Machine Learning

Lecture 3

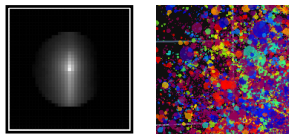
-

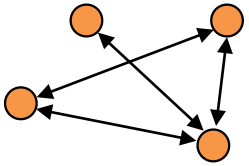
Learning a Process

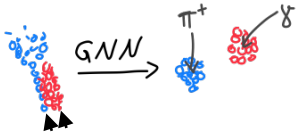
Jan Kieseler

jan.kieseler@kit.edu

DIY conclusions

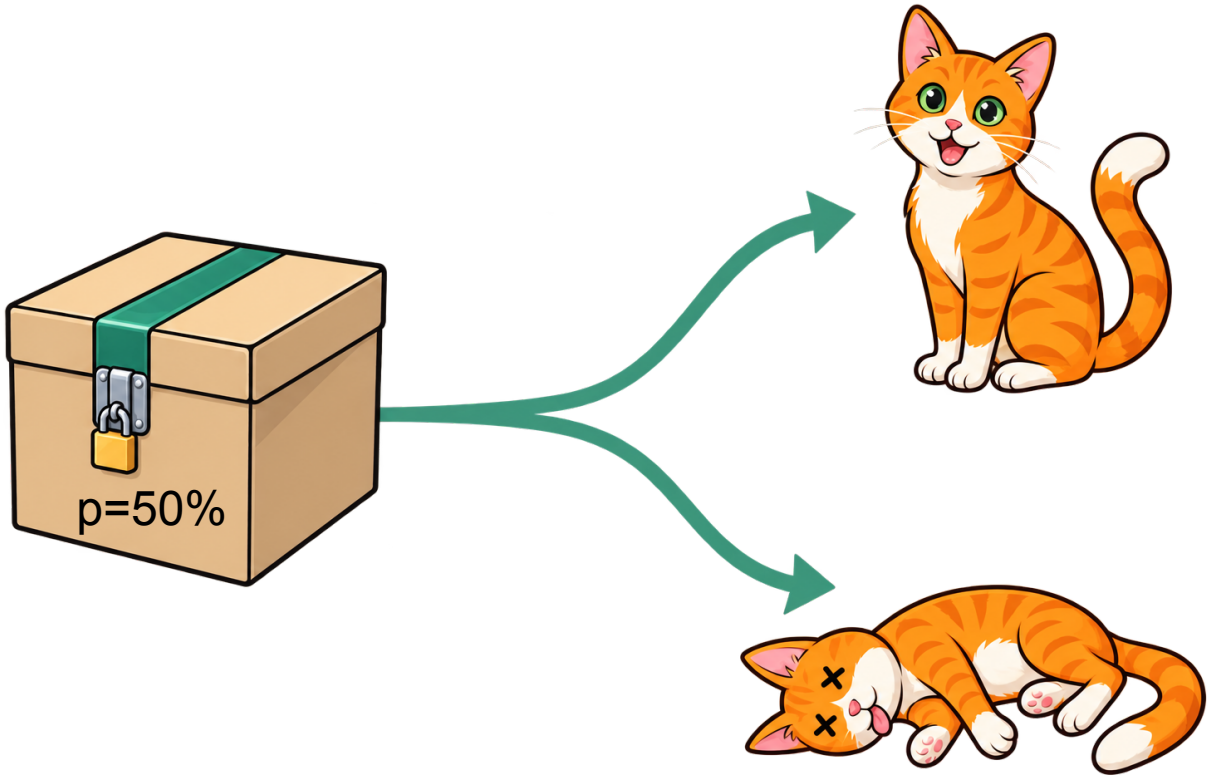




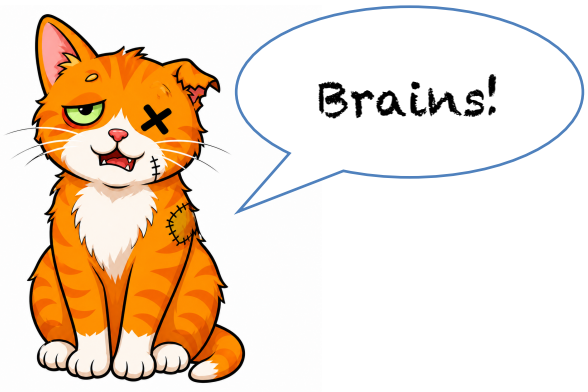


- Finding structures and symmetries can helps reduce parameter counts and makes computation cheaper
- CNNs find relations between neighbouring pixels and abstract them
- Particle relations can be represented through graphs (vertices and edges)
- Dynamical graphs can be less computationally expensive
- Set-to-set problems require special losses and methods
- Reconstruction tasks in HEP are similar to object section tasks in CV, but methods don’t apply 1:1
- It must be guaranteed that the algorithms can be calibrated

Decisions are Hard: Averages can be impossible



$\Phi(\square;\omega) \rightarrow$



Goal: Turn your own Physics Problem into a Well-Posed ML Problem

Lecture 1: Learning a Function

understand the complete basic learning loop and how
a neural network is trained to approximate a deterministic function

Lecture 2: Exploiting Structure

reason from the structure and symmetries of a problem to
an appropriate representation and architecture

Lecture 3: Learning a Process

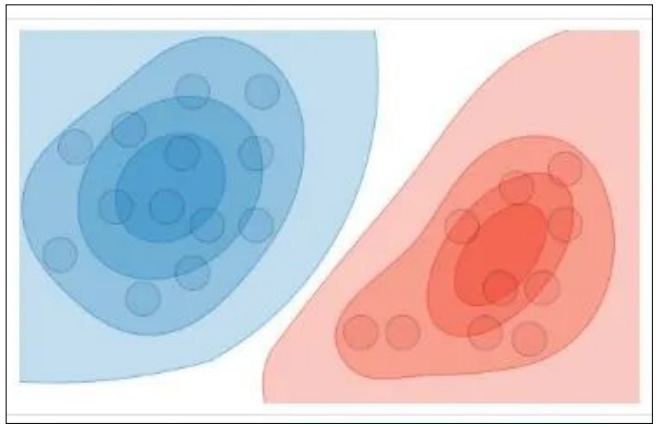
understand how random inputs turn deterministic neural networks
into generators, and how to train and evaluate them.

Ask questions at any time

Distill the most important take-aways:
we will write the conclusions together at the end of each lecture

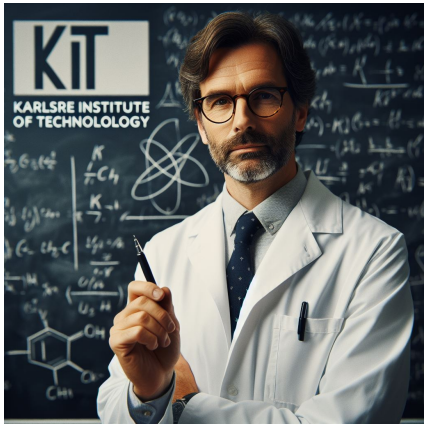
Conditional Generative Networks

- Generative neural networks **sample** from a distribution



- It requires a random component

- In addition, the distribution can be conditioned by other input, e.g. “scientist from Karlsruhe” $p_{x'}(x' \mid c)$

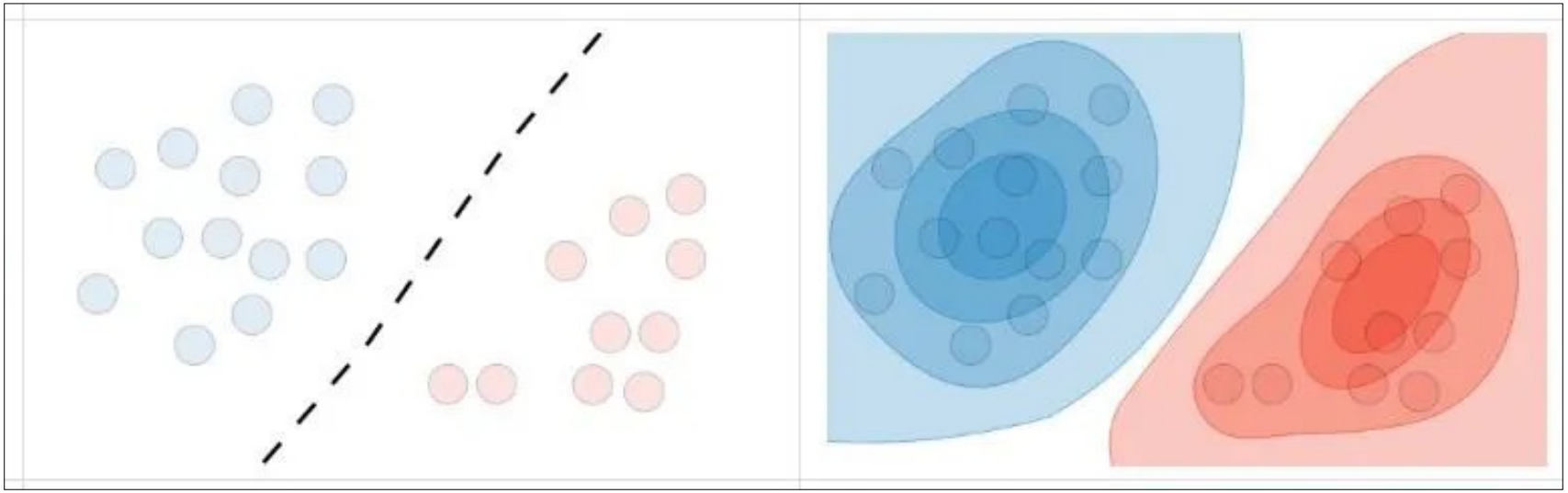


- Add that conditioning to the NN input

The Main Issue: How to Learn a Distribution from Samples

Classifier

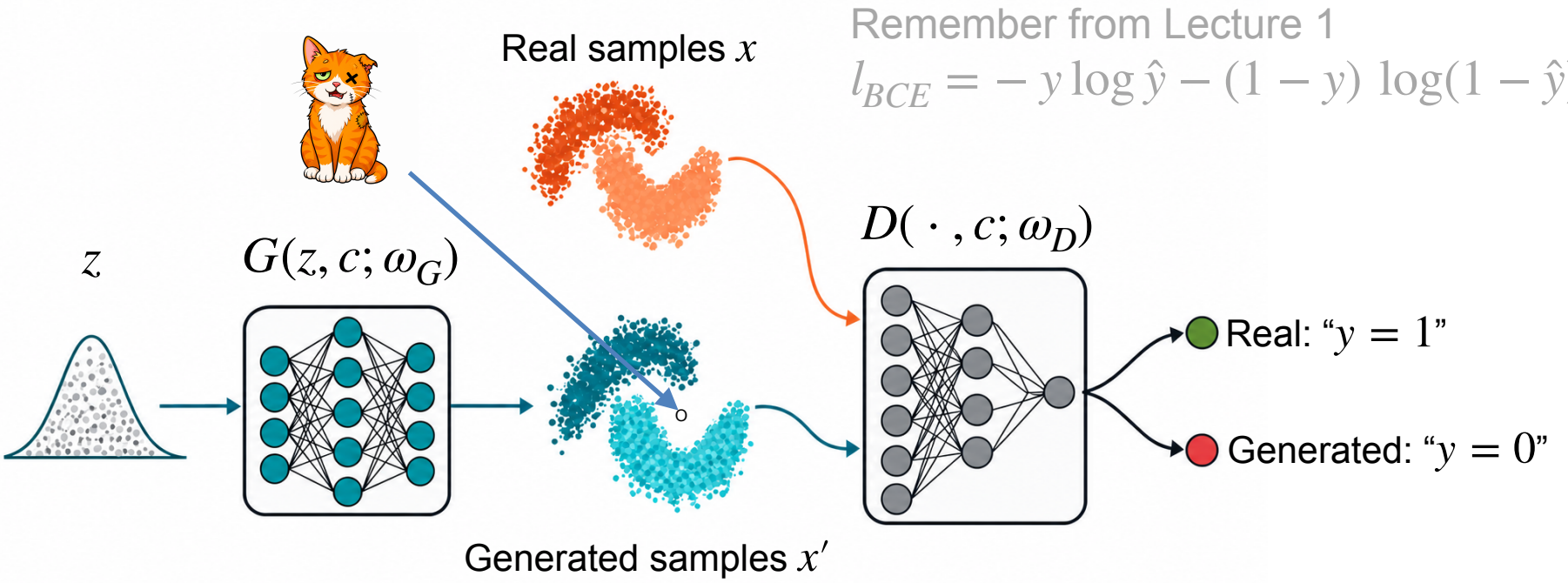
Generative



- Training data: batches of (x, y)
- Loss: $L(\Phi(x; \omega), y)$

- There is no immediate notion of a distribution

Generative Adversarial Networks



Remember from Lecture 1

$$l_{BCE} = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

- The discriminator is trained to distinguish real and generated samples

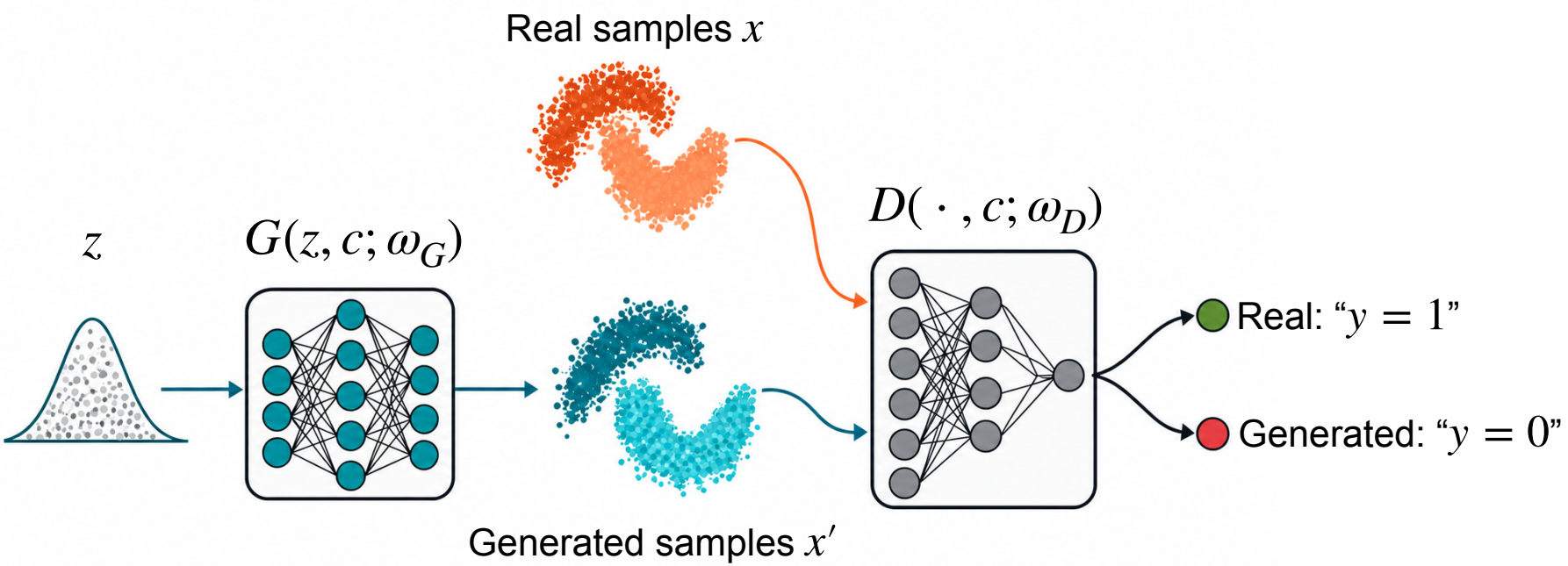
$$l_D = -\log D(x, c; \omega_D) - \log \left(1 - D(x', c; \omega_D)\right)$$

- The generator is trained to make x' appear real to the discriminator

$$l_G = -\log D(x', c; \omega_D) = -\log D\left(G(z, c; \omega_G), c; \omega_D\right)$$

This is the commonly used non-saturating generator loss. It has the same desired equilibrium as the original minimax formulation but gives the generator a more useful learning signal when the discriminator can initially reject generated samples easily.

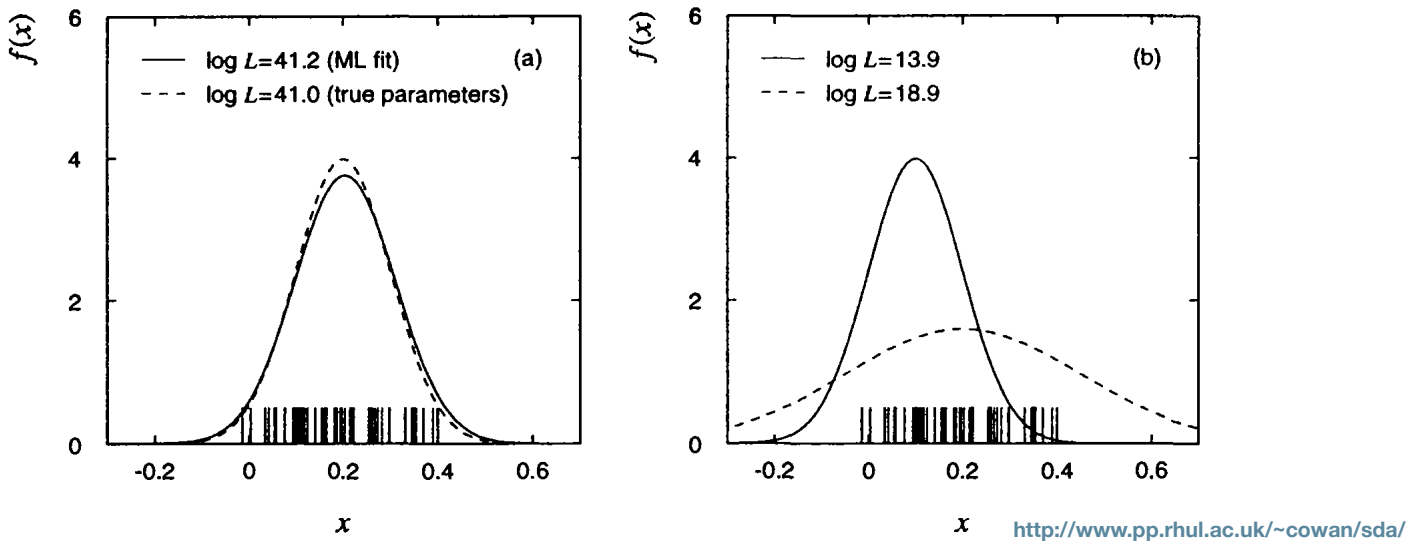
GAN Training Steps



1. Generate x' under condition c
2. Update ω_D using l_D, keep ω_G fixed
3. Update ω_G using l_G, keep ω_D fixed
4. Repeat

- GANs are fast and allow flexible generator architectures; but adversarial training can be brittle and suffer from mode collapse.

How well Does a Distribution Describe a Sample of Data?



http://www.pp.rhul.ac.uk/~cowan/sda/

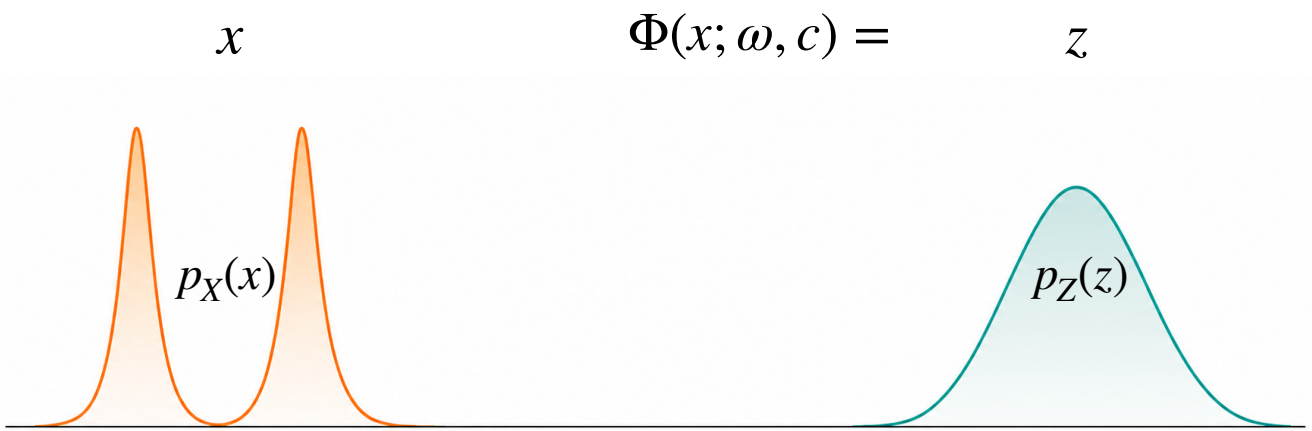
- From statistics: Maximum Likelihood!

$$p(\{x_o, \ldots, x_N\}; \omega) = \prod_i p(x_i; \omega)$$

$$-\log p(x; \omega) = -\sum \log p(x_i; \omega)$$

- Minimising this expression gives us the best-fit values for ω on the data x

Normalising Flows: 1D Example



- Probability conservation:
  $p_X(x)\,\mathrm{d}x = p_Z(z)\,\mathrm{d}z$

$$p_X(x) = p_Z(z) \left| \frac{\mathrm{d}z}{\mathrm{d}x} \right|$$

- Insert the neural network and ω dependence:

$$p_X(x;\omega,c) = p_Z(\Phi(x,c;\omega)) \left| \frac{\partial \Phi(x,c;\omega)}{\partial x} \right|$$

- We can evaluate the model density and construct a likelihood

Train Forward, Generate Backward

- $p_X(x, c; \omega) = p_Z(\Phi(x, c; \omega)) \left| \frac{\mathrm{d}(\Phi(x, c; \omega))}{\mathrm{d}x} \right|$

NB: in more dimensions: det (Jacobian)

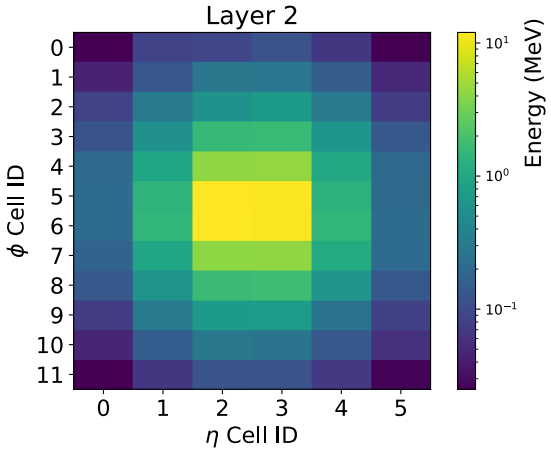
- The negative log-likelihood **loss** becomes

$$L = -\frac{1}{N} \sum_{i}^{N} \left[\log p_Z(\Phi(x, c; \omega)) + \log \left| \frac{\partial \Phi(x, c; \omega)}{\partial x} \right| \right]$$

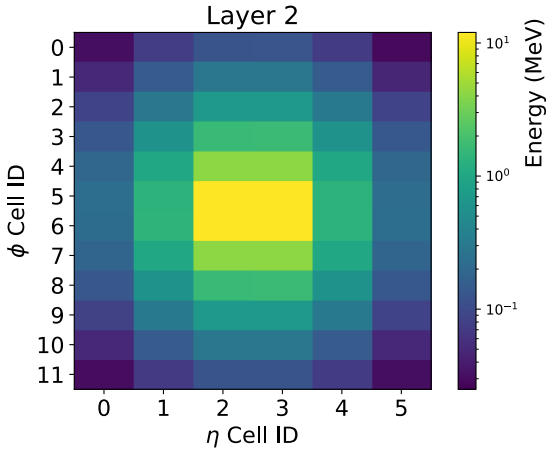
Normalising Flows in Practice

Particle properties →

Flow-generated deposits



Geant 4 baseline

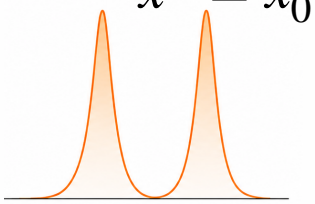


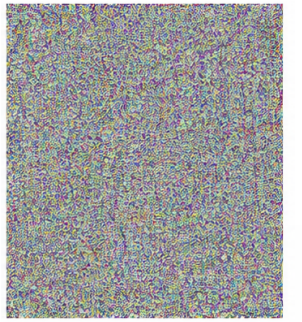
C. Krause, D. Shih, arXiv:2106.05285

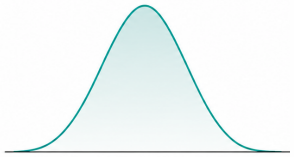
- Can be a very natural way to generate certain physics data.
- Direct likelihood objective, provide generally stable training, but constrain architecture/model to be **invertible**, awkward for variable-size particle sets
- **Flows gain an explicit probability density—but pay for it with architectural constraints**

Diffusion Models: Gradual Corruption Path

$x = x_0$ x_1 x_2 x_3 x_4 $x_T = z$

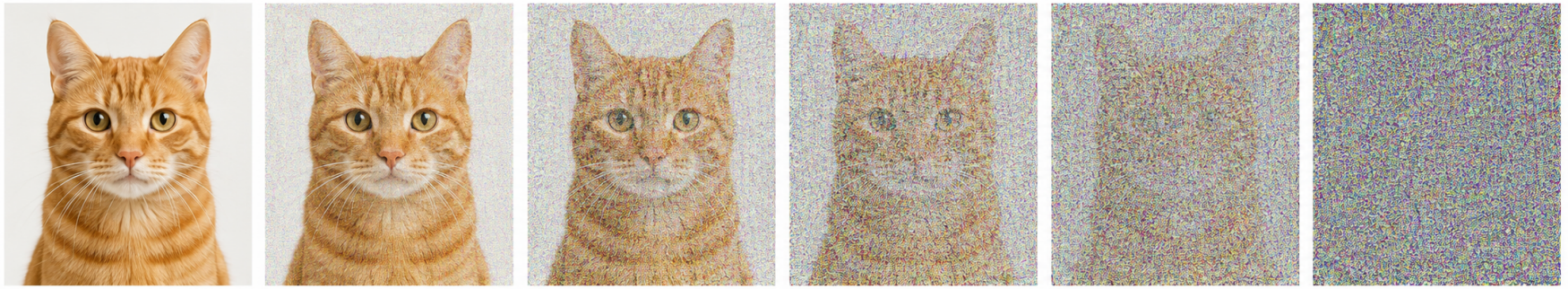






- Flow: $z = \Phi(x, c; \omega)$, and invert Φ
- Diffusion prescribes a gradual stochastic corruption path. We no longer need to learn one algebraically invertible transformation.
- Idea: learn to undo that corruption statistically step by step to get $z \rightarrow x'$

Noise Removal, a Supervised Regression Problem



$x = x_0$ x_1 x_2 x_3 x_4 $x_T = z$

- Make neural network predict total aggregate noise at step t

$$\hat{\epsilon} = \Phi(x_t, t, c; \omega)$$

- We add the noise: we know the amount
- This is a straight-forward supervised regression problem with MSE loss

$$l_{MSE} = (\epsilon - \hat{\epsilon})^2$$

Repeated Regression Predictions Become a Generator

$z = x'_T$ x'_4 x'_3 x'_2 x'_1 $x'_0 = x'$

- Add fresh noise for step t
- Evaluate $\hat{\epsilon}_t = \Phi(x'_t, t, c; \omega)$,
- Use $\hat{\epsilon}_t$ to calculate the denoising update

Start at $t = T$, repeat until $t = 1$

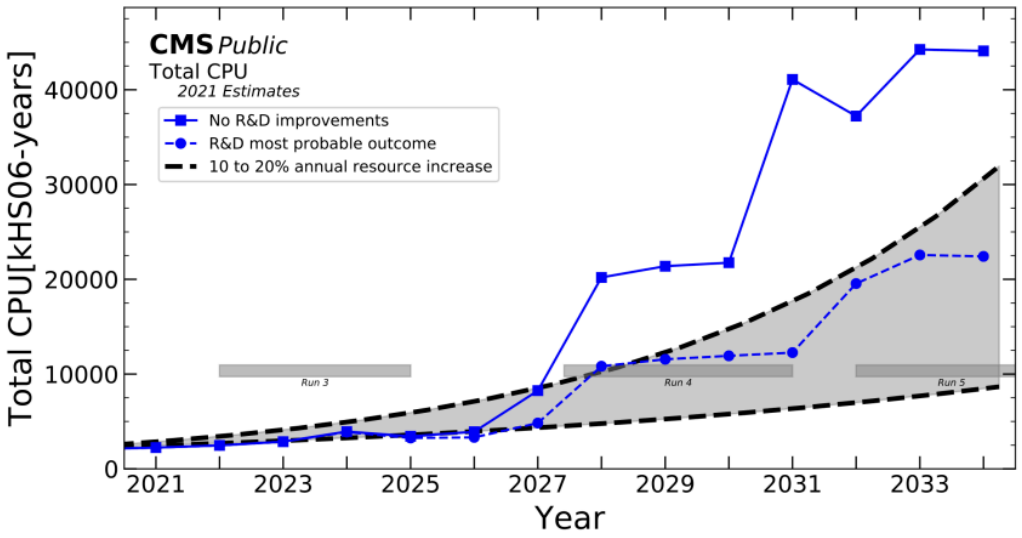
NB: there are complex scaling factors involved

- The process suppresses inconsistent noise while reinforcing plausible structure

- No invertible learned network or Jacobian is required.
- Stable: training uses a direct regression loss rather than an adversarial game.
- Slow: Generation requires repeated network evaluations.

Why ML (3): Resources again

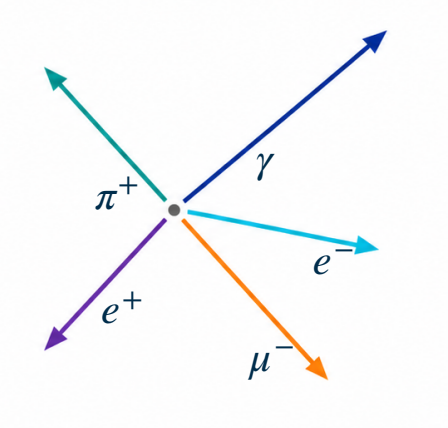
- ML can bring significant gain in physics performance

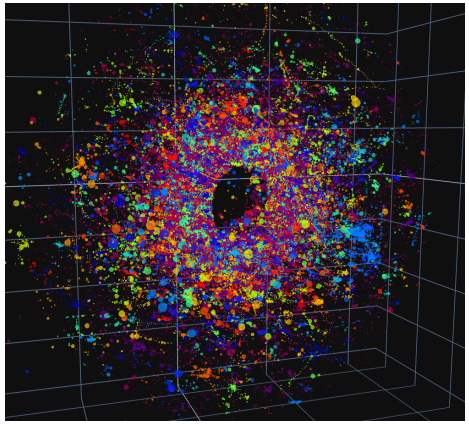


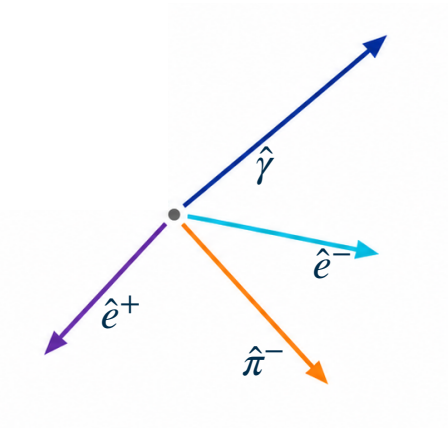
CMS NOTE -2022/008

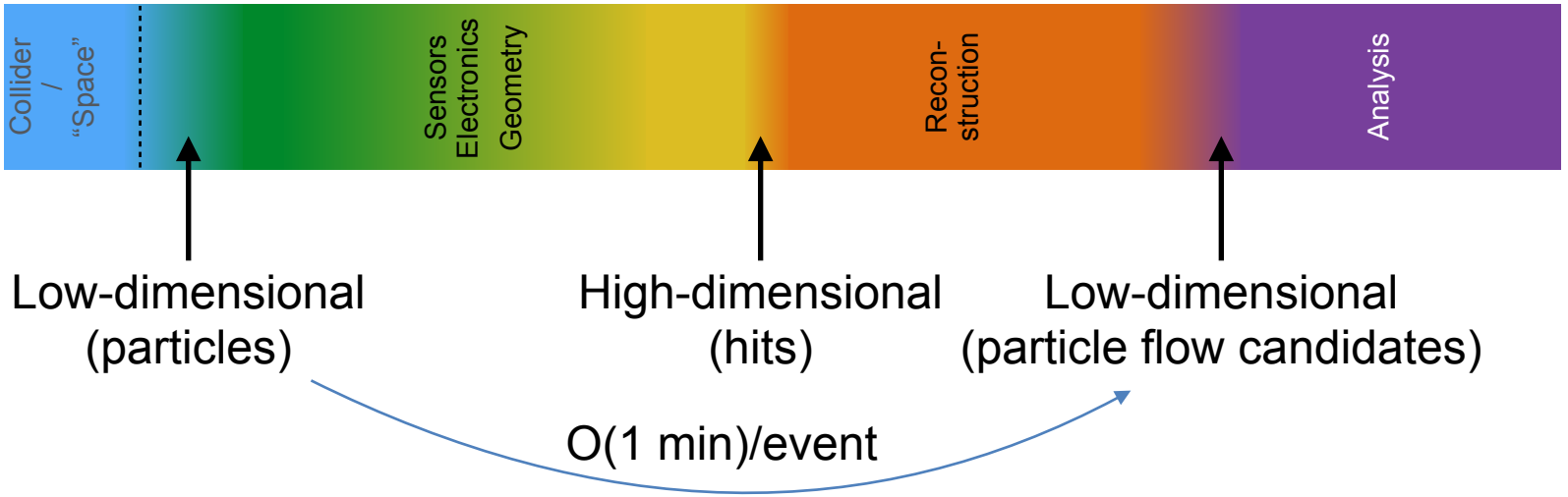
- ML algorithms open the option to lower resource requirements for reconstruction
- **ML algorithms open the option to lower resource requirements for simulation**

Skip the Expensive Part



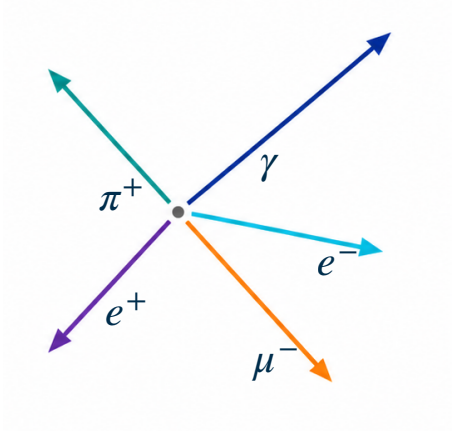






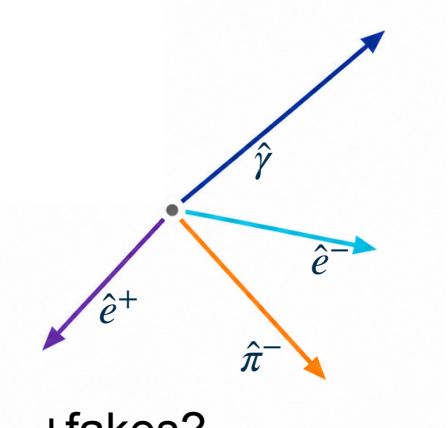
- We can use fully simulated events to train a "shortcut" neural network to create reconstructed particles from MC-generated particles

Set-to-Set Generative Problem



Φ → # particles

Φ → properties



+fakes?

- Generative NN problem: the same MC-generated event can have very different detector responses

Flow Matching

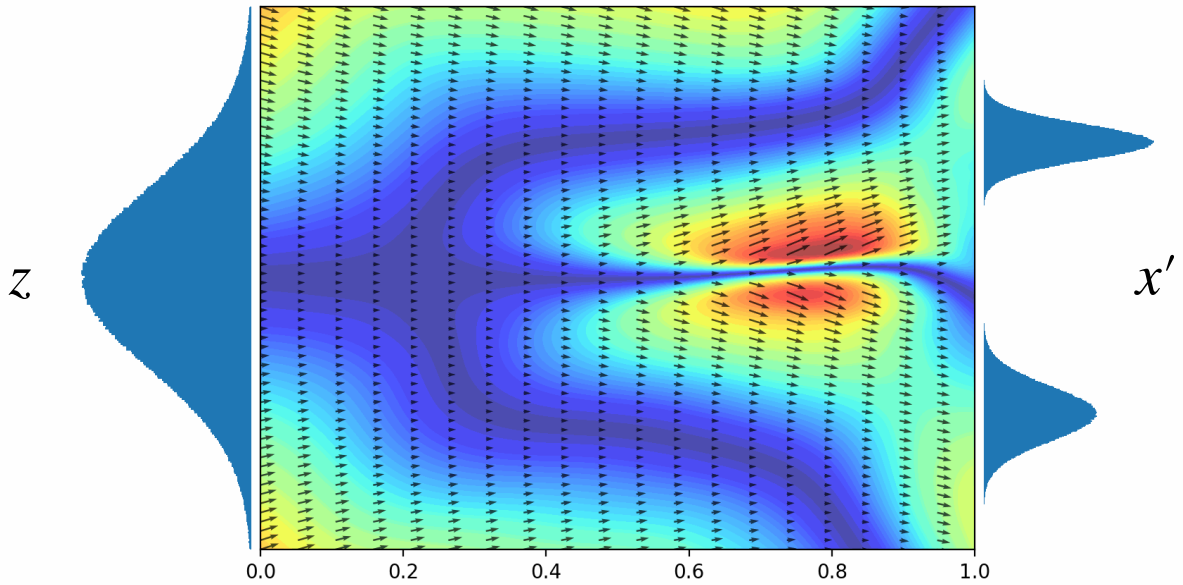
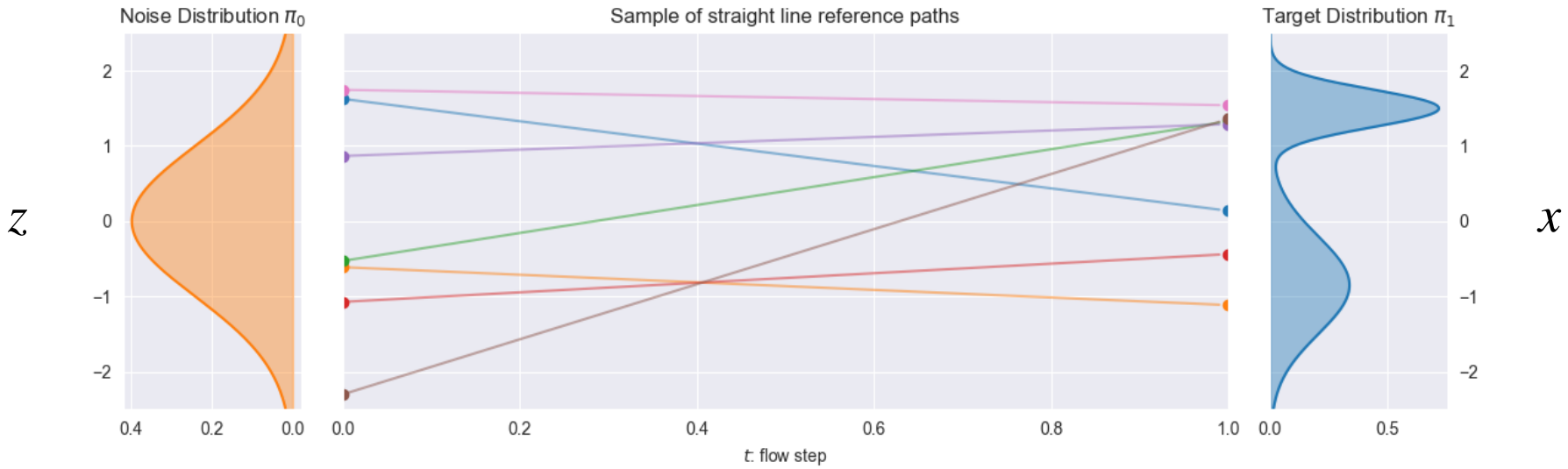


Figure by Dmitrii Kobylianskii

- Normalising Flow: learn one ***explicitly invertible*** map; use its Jacobian to calculate an exact density.
- Diffusion: learn incremental denoising corrections along a stochastic noise path.
- Flow matching: learn incremental transport velocities along a chosen probability path.

Flow Matching Training (Concept)



https://peterroelants.github.io/posts/flow_matching_intro/

- Define linear connections between pairs of sampled points in p_z and in $p_x(x \mid c)$

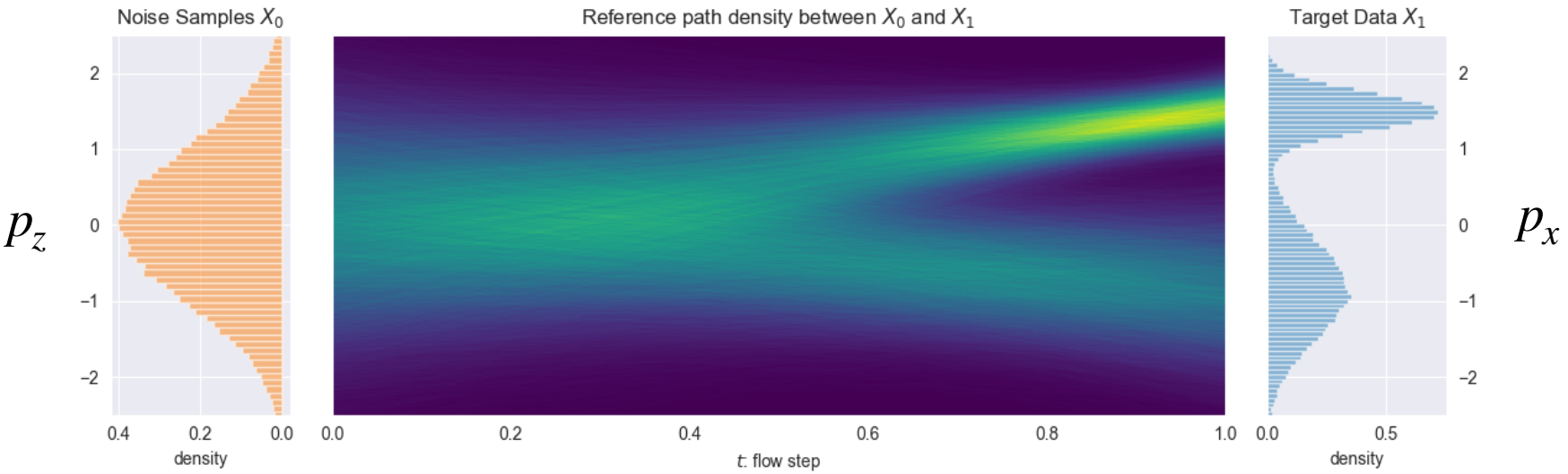
  $x_t = (1-t)z + tx$

  $v_t = x - z$

  Draw t

- Learn direction of motion: $\hat{v}_t = \Phi(x_t, t, c; \omega)$
- $l_{FM} = (v_t - \hat{v}_t)^2$

Learning the Whole Intermediate Distribution



https://peterroelants.github.io/posts/flow_matching_intro/

- The regressor will learn the expectation value

➡Once trained, $\hat{v}_t = \Phi(x_t, t, c; \omega)$ will describe the whole velocity field

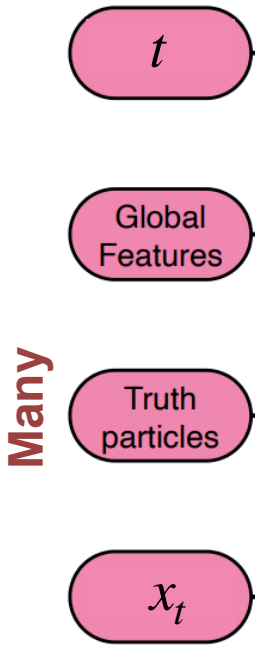
➡Generation can approximately follow this field (integrate) in steps from a z to an x'

Parnassus: Event Level Network → How Many?

arXiv:2503.19981

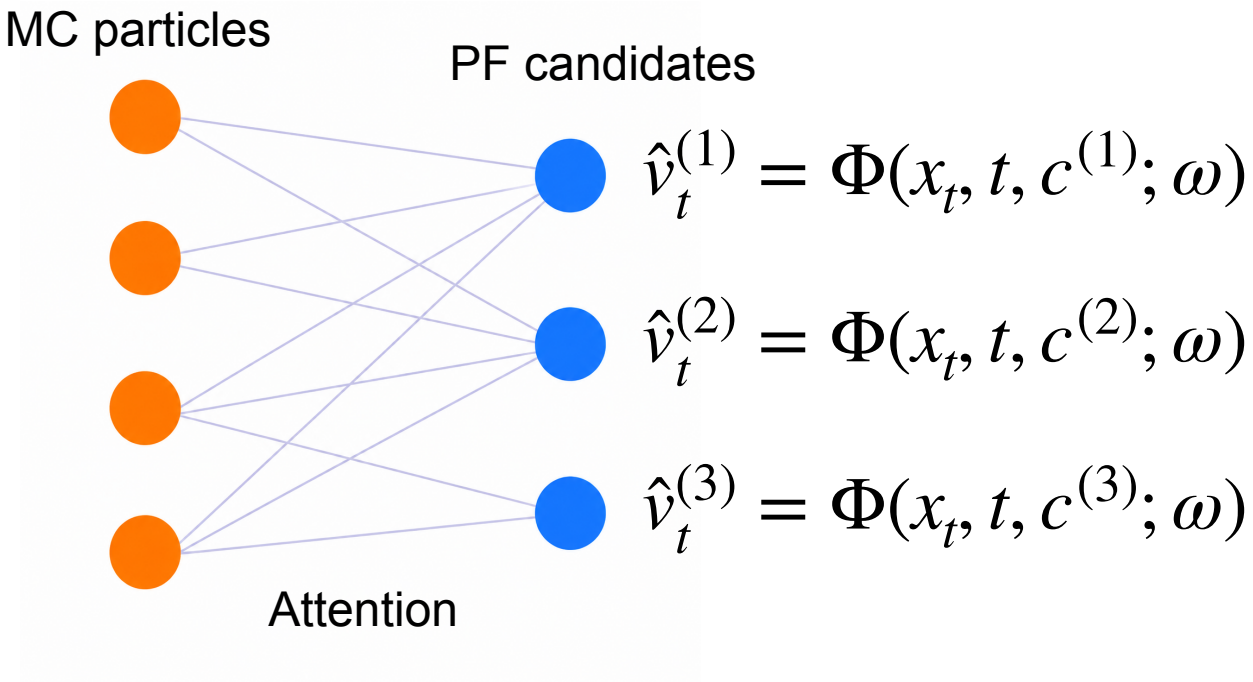
$$\hat{v}_t = \Phi(x_t, t, c; \omega)$$

Figure adapted from Eilam Gross



- Event level network generates global event properties and the **number of reconstructed particles**

Particle Property Network

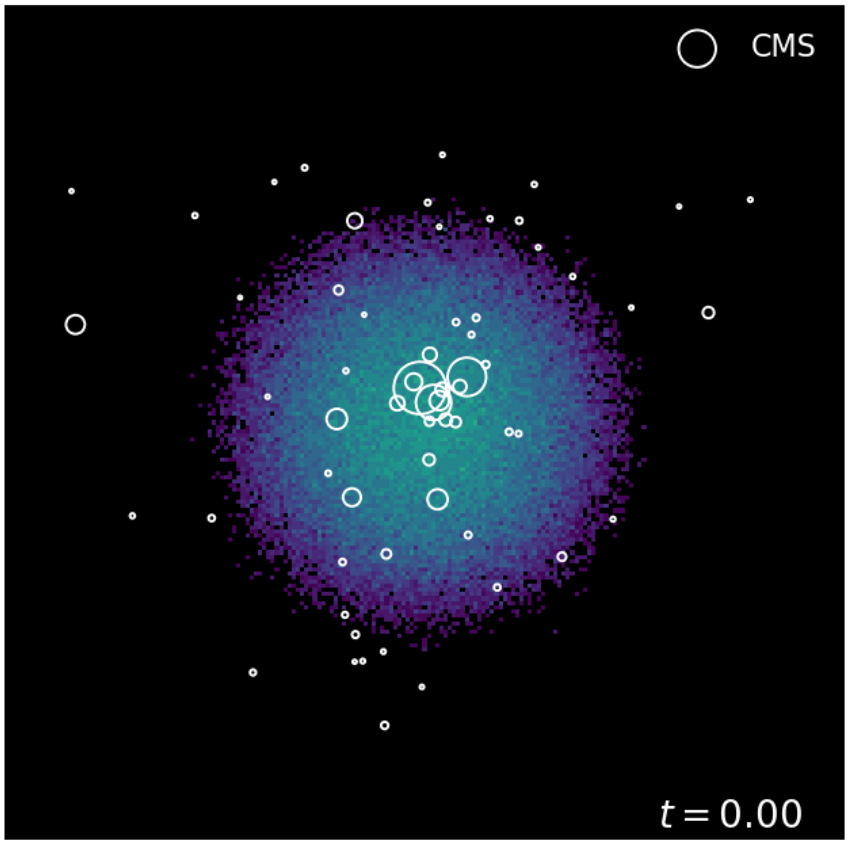


- Graph between truth particles and candidates (and global context) creates an individual velocity vector for the properties of each PF candidate

- Determines the **properties of the PF candidates**

*cross-attention

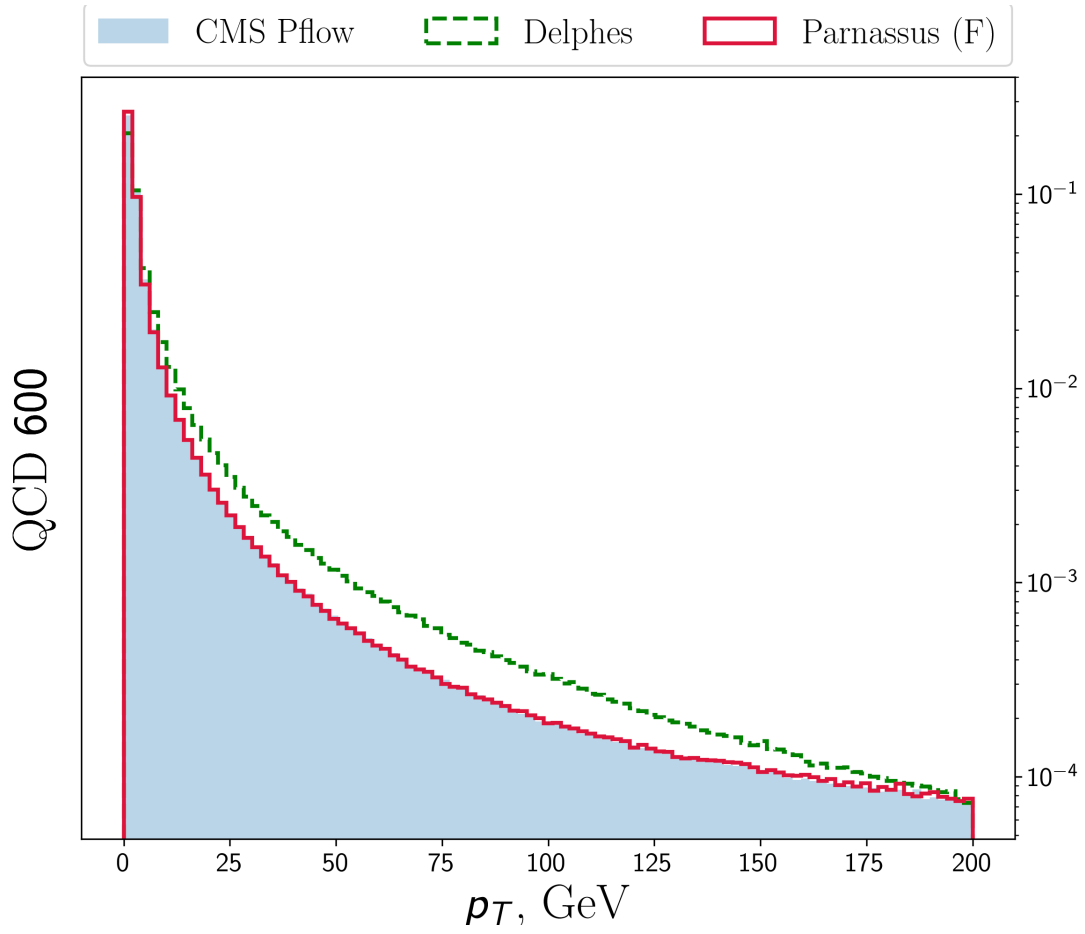
The Flow Matching in Action



Animation by Etienne Dreyer

- Parnassus uses 25 integration steps

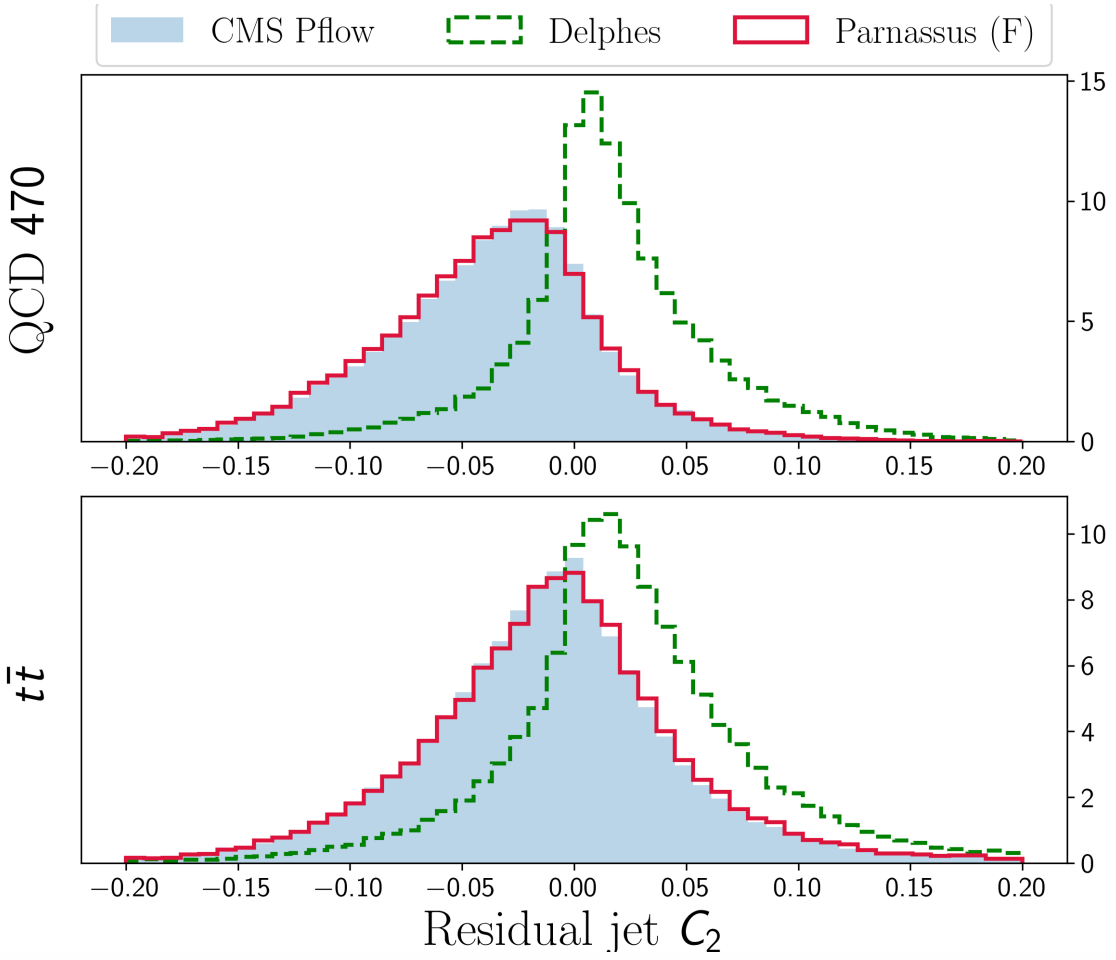
Particle Level Performance is Convincing



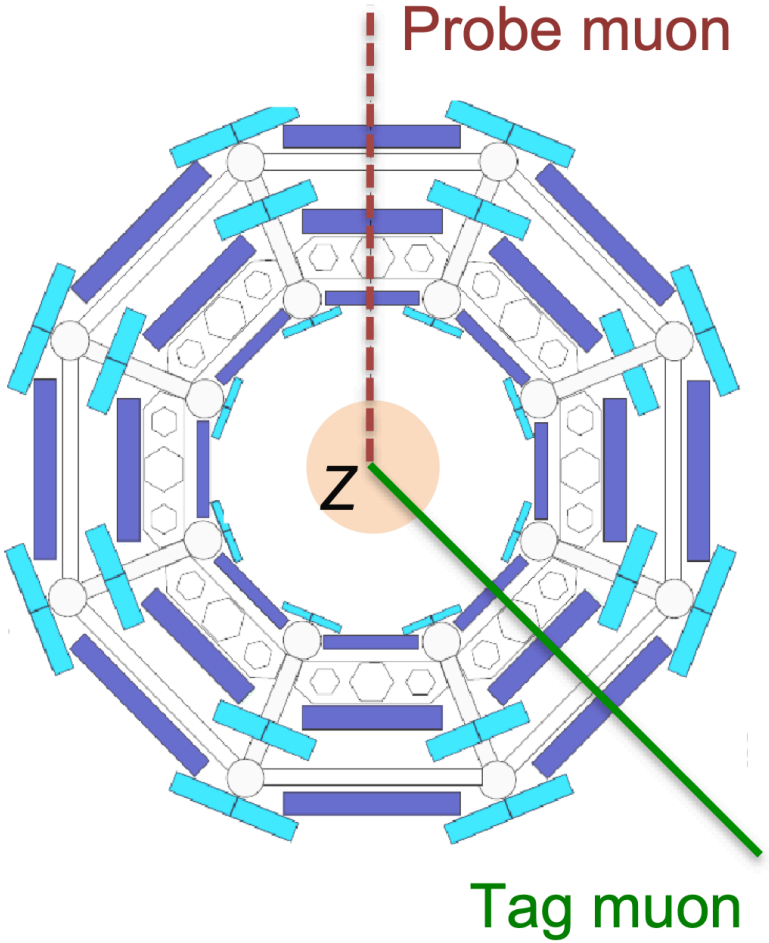
Jet Substructure is a Particularly Sensitive Test

- Very sensitive to the radiation pattern within the jet
- Low C2->Single Hard Core (q/g jets)

  High C2->Resolved Substructure (W->qq)
- Parnassus provides high fidelity fast simulation
  20 events / s



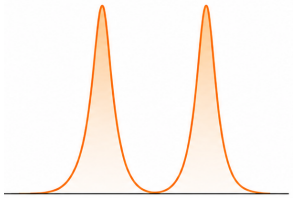
Calibration? → !

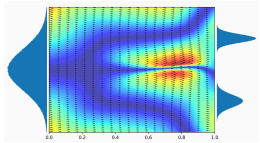


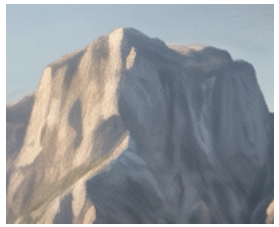
DIY Conclusions

Real: “$y = 1$”

Generated: “$y = 0$”







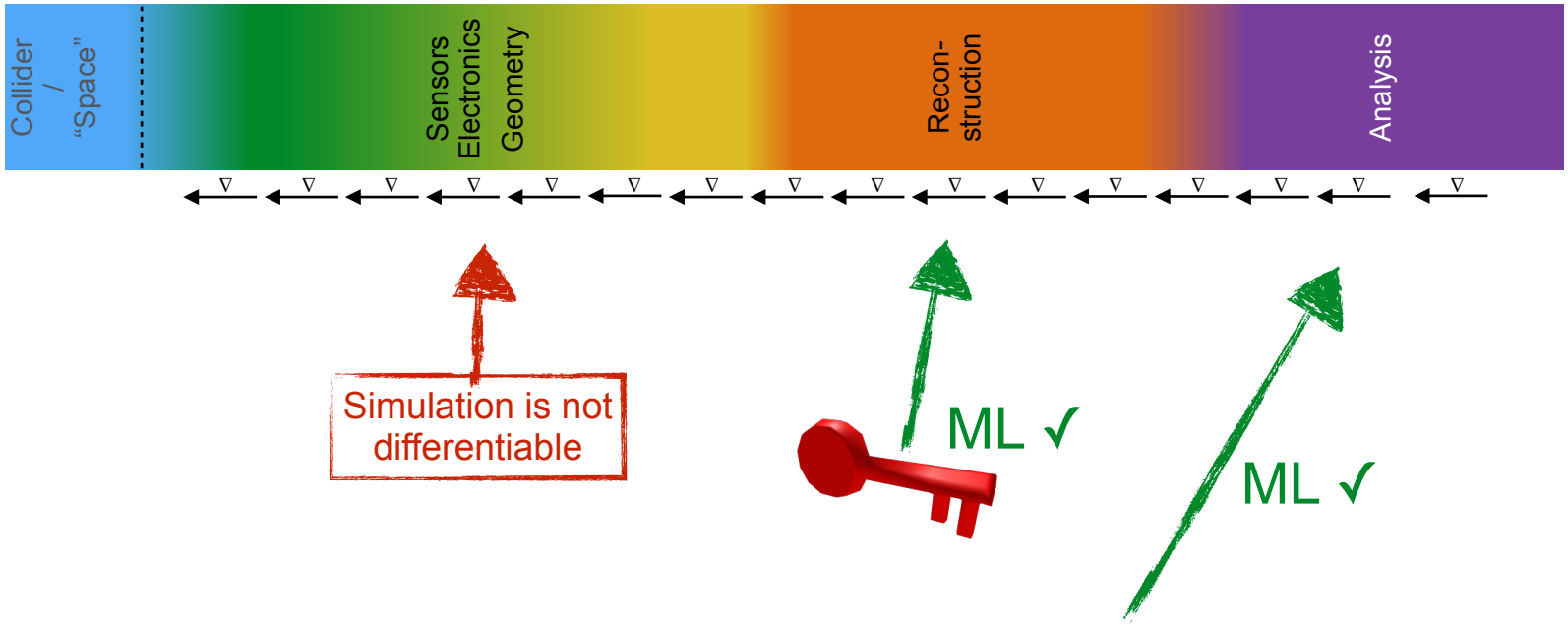
- The average outcome might not be plausible/realistic
- Generative adversarial approaches suppress non-plausible outputs
- Normalising flows provide an invertible function between a random distribution and the desired distribution
- A diffusion model breaks down the inversion into several stochastic steps (invertibility not needed anymore)
- Flow matching: We can map distributions by learning the velocity field through linear mapping
- We can speed up simulation with generative NN a lot; it canals be high fidelity, but more work is needed for ensuring we can calibrate these methods robustly

EXTRA SLIDE(S)

We have put it all Together in Parnassus

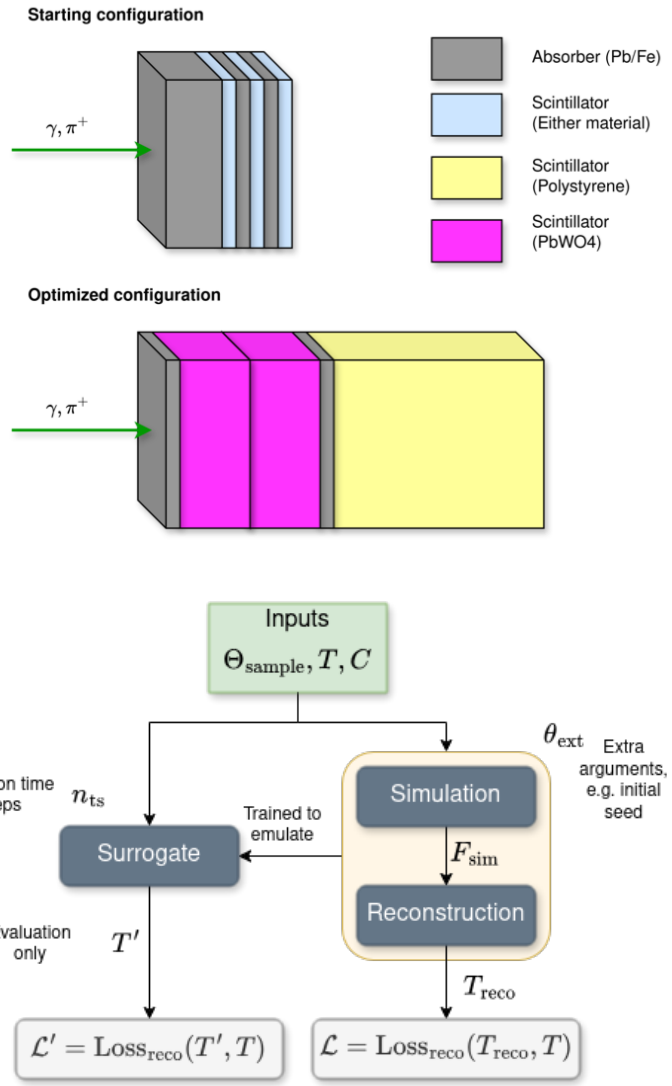
- Flow matching combines ideas from normalising flows and diffusion models
- The architectures use the structure of the input data
- Aggregation functions obey their symmetries
- Graphs with attention create the context to derive the PF candidate properties
- The resulting physics performance is very convincing

A problem with the chain



Diffusion models in physics

- One example: AI driven detector optimisation
- Learn to emulate simulation and reconstruction performance conditioned on detector parameters
- ML model is differentiable:
  we can extract gradients for the detector parameters and perform gradient descent on a detector design!



K. Schmidt, N. Kota, JK, et al; arXiv:2502.02152