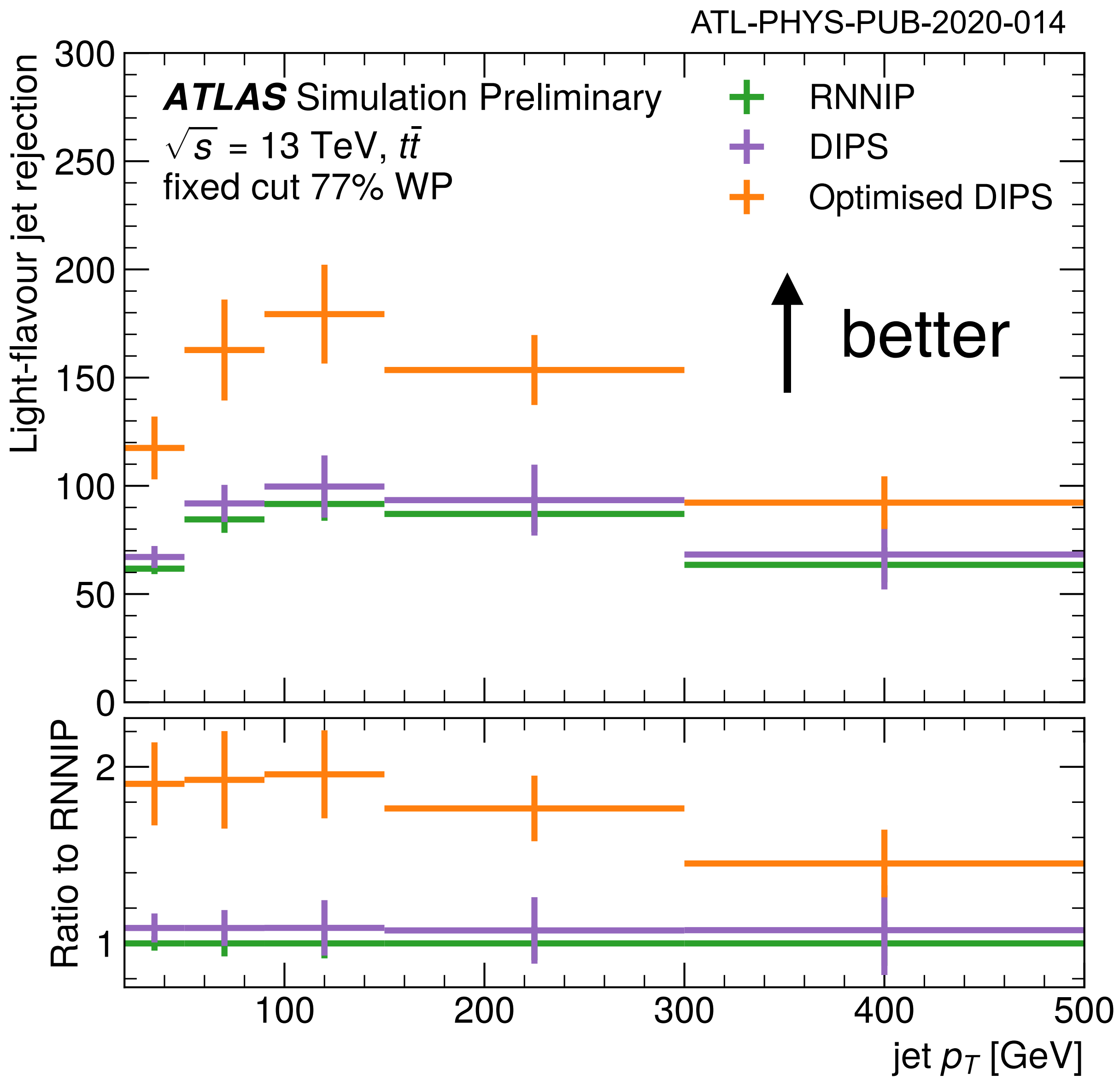




Latent space in IRC safe case is interpretable (and predictable!)

See also equivalent / covariant networks (e.g. 2203.06153)

Solution 1: Deep sets



Faster to train than RNN so can do R&D on input features to improve overall performance.

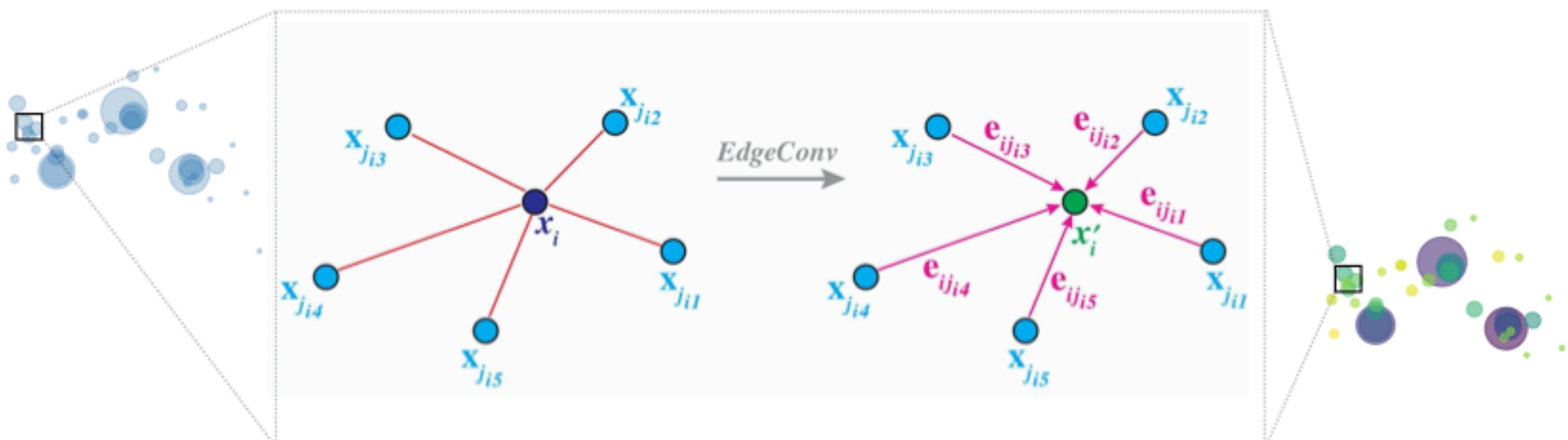
Latent space in IRC safe case is interpretable (and predictable!)

M. Zaheer et al. https://arxiv.org/abs/1703.06114; P. Komiske, E. Metodiev, & J. Thaler, JHEP 01 (2019) 121

Solution 2+: Graphs and Transformers

Classic CNNs operate on a fixed grid and are not invariant under the permutation of points

Can generalize CNNs to act on graphs



Need to define distances using particle properties

1801.07829 , 1902.08570

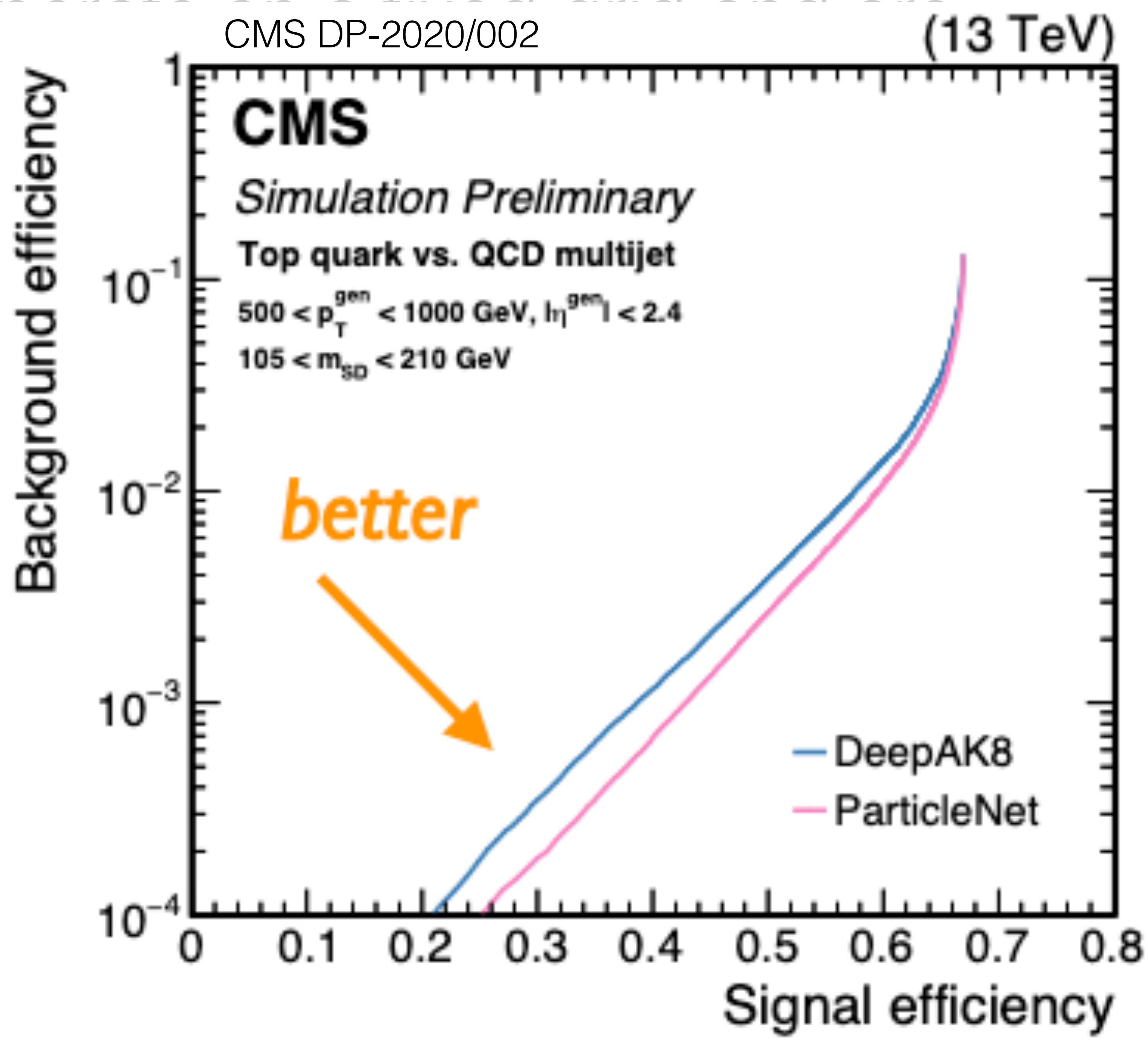
Solution 2+: Graphs and Transformers

Classic CNNs o

not invariant un

Can generaliz

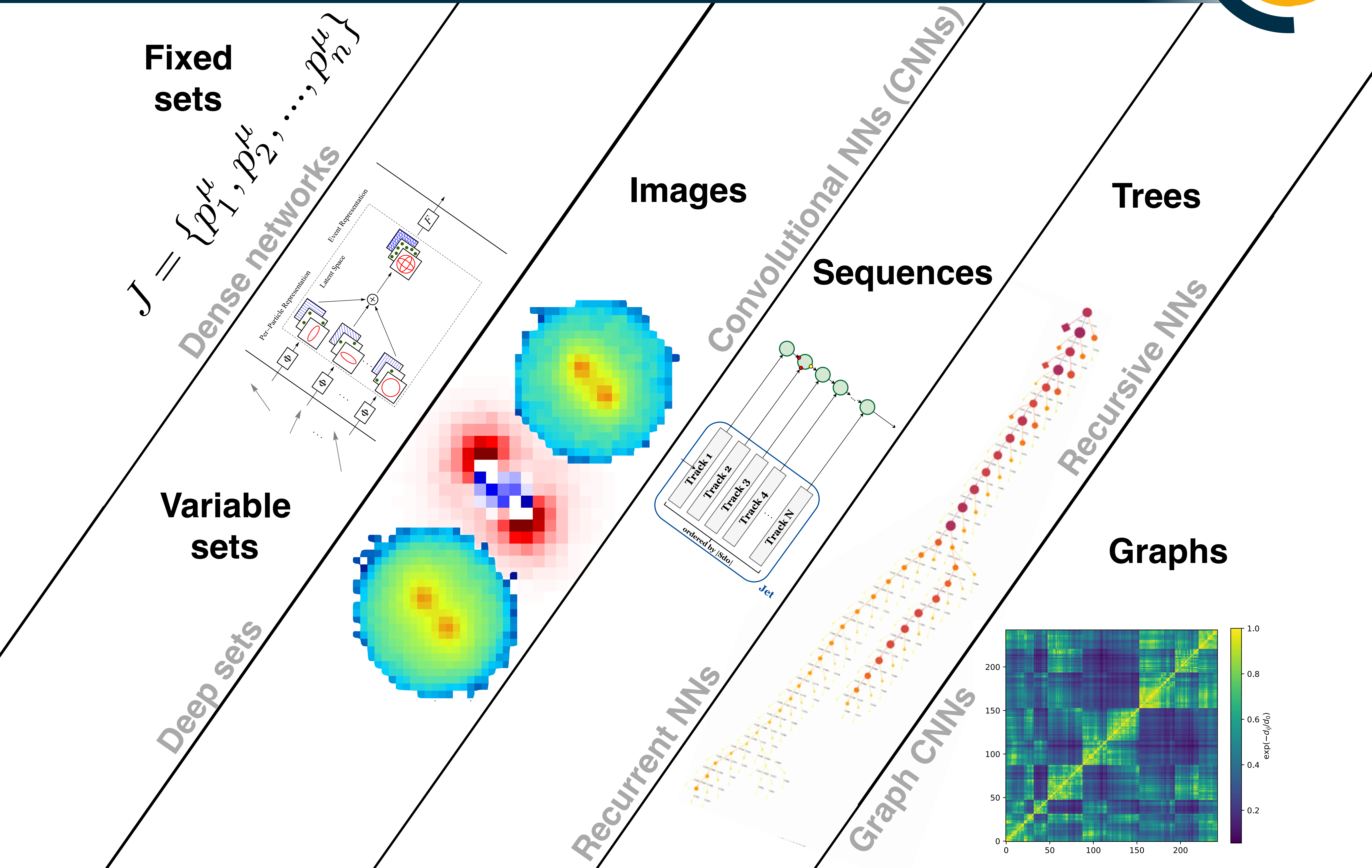
Competitive performance to other state-of-the-art methods



Need to define dis

1801.07829 , 1902.08570

Step 1: how to represent our data



Step 2: set up the learning task

One way to categorize methods is based on their level of ***supervision***

Unsupervised = no labels
Weakly-supervised = noisy labels
Semi-supervised = partial labels
Supervised = full label information

Supervised

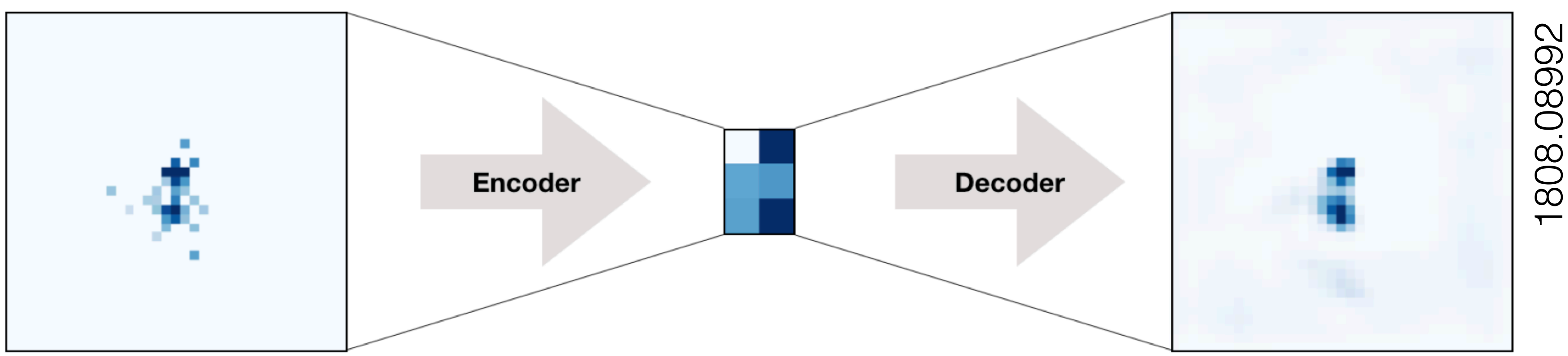
This is 99% of the ML. We have labeled examples and we train a model to predict the labels from the examples.

Need to be careful about what loss function to pick (more on that in a little bit…)

Unsupervised

Unsupervised = no labels

Typically, the goal of these methods is to implicitly or explicitly estimate p(x).



One strategy (autoencoders) is to try to compress events and then uncompress them. When x is far from uncompres(compress(x)), then x probably has low p(x).

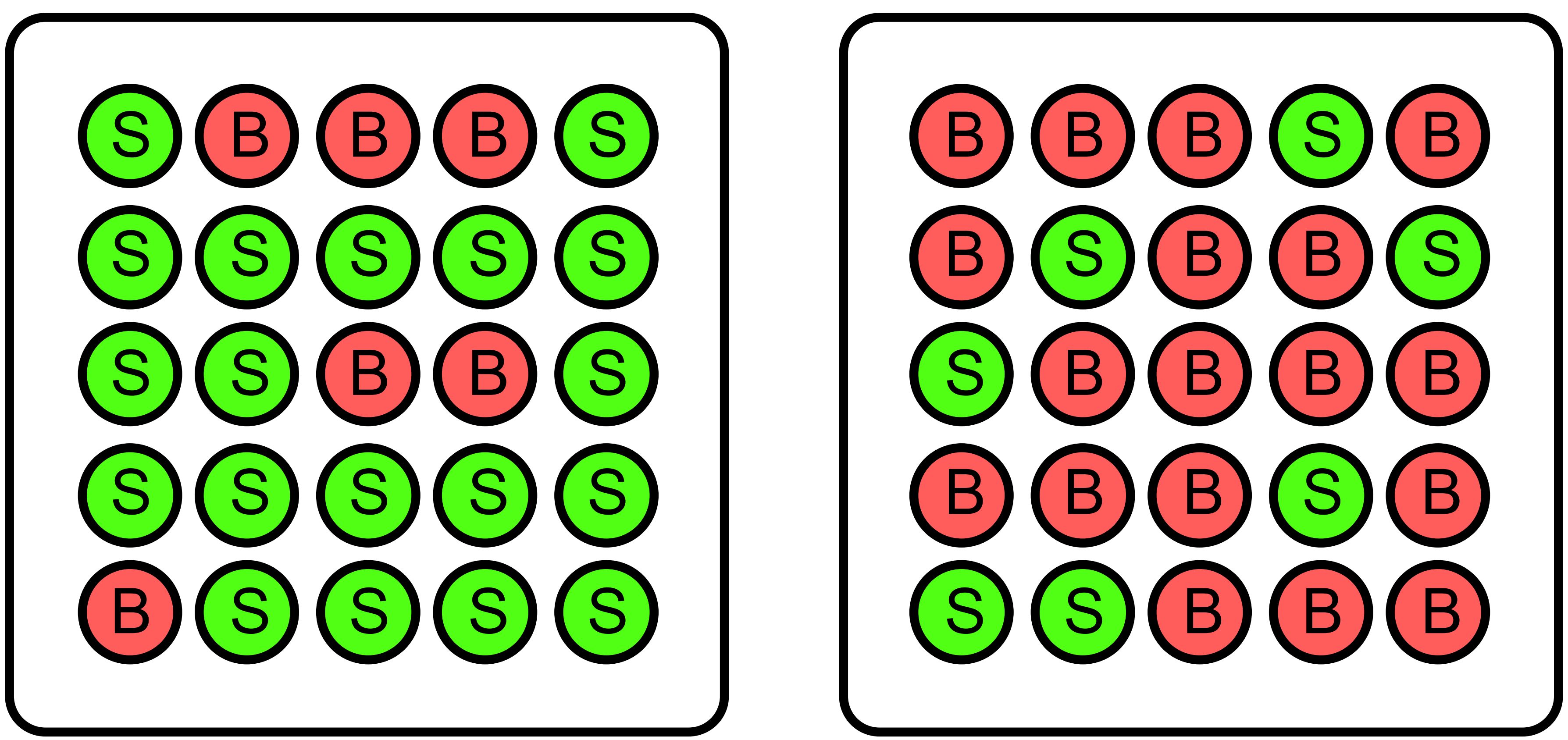
Talking point: anomaly detection!

Weakly-supervised

Weakly-supervised = noisy labels

Typically, the goal of these methods is to estimate *p(possibly signal-enriched)/p(possibly signal-depleted)*

Signal enriched | Signal depleted



1708.02949

Semi-supervised

Semi-supervised = partial labels

Typically, these methods use some signal simulations to build signal sensitivity

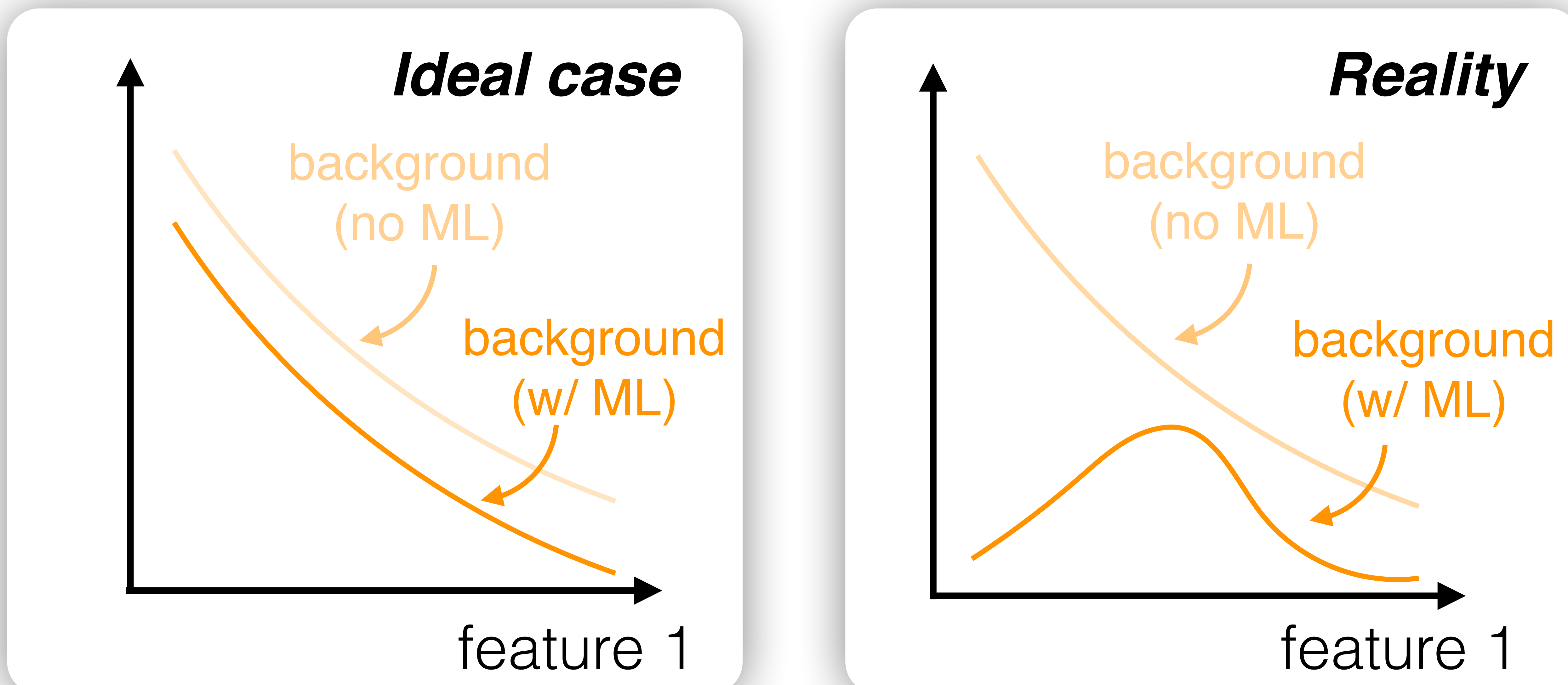
Image credit: https://www.particlezoo.net

vs

e.g. SM background versus many signals

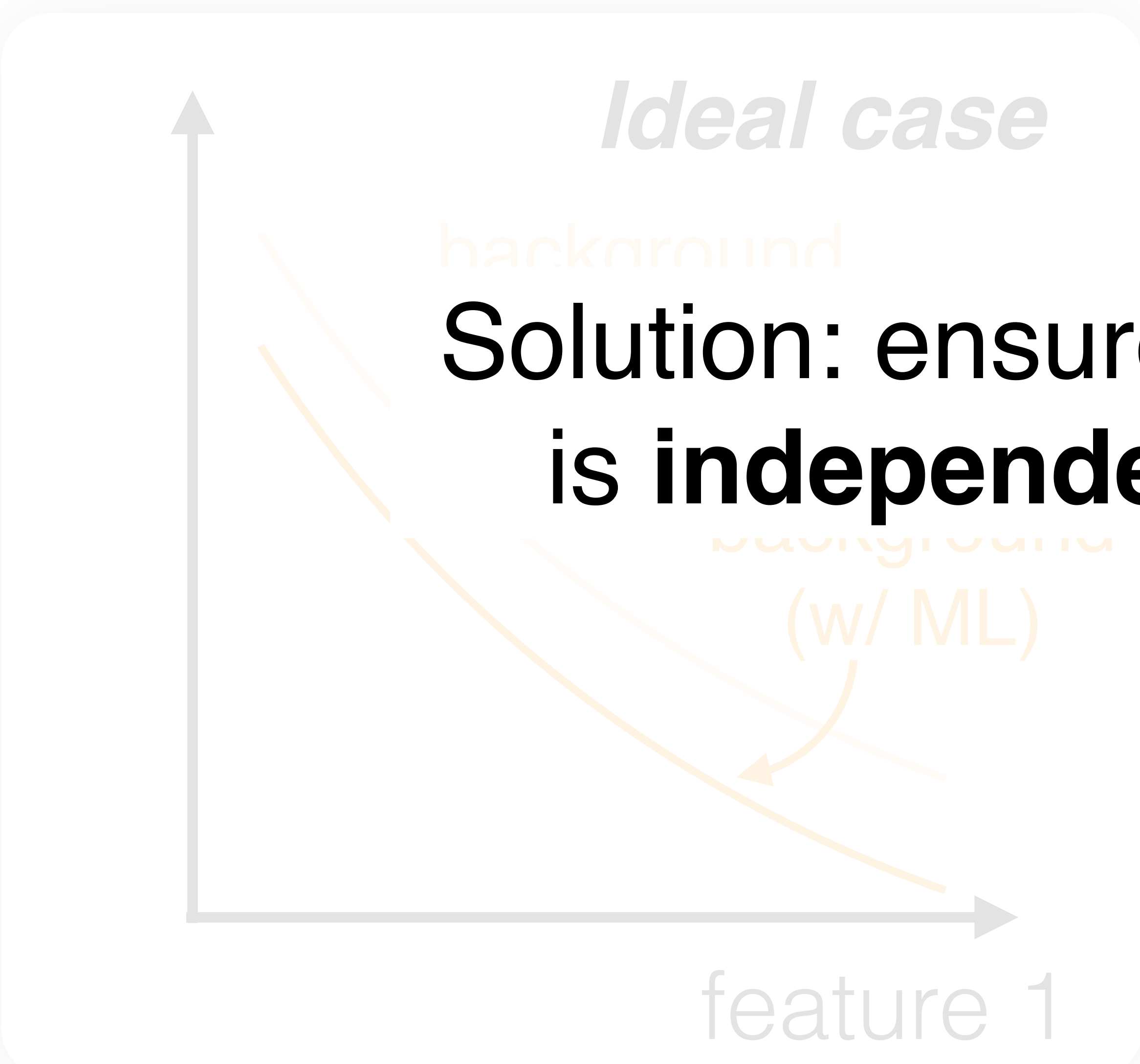
Caution Part I

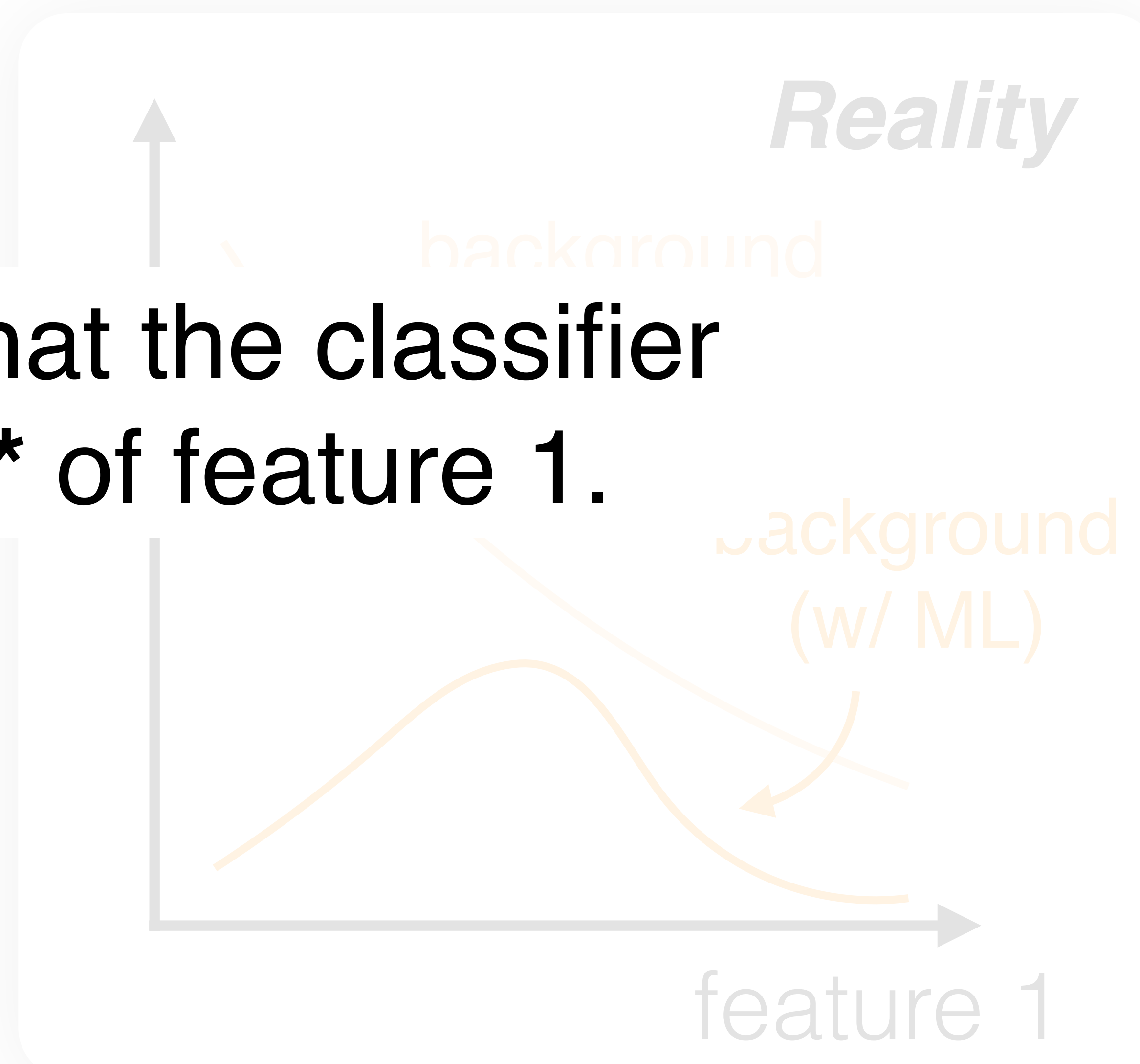
How can we learn a classifier that does not sculpt a bump in the background?



Caution Part I

How can we learn a classifier that does not sculpt a bump in the background?



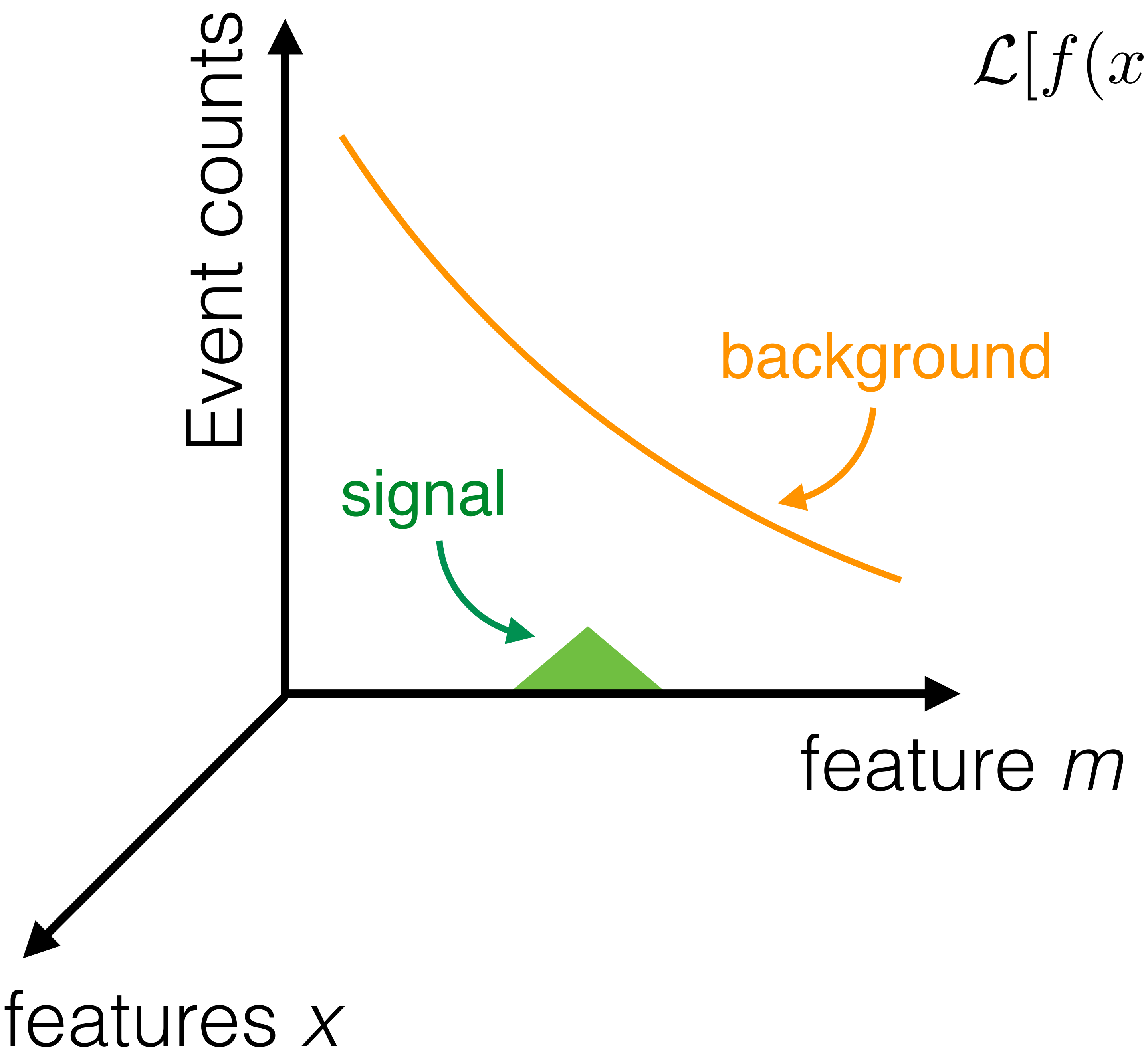


Solution: ensure that the classifier is **independent*** of feature 1.

**This is actually sufficient but unnecessary. There are many dependencies (e.g. linear) that would not sculpt bumps.*

Caution Part I: decorrelation

Train e.g. a neural network

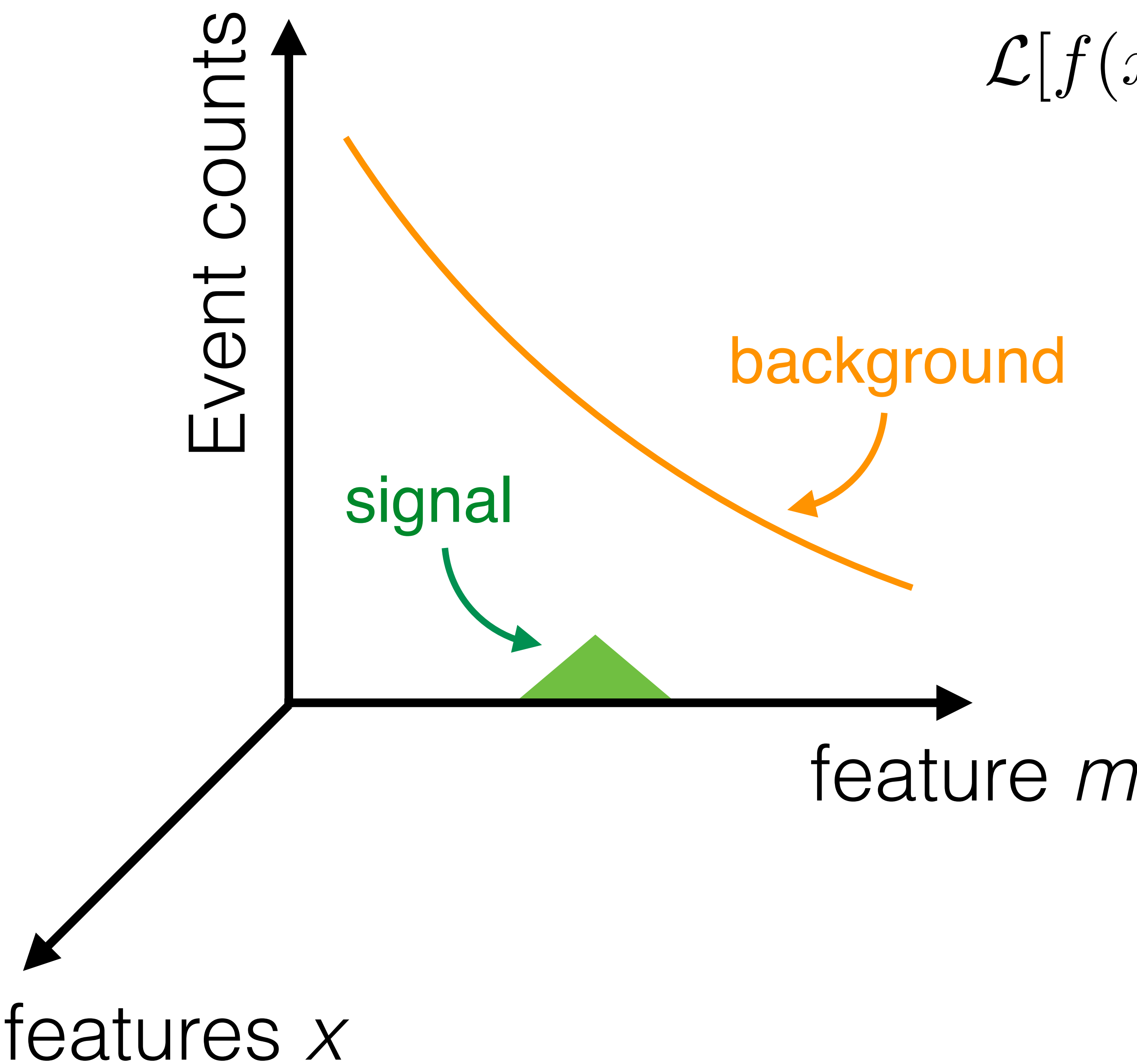


$$\mathcal{L}[f(x)] = \sum_{i\in s} L_{\text{classifier}}(f(x_i), 1) + \sum_{i\in b} L_{\text{classifier}}(f(x_i), 0)$$

$L_{classifier}$ is the usual classifier loss, e.g. cross entropy or mean squared error.

Caution Part I: decorrelation

Train e.g. a neural network with a **custom loss functional**



$$\mathcal{L}[f(x)] = \sum_{i\in s} L_{\text{classifier}}(f(x_i),1) + \sum_{i\in b} L_{\text{classifier}}(f(x_i),0) + \lambda \sum_{i\in b} L_{\text{decor}}(f(x_i),m_i)$$

$L_{classifier}$ is the usual classifier loss, e.g. cross entropy or mean squared error.

L_{decor} is large when *f(x)* and *m* are "correlated"

Enforcing Independence

Train e.g. a neural network with a **custom loss functional**

$$\mathcal{L}[f(x)] = \sum_{i\in s} L_{\text{classifier}}(f(x_i),1) + \sum_{i\in b} L_{\text{classifier}}(f(x_i),0) + \lambda \sum_{i\in b} L_{\text{decor}}(f(x_i),m_i)$$

Recent proposals:

Adversaries: L_{decor} is the loss of **a 2nd NN** (adversary) that tries to learn *m* from *f(x)*.

Distance Correlation: L_{decor} is **distance correlation** (generalizes Pearson correlation) between *m* and *f(x)*.

Mode Decorrelation: L_{decor} is small when the **CDF** of *f(x)* is the same across different values of *m*.

Enforcing Independence

Train e.g. a neural network with a **custom loss functional**

$$\mathcal{L}[f(x)] = \sum_{i\in s} L_{\text{classifier}}(f(x_i), 1) + \sum_{i\in b} L_{\text{classifier}}(f(x_i), 0) + \lambda \sum_{i\in b} L_{\text{decor}}(f(x_i), m_i)$$

Recent proposals:

Adversaries: L_{decor} is the loss of **a 2nd NN** (adversary) that tries to learn *m* from *f(x)*.

Distance Correlation: L_{decor} is **distance correlation** (generalizes Pearson correlation) between *m* and *f(x)*.

Mode Decorrelation: L_{decor} is small when the **CDF** of *f(x)* is the same across different values of *m*.

Enforcing Independence

Train e.g. a neural network with a **custom loss functional**

$$\mathcal{L}[f(x)] = \sum_{i\in s} L_{\text{classifier}}(f(x_i), 1) + \sum_{i\in b} L_{\text{classifier}}(f(x_i), 0) + \lambda \sum_{i\in b} L_{\text{decor}}(f(x_i), m_i)$$

Recent proposals:

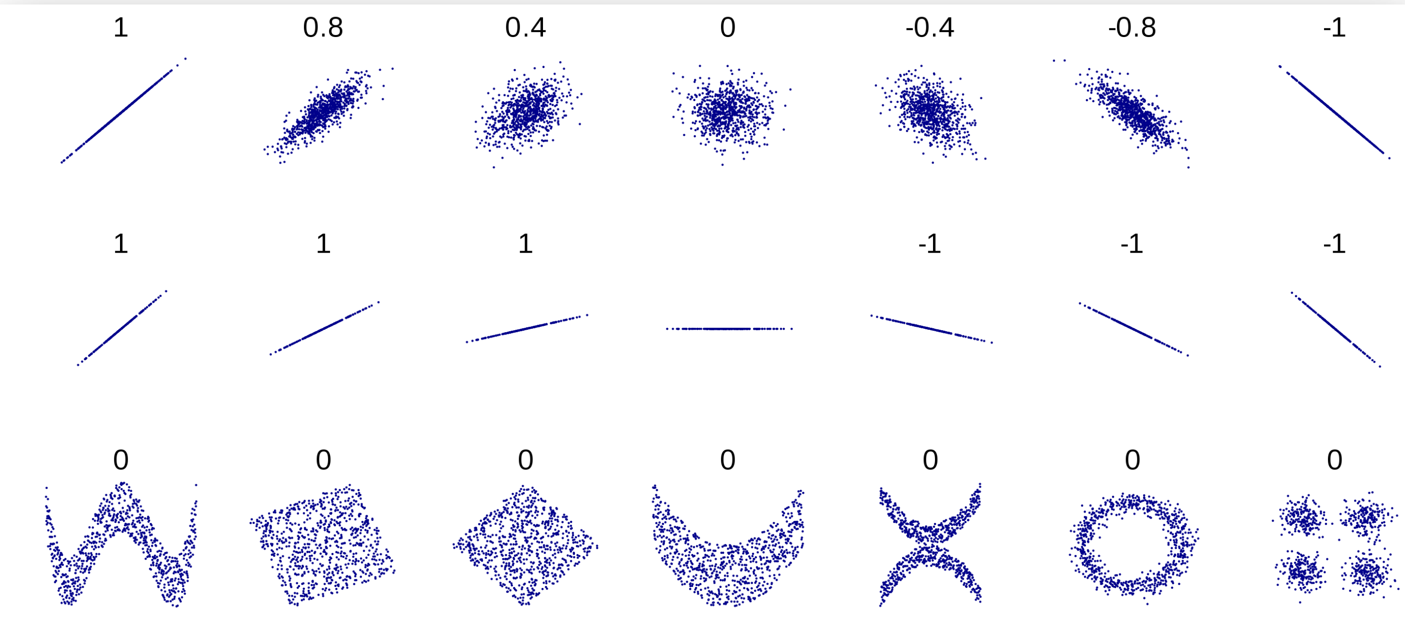
Adversaries: L_{decor} is the loss of **a 2nd NN** (adversary) that tries to learn *m* from *f(x)*.

Distance Correlation: L_{decor} is **distance correlation** (generalizes Pearson correlation) between *m* and *f(x)*.

Mode Decorrelation: L_{decor} is small when the **CDF** of *f(x)* is the same across different values of *m*.

Enforcing Independence

Pearson Correlation



Distance Correlation

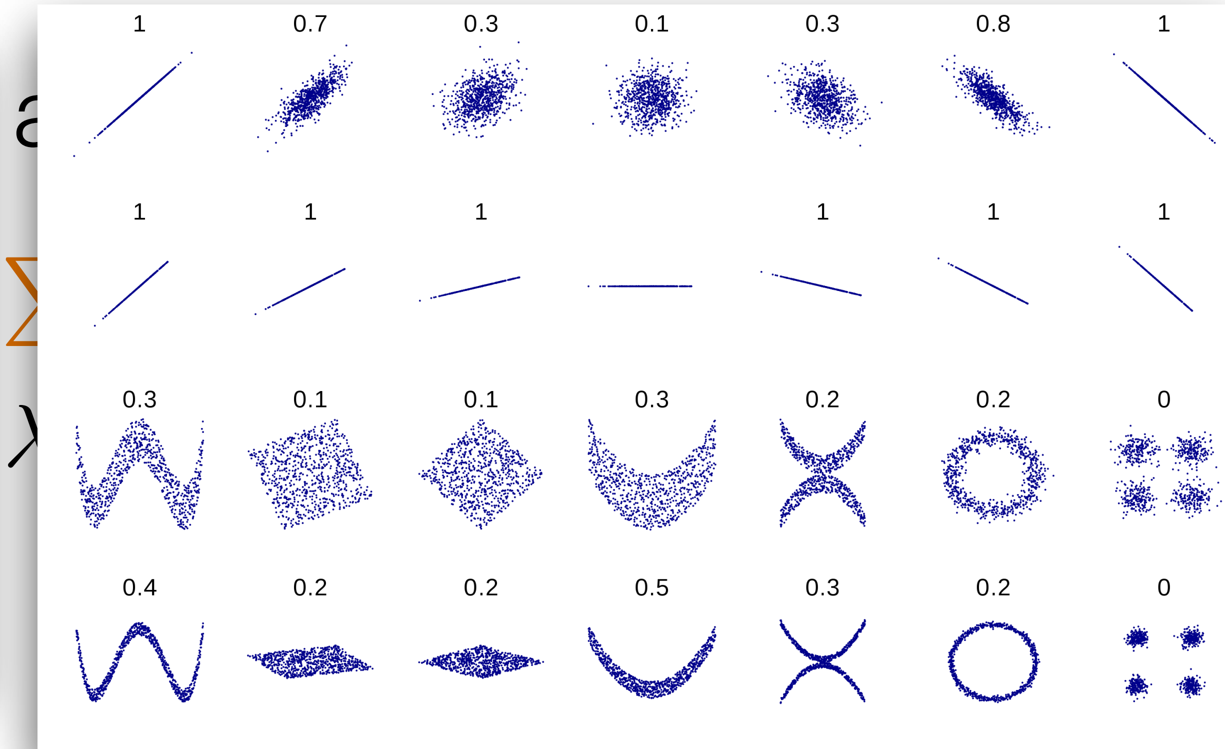


Image credit: Denis Boigelot

Adversaries: L_{decor} is the loss of **a 2nd NN** (adversary) that tries to learn *m* from *f(x)*.

Distance Correlation: L_{decor} is **distance correlation** (generalizes Pearson correlation) between *m* and *f(x)*.

Mode Decorrelation: L_{decor} is small when the **CDF** of *f(x)* is the same across different values of *m*.

Enforcing Independence

Train e.g. a neural network with a **custom loss functional**

$$\mathcal{L}[f(x)] = \sum_{i\in s} L_{\mathrm{classifier}}(f(x_i), 1) + \sum_{i\in b} L_{\mathrm{classifier}}(f(x_i), 0) + \lambda \sum_{i\in b} L_{\mathrm{decor}}(f(x_i), m_i)$$

Recent proposals:

Adversaries: L_{decor} is the loss of **a 2nd NN** (adversary) that tries to learn *m* from *f(x)*.

Distance Correlation: L_{decor} is **distance correlation** (generalizes Pearson correlation) between *m* and *f(x)*.

Mode Decorrelation: L_{decor} is small when the **CDF** of *f(x)* is the same across different values of *m*.

Adversaries

Adversaries: L_{decor} is the loss of **a 2nd NN** (adversary) that tries to learn m from $f(x)$.

Pros: Very flexible and m can be multidimensional

Cons: Hard to train (minimax problem) & many parameters

1611.01046,1703.03507

Distance Correlation

Distance Correlation: L_{decor} is **distance correlation** (generalizes Pearson correlation) between m and $f(x)$.

Pros: Convex (easier to train) and no free parameters

Cons: Memory intensive to compute distance correlation

2001.05310, 2007.14400

Mode Decorrelation

Mode Decorrelation (MoDE): L_{decor} is small when the **CDF** of $f(x)$ is the same across different values of m.

Pros: Readily generalizes beyond independence (can require linear, quadratic (+monotonic), …

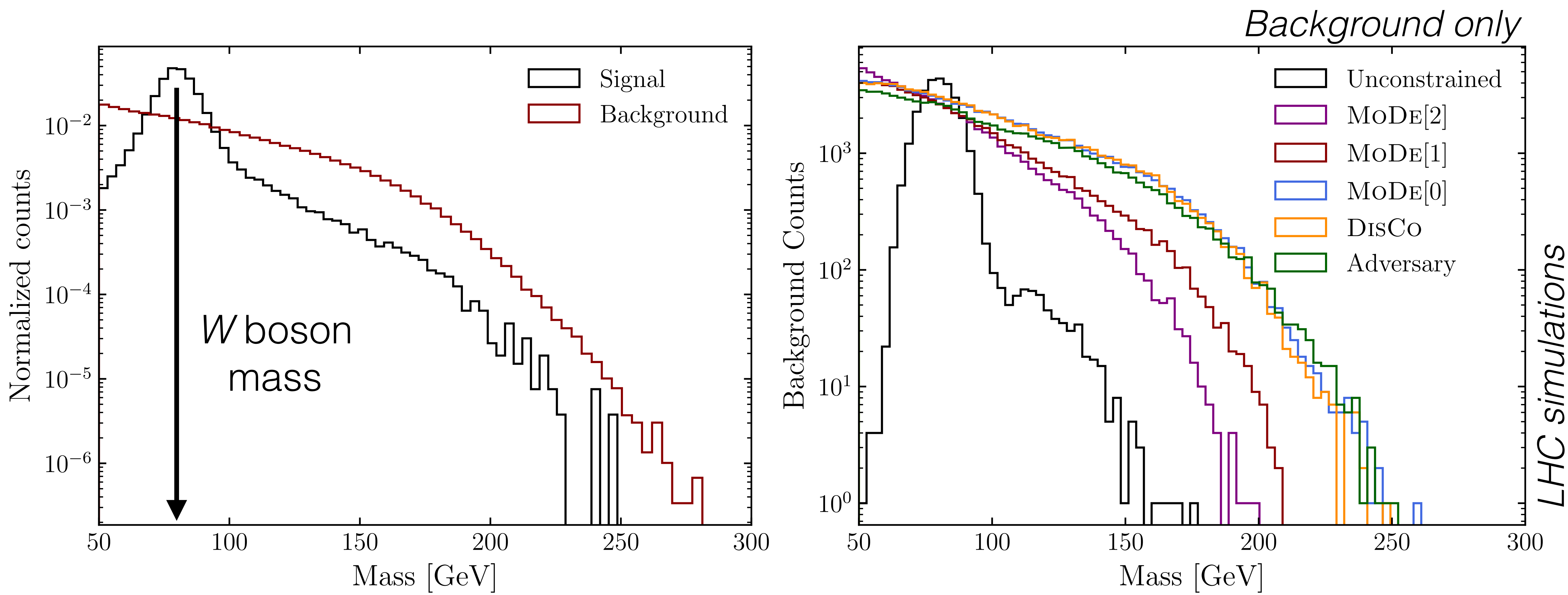
No free parameters and small memory footprint

Cons: In its simplest form, need discrete bins in m (does not seem to be fundamental)

2010.09745

Overview

Real world example: the search for Lorentz-boosted *W* bosons at the Large Hadron Collider



MoDe[0] enforces independence, [1] is linear, [2] is monotonic quadratic, …

2010.09745

N.B. think twice about using decorrelation for uncertainties! See 2109.08159.

Caution Part II: prior dependence

Sometimes, we need a model (often for calibration) that does not depend on the training sample properties.

For example, a particle of a given energy hits our detector and registers measurements in a number of sensors

e.g. the particle energy is uniform during training, but exponential for certain running conditions.

(usually not an issue for classification)

Caution Part II: prior dependence

Sometimes, we need a model (often for calibration) that does not depend on the training sample properties.

For example, a particle of a given energy hits our detector and registers measurements in a number of sensors

e. ng,

s.

Your first instinct here might have been to train a classifier to estimate the true value given measured values using simulated data.

Caution Part II: prior dependence

Claim: this is prior dependent !

For example, a particle of a given energy hits our detector and registers measurements in a number of sensors

e. ng,
s.

Your first instinct here might have been to train a classifier to estimate the true value given measured values using simulated data.

What goes wrong?

Suppose you have some features x and you want to predict y.

detector energy *true energy*

One way to do this is to find an f that minimizes the mean squared error (MSE):

$$f = \mathrm{argmin}_g \sum_i (g(x_i) - y_i)^2$$

Then*, f(x) = E[y|x].

*If you have not seen this before, please let me know if you need help with the proof!

What goes wrong?

Suppose you have some features x and you want to predict y.

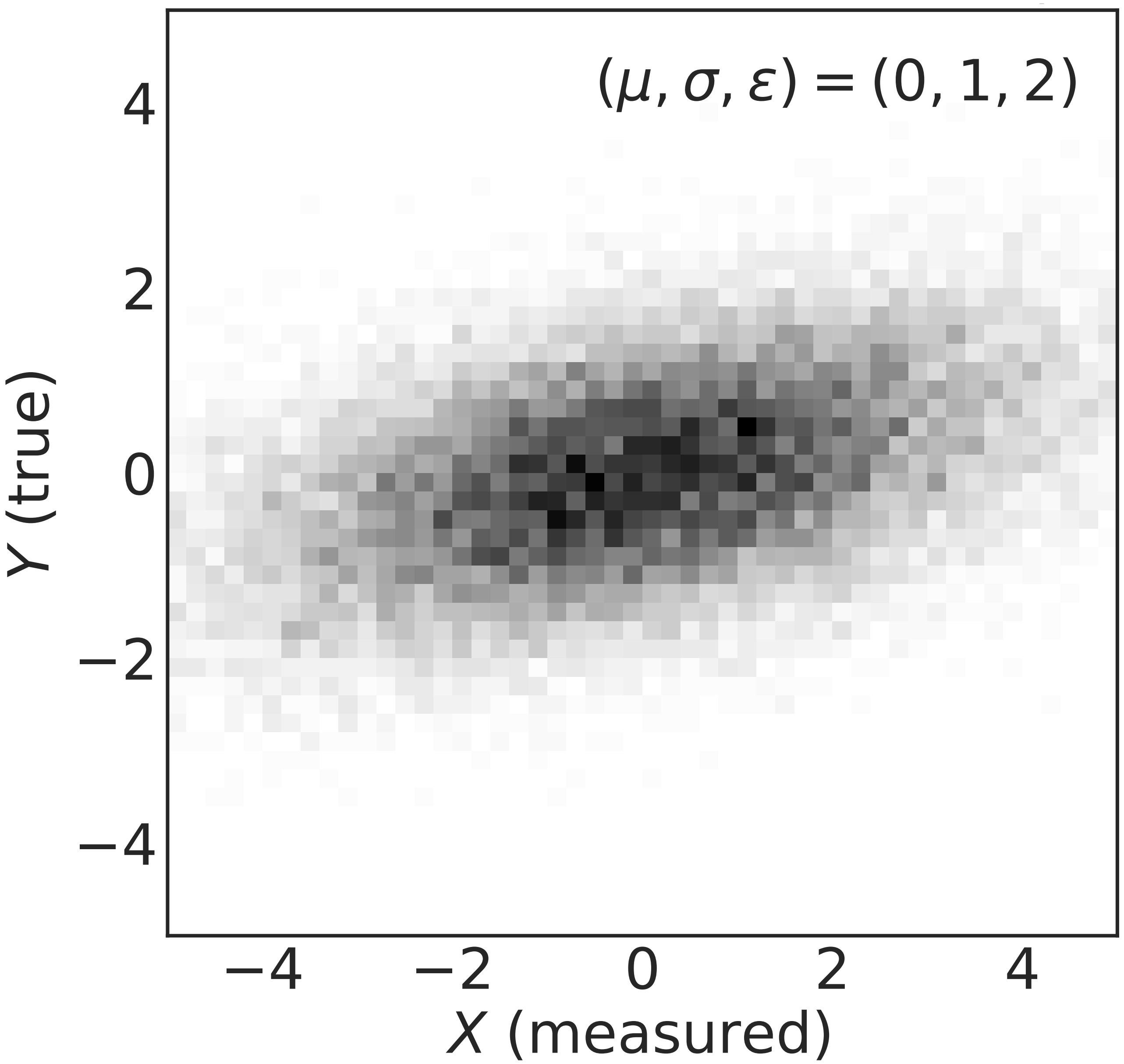
detector energy *true energy*

$$f(x) = E[y|x] = \int \mathrm{d}y\, y\, p(y|x)$$

$$E[f(x)|y] = \int \mathrm{d}x\, \mathrm{d}y'\, y'\, p_{\mathrm{train}}(y'|x)\, p_{\mathrm{test}}(x|y)$$

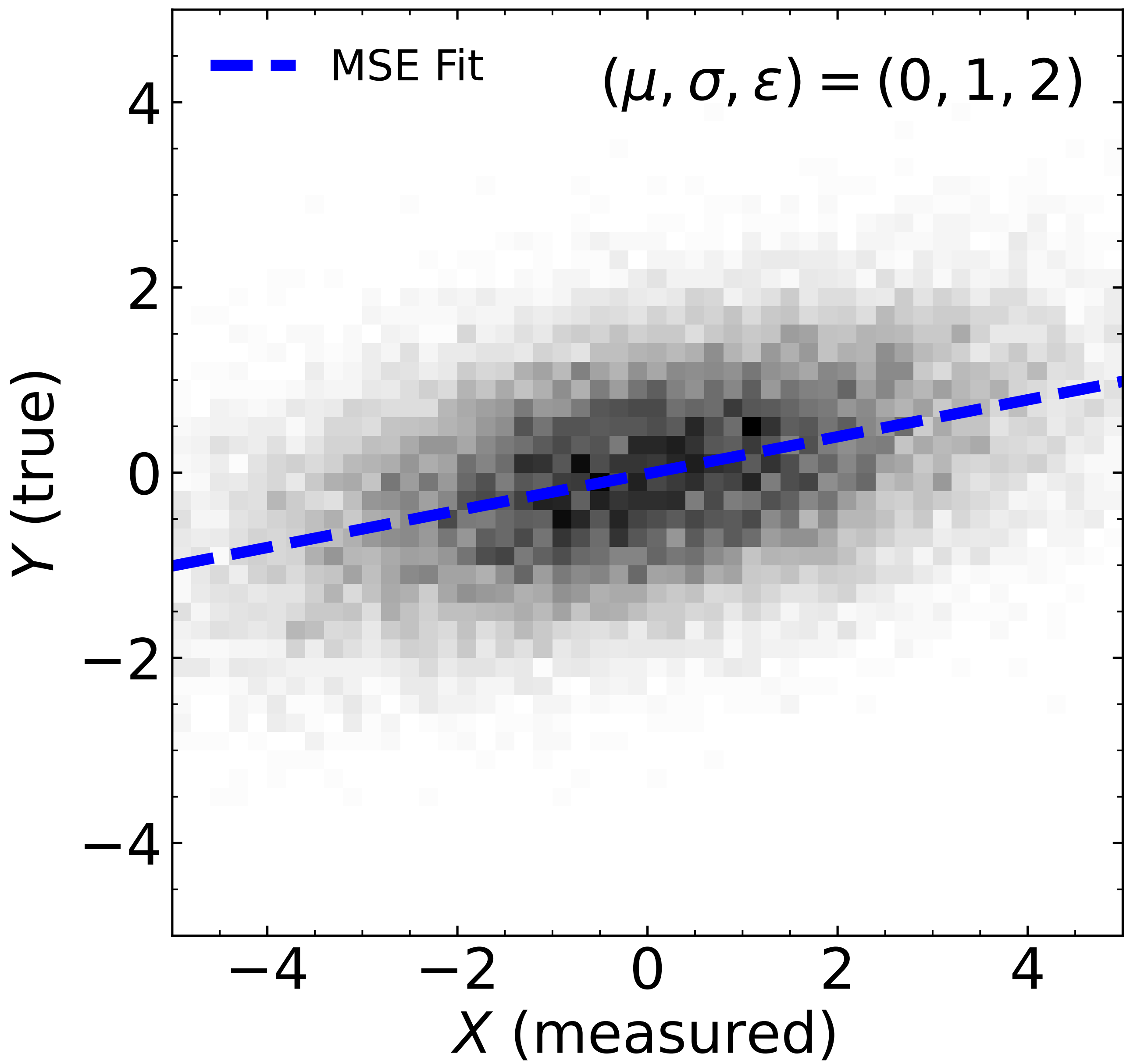
this need not be y even if $p_{train} = p_{test}$ (!)

Gaussian Example



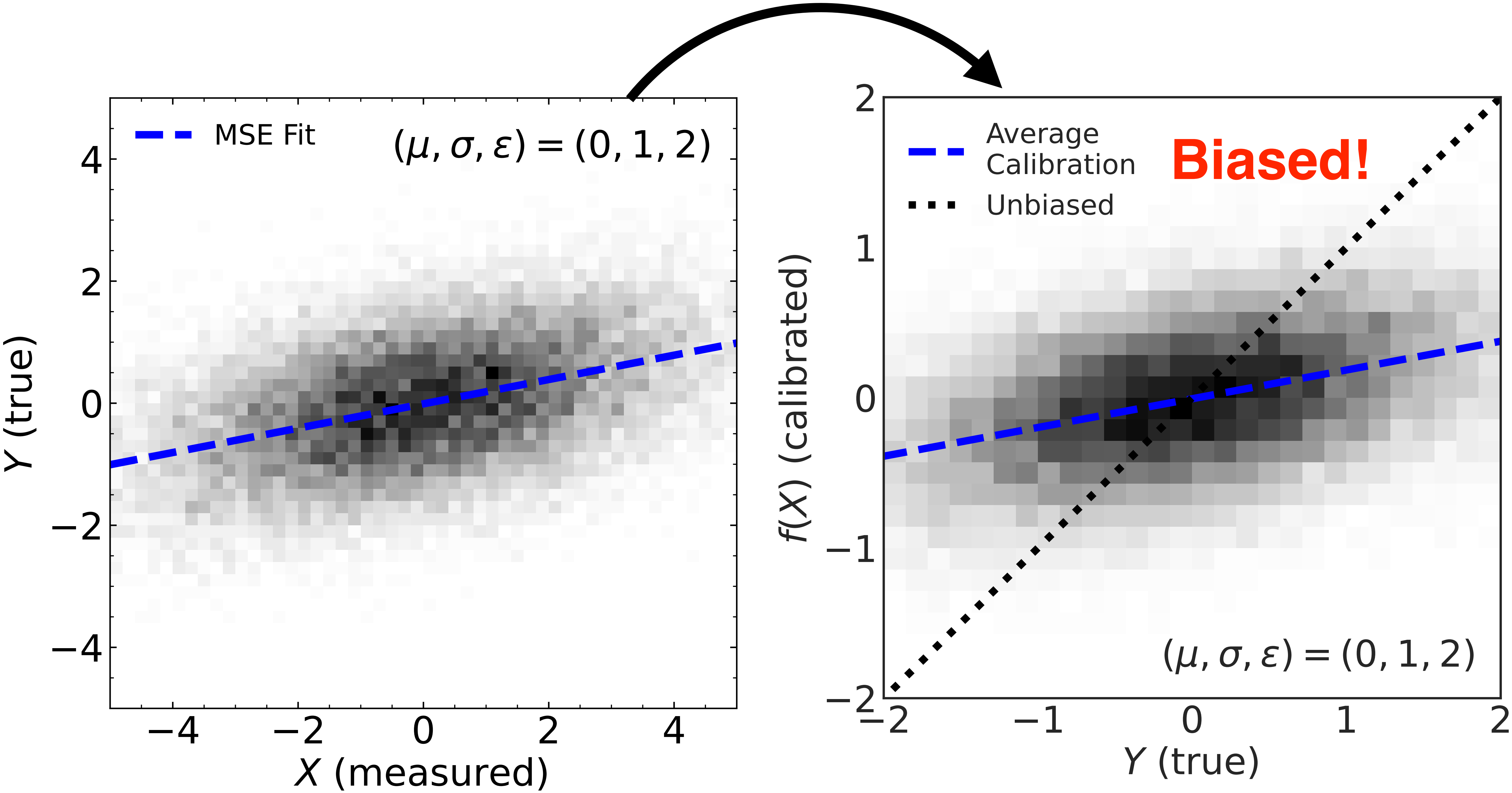
2205.05084

Gaussian Example



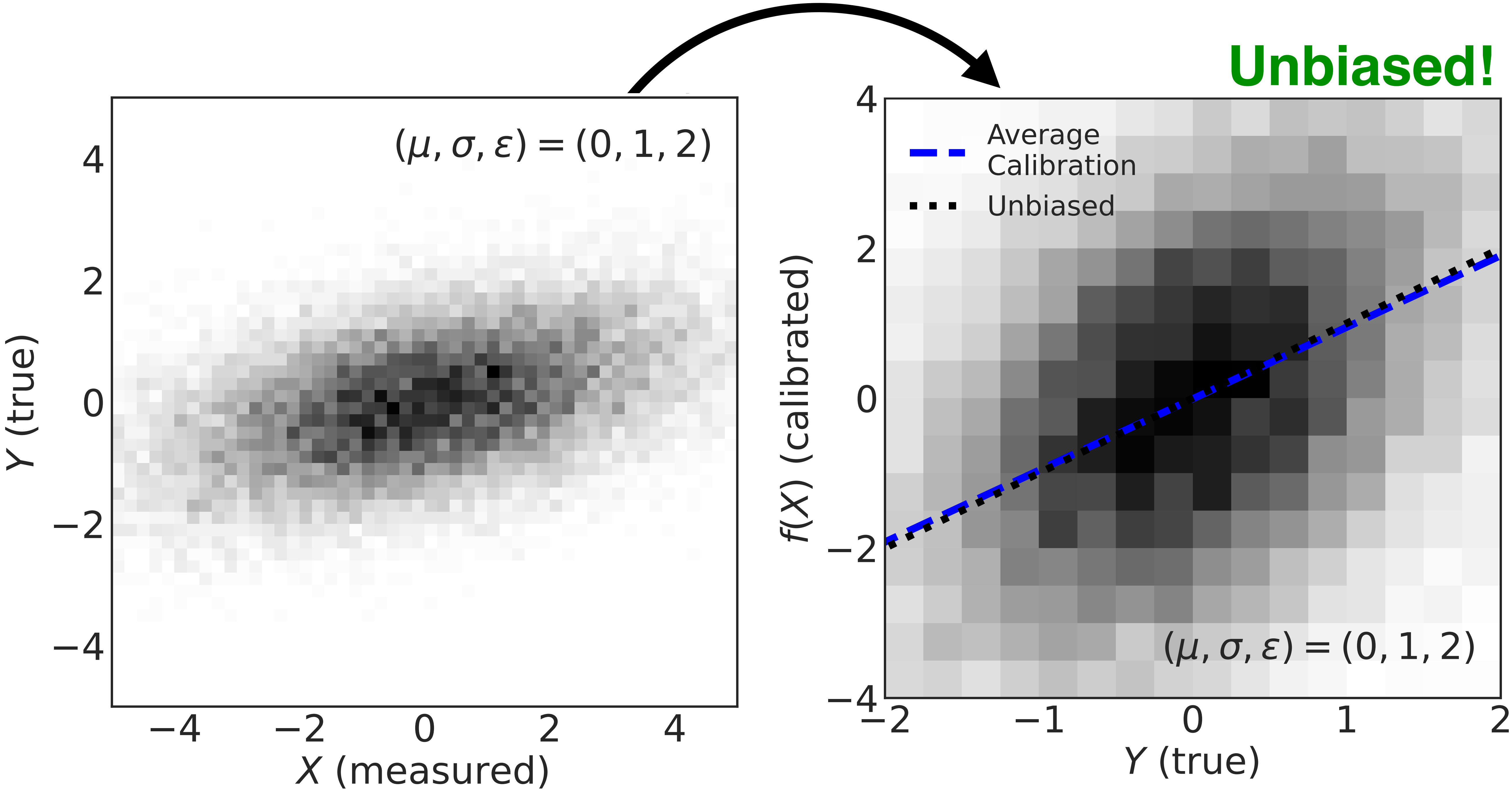
2205.05084

Gaussian Example



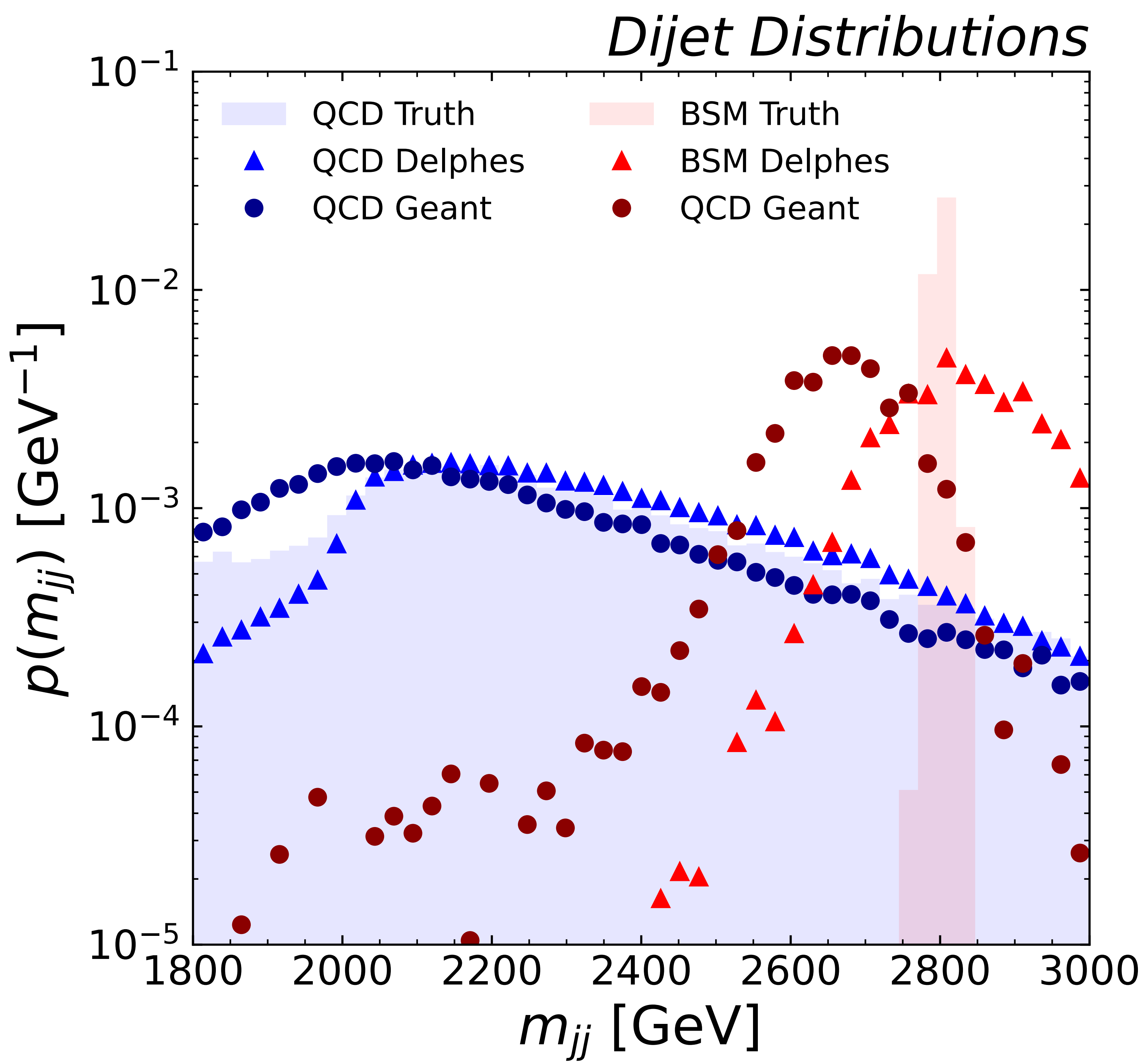
2205.05084

Gaussian Example: MLE instead!



2205.05084, 2404.18992

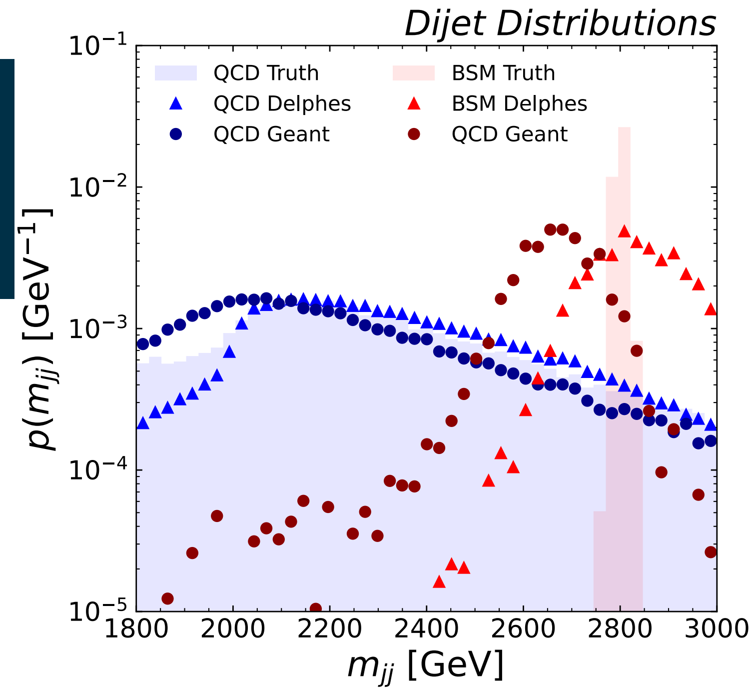
Physics Example

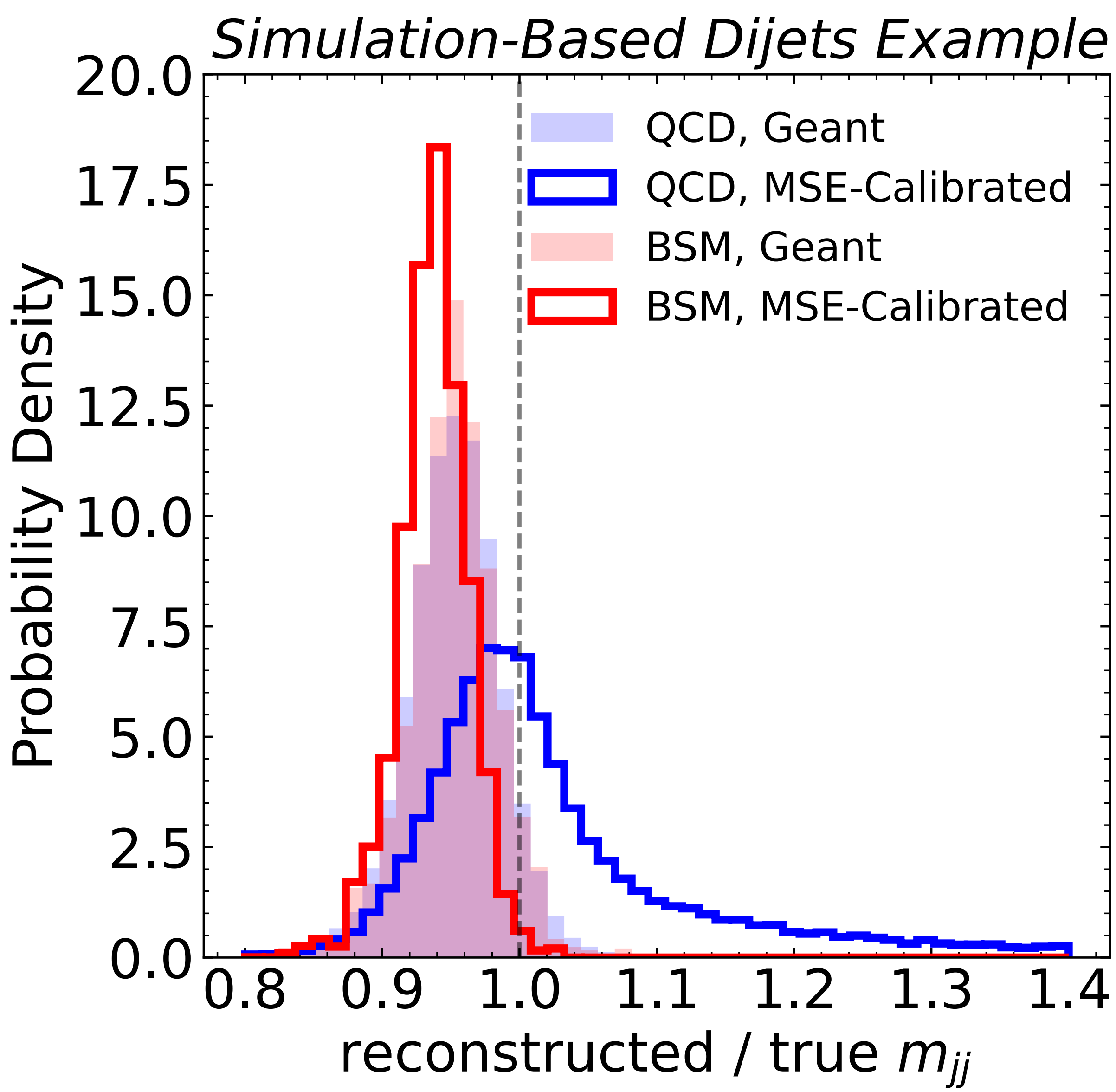


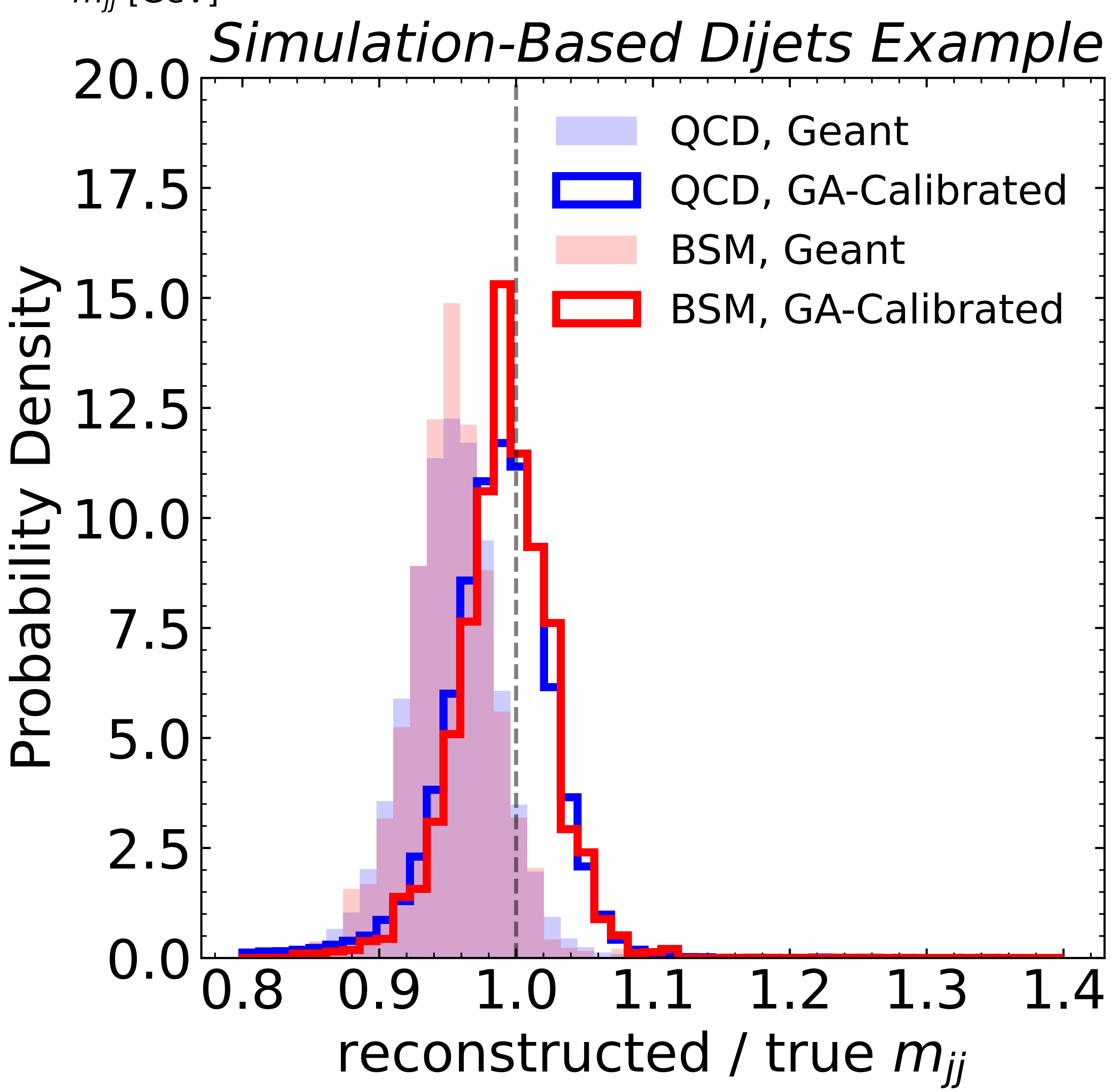
2205.05084; see also ATL-PHYS-PUB-2018-013

Physics Example

GA = (local) MLE







2205.05084; see also ATL-PHYS-PUB-2018-013

Particle Physics + Machine Learning

(II) Theory of everything

Fast simulation / phase space

↓

Physics simulators

↓

Detector-level observables

↓

Pattern recognition

↔

Nature

↓

Experiment

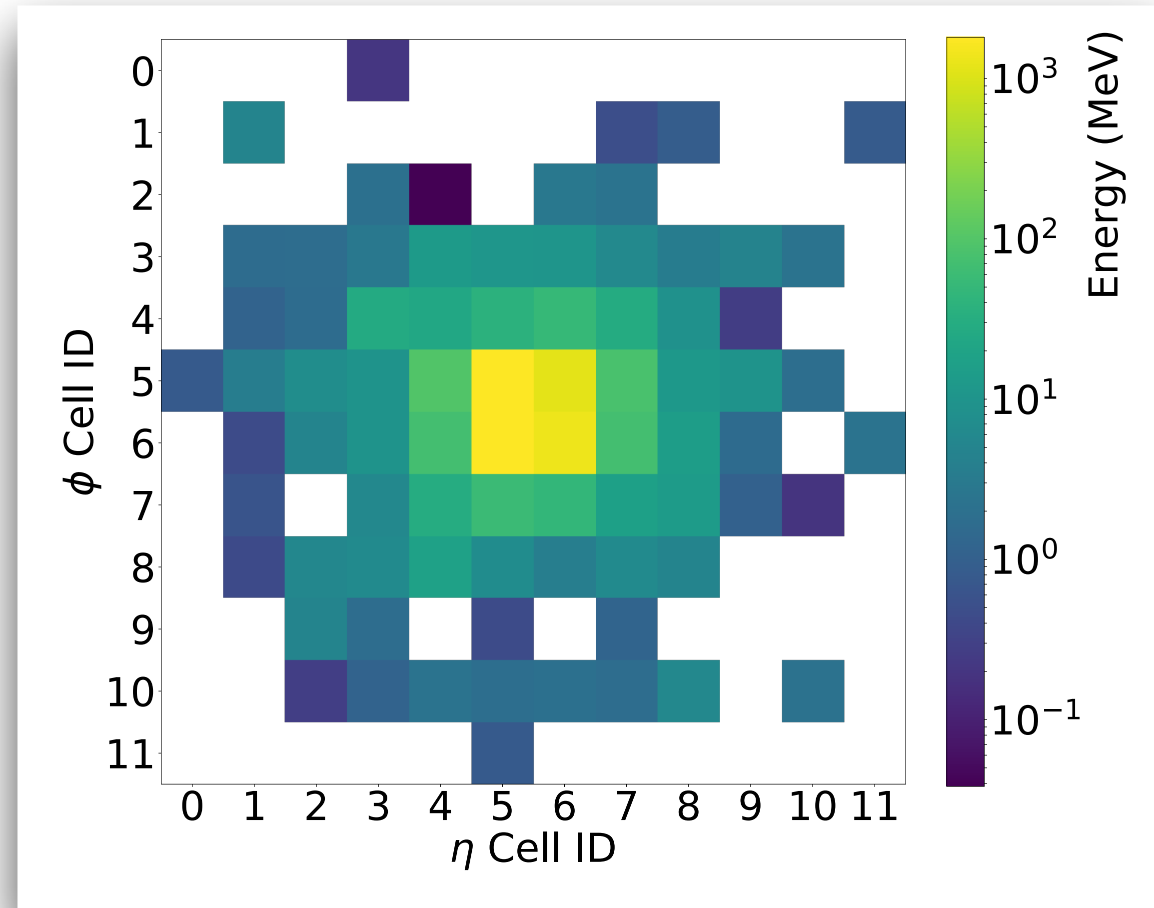
↓

Detector-level observables

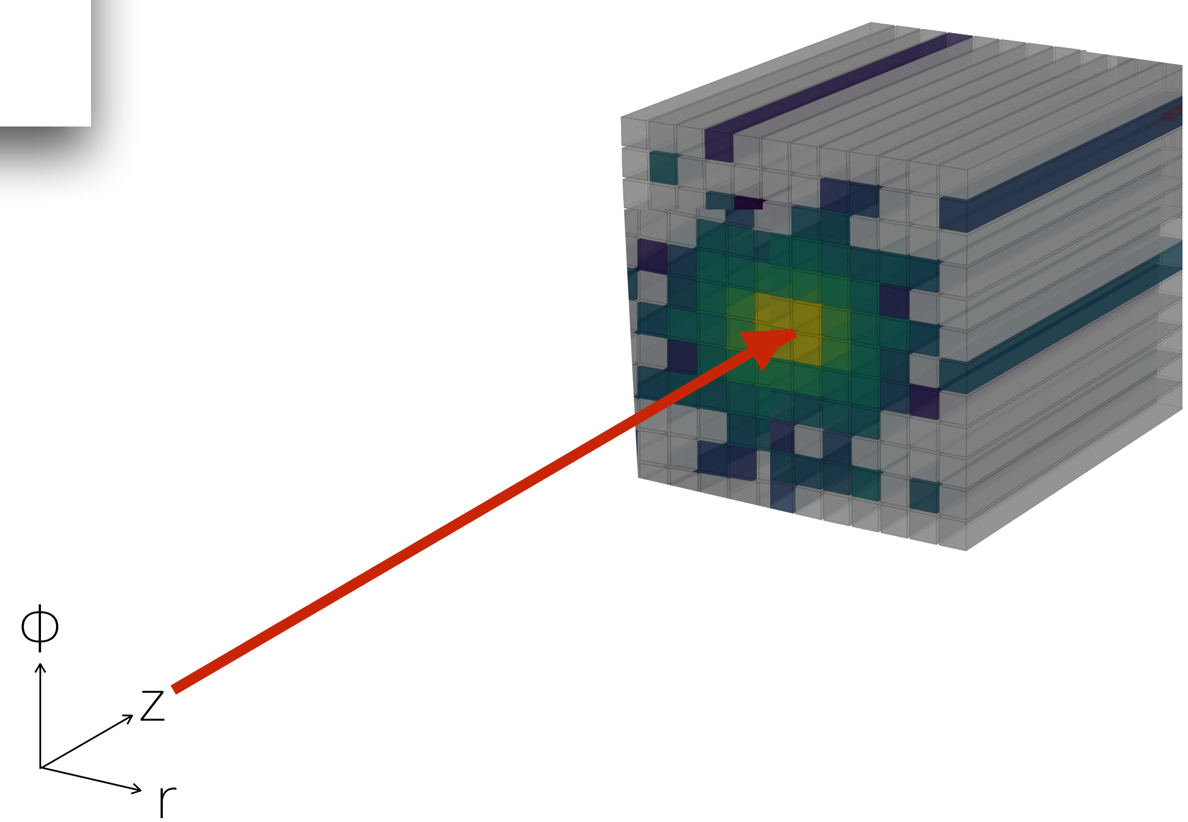
↓

Pattern recognition

Surrogate Models with ML



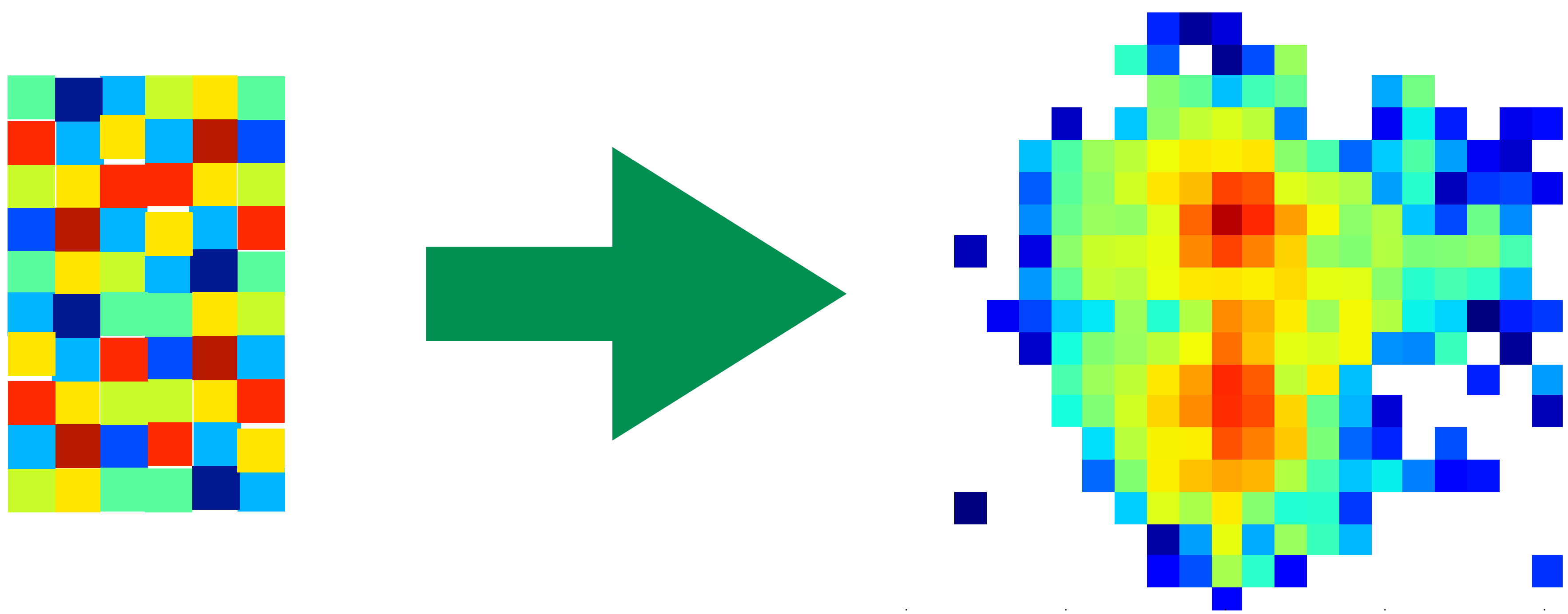
Can we train a neural network to emulate the detector simulation?



Grayscale images: Pixel intensity = energy deposited

Introduction: generative models

A **generator** is nothing other than a function that maps random numbers to structure.



Deep generative models: the map is a deep neural network.

Tools

GANs

Generative Adversarial Networks

Score-based

NFs

Normalizing Flows

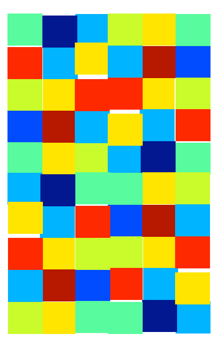
VAEs

Variational Autoencoders

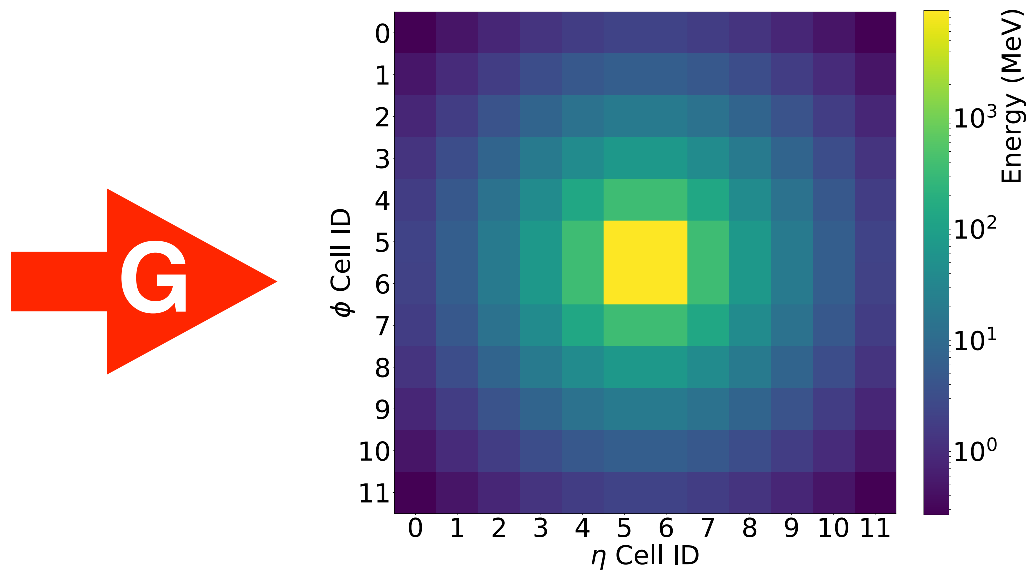
Deep generative models: the map is a deep neural network.

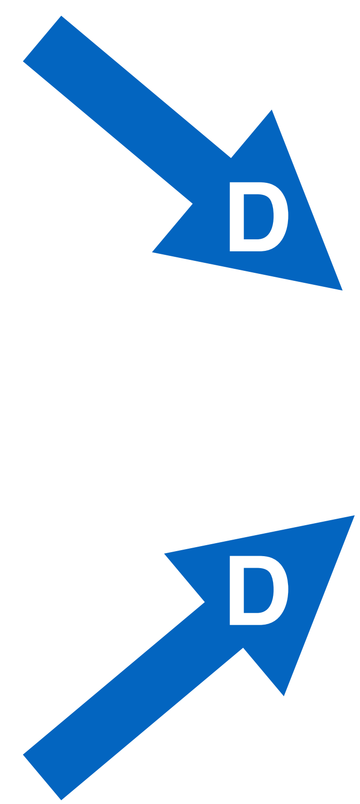
Introduction: GANs

Generative Adversarial Networks (GANs):
A two-network game where one ***maps noise to structure*** *and one* ***classifies images as fake or real****.*



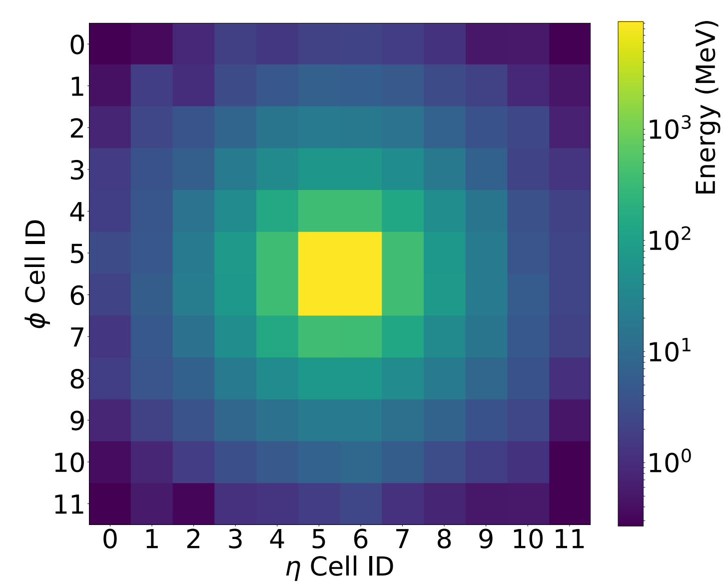
noise





{real,fake}

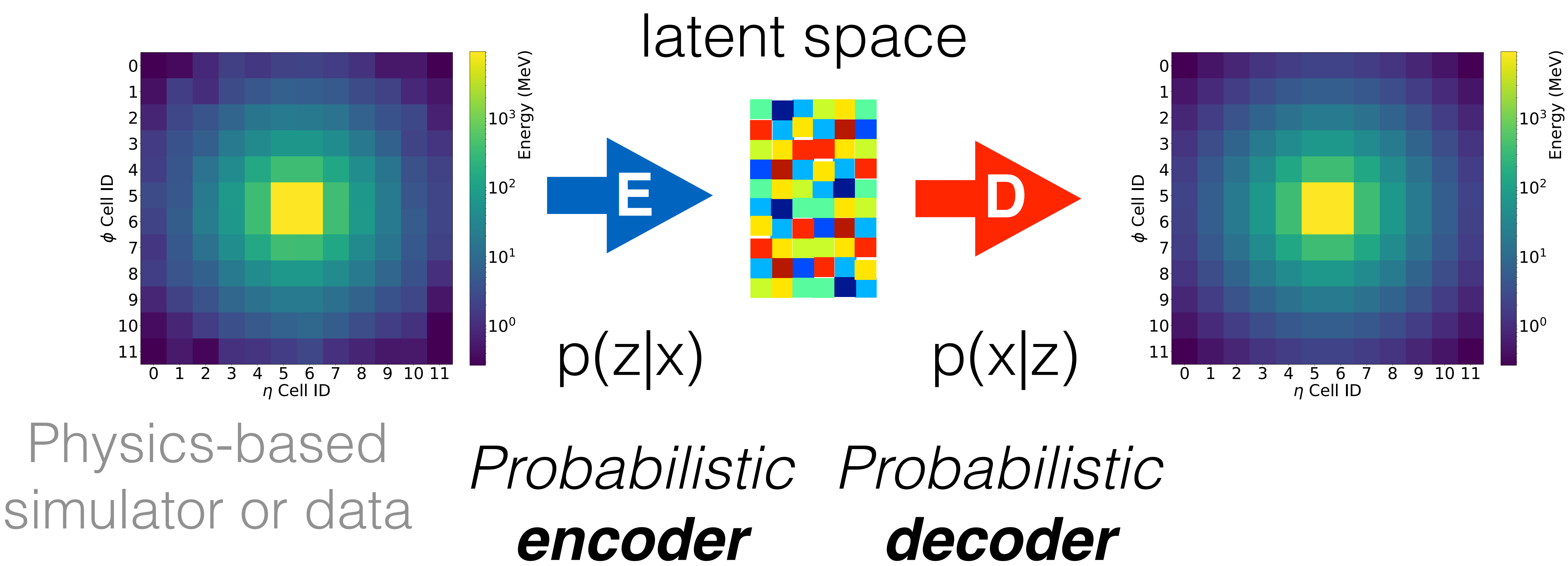
When **D** is maximally confused, **G** will be a good generator



Physics-based simulator or data

Introduction: VAEs

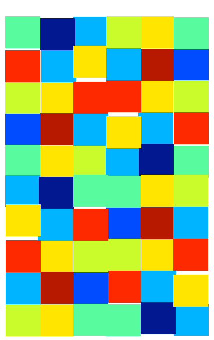
Variational Autoencoders (VAEs):
A pair of networks that embed the data into a latent space with a given prior and decode back to the data space.



Introduction: NFs

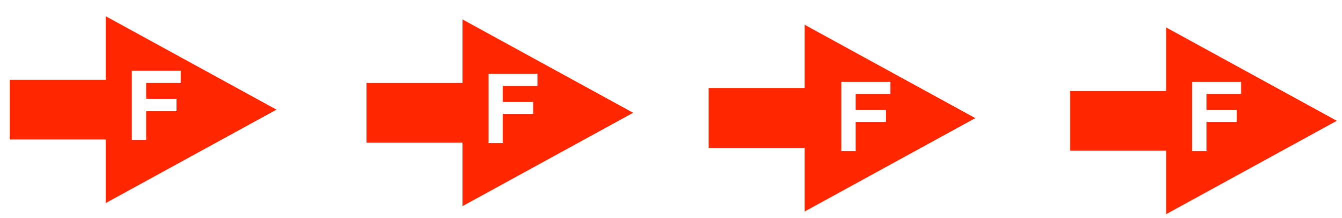
Normalizing Flows (NFs):
A series of invertible transformations mapping a known density into the data density.

Optimize via maximum likelihood

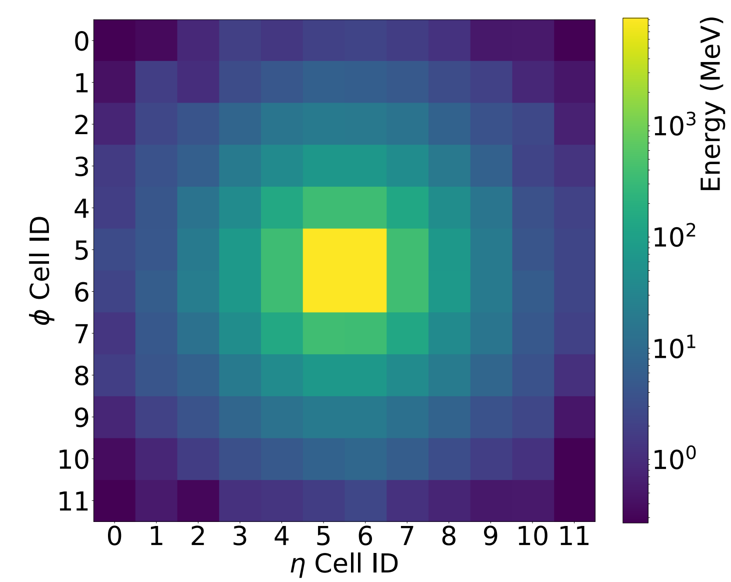


latent space

$p(z)$



Invertible transformations with tractable Jacobians



$$p(x) = p(z)\, |dF^{-1}/dx|$$

$p(x)$

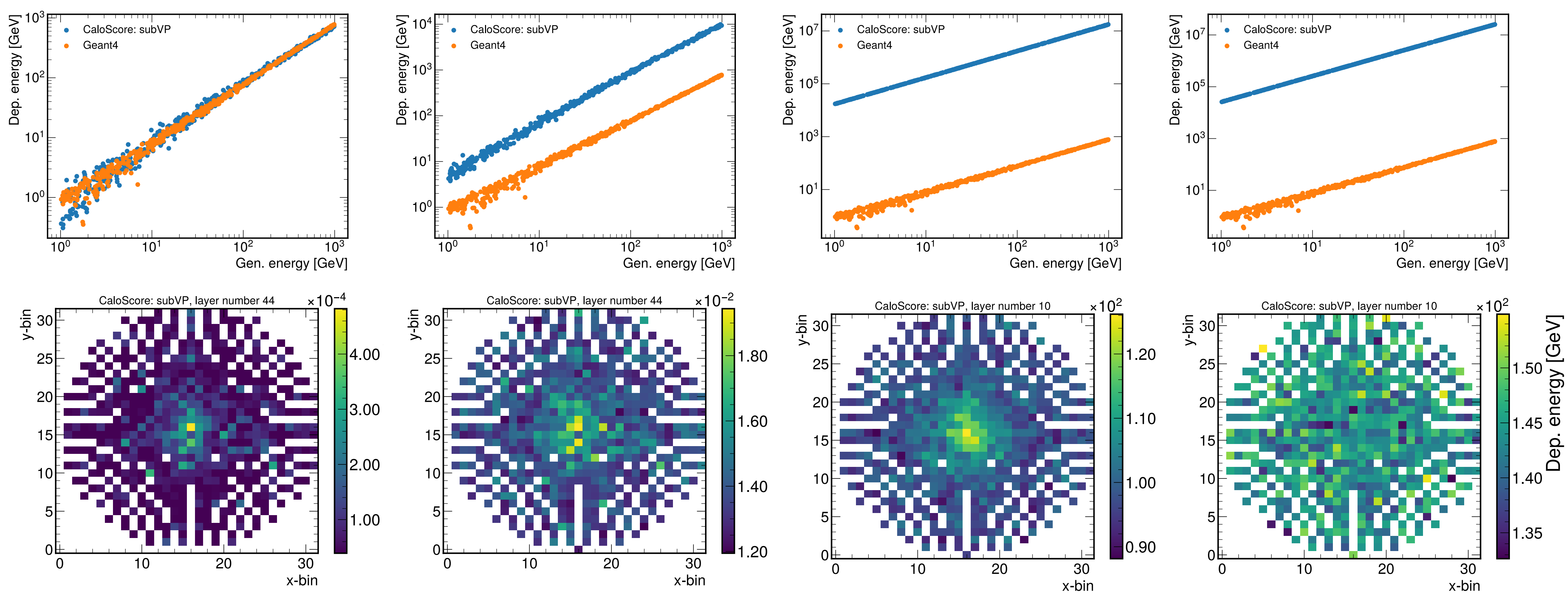
Introduction: Score-based

Score-based

Learn the gradient of the density instead of the probability density itself.

Forward diffusion (training)

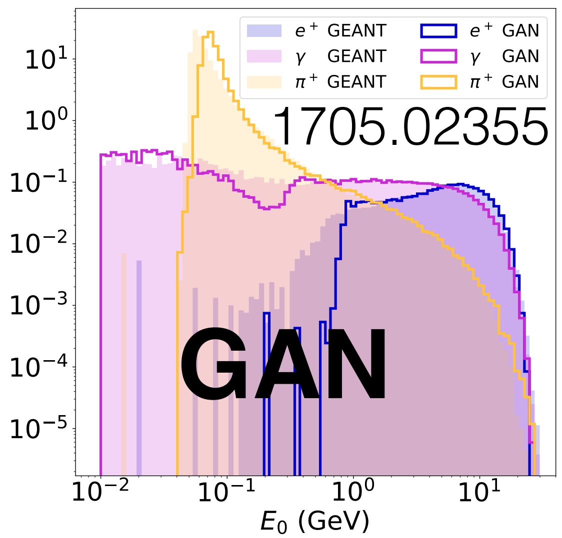
t=0 **t=0.25** **t=0.75** **t=1**

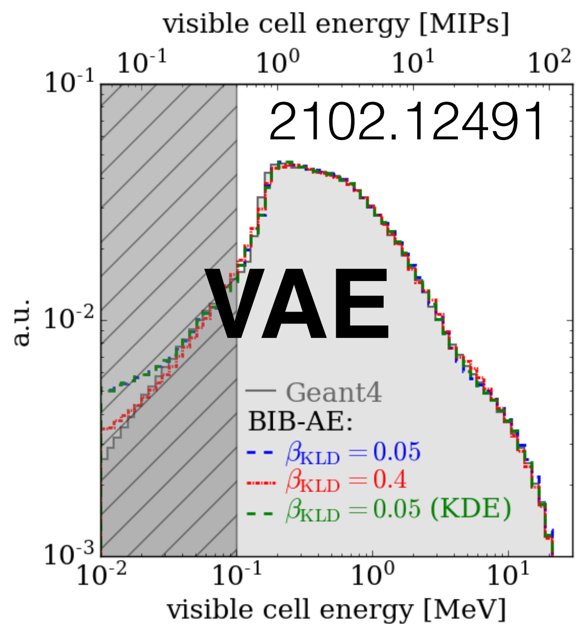


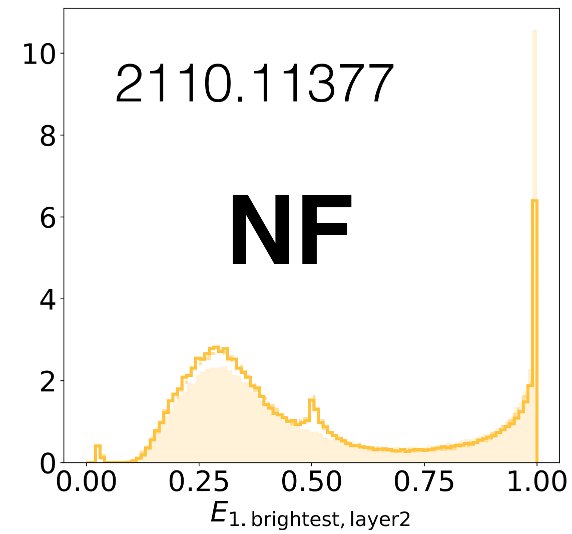
Reverse-time diffusion (data generation)

From 2206.11898

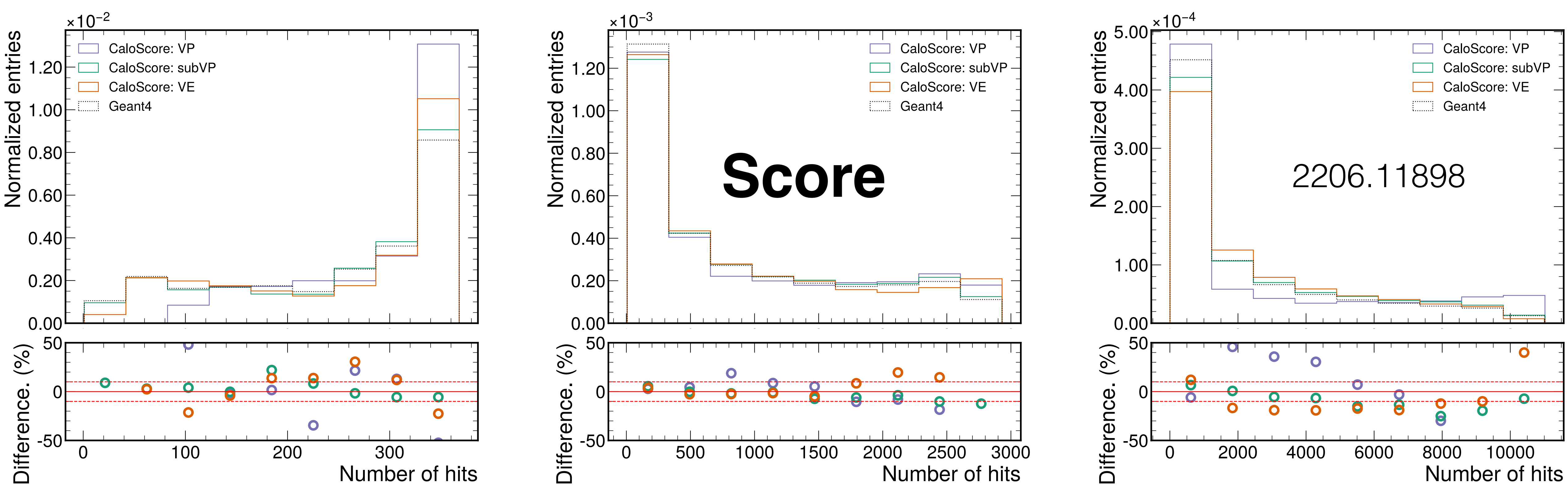
Calorimeter ML Surrogate Models





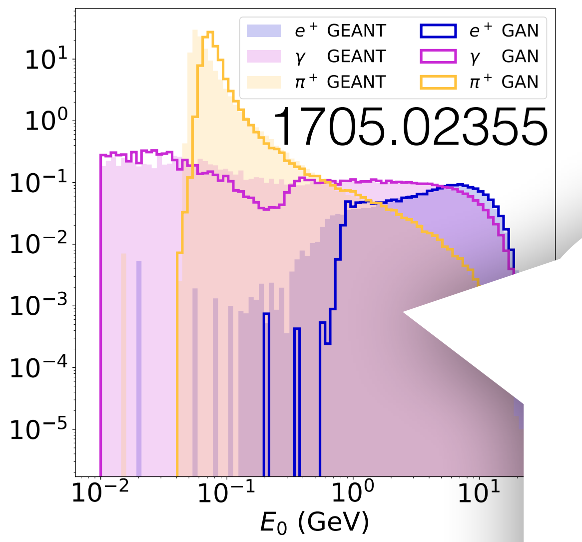


Many papers on this subject - see the living review for all



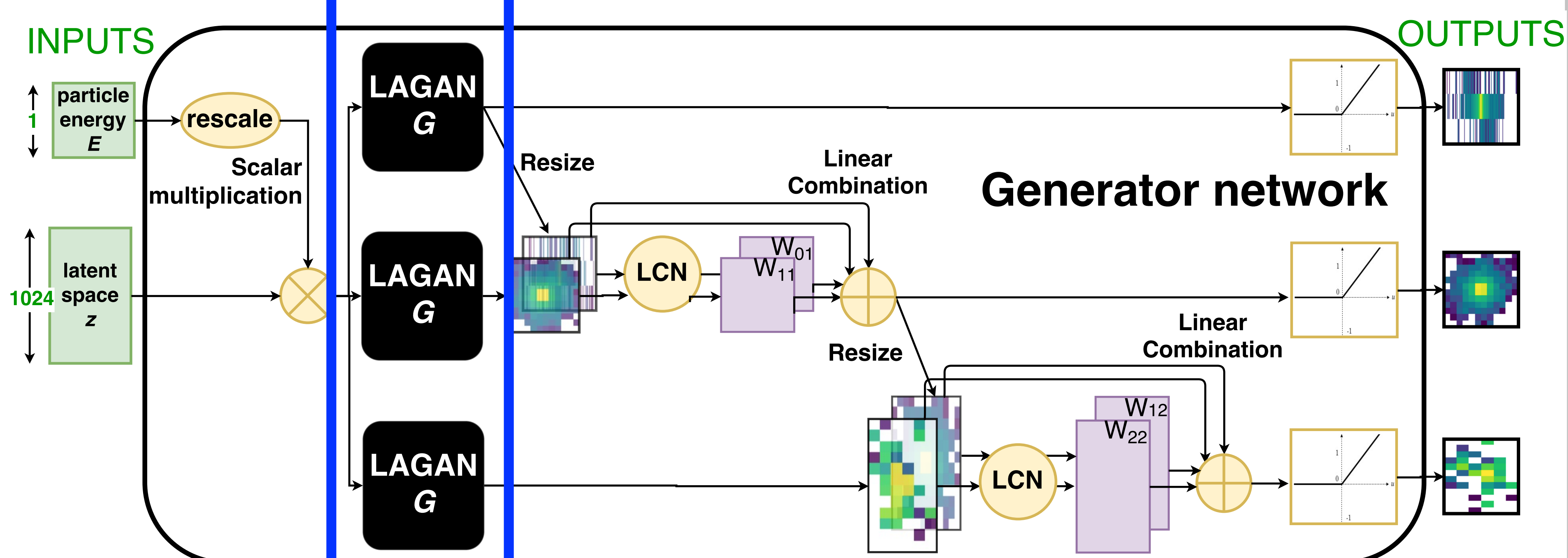
See also https://calochallenge.github.io/homepage/ and https://calochallenge.github.io/homepage/

Calorimeter ML Surrogate Models



One image per calo layer

One network per particle type; input particle energy



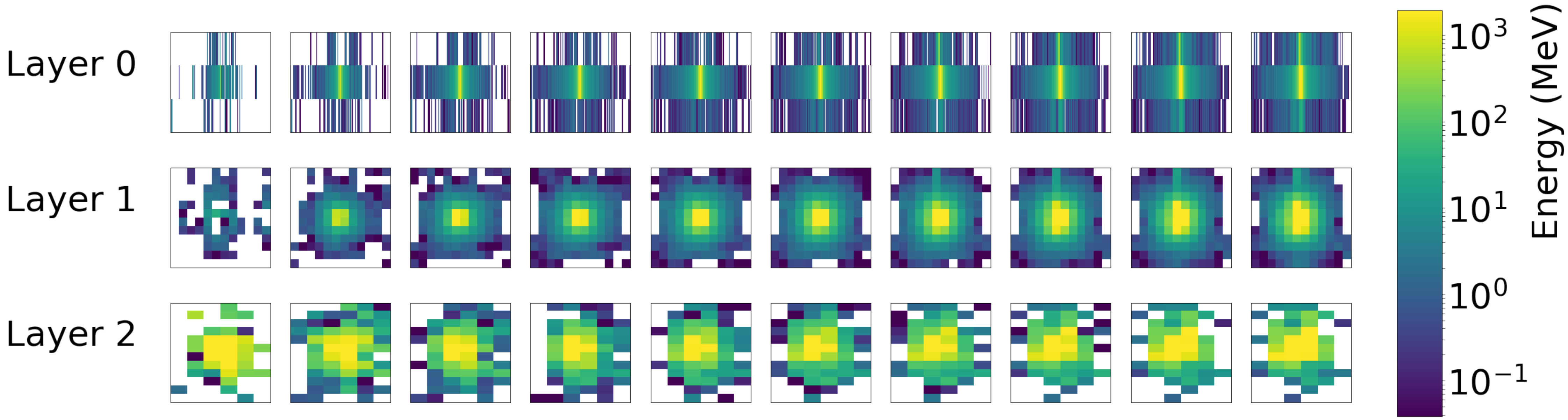
LA = Locally Aware, like a CNN

use layer i as input to layer i+1

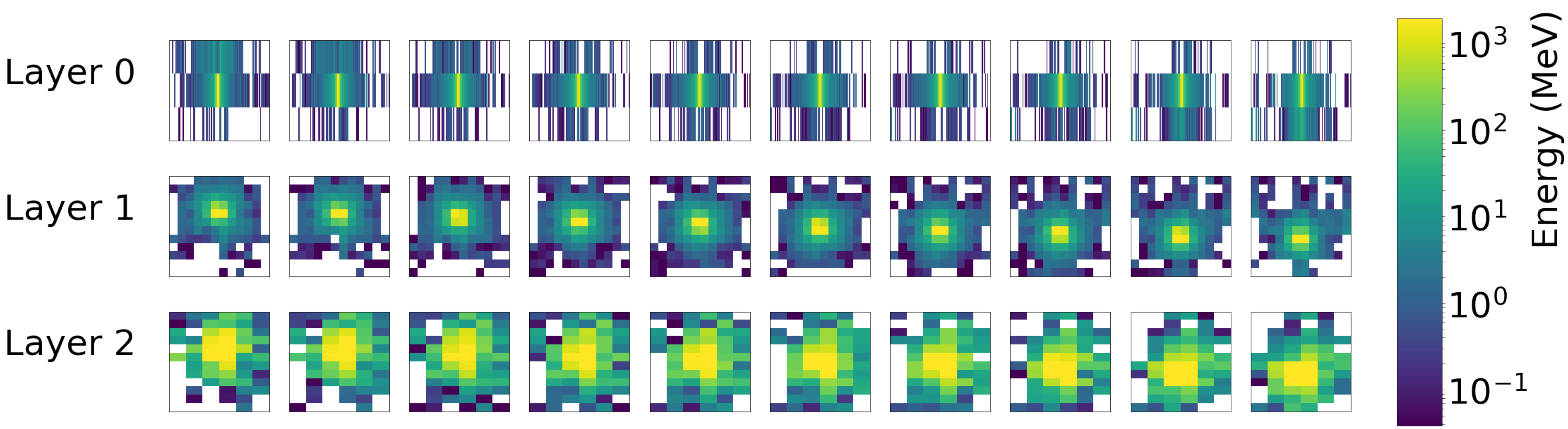
ReLU to encourage sparsity

Conditioning

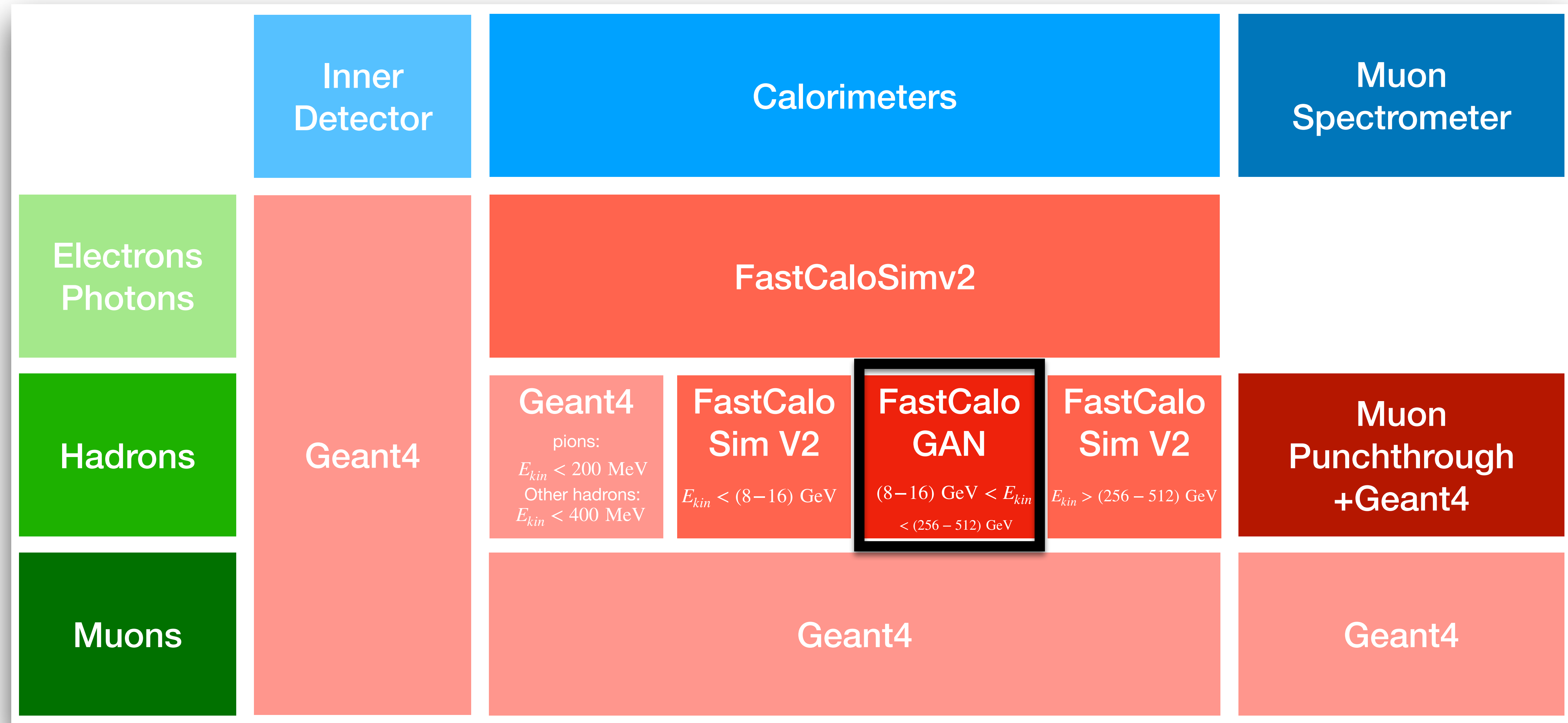
Fix noise, scan latent variable corresponding to energy



Fix noise, scan latent variable corresponding to x-position

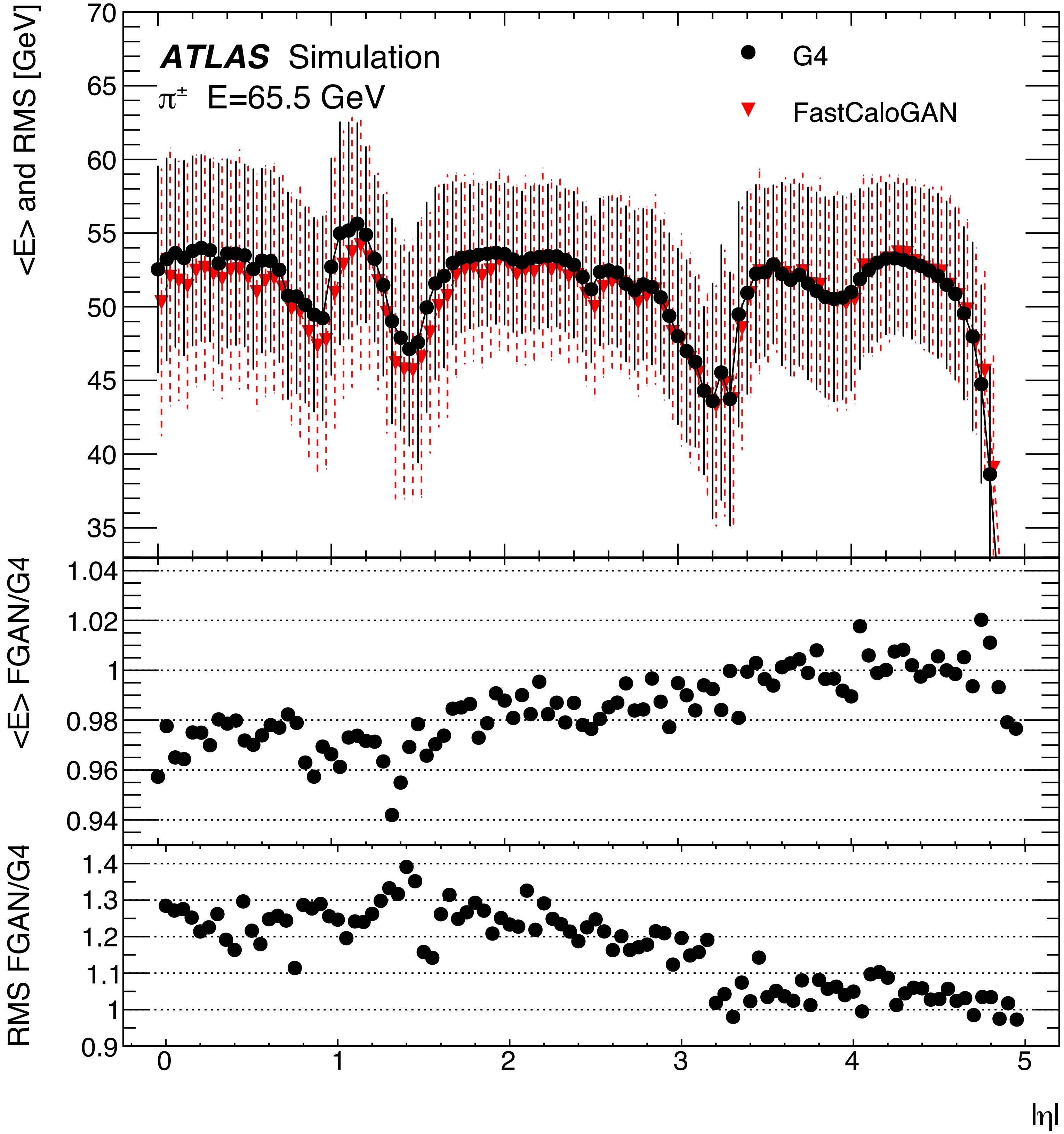


Integration into real detector sim.



Our (ATLAS Collaboration) fast simulation (AF3) now includes a GAN at intermediate energies for pions

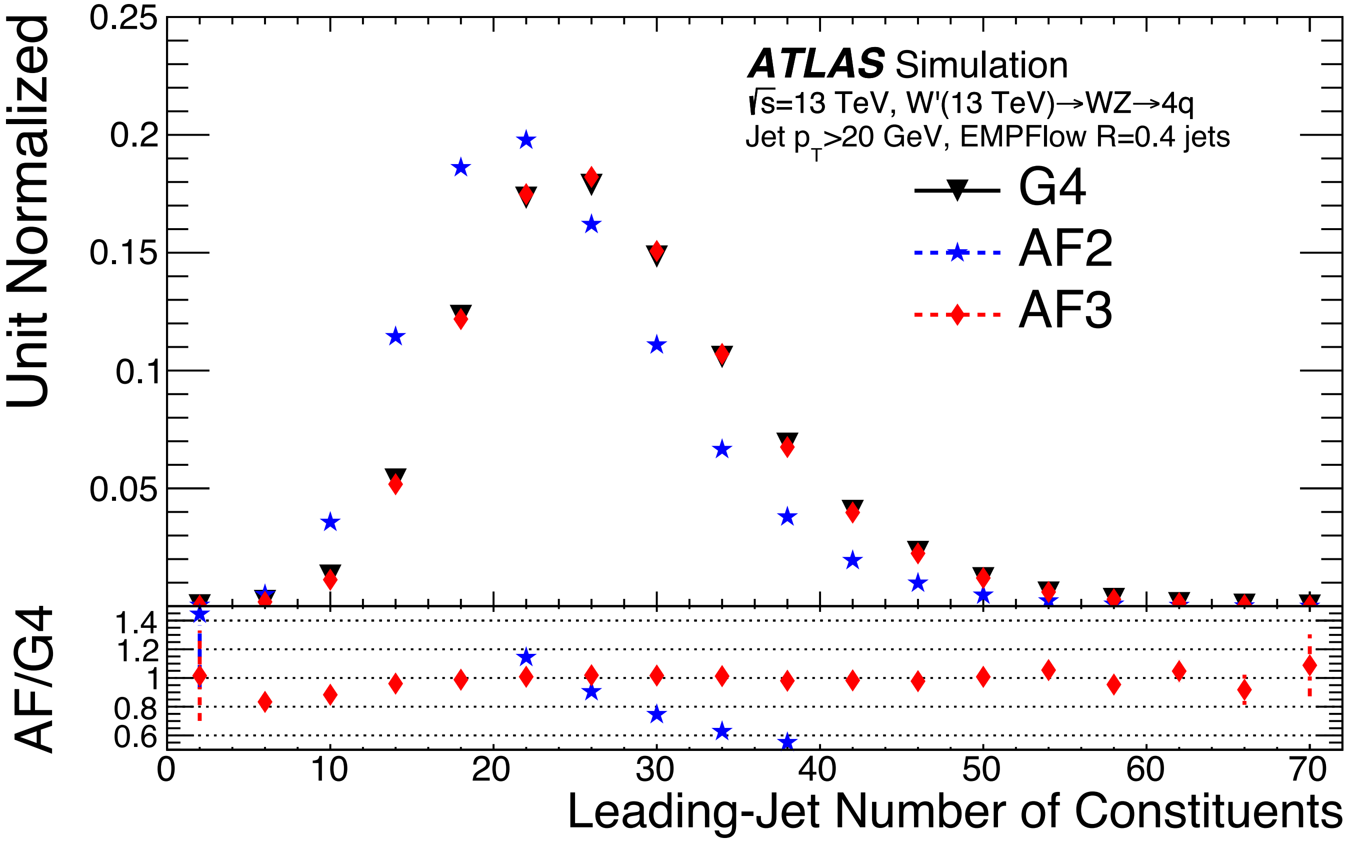
Integration into real detector sim.



The GAN architecture is relatively simple, but it is able to match the energy scale and resolution well.

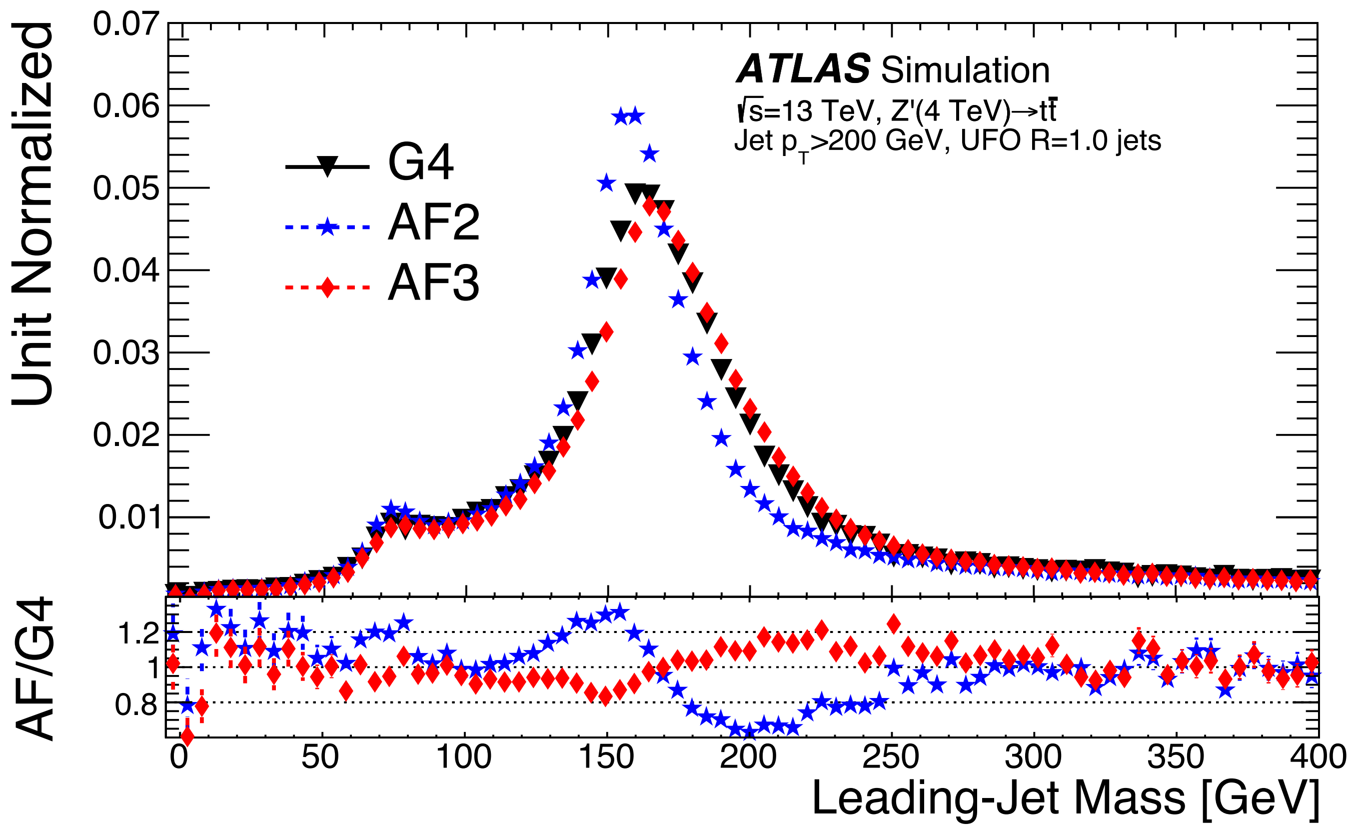
There is one GAN per η slice

Integration into real detector sim.



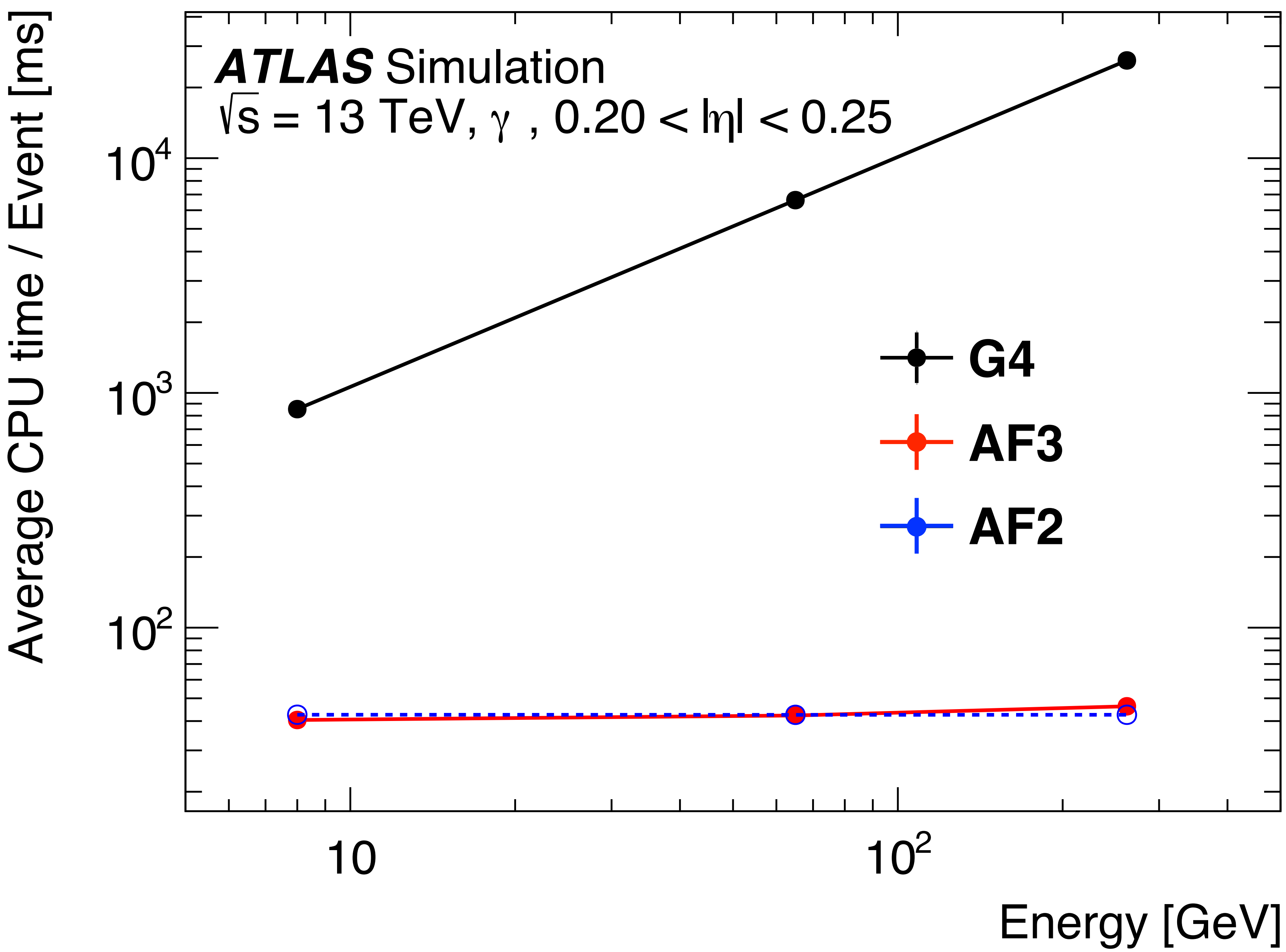
The new fast simulation (**AF3**) significantly improves jet substructure with respect to the older one (**AF2**)

Ideally, the same calibrations derived for full sim. (Geant4-based) can be applied to the fast sim.



ATLAS Collaboration, 2109.02551

Integration into real detector sim.



As expected, the fast sim. timing is independent of energy, while Geant4 requires more time for higher energy.

Statistical Amplification

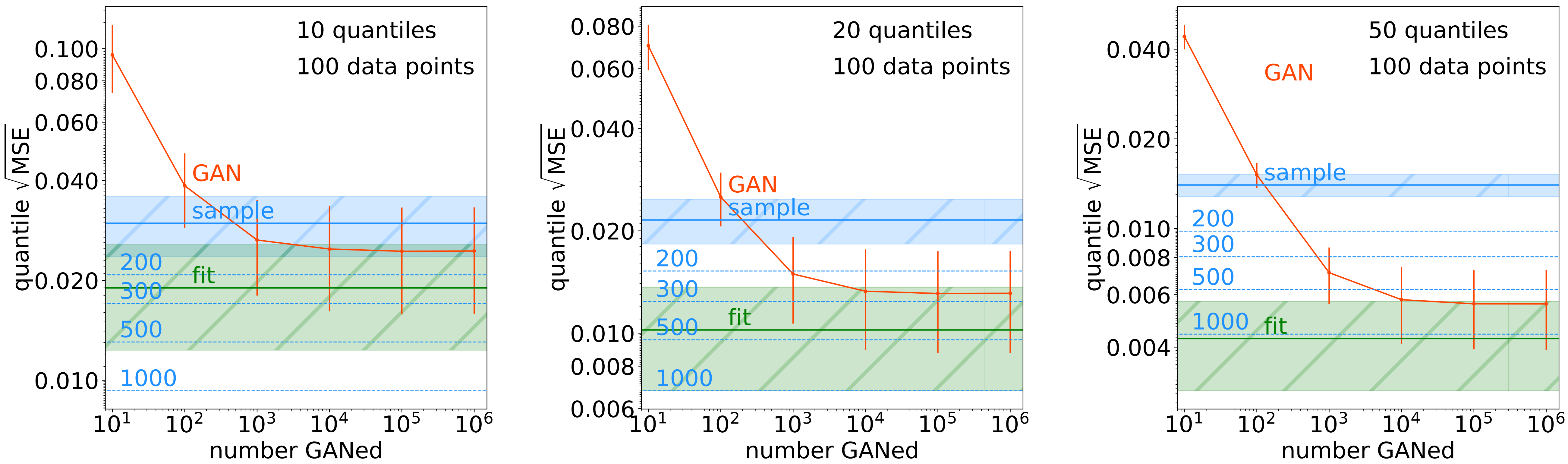
Common question: if we train on N events and sample M >> N events, do we have the statical power of M or N?

No free lunch - only win with **inductive bias**. Examples: factorization, symmetries, smoothness, …

Statistical Amplification

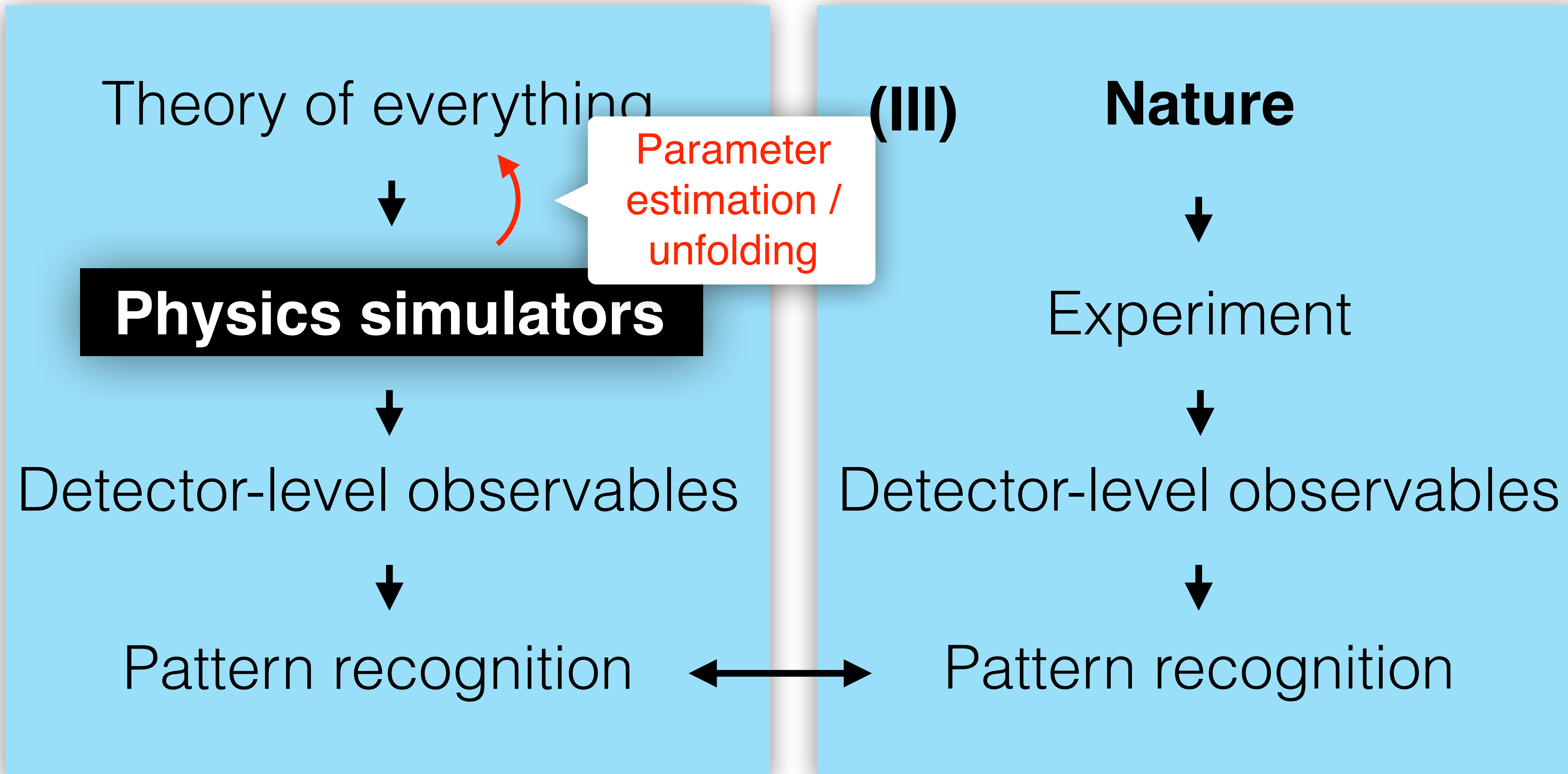
Common question: if we train on N events and sample M >> N events, do we have the statical power of M or N?

No free lunch - only win with **inductive bias**. Examples: factorization, symmetries, smoothness, …



2006.06685; 2202.07352

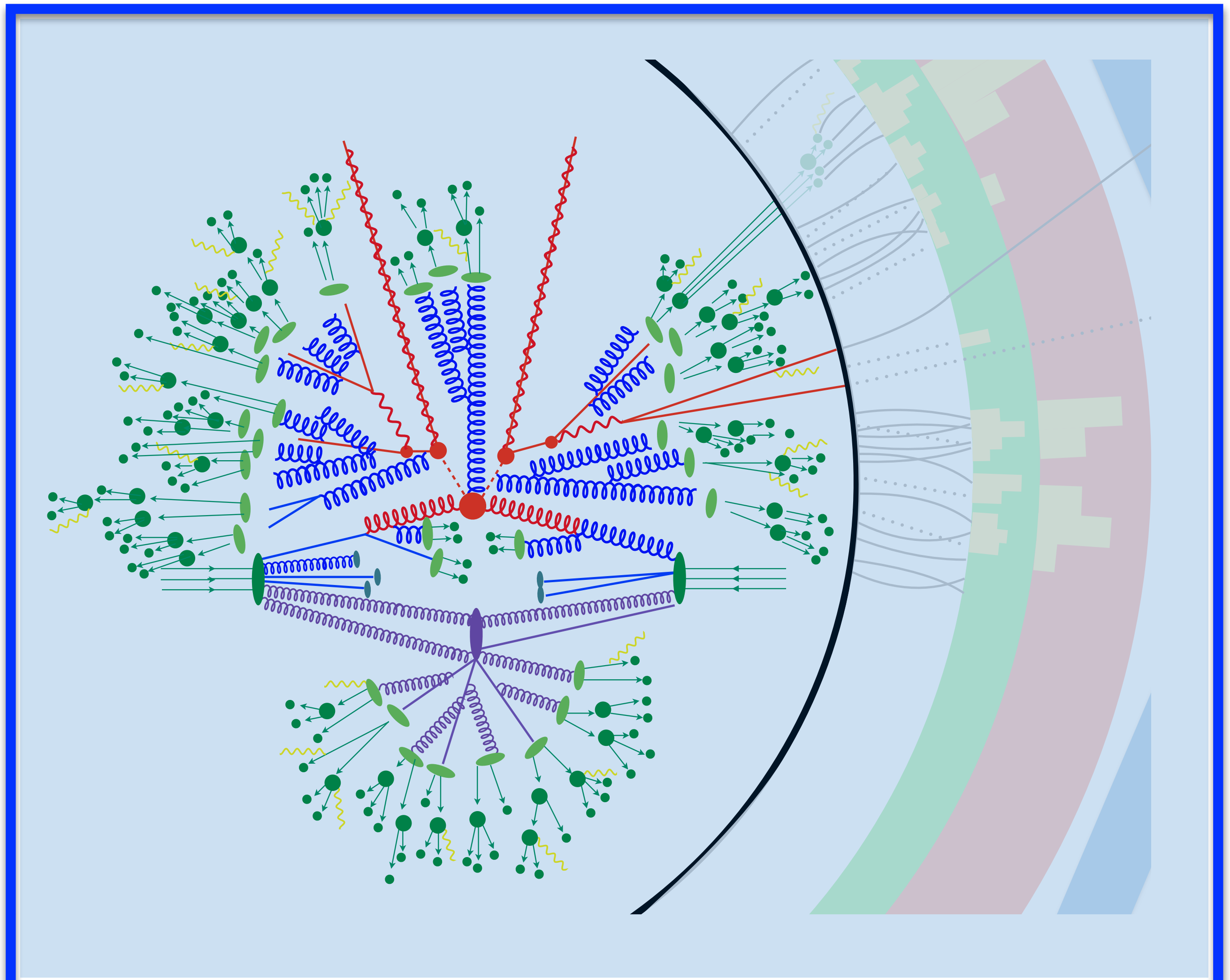
Particle Physics + Machine Learning



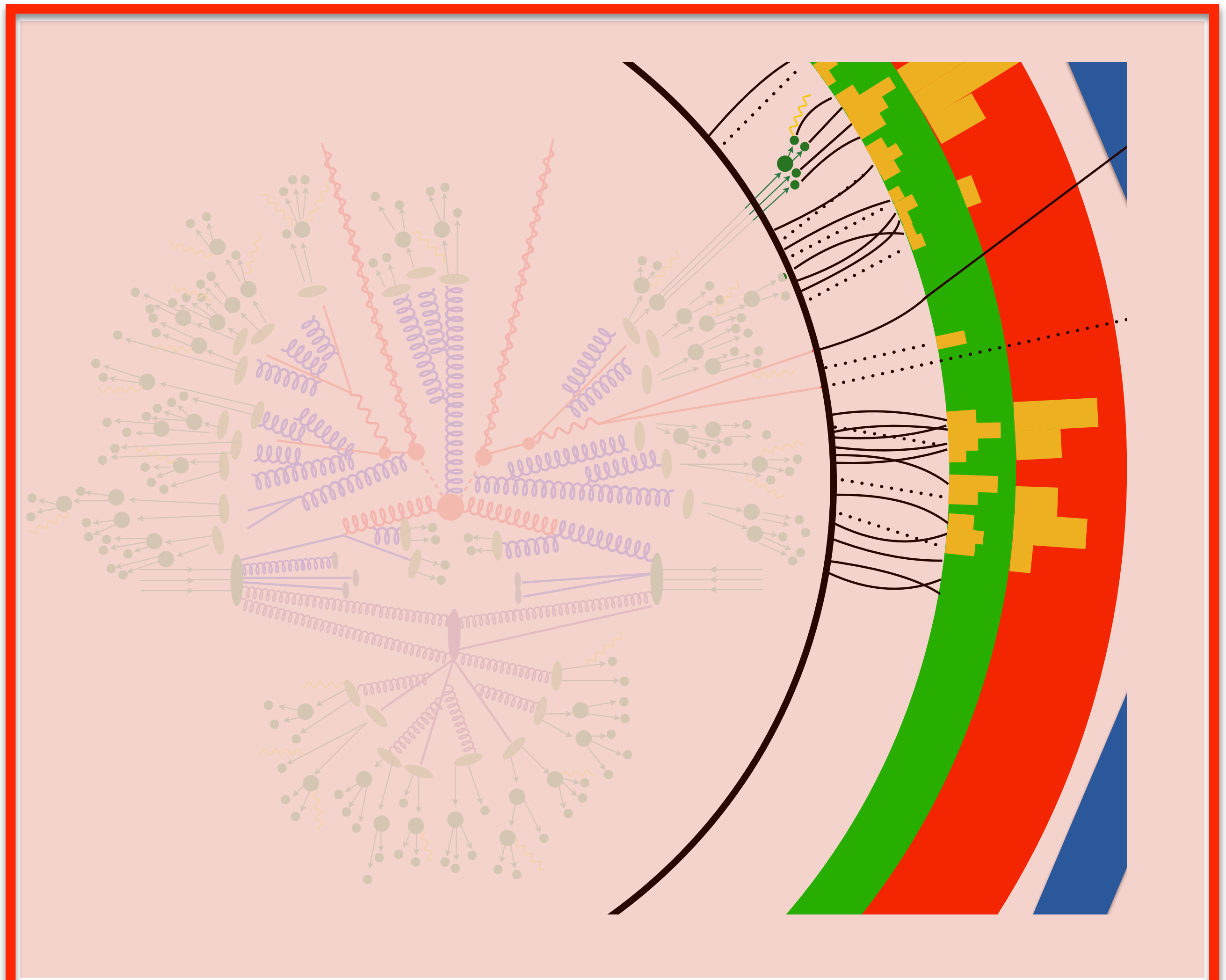
Inverse Problems

Want this

(or the parameters of the generative model)



Measure this

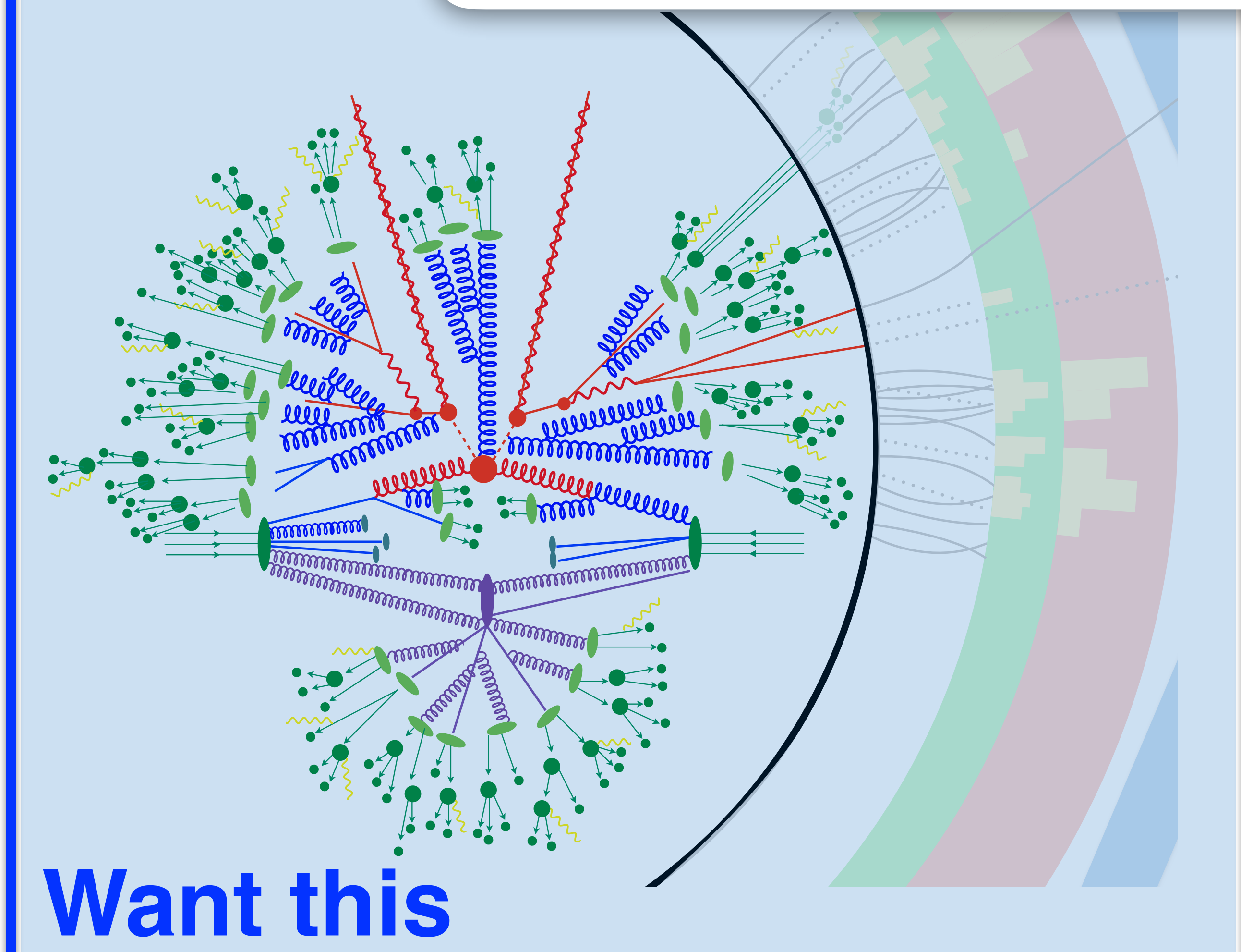


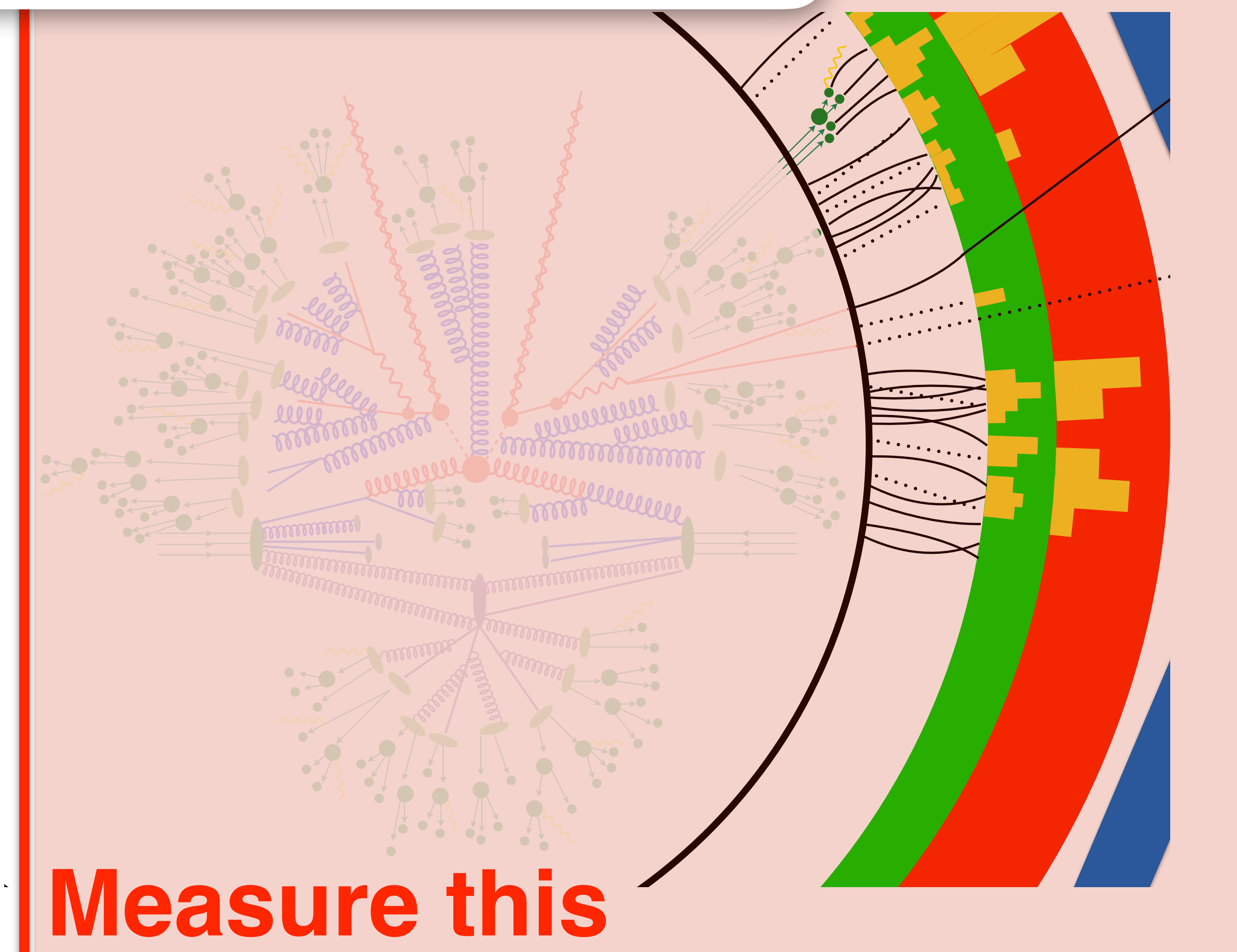
remove detector distortions (unfolding) or parameter estimation

Inverse Problems

If you know *p*(***meas.*** / ***true***), could do maximum likelihood, i.e.

$$\text{unfolded} = \underset{\text{true}}{\operatorname{argmax}}\ p(\text{measured} \mid \text{true})$$





For parameter estimation, replace ***true*** *with* θ

Inverse Problems

If you know *p(**meas.** | **true**)*, could do maximum likelihood, i.e.

$$\textit{unfolded} = \underset{\textbf{true}}{\operatorname{argmax}}\; p(\textbf{measured} \mid \textbf{true})$$

Challenge: **measured** is hyperspectral and **true** is hypervariate … *p(**meas.** | **true**) is **intractable** !*

*For parameter estimation, replace **true** with θ*

Inverse Problems

If you know *p(**meas.** / **true**)*, could do maximum likelihood, i.e.

$$\text{unfolded} = \underset{\text{true}}{\operatorname{argmax}}\; p(\text{measured} \,/\, \text{true})$$

Challenge: **measured** is hyperspectral and **true** is hypervariate … *p(**meas.** | **true**) is **intractable** !*

However: we have **simulators** that we can use to sample from *p(**meas.** | **true**)*

→ **Simulation-based (likelihood-free) inference**

*For parameter estimation, replace **true** with θ*

I’ll briefly show you one solution to give you a sense of the power of likelihood-free inference.

Reweighting

I'll briefly show you one solution to give you a sense of the power of likelihood-free inference.

The solution will be built on ***reweighting***

dataset 1: sampled from ***p(x)***
dataset 2: sampled from ***q(x)***

Create weights ***w(x) = q(x)/p(x)*** so that when dataset 1 is weighted by ***w***, it is statistically identical to dataset 2.

Reweighting

I'll briefly show you one solution to give you a sense of the power of likelihood-free inference.

The solution will be built on ***reweighting***

dataset 1: sampled from ***p(x)***
dataset 2: sampled from ***q(x)***

Create weights ***w(x) = q(x)/p(x)*** so that when dataset 1 is weighted by ***w***, it is statistically identical to dataset 2.

What if we don't (and can't easily) know ***q*** and ***p***?

Classification for reweighting

Fact: Neutral networks learn to approximate the likelihood ratio $= q(x)/p(x)$

(or something monotonically related to it in a known way)

Solution: train a neural network to distinguish the two datasets!

This turns the problem of **density estimation** (**hard**) into a problem of **classification** (**easy**)

Proof of fact

$$L[f] = \sum (f(x_i) - c)^2$$

Try yourself with BCE!

$$\approx \int dx\, p(x, c)\, (f(x) - c)^2$$

$$\frac{\delta L[f, f']}{\delta f} = \frac{\partial L}{\partial f} - \frac{d}{dx}\frac{\partial L}{\partial f'} = 0$$

Euler-Lagrange Equation

Proof of fact

$$L[f] = \sum (f(x_i) - c)^2$$

Try yourself with BCE!

$$\approx \int dx\, p(x, c)\, (f(x) - c)^2$$

$$\frac{\delta L[f, f']}{\delta f} = \frac{\partial L}{\partial f} - \cancel{\frac{d}{dx}\frac{\partial L}{\partial f'}} = 0$$

Euler-Lagrange Equation

Basically just a regular derivative:

$$\int dc\, p(x, c)(f(x) - c) = 0 \implies f(x) = E[c \,|\, x]$$

Classification for reweighting

Fact: Neutral networks learn to approximate the likelihood ratio $= q(x)/p(x)$

(or something monotonically related to it in a known way)

Solution: train a neural network to distinguish the two datasets!

This turns the problem of **density estimation** (**hard**) into a problem of **classification** (**easy**)

Example

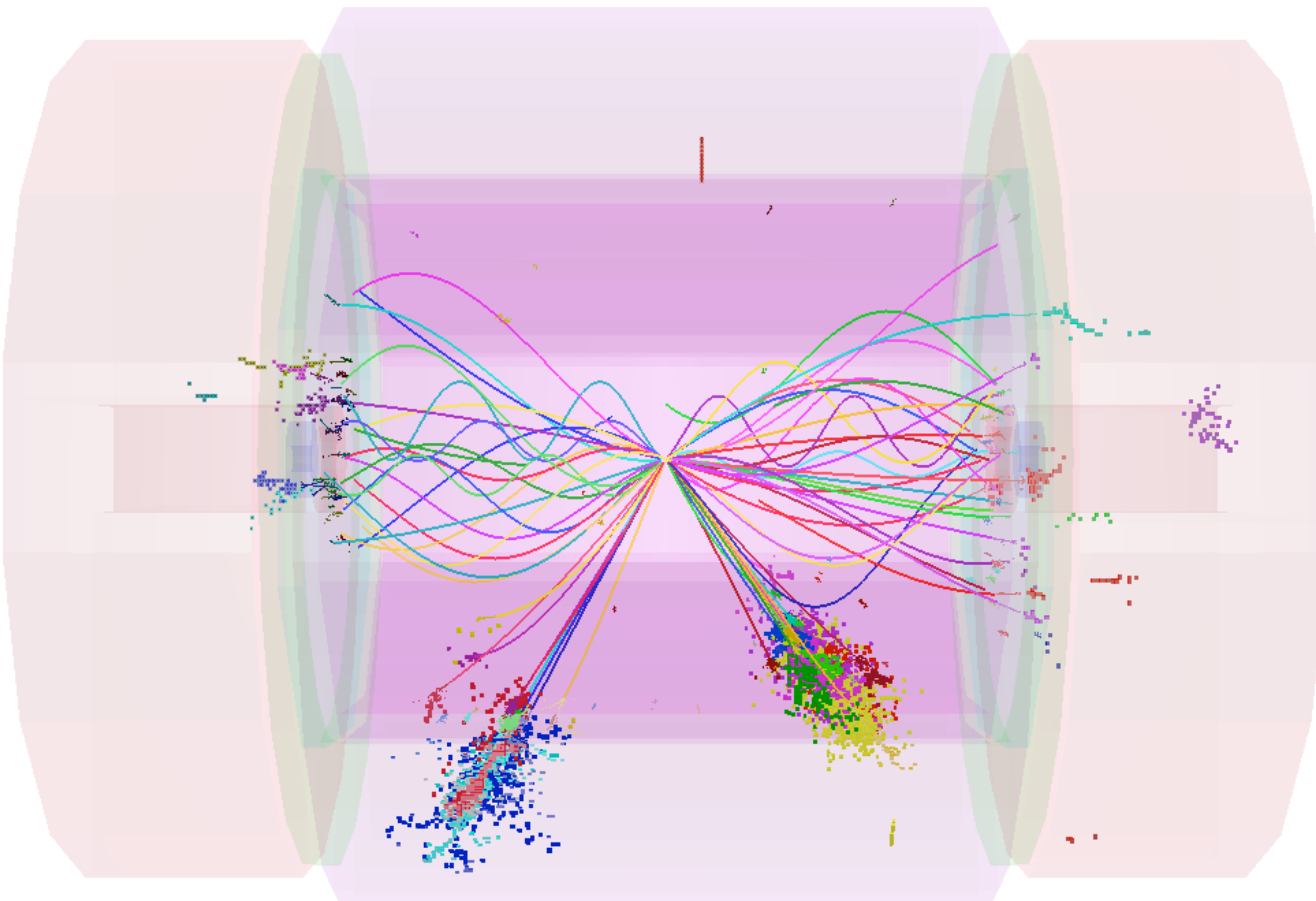
Here, instead of emulating $p(x\,|\,\theta)$ directly, we learn $\dfrac{p(x\,|\,\theta)}{p(x\,|\,\theta_0)}$

(turns the problem of generation into classification)

Example

Here, instead of emulating $p(x\,|\,\theta)$ directly, we learn $\dfrac{p(x\,|\,\theta)}{p(x\,|\,\theta_0)}$

(turns the problem of generation into classification)



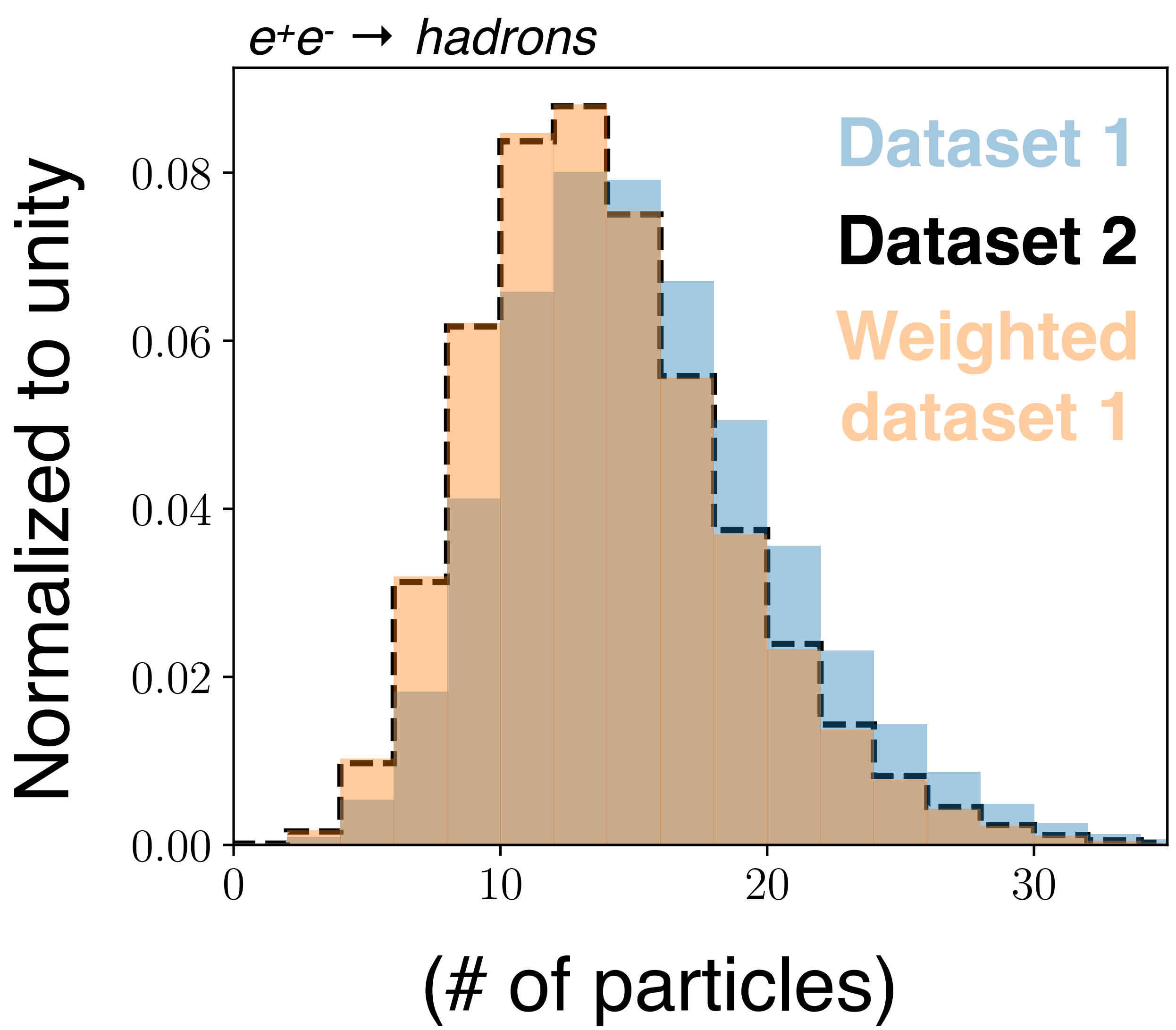
Benefit: easy to integrate complex data structure (symmetries, etc.)

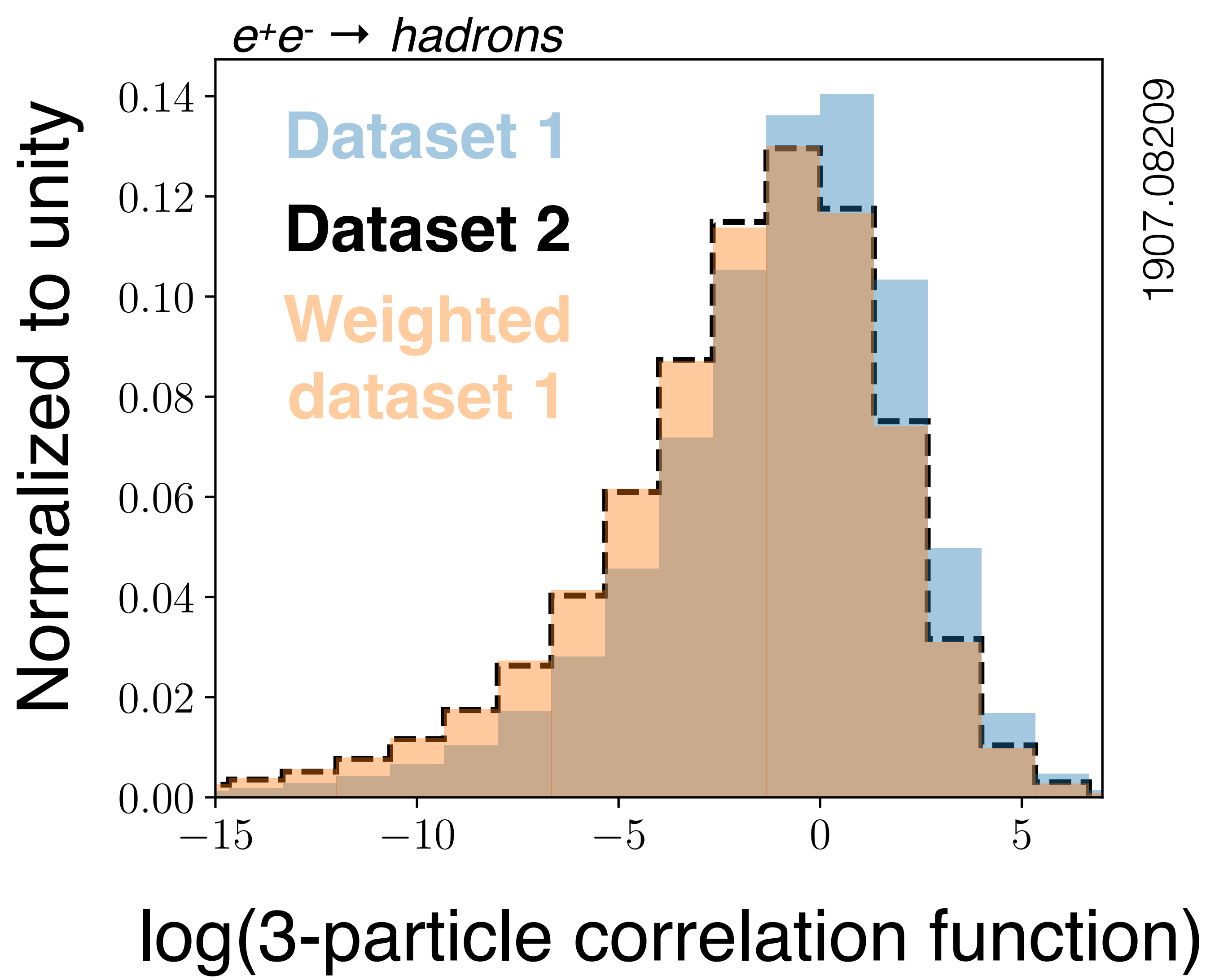
Downside: large weights when θ is far from θ_0

Image: Linear Collider Detector Project

Classification for reweighting

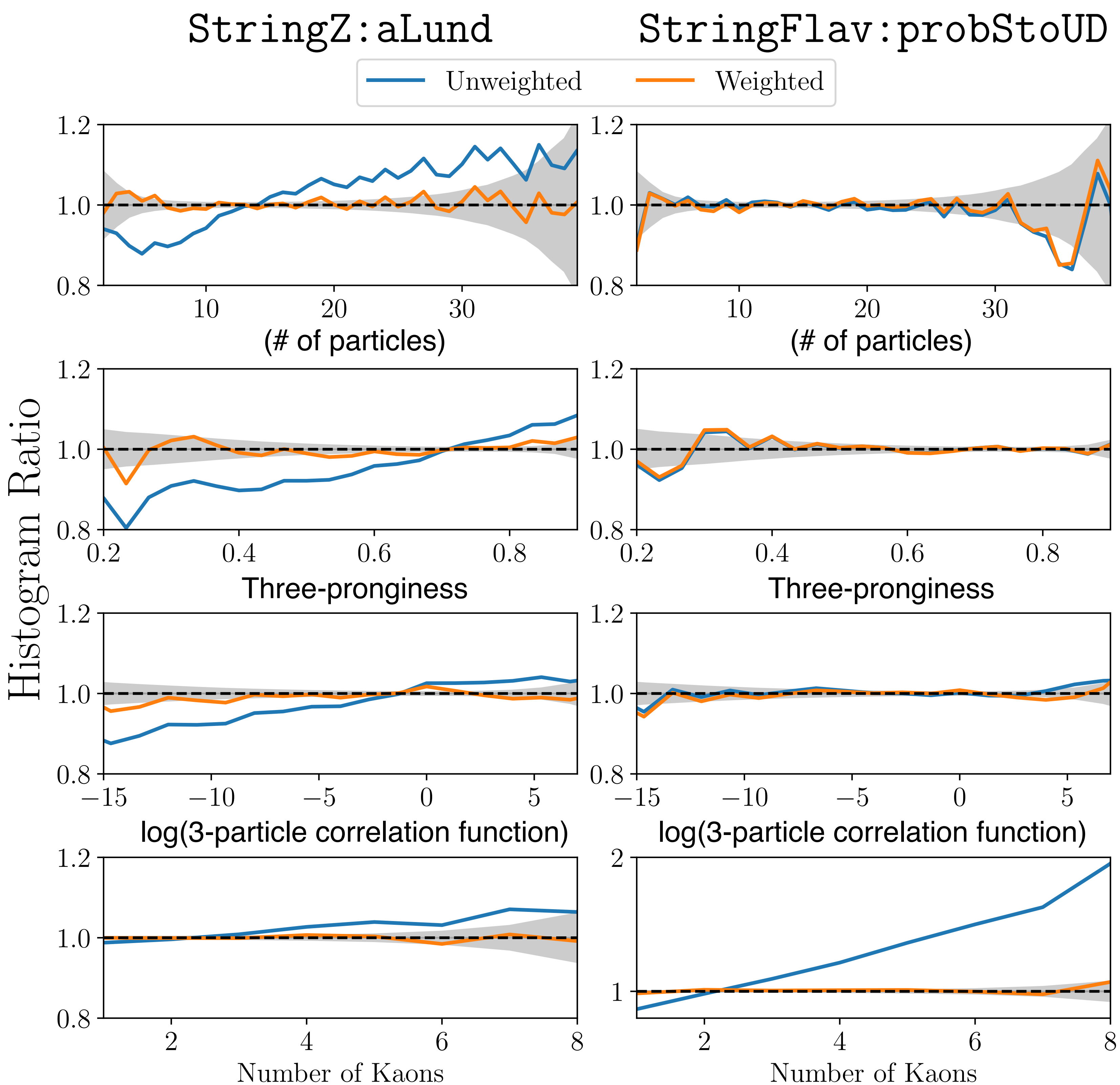
Reweight the **full phase space** and then check for various binned 1D observables.





1907.08209

Achieving precision



Works also when the differences between the two simulations are **small** (left) or **localized** (right).

These are histogram ratios for a series of one-dimensional observables

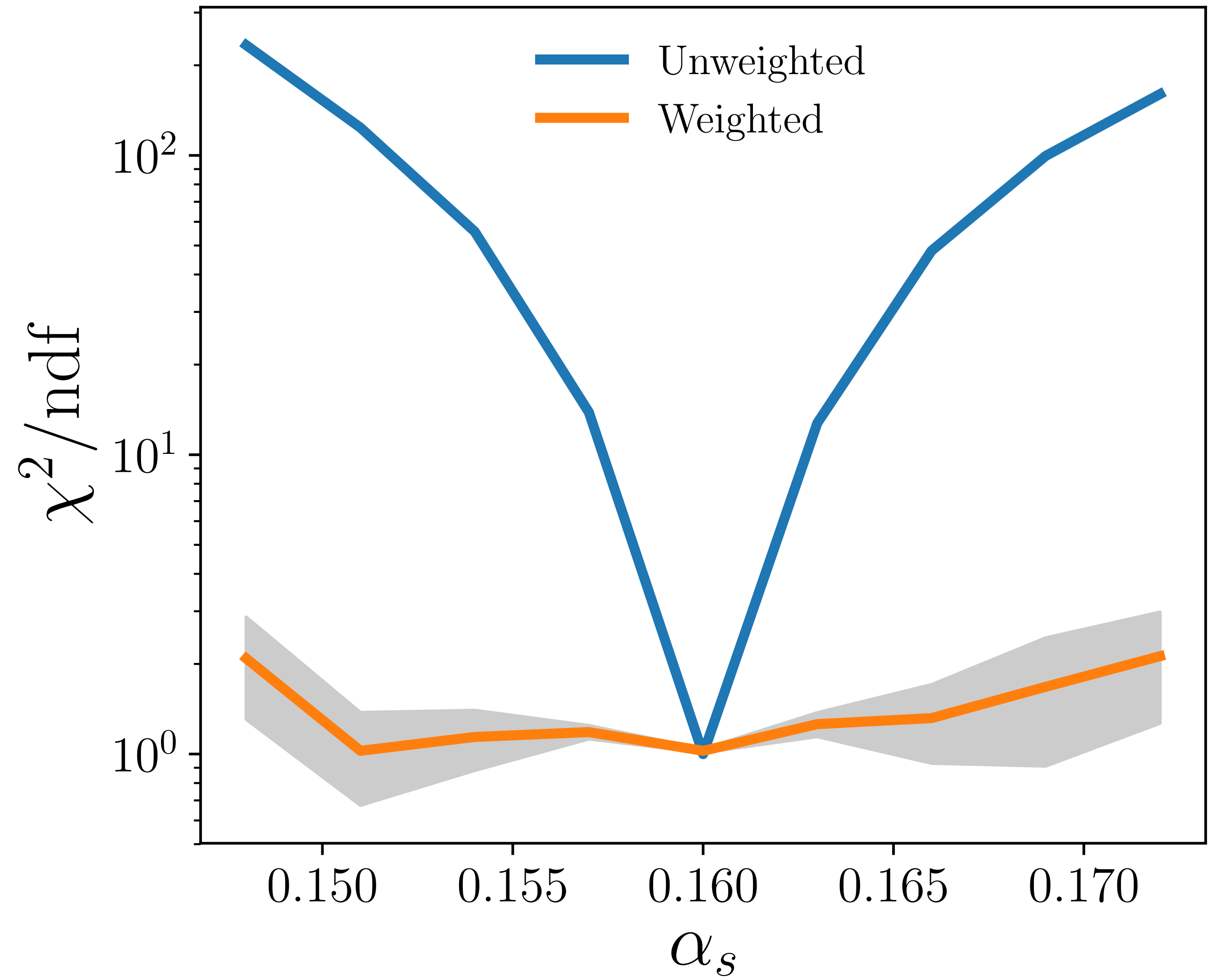
1907.08209

Parameterized reweighting

What if we have a new simulation with multiple continuous parameters θ?

Easy - learn a parameterized classifier* !

…simply add the parameter as a feature to the network during training and let it learn to interpolate.



1907.08209

*1506.02169

Example Fit

Step 1: Differentiable Surrogate Model

$$f(x,\theta) = \underset{f'}{\operatorname{argmax}} \sum_{i \in \boldsymbol{\theta_0}} \log f'(x_i,\theta) + \sum_{i \in \boldsymbol{\theta}} \log(1 - f'(x_i,\theta))$$

See also 1805.00020

Example Fit

Step 1: Differentiable Surrogate Model

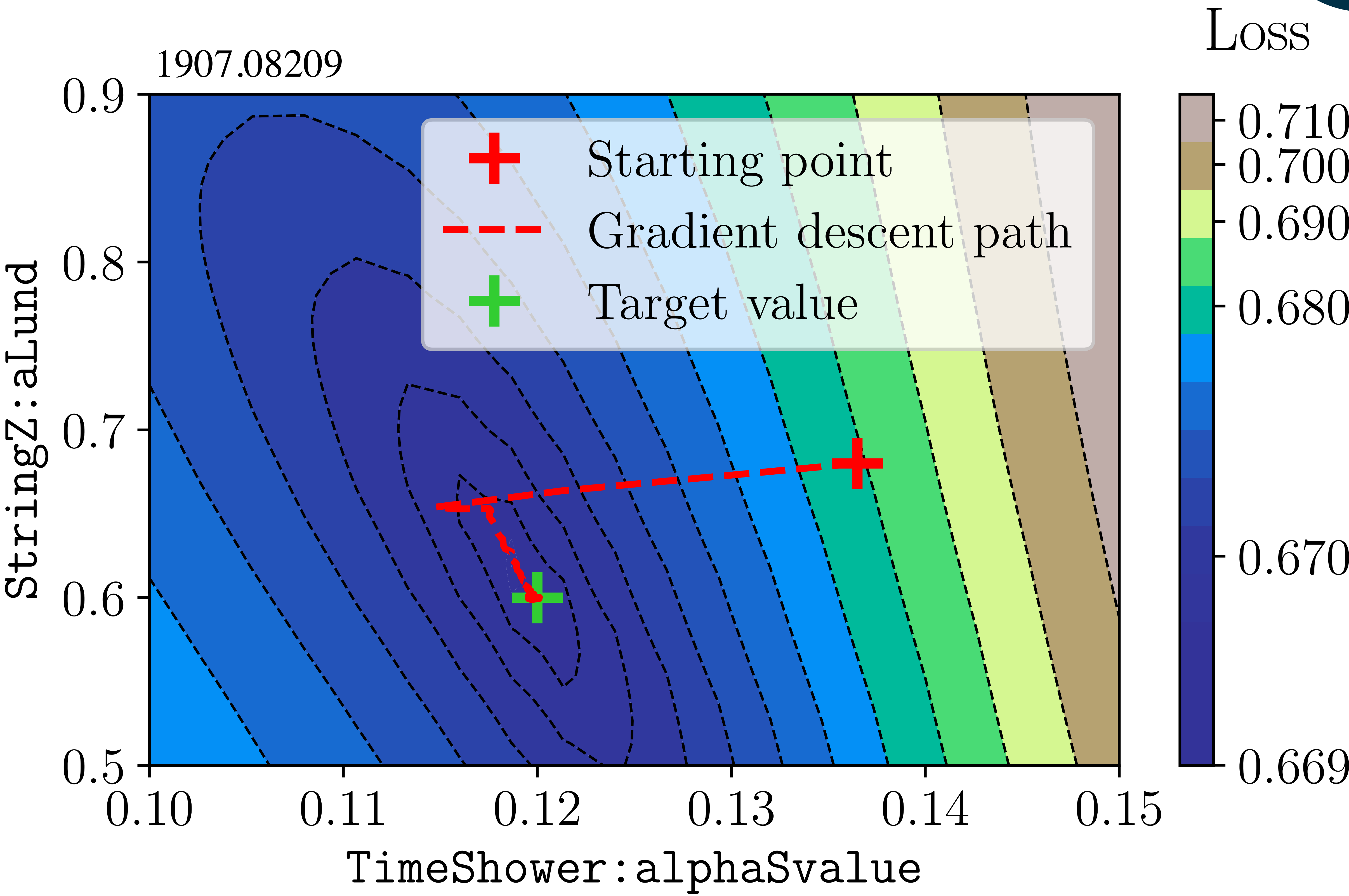
$$f(x,\theta) = \underset{f'}{\operatorname{argmax}} \sum_{i\in\boldsymbol{\theta_0}} \log f'(x_i,\theta) + \sum_{i\in\boldsymbol{\theta}} \log(1 - f'(x_i,\theta))$$

Step 2: Gradient-based optimization

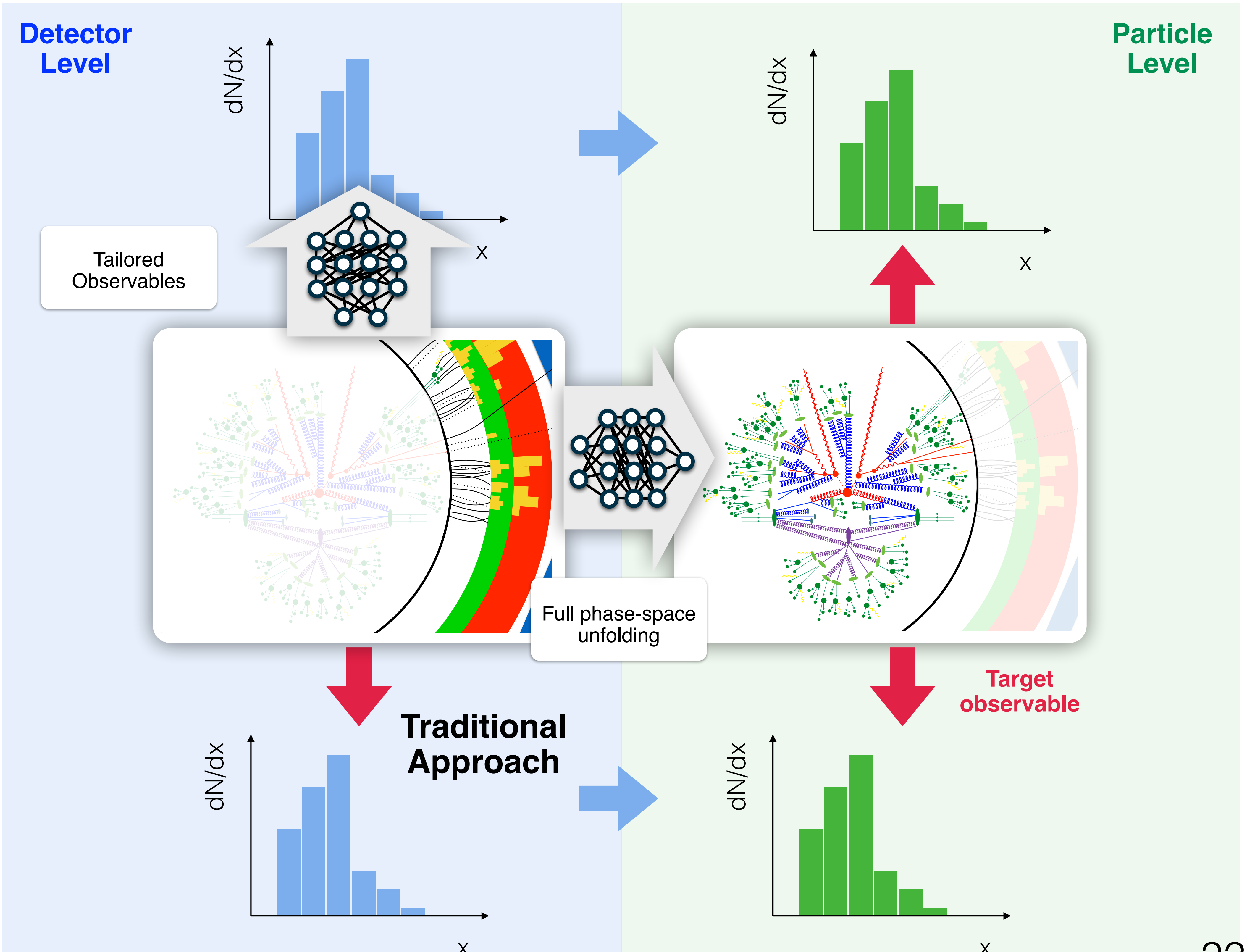
$$\theta^* = \underset{\theta'}{\operatorname{argmax}} \sum_{i\in\boldsymbol{\theta_0}} \log f(x_i,\theta') + \sum_{i\in\boldsymbol{\theta_1}} \log(1 - f(x_i,\theta'))$$

See also 1805.00020

Example Fit

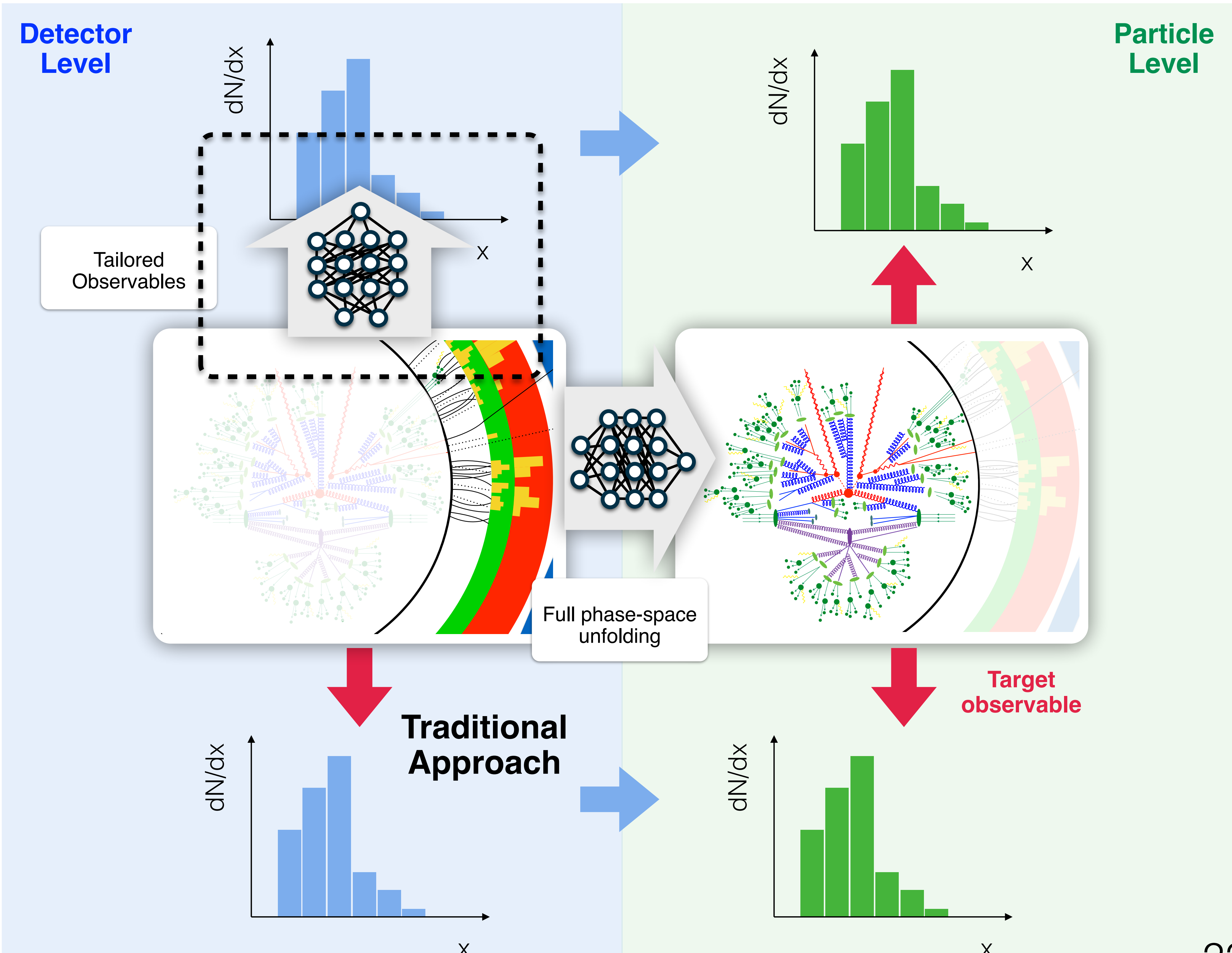

1907.08209
Loss
Starting point
Gradient descent path
Target value
0.9
0.8
0.7
0.6
0.5
StringZ:aLund
0.10
0.11
0.12
0.13
0.14
0.15
TimeShower:alphaSvalue
0.710
0.700
0.690
0.680
0.670
0.669

Unfolding



2203.16722

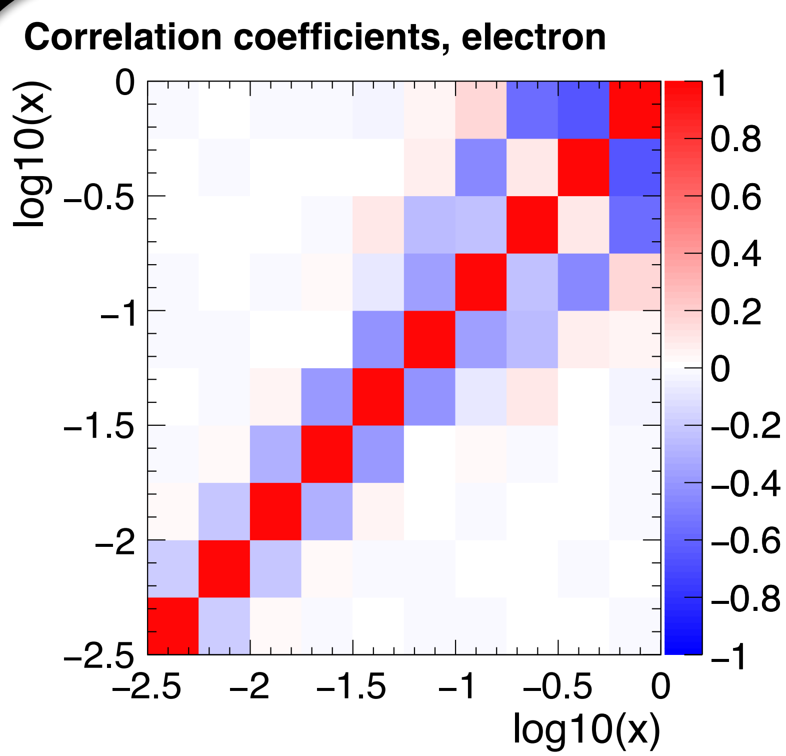
Unfolding

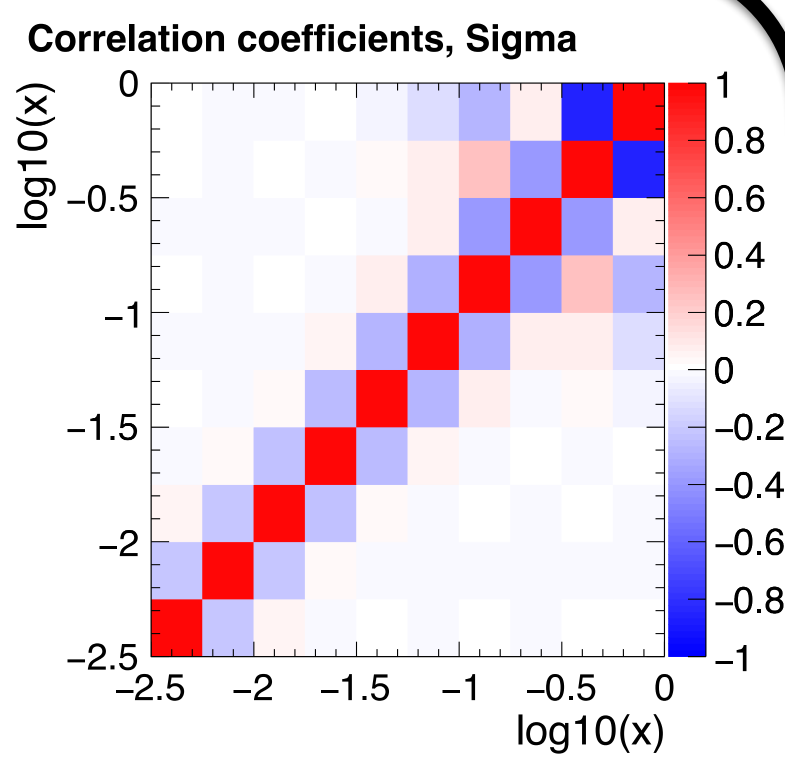


2203.16722

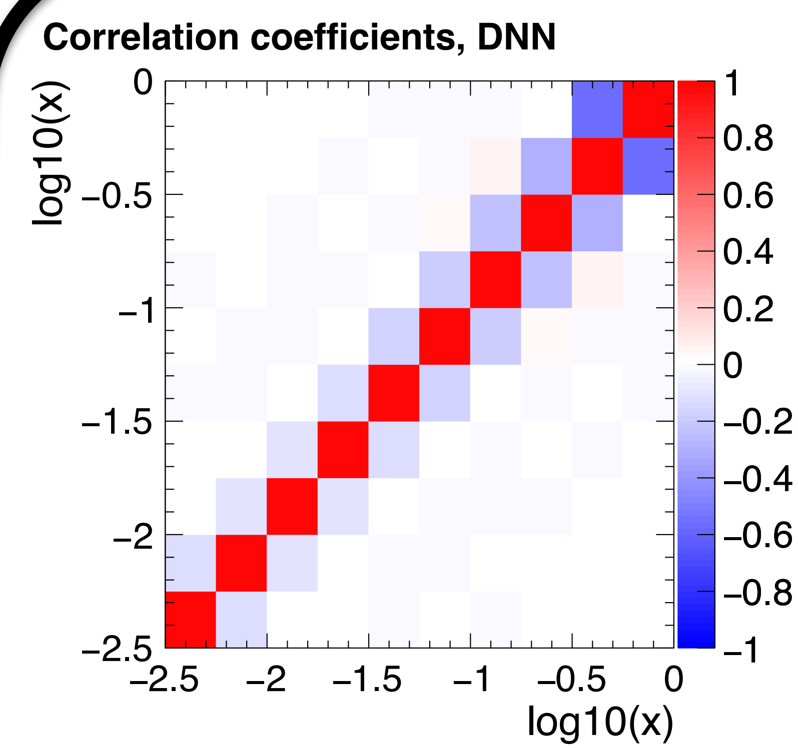
Unfolding

Learn tailored observables; no reason detector level needs to be same observable as particle level!





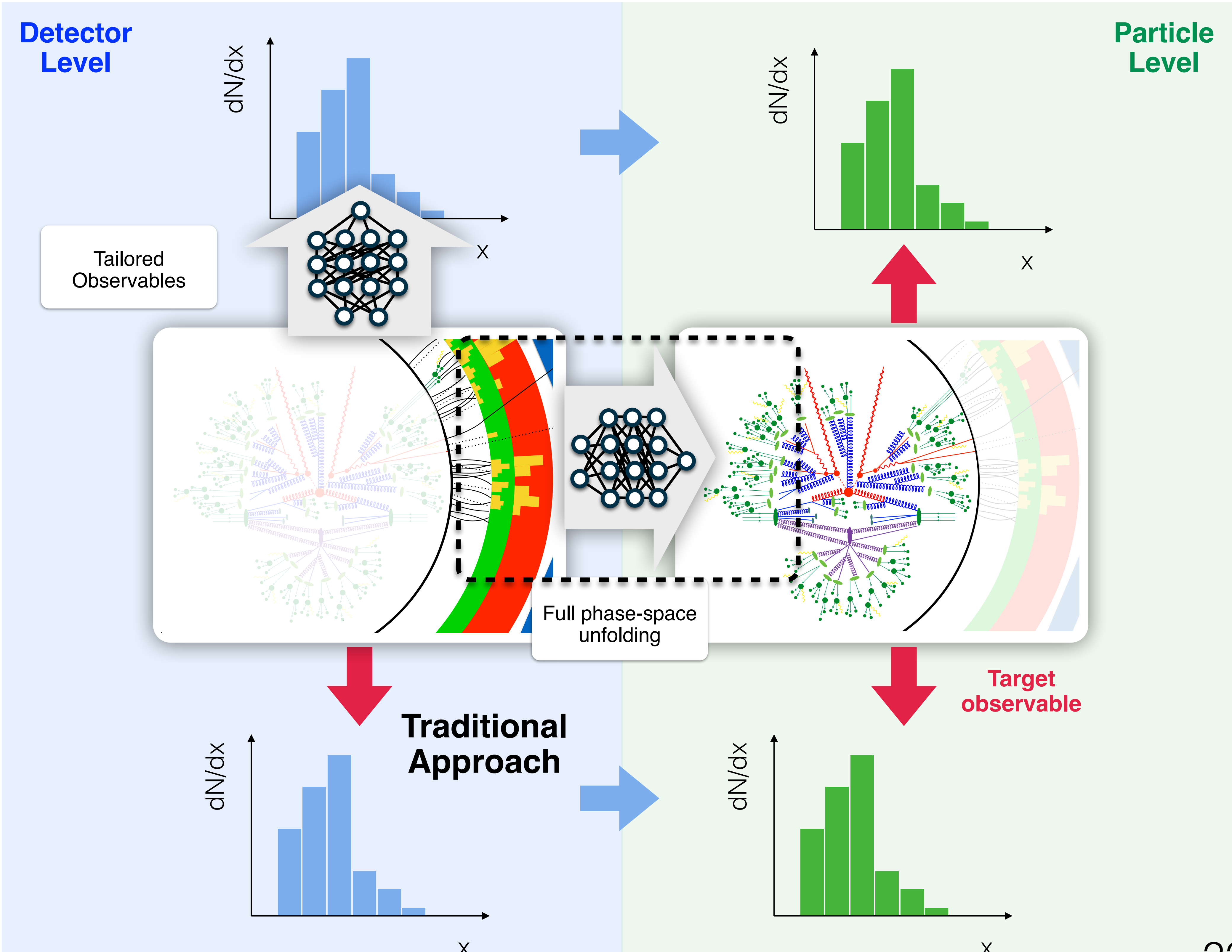
Classical Observables



Neural Network

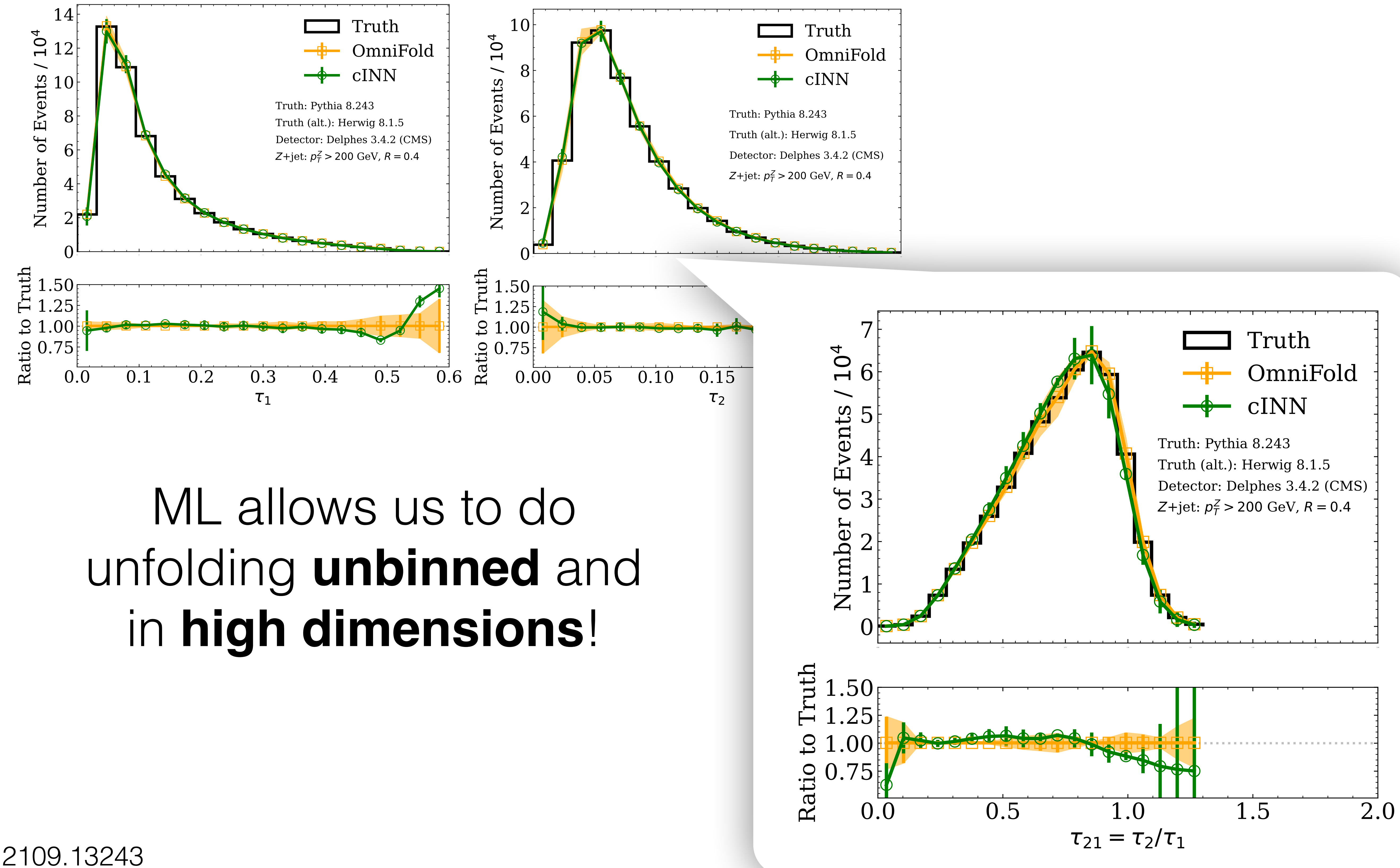
2203.16722

Unfolding



2203.16722

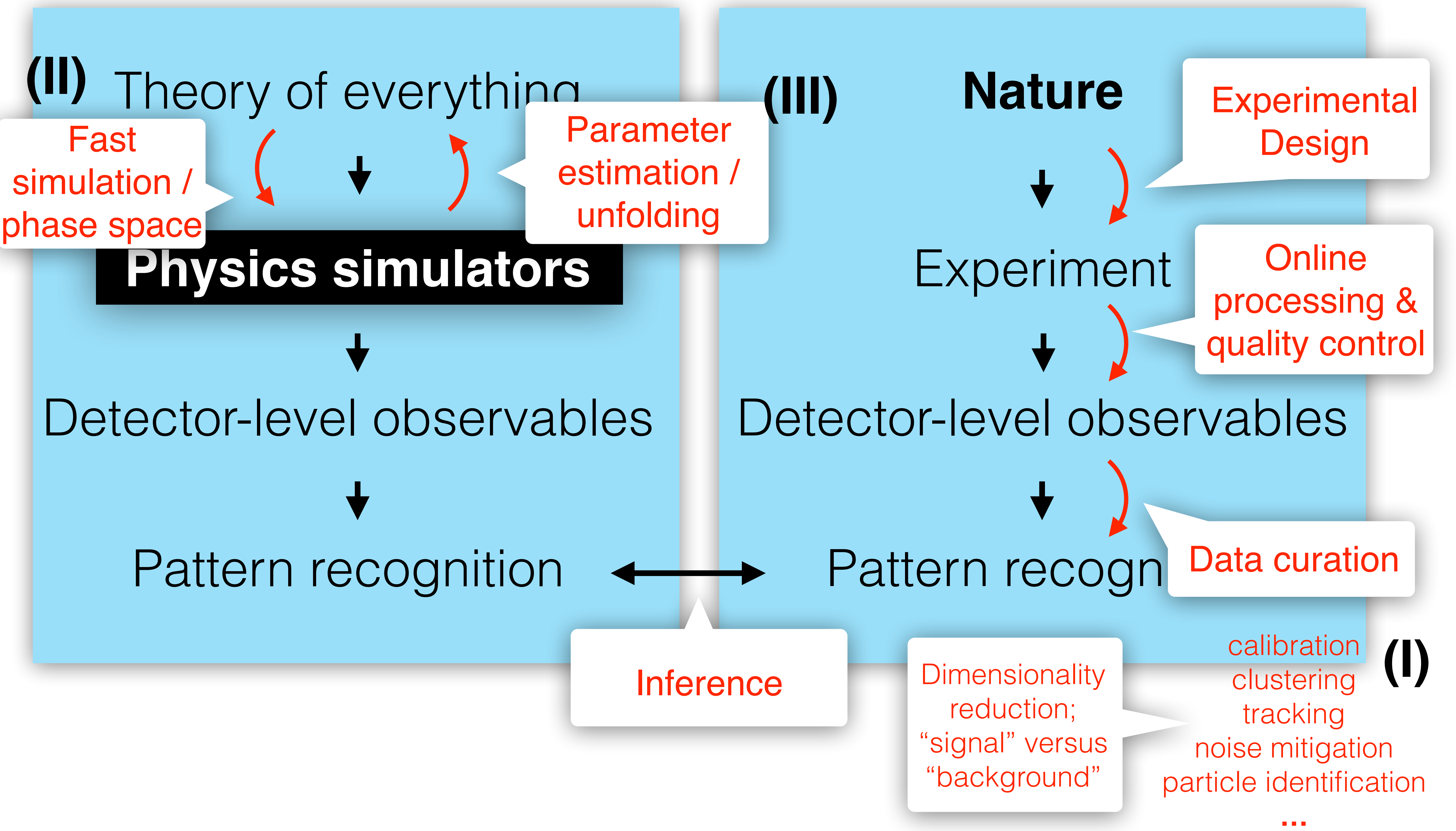
Unfolding



ML allows us to do unfolding **unbinned** and in **high dimensions**!

2109.13243

Particle Physics + Machine Learning

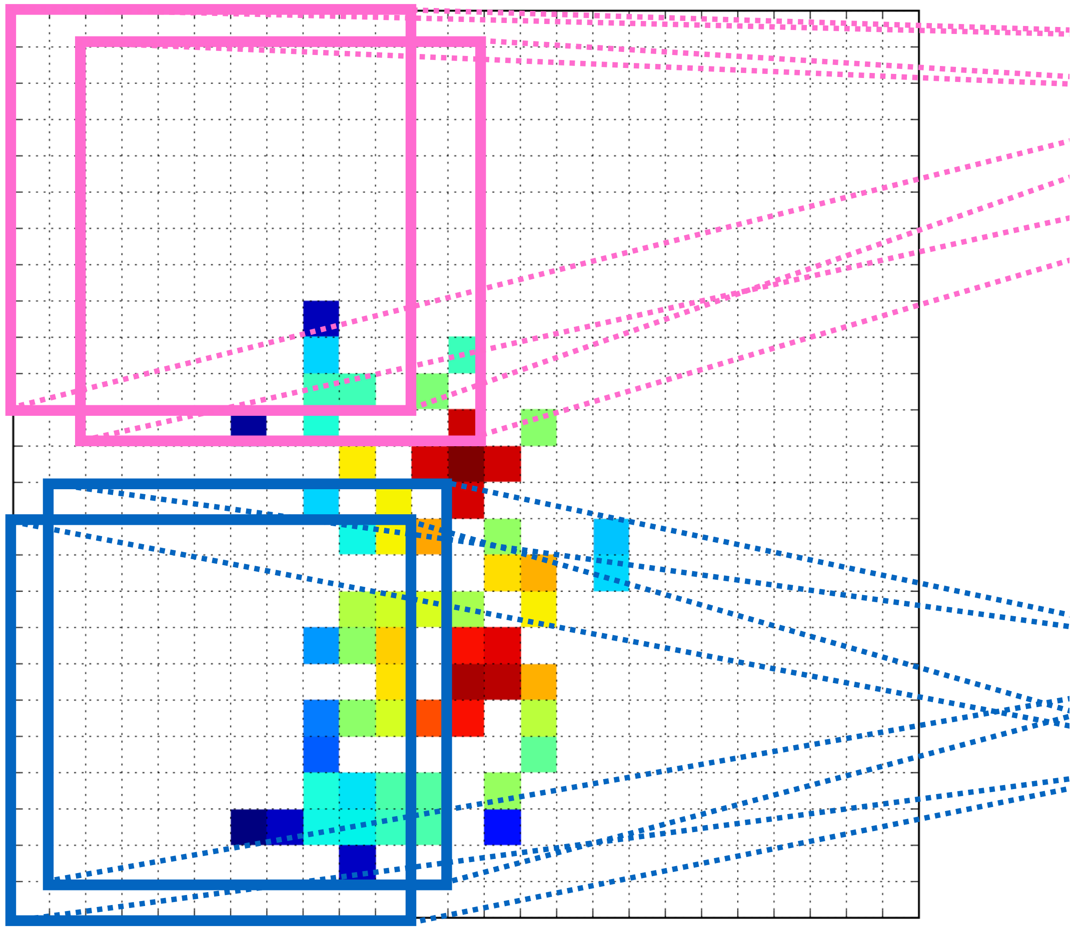


Conclusions and Outlook

AI/ML has a great potential to **enhance**, **accelerate**, and **empower** all areas of HEP

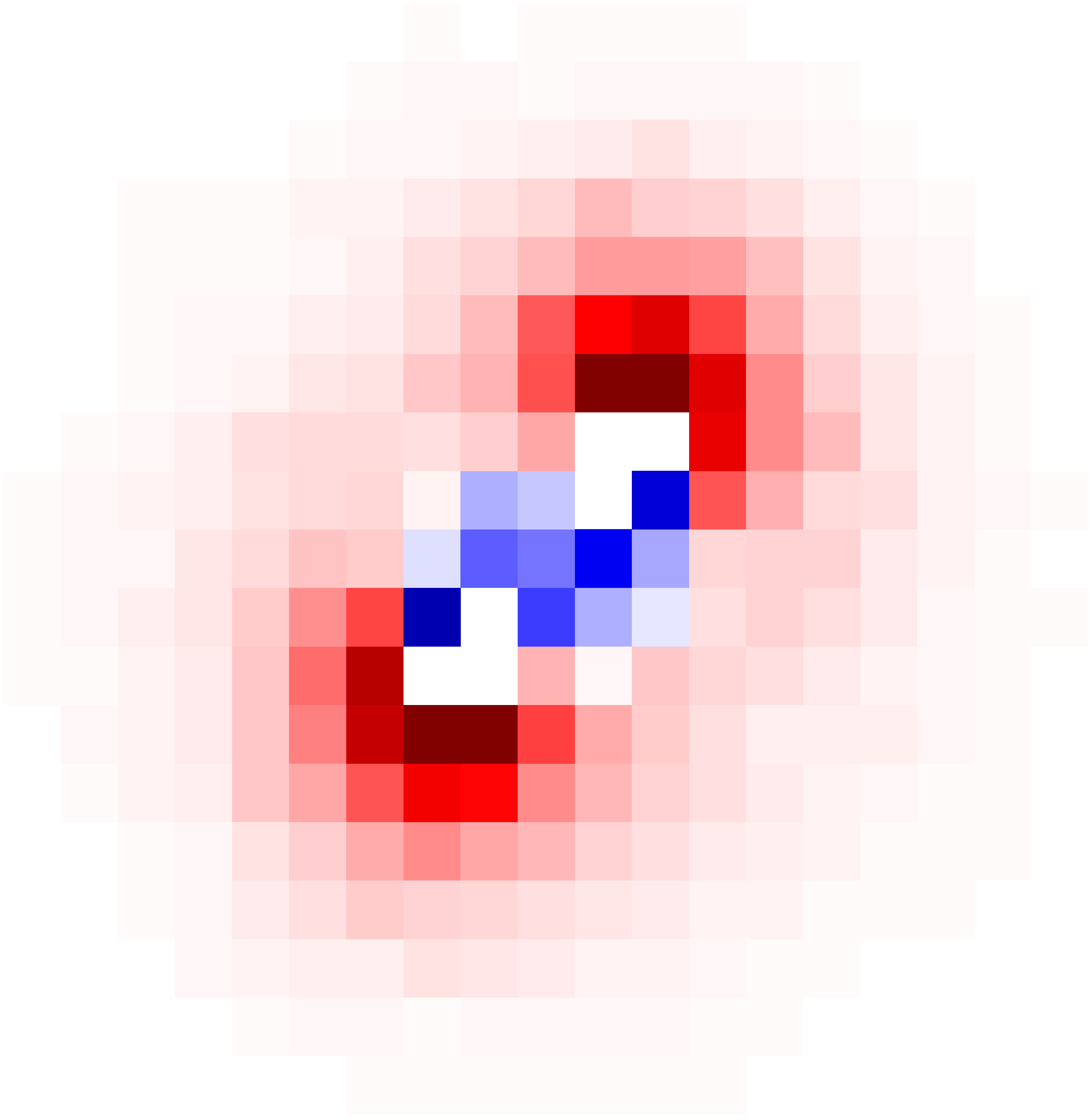
There are applications now that were unthinkable before ML and new ideas are incoming!

We need you to help develop, adapt, and deploy new methods



I've provided some specific examples today, but see the Living Review, 2102.02770, for more!

Note that I could not cover everything! e.g. anomaly detection



Fin.