Fast, accurate, and precise detector simulation with vision transformers

Luigi Favaro, Ayodele Ore, Sofia Palacios Schweitzer,
Tilman Plehn, Andrea Giammanco

16/06/25 — EuCAIFCon25

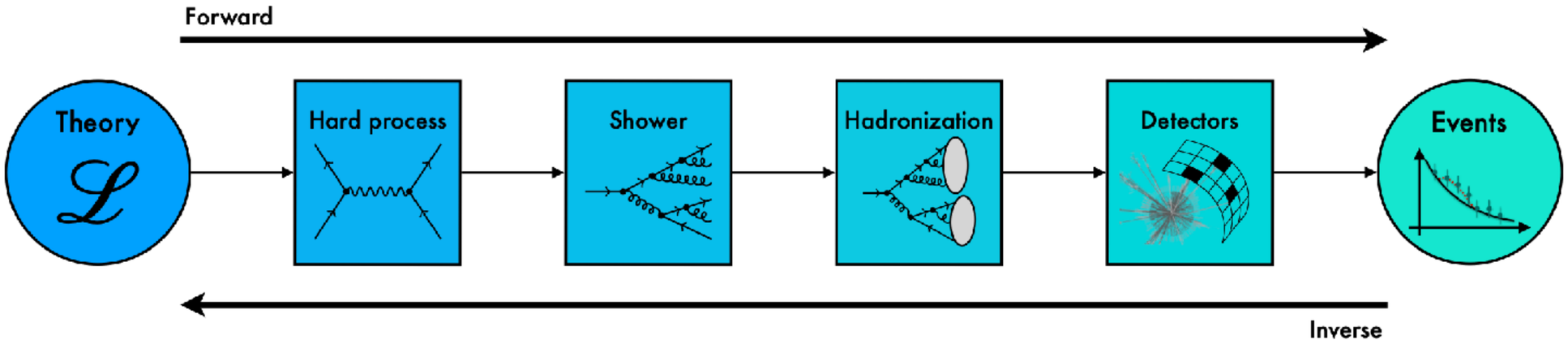
FREEDOM TO RESEARCH

UCLouvain

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

HEP simulations

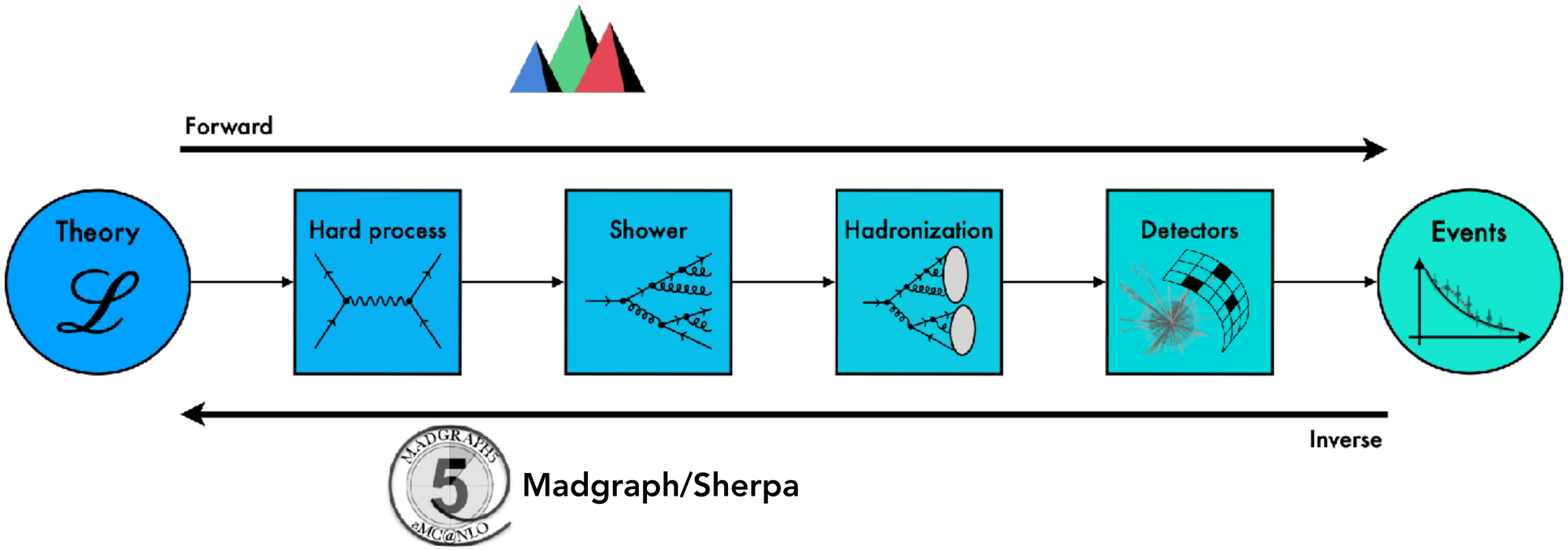
Simulation chain



- MCMC samplers are the foundations of HEP simulations.

HEP simulations

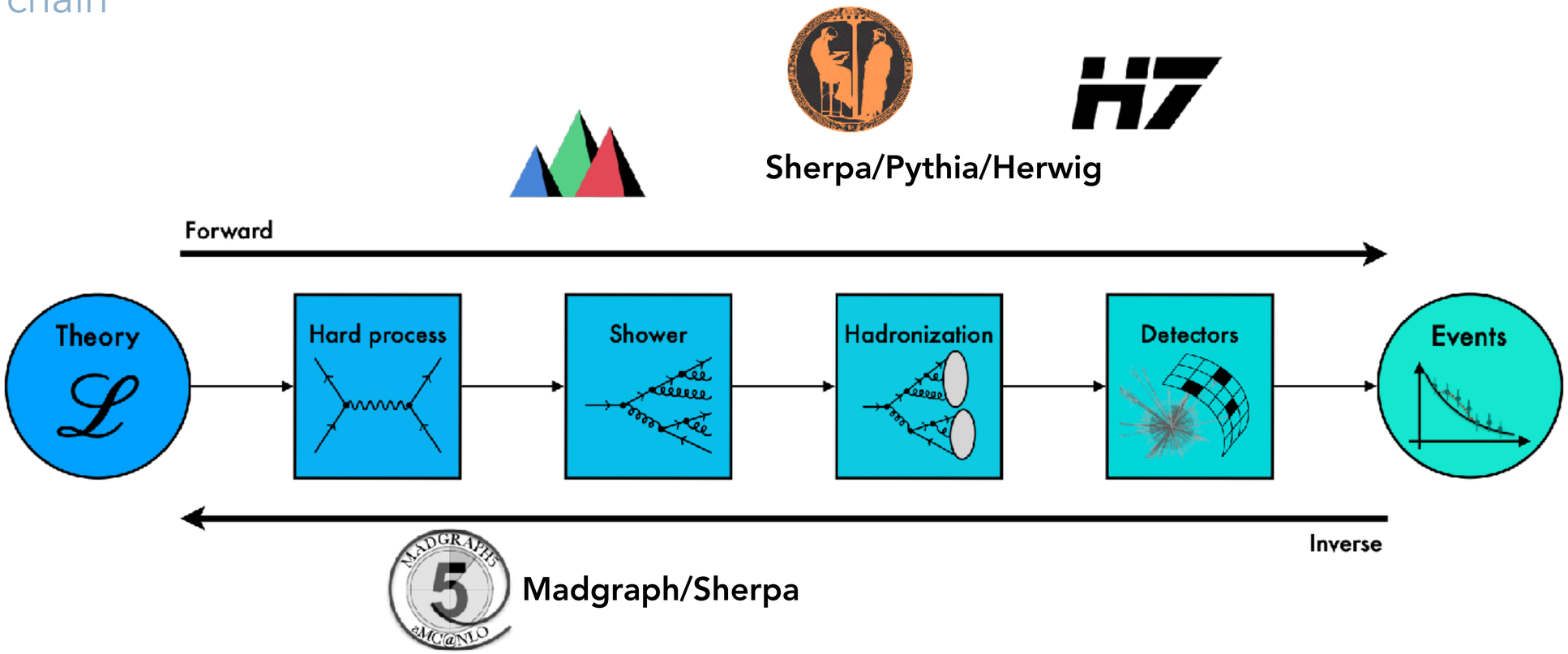
Simulation chain



- MCMC samplers are the foundations of HEP simulations.

HEP simulations

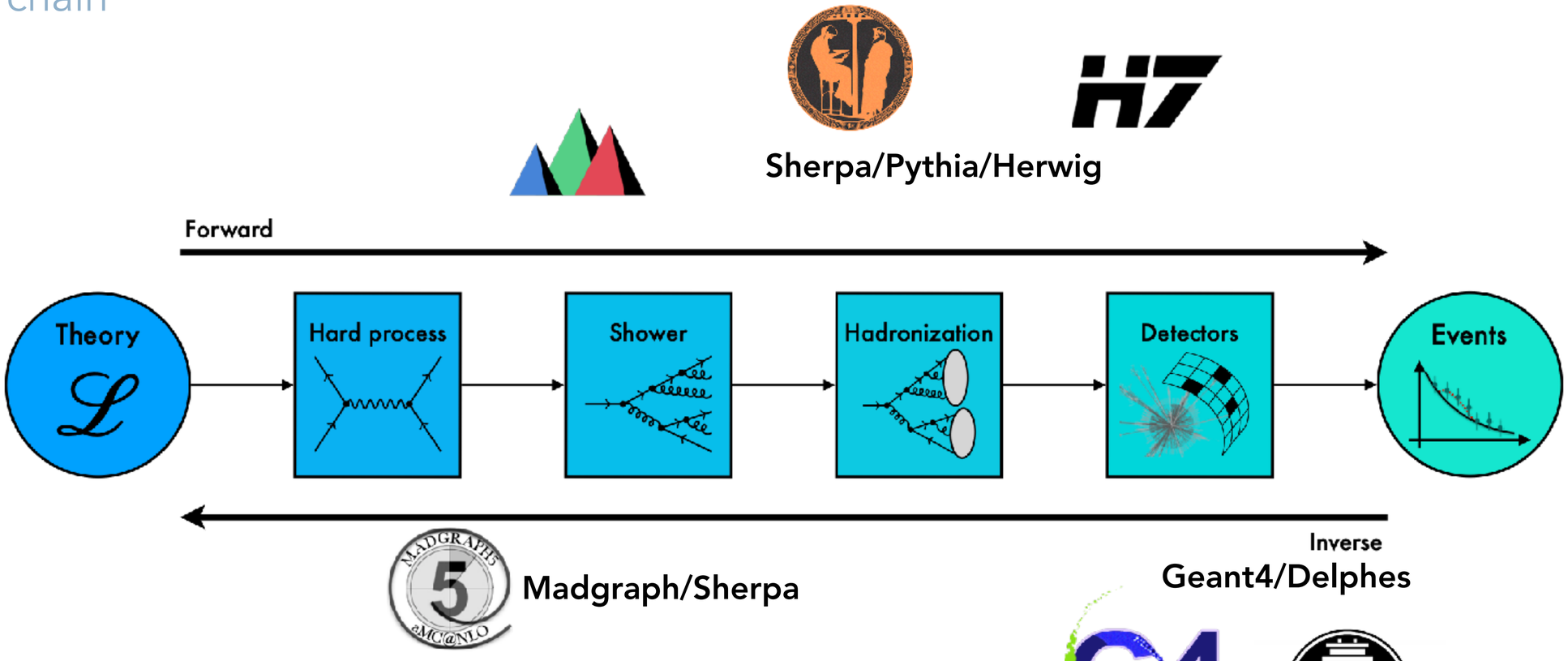
Simulation chain



- MCMC samplers are the foundations of HEP simulations.

HEP simulations

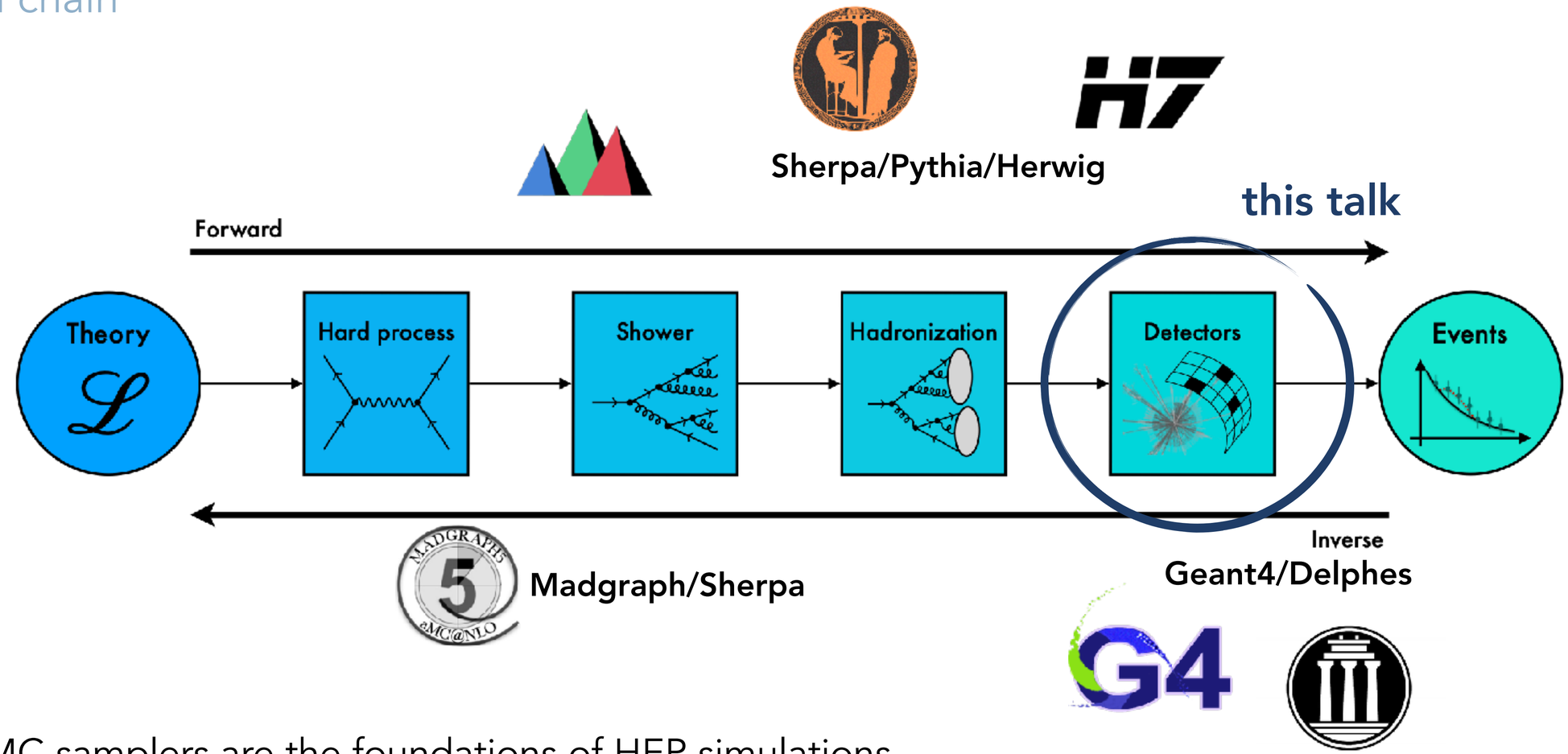
Simulation chain



- MCMC samplers are the foundations of HEP simulations.

HEP simulations

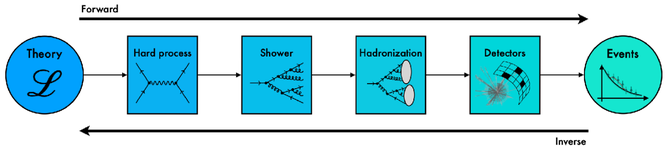
Simulation chain



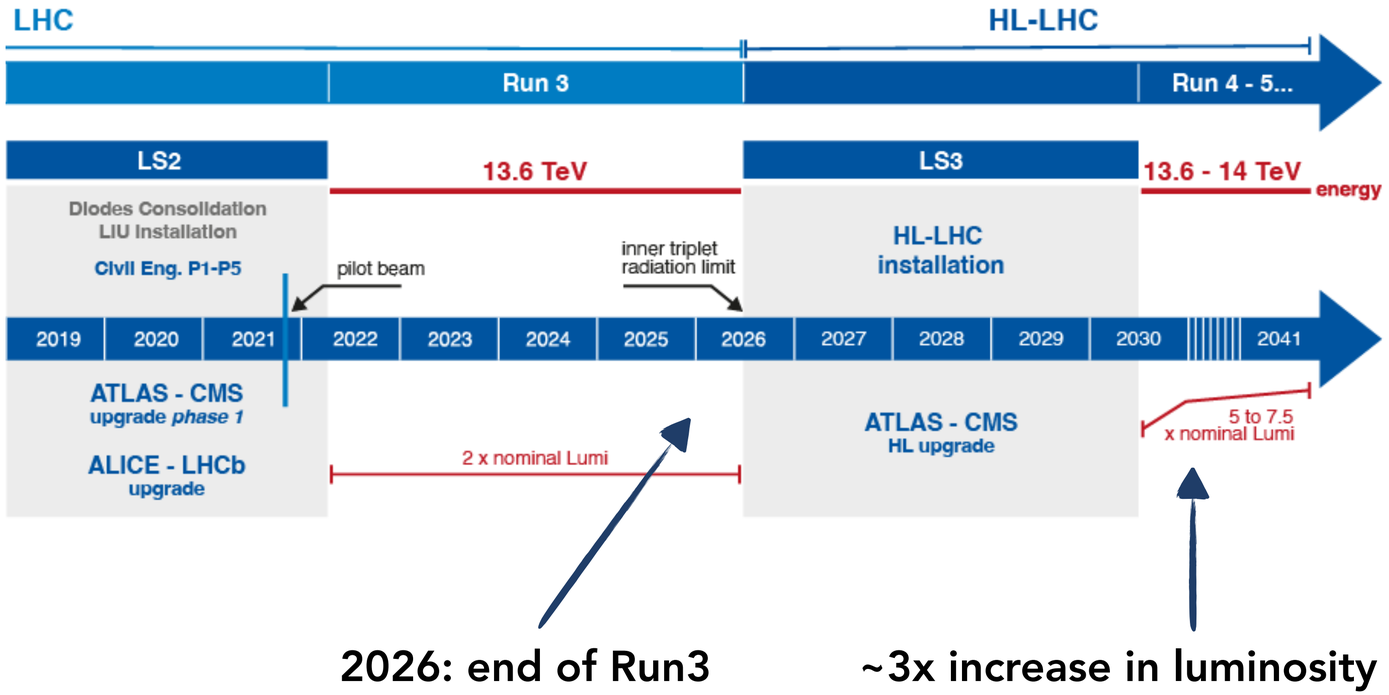
- MCMC samplers are the foundations of HEP simulations.

HEP simulations

Detector simulation

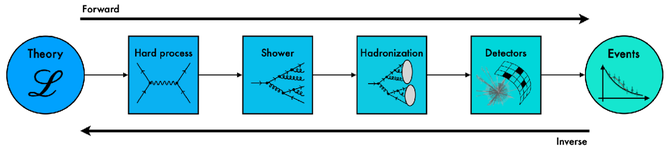


- High-Luminosity LHC will produce unmatched amount of data;
- simulations need to match the collected statistics.



HEP simulations

Detector simulation



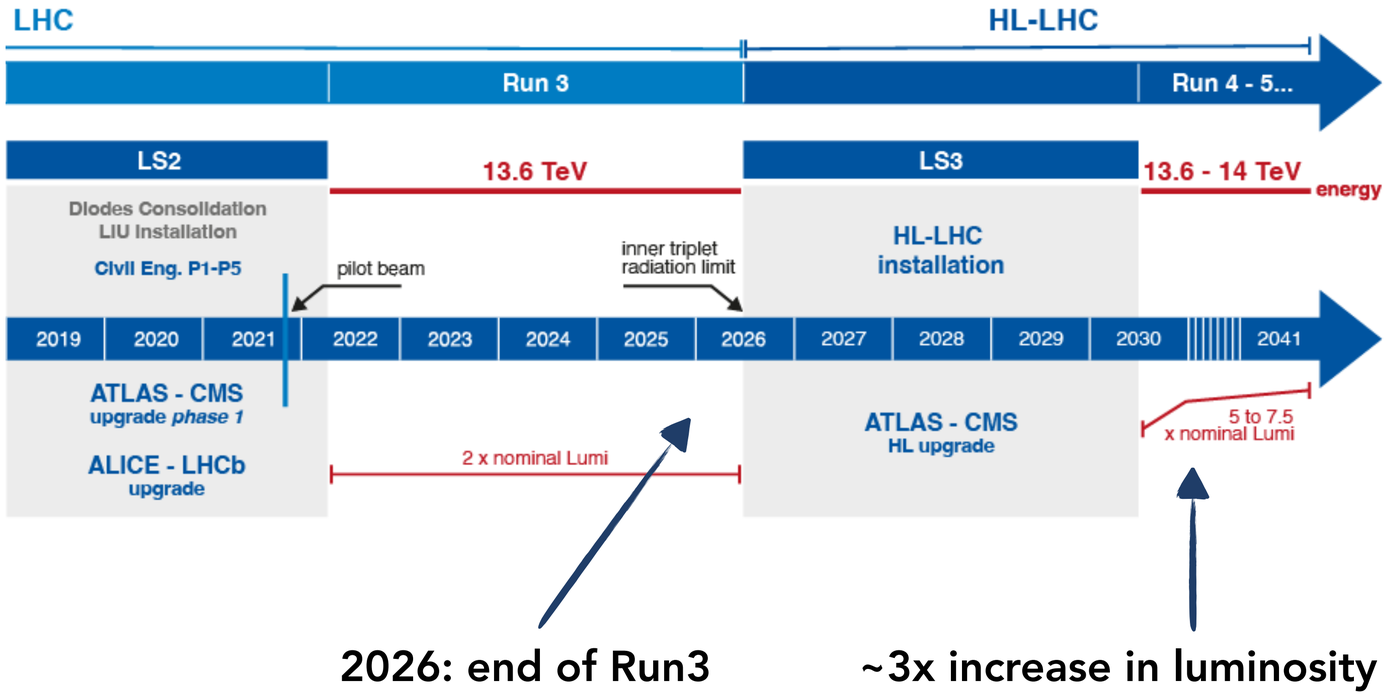
- High-Luminosity LHC will produce unmatched amount of data;
- simulations need to match the collected statistics.

Overarching goal:

Maximise LHC potential

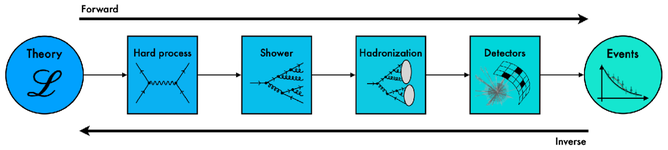
- find BSM physics;
- understand LHC data w/ SM.

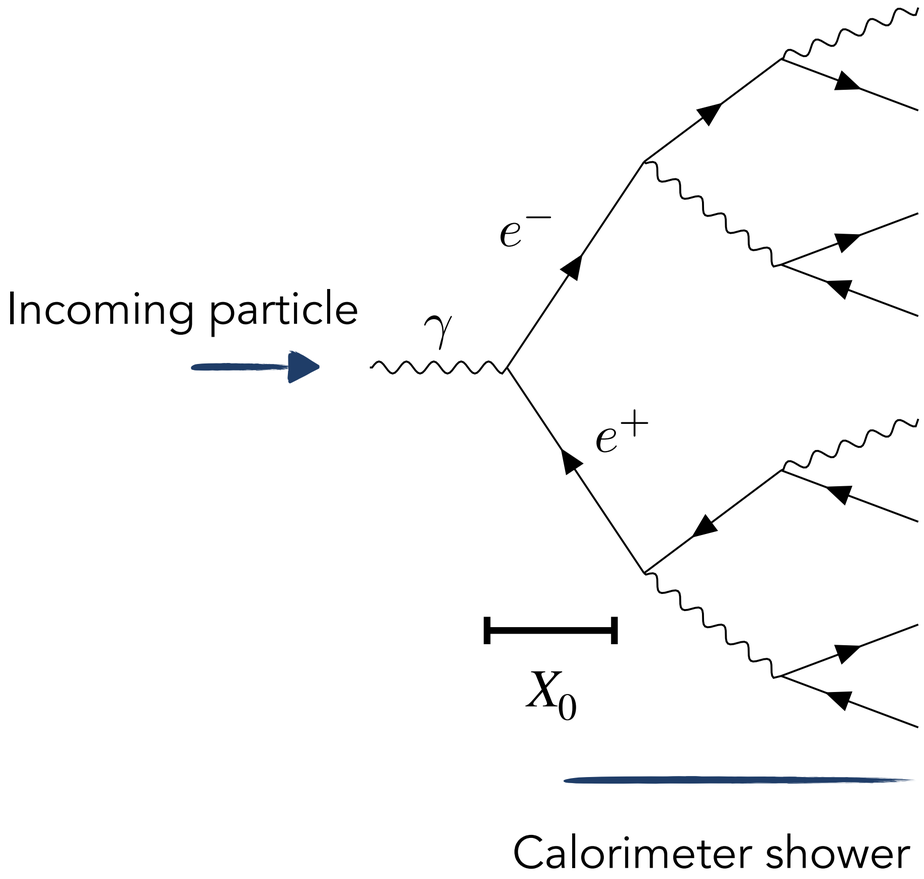
Requires better and faster simulators



HEP simulations

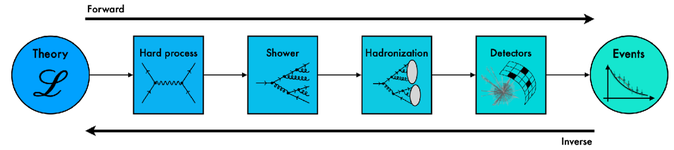
Detector simulation



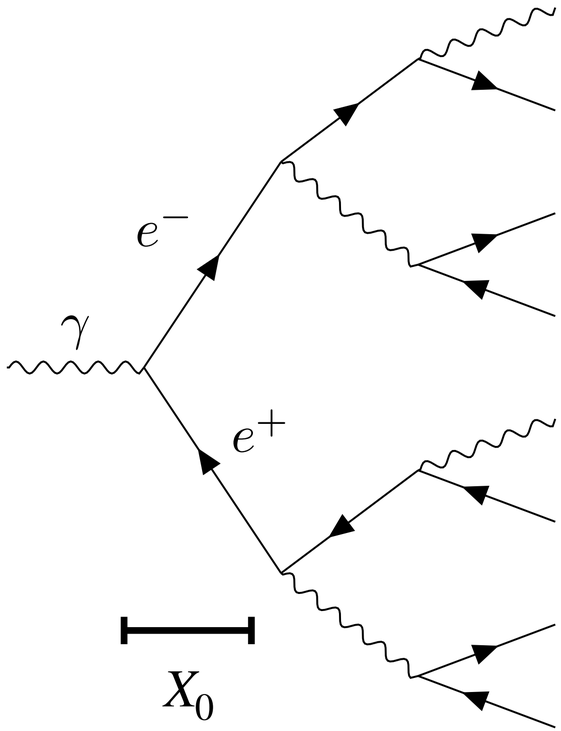


HEP simulations

Detector simulation

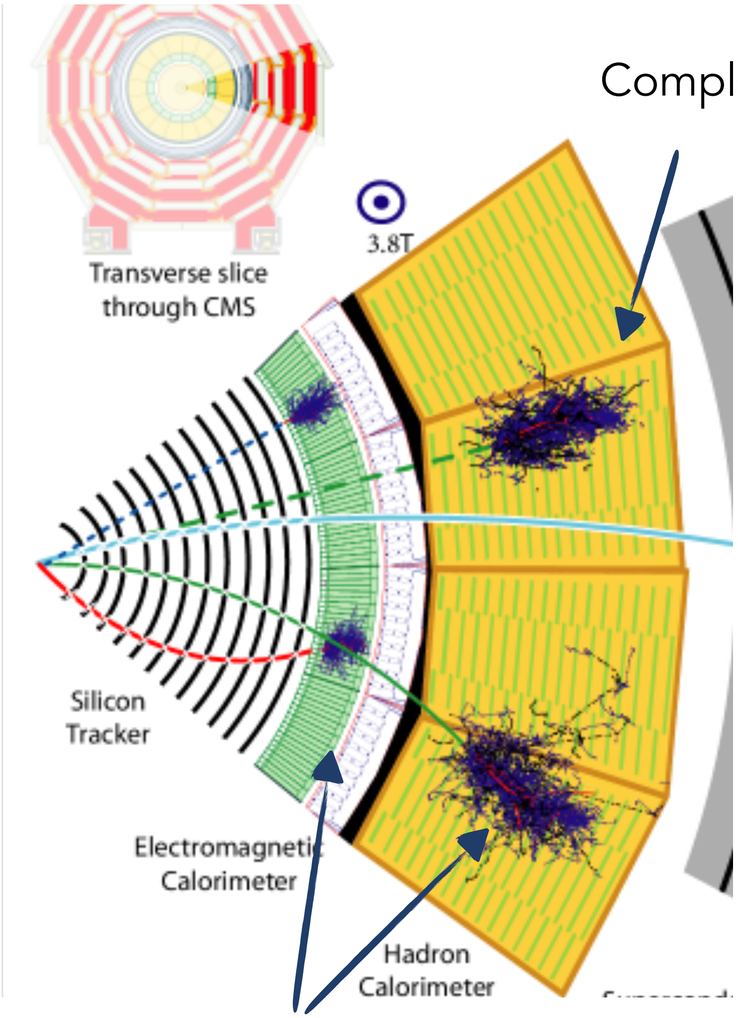


Incoming particle



Calorimeter shower

Reality

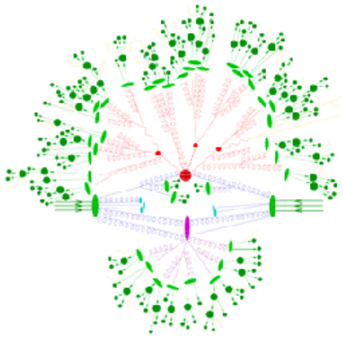


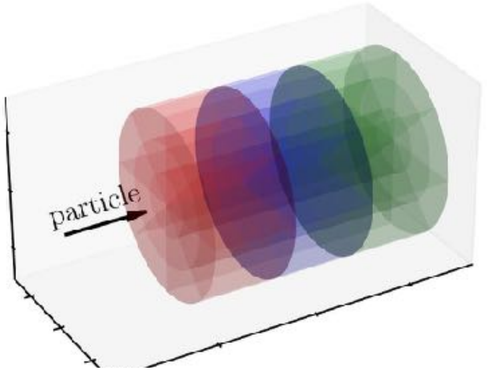
Complex structure

Different physics

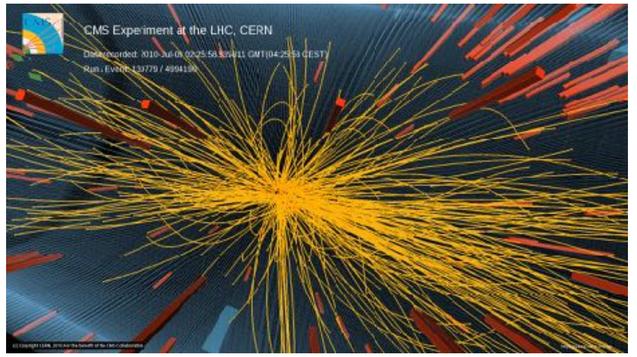
HEP simulations

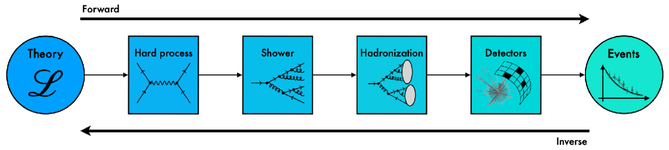
ML emulators





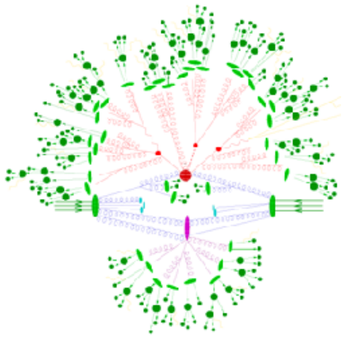
Slow

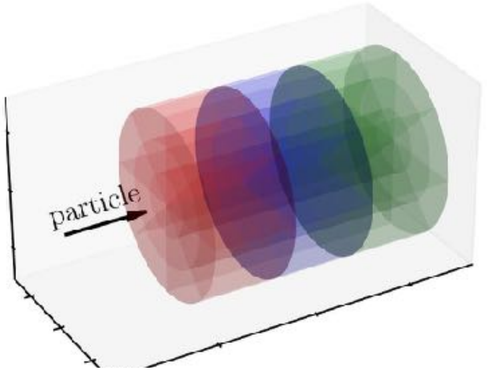




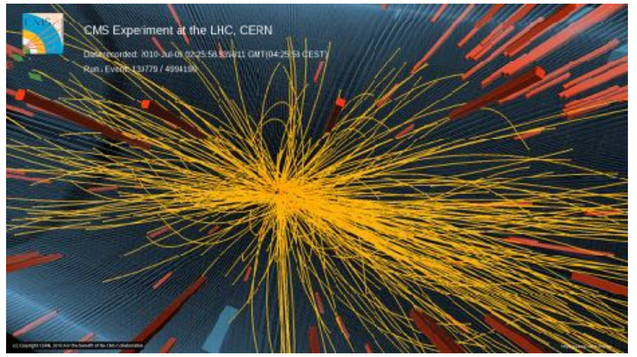
HEP simulations

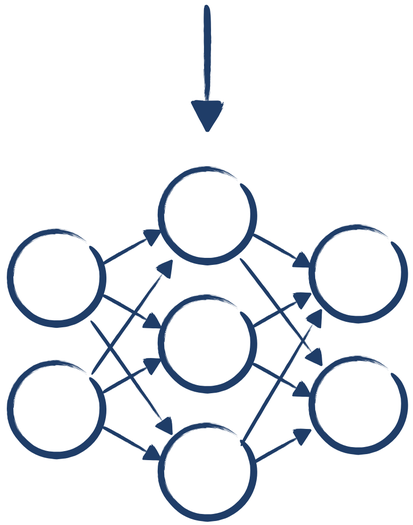
ML emulators



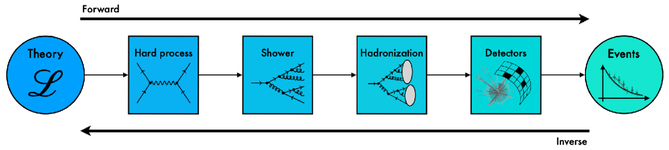


Slow



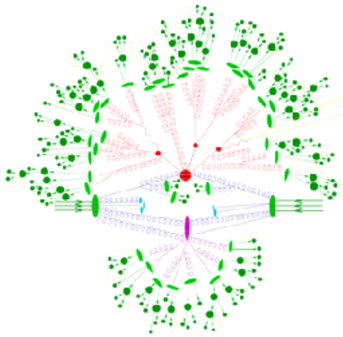


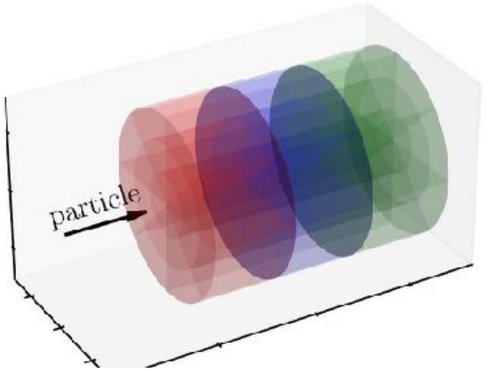
Train with $x \sim p(\text{G4} \mid E_{\text{inc}})$



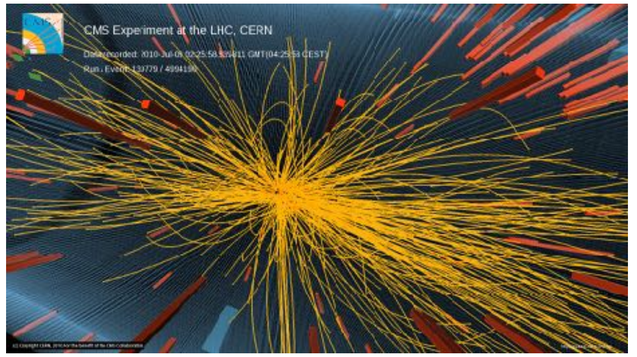
HEP simulations

ML emulators

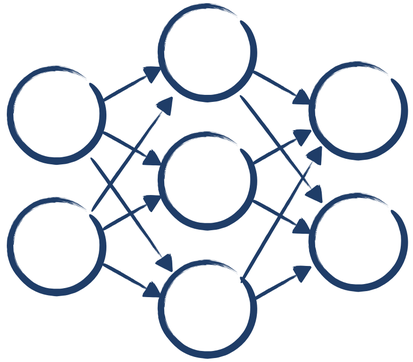




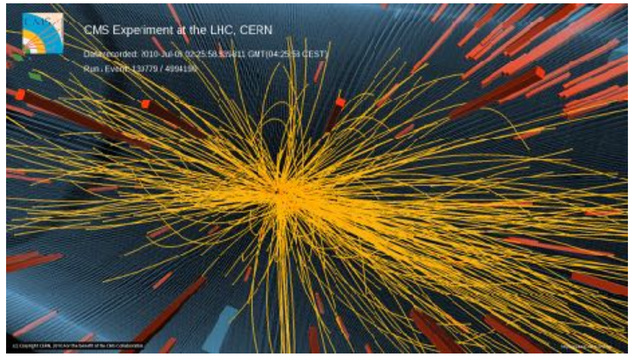
Slow

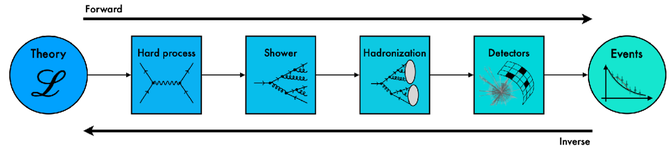


Train with $x \sim p(\text{G4} \mid E_{\text{inc}})$



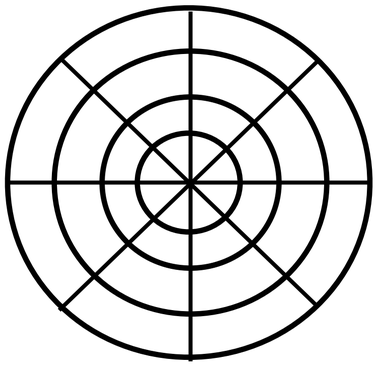
Fast





Vision Transformers

Paradigm

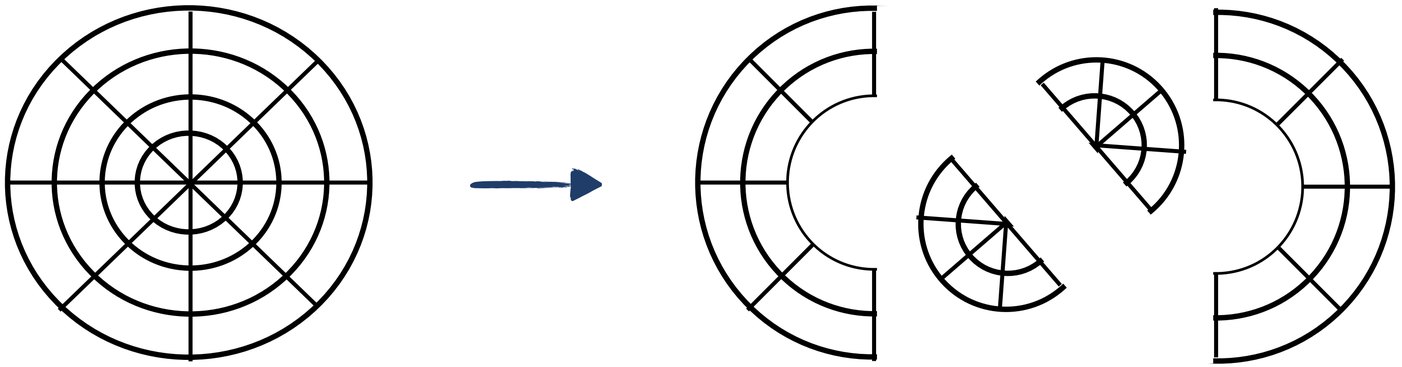


Detector geometry

- Energy deposition in each cell;
- high-dimensional input;
- 3D structure: (x, y, z) or (r, α, z).

Vision Transformers

Paradigm



Detector geometry

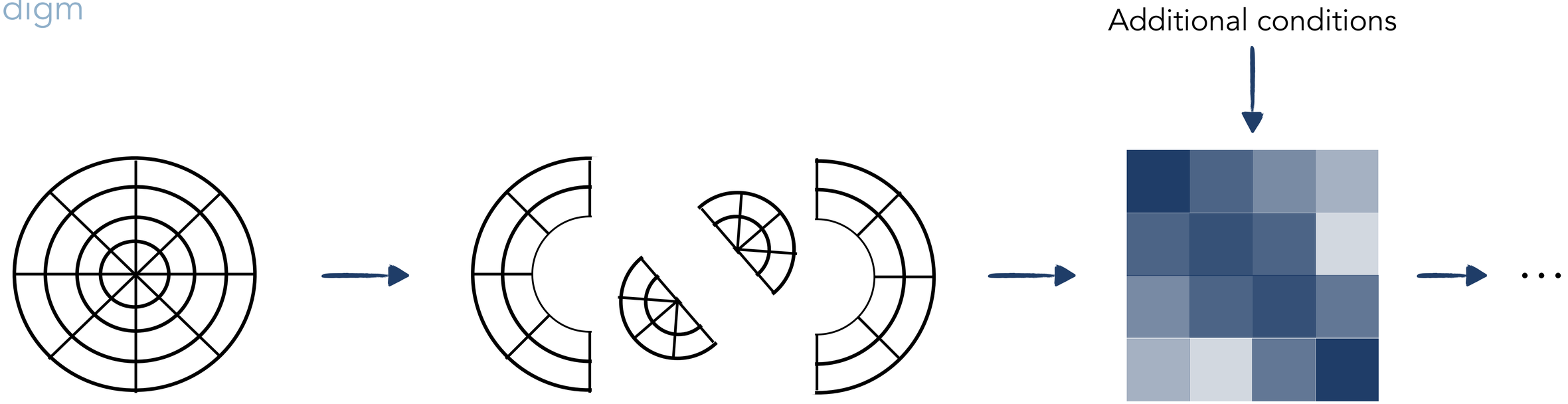
- Energy deposition in each cell;
- high-dimensional input;
- 3D structure: (x, y, z) or (r, α, z).

Divide into patches

- Define a recipe for patching;
- domain-knowledge input.

Vision Transformers

Paradigm



Detector geometry

- Energy deposition in each cell;
- high-dimensional input;
- 3D structure: (x, y, z) or (r, α, z).

Divide into patches

- Define a recipe for patching;
- domain-knowledge input.

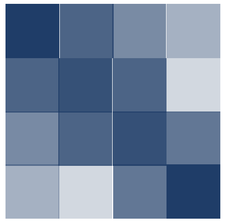
Self-attention on patched objects

- Stack of transformer blocks;
- conditions: $(E_{\text{inc}}, E_i, \ldots)$.

Vision Transformers

& Generative networks

Normalizing Flows



$$\theta(x) \in \mathbb{R}^{(3m-1)n}$$

Learn m bins RQS for n input feature:

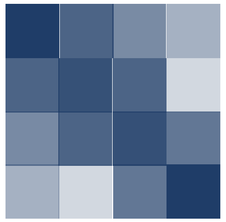
$$\mathscr{L} = -\left\langle \log p_z(f_\theta(x)) + \log \left| \frac{\partial f_\theta}{\partial x} \right| \right\rangle$$

- Fast
- Discrete, less expressive

Vision Transformers

& Generative networks

Normalizing Flows



$$\theta(x) \in \mathbb{R}^{(3m-1)n}$$

Learn m bins RQS for n input feature:

$$\mathscr{L} = -\left\langle \log p_z(f_\theta(x)) + \log \left| \frac{\partial f_\theta}{\partial x} \right| \right\rangle$$

- Fast
- Discrete, less expressive

Conditional Flow Matching

$$v_\phi(x,t) \in \mathbb{R}^n$$

Regress a velocity vector comparing to a target conditional velocity:

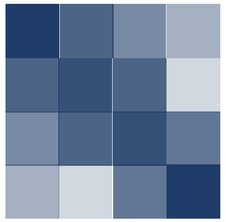
$$\mathscr{L} = \left\langle \left|\left| v_\phi(x,t) - v(x,t\,|\,x_0) \right|\right| \right\rangle$$

- Very expressive
- Slower, multiple function evaluations

Vision Transformers

& Generative networks

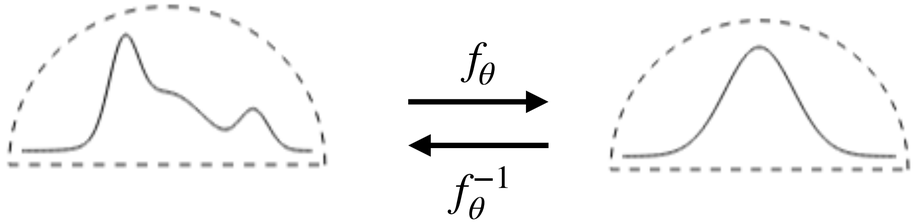
Normalizing Flows



$$\theta(x) \in \mathbb{R}^{(3m-1)n}$$

Learn m bins RQS for n input feature:

$$\mathscr{L} = -\left\langle \log p_z(f_\theta(x)) + \log \left| \frac{\partial f_\theta}{\partial x} \right| \right\rangle$$



Conditional Flow Matching

$$v_\phi(x,t) \in \mathbb{R}^n$$

Regress a velocity vector comparing to a target conditional velocity:

$$\mathscr{L} = \left\langle \left|\left| v_\phi(x,t) - v(x,t\,|\,x_0) \right|\right| \right\rangle$$

Sampling $\longrightarrow$ $x(t=0) = x(t=1) - \int_0^1 v_\phi(x,t)dt$

Vision Transformers

arXiv:2405.09629, **LF**, Ore A., Palacios Schweitzer S., Plehn T.

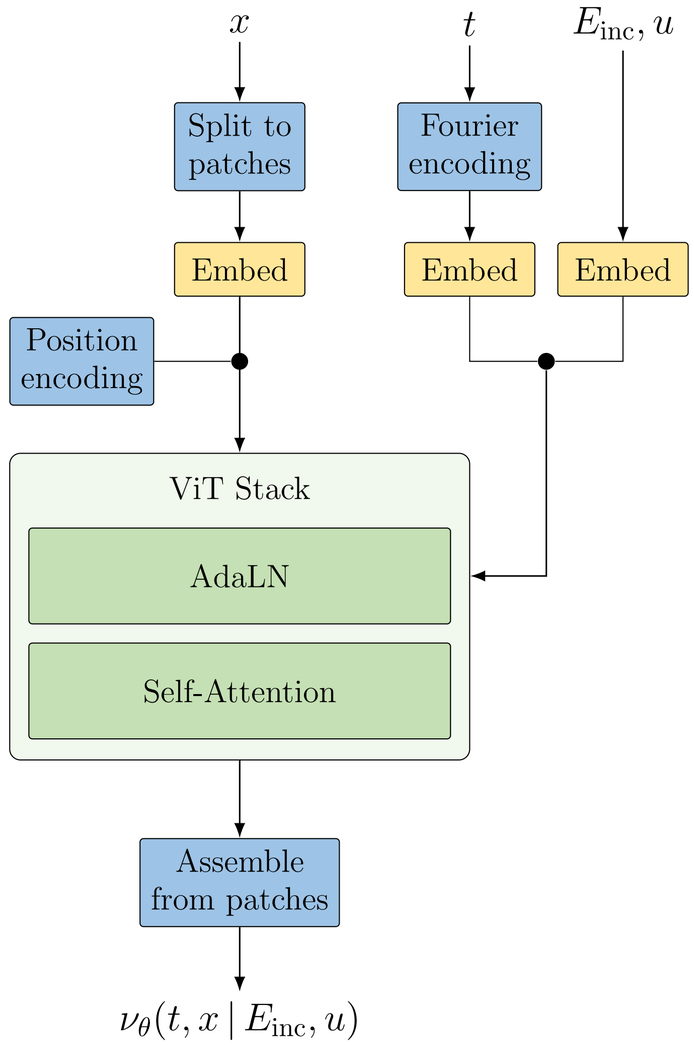
CaloDREAM — example

- Split generation of layer energies and voxels;
- split detector into patches;
- embed patches and conditions;
- apply a residual transformation to the inputs:
  - multi-head self-attention;
  - fully-connected network.

$$x_l = x_h + \gamma_l \cdot g_l(a_l x_h + b_l)$$

AdaLN: γ_l, a_l, b_l learnable, conditioned on t, E_{inc}, u

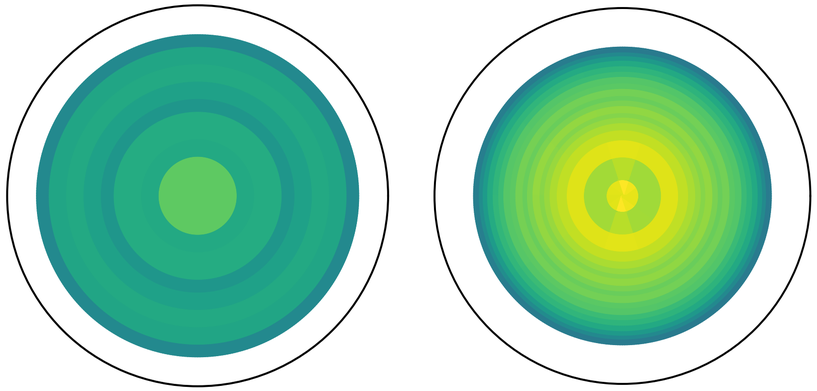
- predict a v_θ for each voxel.



Datasets

CaloChallenge

All the datasets are available on Zenodo and can be accessed from:
https://calochallenge.github.io/homepage/



Dataset 1 (AtlFast3)

γ showers — 368 voxels

π^+ showers — 533 voxels

121k showers

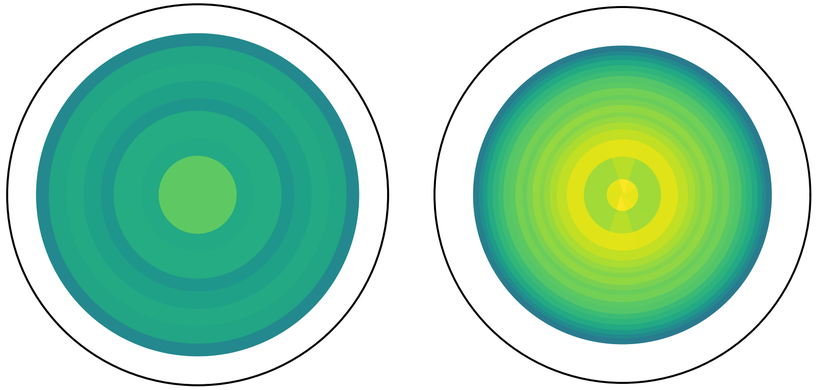
discrete E_{inc}

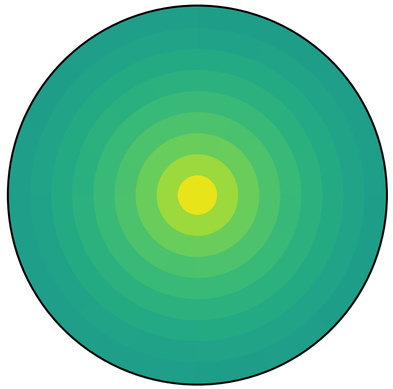
+ irregular geometries

Datasets

CaloChallenge

All the datasets are available on Zenodo and can be accessed from:
https://calochallenge.github.io/homepage/





Dataset 1 (AtlFast3)

γ showers — 368 voxels

π^+ showers — 533 voxels

121k showers

discrete E_{inc}

+ irregular geometries

Dataset 2

e^+ showers — 6480 voxels

100k showers

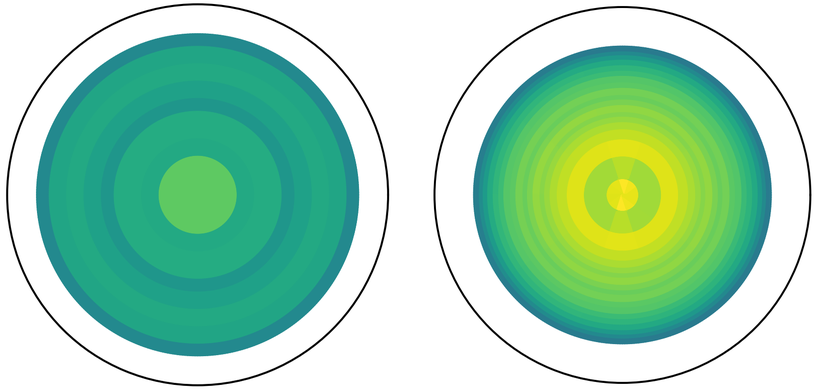
$E_{\text{inc}} \in \text{LogUnif}\,(1, 10^3)$ GeV

(r, α, z): (9, 16, 45)

Datasets

CaloChallenge

All the datasets are available on Zenodo and can be accessed from:
https://calochallenge.github.io/homepage/



Dataset 1 (AtlFast3)

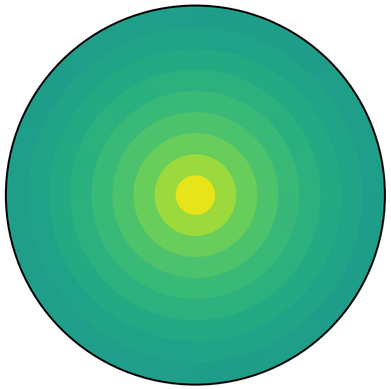
γ showers — 368 voxels

π^+ showers — 533 voxels

121k showers

discrete E_{inc}

+ irregular geometries



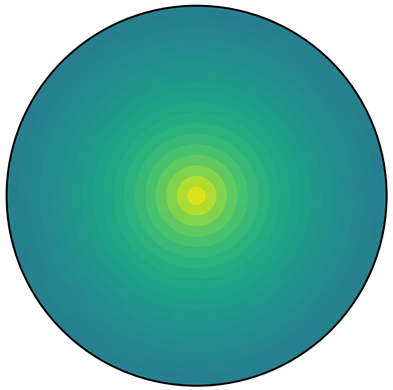
Dataset 2

e^+ showers — 6480 voxels

100k showers

$E_{\text{inc}} \in \text{LogUnif}\,(1,10^3)$ GeV

(r, α, z): (9, 16, 45)



Dataset 3

e^+ showers — 40500 voxels

100k showers

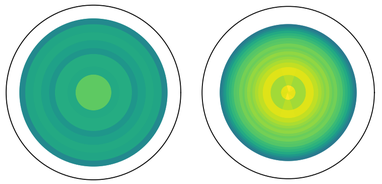
$E_{\text{inc}} \in \text{LogUnif}\,(1,10^3)$ GeV

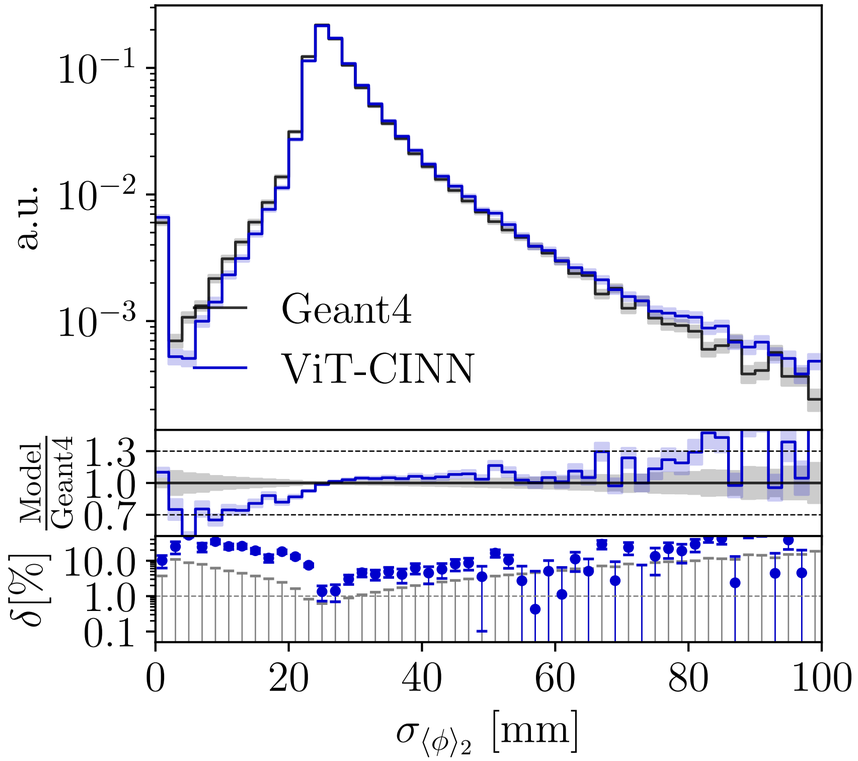
(r, α, z): (18, 50, 45)

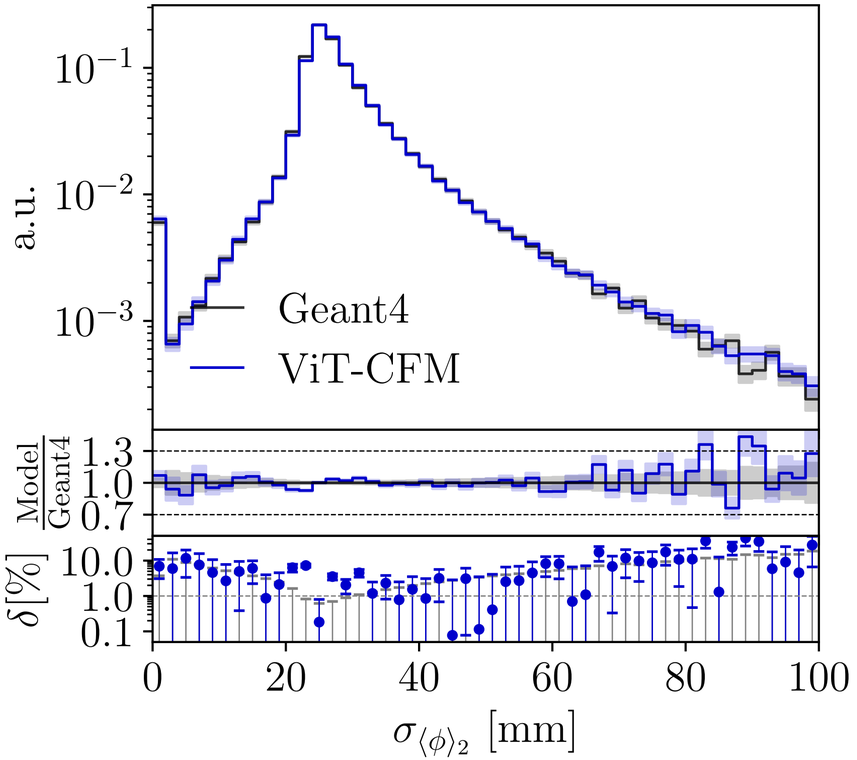
High-Level Features

Irregular geometry

Dataset 1 — γ





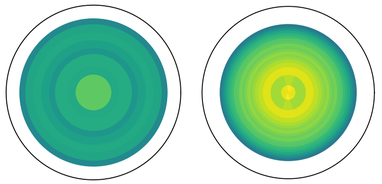


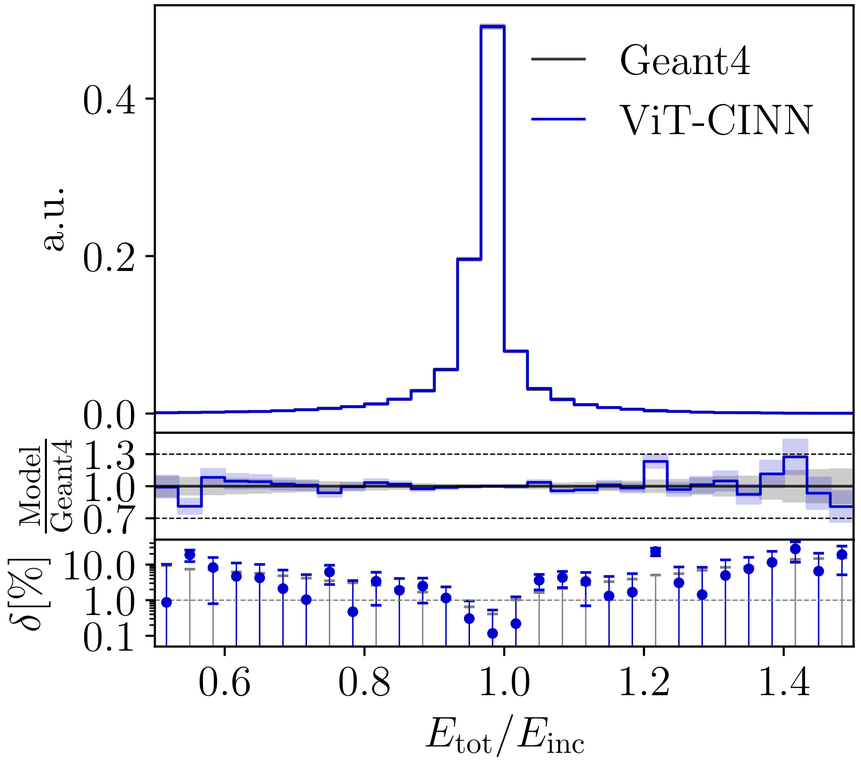
per-layer shower width: $\sigma_{\langle\xi\rangle} = \sqrt{\frac{\xi^2 \cdot x}{\sum_i x_i} - \langle\xi\rangle^2}, \quad \xi \in \{\phi, \eta\}$

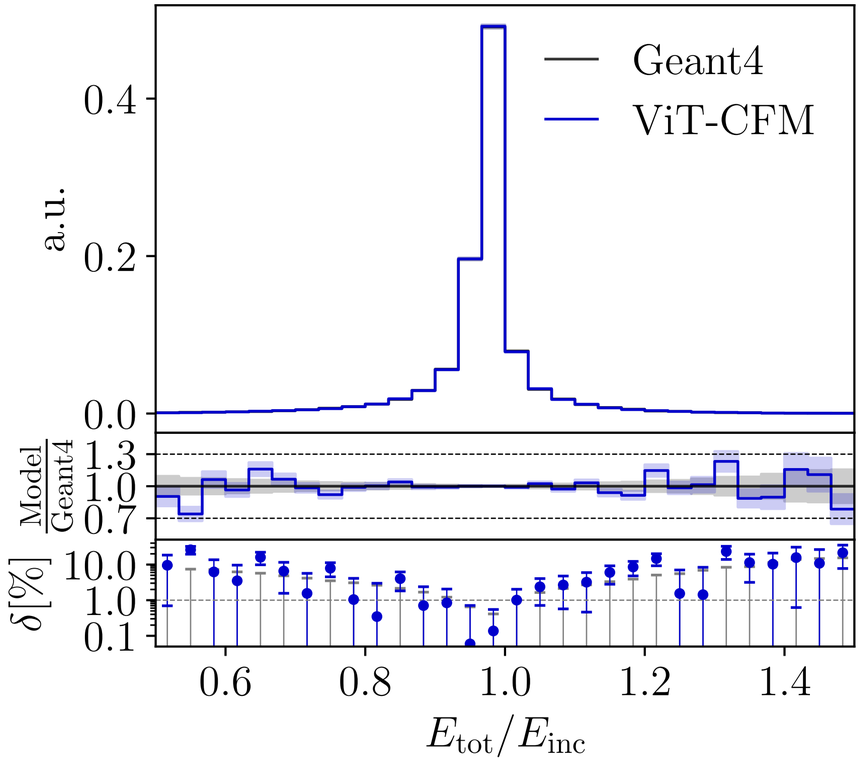
High-Level Features

Irregular geometry

Dataset 1 — γ





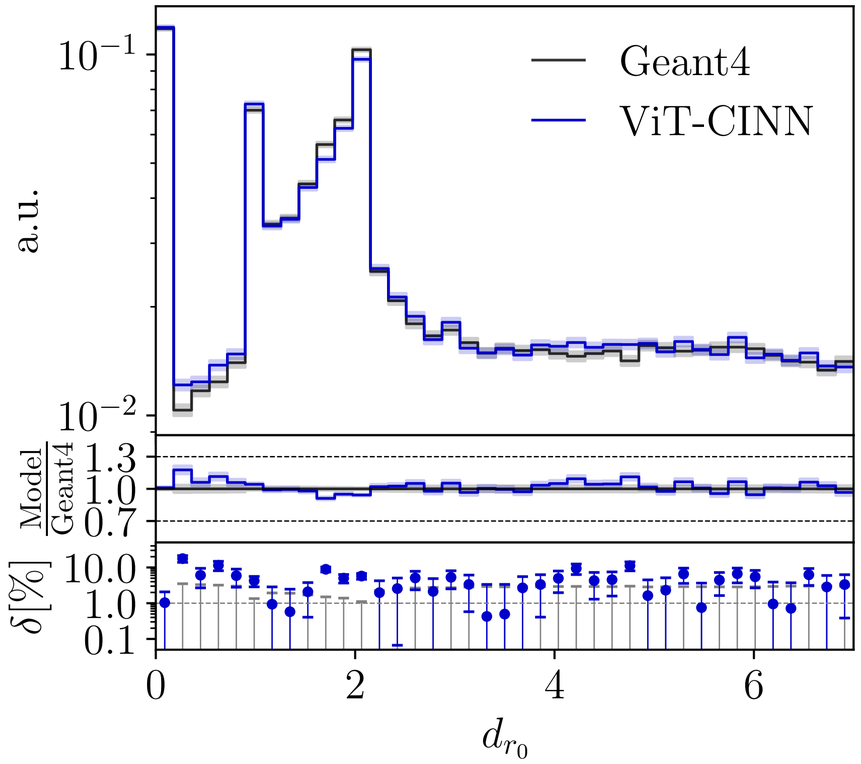


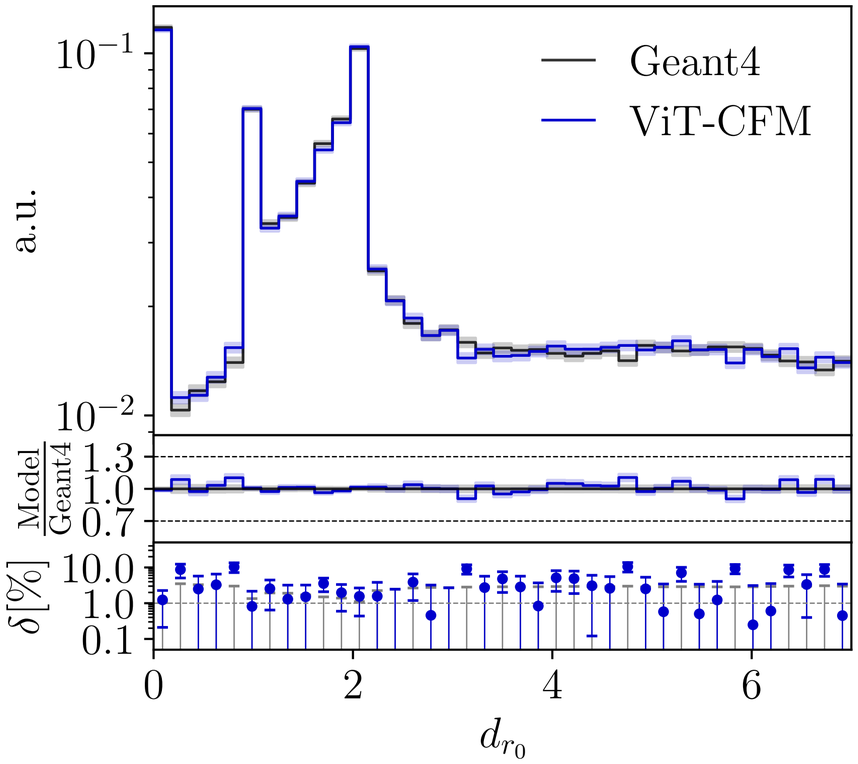
$$\frac{\text{Total energy}}{\text{Incident energy}}$$

High-Level Features

Hadronic showers

Dataset 1 — π^+



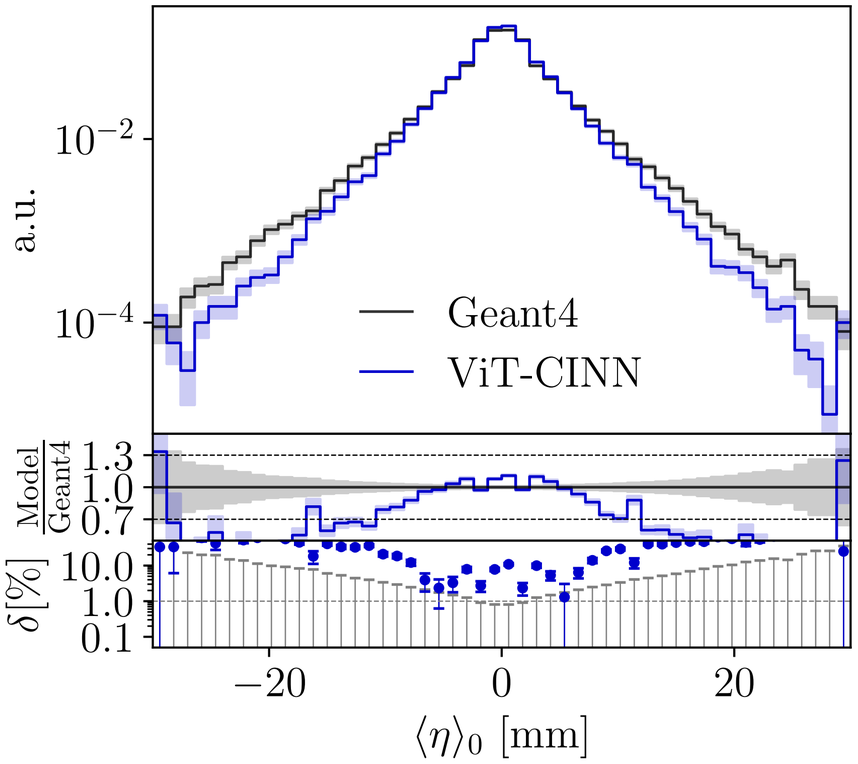


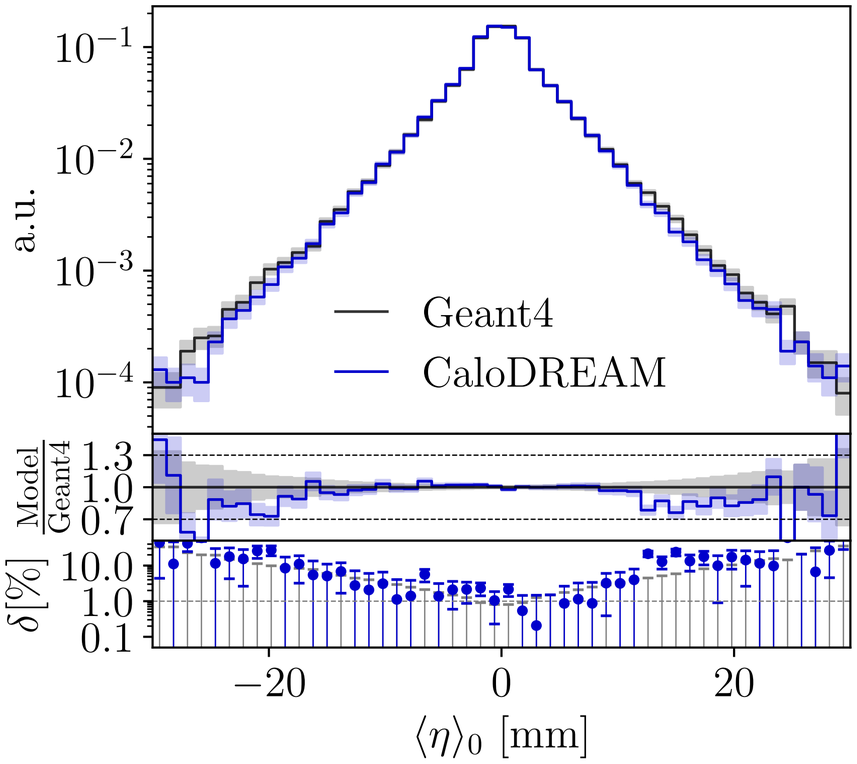
energy-weighted shower depth: $d_{r_j} = \frac{\sum_i^N k_i E_{i,r_j}}{E_{\text{tot},j}}$

High-Level Features

Scaling to high-dimensionalities

Dataset 2



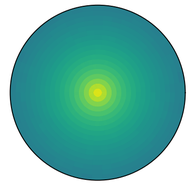


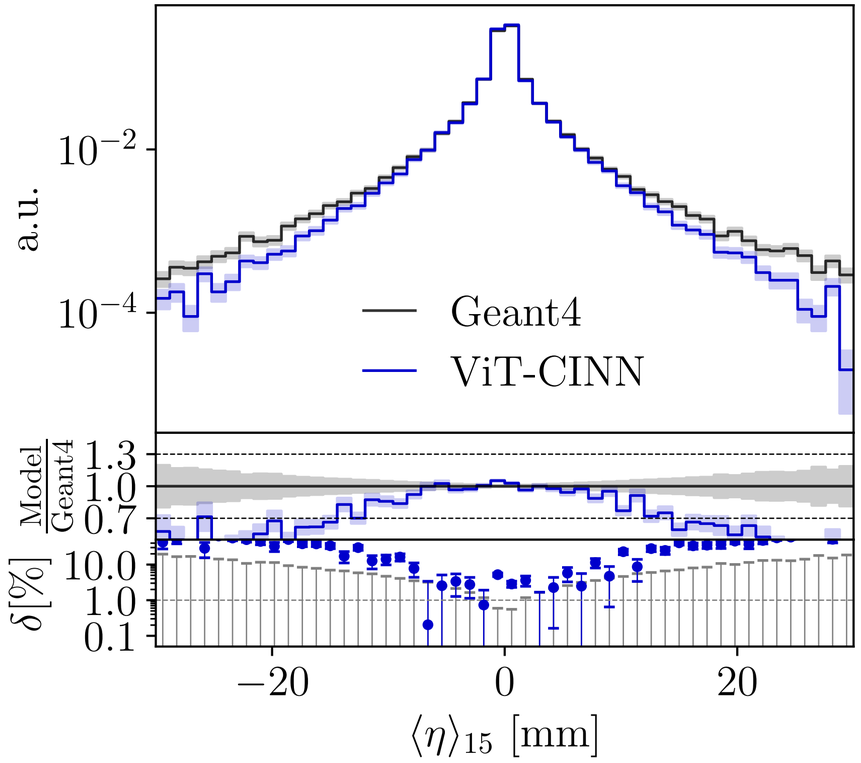
per-layer shower center: $\langle\xi\rangle = \frac{\xi \cdot x}{\sum_i x_i}, \quad \xi \in \{\phi, \eta\}$

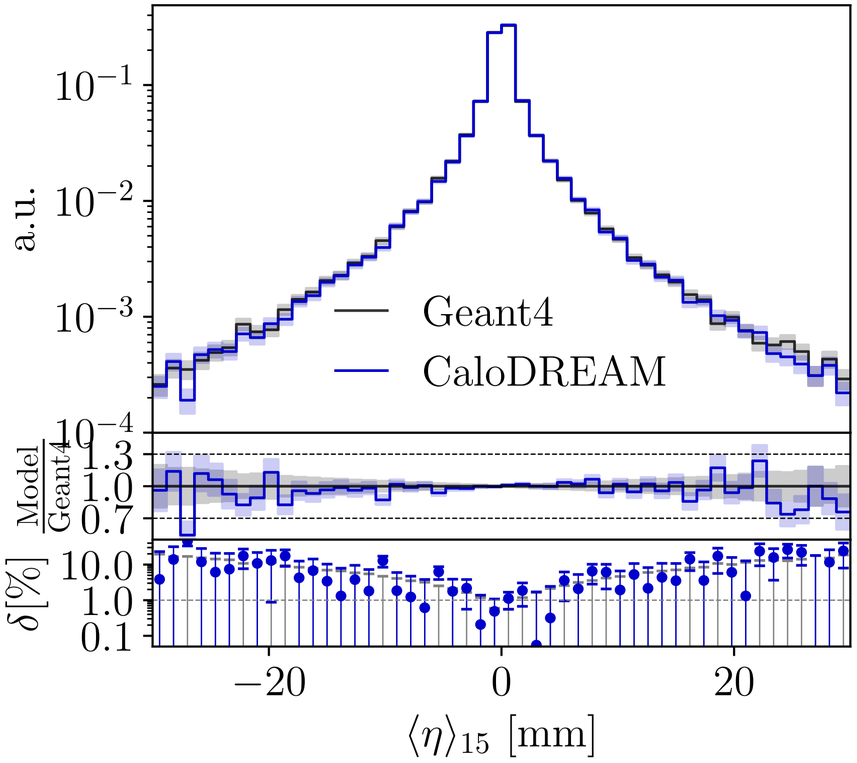
High-Level Features

Scaling to high-dimensionalities

Dataset 3





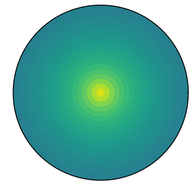


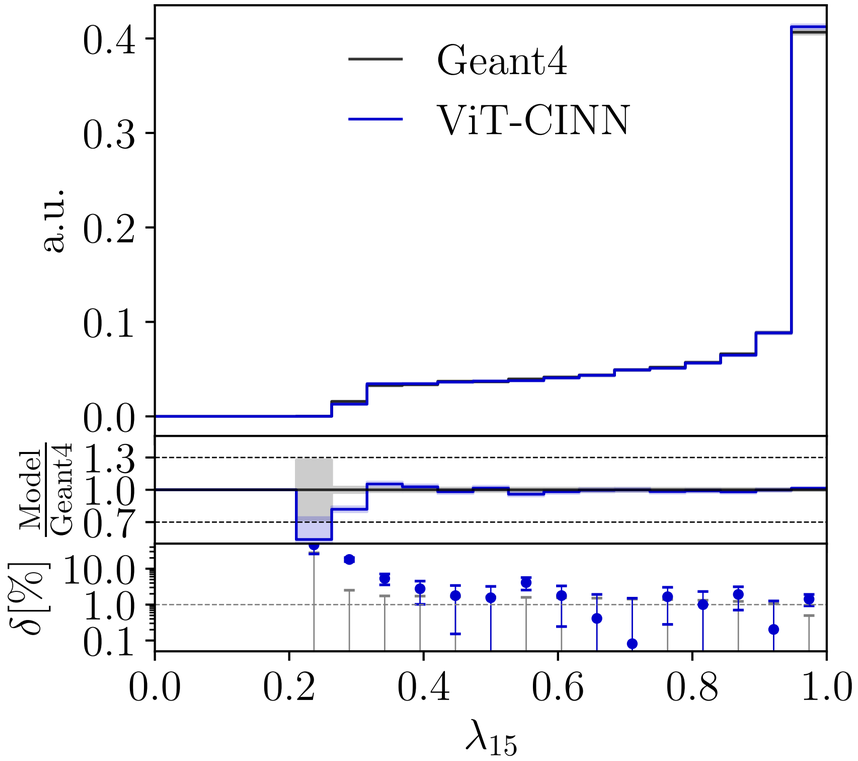
per-layer shower center: $\langle\xi\rangle = \frac{\xi \cdot x}{\sum_i x_i}, \quad \xi \in \{\phi, \eta\}$

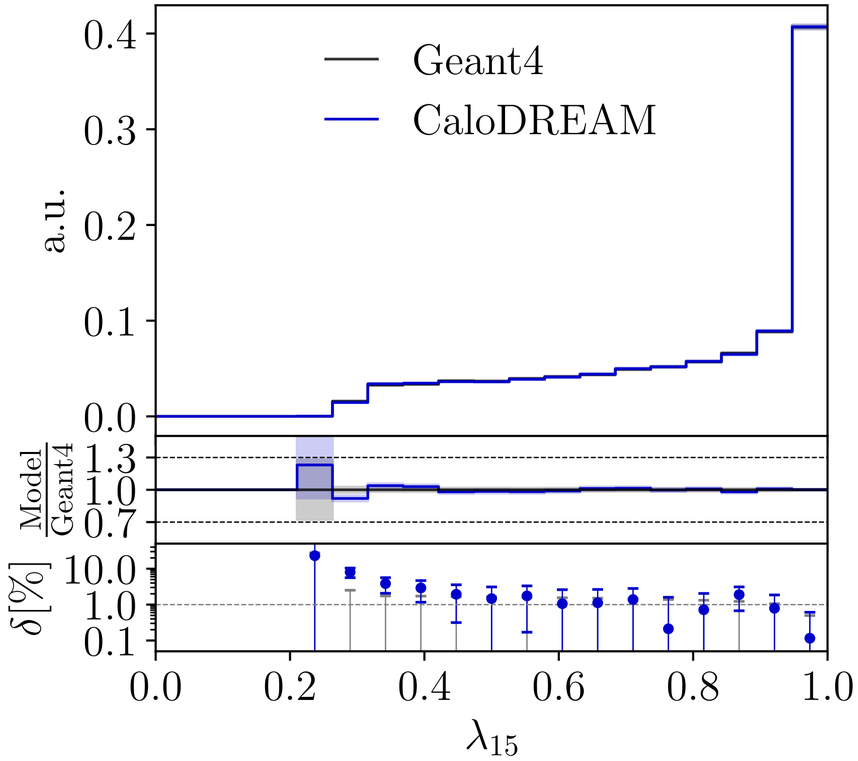
High-Level Features

Scaling to high-dimensionalities

Dataset 3







layer sparsity

Evaluation

Fidelity — Classifier test

How to evaluate the performance of gen. neural networks?

Train a classifier:

- LR optimal observable in simple two hypotheses test;
- Low-level (LL) classifier: inputs are all the voxels;
- High-level (HL) classifier: inputs are set of HLFs.

NFs	LL AUC	HL AUC
DS1 - γ	0.67	0.54
DS1 - π^+	0.67	0.63
DS2	0.76	0.70
DS3	0.84	0.71

CFMs	LL AUC	HL AUC
DS1 - γ	0.51	0.51
DS1 - π^+	0.52	0.63
DS2 (DREAM)	0.53	0.52
DS3 (DREAM)	0.63	0.52

Evaluation

Fidelity — Classifier test

How to evaluate the performance of gen. neural networks?

Train a classifier:

- LR optimal observable in simple two hypotheses test;
- Low-level (LL) classifier: inputs are all the voxels;
- High-level (HL) classifier: inputs are set of HLFs.

Extract a weight distribution: $w(x) = \frac{C(x)}{1 - C(x)} \approx \frac{p_{\text{G4}}}{p_{\text{model}}}(x)$

Calculate Area Under the Curve (AUC) score:

AUC = 0.5 $\Longrightarrow$ indistinguishable samples

NFs	LL AUC	HL AUC
DS1 - γ	0.67	0.54
DS1 - π^+	0.67	0.63
DS2	0.76	0.70
DS3	0.84	0.71

CFMs	LL AUC	HL AUC
DS1 - γ	0.51	0.51
DS1 - π^+	0.52	0.63
DS2 (DREAM)	0.53	0.52
DS3 (DREAM)	0.63	0.52

Evaluation

Generation speed

Many factors can affect the generation speed, here:

- generation time per shower;
- timed on one A100 GPU;
- batch size 100
- including overheads from moving data to GPU

Conditional Flow Matching:

- time for one step of Runge-Kutta 4 (4 n.f.e.);
- previous results use 20 steps with RK4 ODE solver.

	NFs [ms per shower]	CFMs [ms per shower]
	full gen.	1 step RK4
DS1 - γ	2	1
DS1 - π^+	2	2
DS2	15	2
DS3	60	10

Evaluation

Generation speed

Many factors can affect the generation speed, here:

- generation time per shower;
- timed on one A100 GPU;
- batch size 100
- including overheads from moving data to GPU

Conditional Flow Matching:

- time for one step of Runge-Kutta 4 (4 n.f.e.);
- previous results use 20 steps with RK4 ODE solver.

	NFs [ms per shower]	CFMs [ms per shower]
	full gen.	1 step RK4
DS1 - γ	2	1
DS1 - π^+	2	2
DS2	15	2
DS3	60	10

Conservative Geant4 shower time: 1-10 s/shower

(highly dependent on E_{inc})

$\Longrightarrow$ **much faster generation time!**

Bespoke samplers

Speeding up CFMs

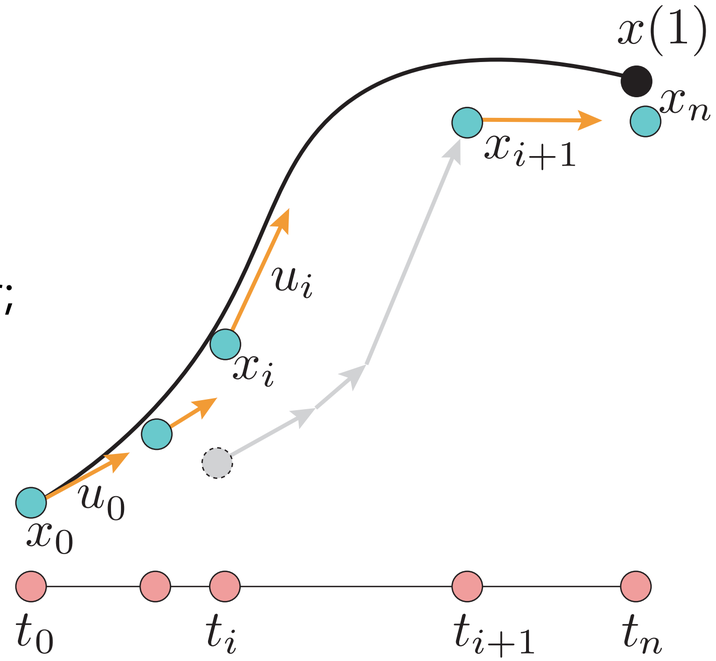
How to reduce the number of function evaluations?

- Bespoke samplers: keep the model fixed and learn a model-specific solver;
- use the general expression for a non-stationary solver:

$$(t_i, x_i)_{i=0}^{N}$$

$$x_{i+1} = a_i\, x_0 + b_i \cdot V_i$$

x_0 latent point
V_i vector fields
a_i, b_i, t_i learnable parameters



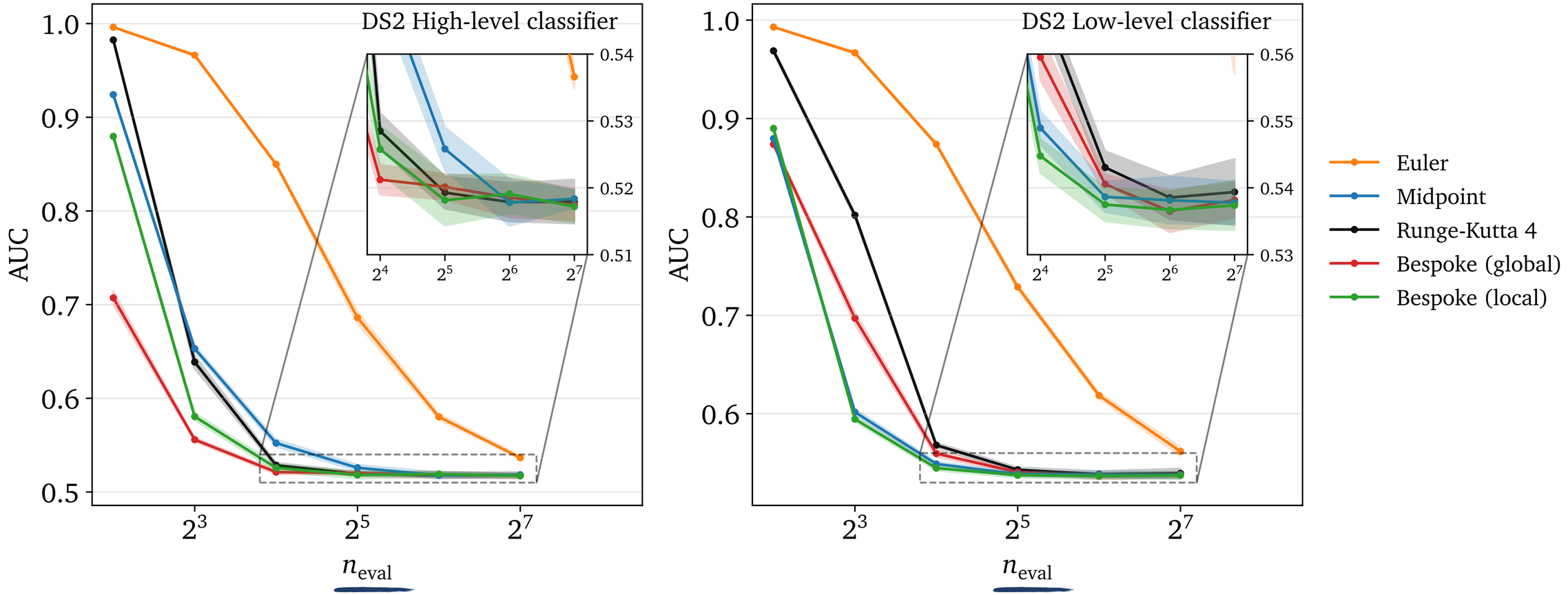
from Bespoke solvers, Shaul N. et al.

- minimize either the global or the local truncation error wrt. a reference solver:

$$\mathscr{L}_{GTE} = \langle [x_{ref}(1) - x_N]^2 \rangle_{x_0 \sim \mathcal{N}}\,, \qquad \mathscr{L}_{LTE} = \langle \sum_{i=0}^{N-1} \left[x_{ref}(t_{i+1}) - (a_i x_0 + b_i \cdot V_{ref,i}) \right]^2 \rangle_{x_0 \sim \mathcal{N}}$$

Bespoke samplers

Speeding up CFMs



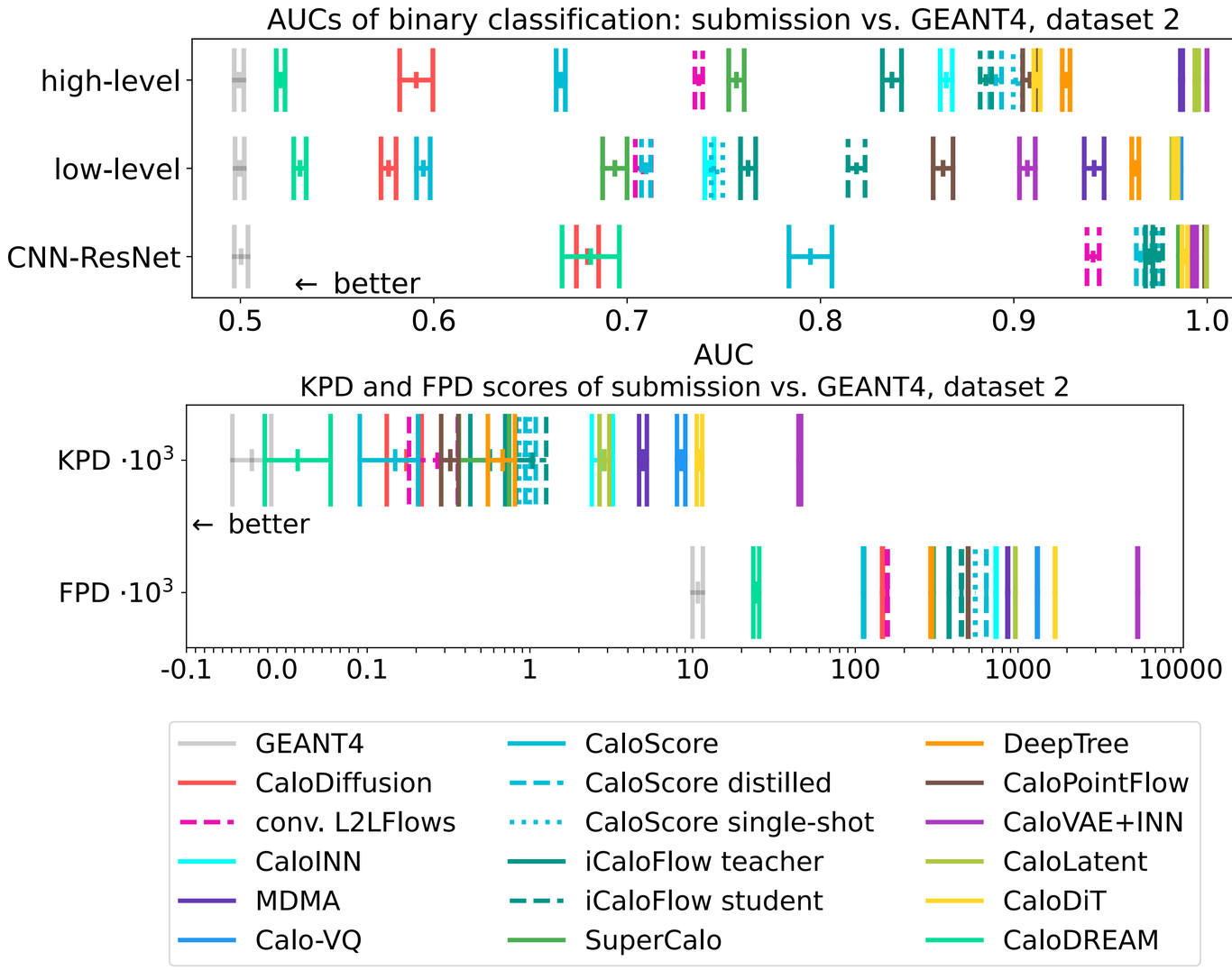
Outlook

CaloChallenge

Largest comparison of gen. networks:

- multi-class classifiers;
- other evaluation metrics, KPD/FPD;
- CPU/GPU generation times.

More in the writeup!
arXiv:2410.21611



Outlook

Vision Transformers — messages

Fast detector simulation:

- ViT can be used to train (continuous) normalizing flows;
- NFs are faster but less accurate;
- CFMs are more accurate but require multiple evaluations

  ⟶ Bespoke solvers can reduce the number of function evaluations;

- now need to go from simulated cells to downstream tasks…

Outlook

Vision Transformers — messages

Fast detector simulation:

- ViT can be used to train (continuous) normalizing flows;
- NFs are faster but less accurate;
- CFMs are more accurate but require multiple evaluations

  ⟶ Bespoke solvers can reduce the number of function evaluations;
- now need to go from simulated cells to downstream tasks…

Vision transformers:

- ViTs can handle irregular input data;
- scale well to high-dimensional spaces;
- interesting beyond detector simulation, check out Ayo's SKATr.

Outlook

Open-source code

- Stay tuned for an open-source repository;
- easily define patching functions…

  … and train generative networks;
- let me know your use cases!

```python
class BaseModel(nn.Module):
    def __init__(self, shape):
        super().__init__()

        self.shape = shape

    def from_patches(self, x):
        """
        Transform from input geometry to patches

        Parameters
        ----------
        x: torch.Tensor
            Input tensor of the form (batch_size, *dims)

        Returns
        -------
        output: torch.Tensor
            Output patched tensor with shape (batch_size, #patches, patch_dim)
        """
        pass

    def to_patches(self, x):
        """
        Transform from patches back to original geometry

        Parameters
        ----------
        x: torch.Tensor
            Input tensor of the form (batch_size, num_patches, patch_dim)

        Returns
        -------
        output: torch.Tensor
            Output tensor with shape (batch_size, *dims)
        """
        pass

    def forward(self, x, c, rev=False, jac=True):
        """
        Simple forward pass
```

EUROPEAN AI FOR FUNDAMENTAL PHYSICS CONFERENCE EuCAIFCon 2025

Fast, accurate, and precise detector simulation with vision transformers

CaloDREAM

Coming soon

GitHub repo

Fast, accurate, and precise detector simulation with vision transformers

Thank you for your attention!

Coming soon

CaloDREAM

GitHub repo

EUROPEAN AI FOR FUNDAMENTAL PHYSICS CONFERENCE
EuCAIFCon 2025

Backup

Conditional Flow Matching

Promote the discrete transformation to a continuous one:

$$\frac{dx(t)}{dt} = v(x(t), t) \quad \text{with} \quad x \in \mathbb{R}^d \qquad\qquad \frac{\partial p(x,t)}{\partial t} + \nabla_x \left[p(x,t) v(x,t) \right] = 0 \ .$$

We want to impose the boundary conditions for $p(x, t)$:

$$p(x,t) \to \begin{cases} \mathcal{N}(x; 0, 1) & t \to 1 \\ p_{data}(x) & t \to 0 \ . \end{cases}$$

Need to define the training trajectories

$\longrightarrow$ linear, simplest choice

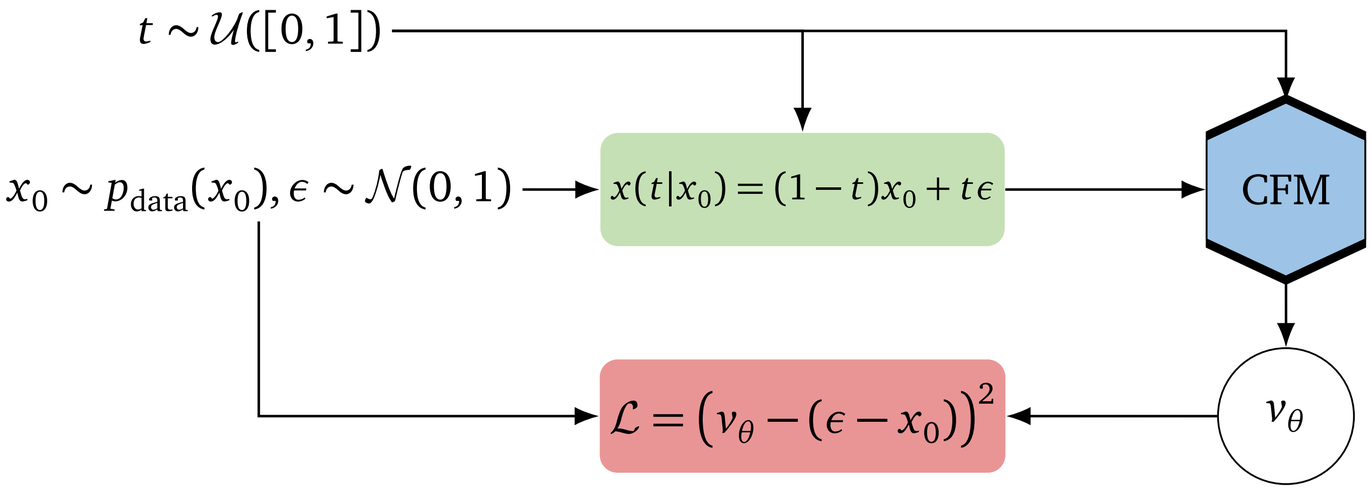
$$x(t \,|\, x_0) = (1-t)x_0 + t\epsilon \qquad \epsilon \sim \mathcal{N}(0,1)$$

Learn this velocity field with a NN:

$$\mathcal{L} = || \, v(x,t) - v_\phi(x,t) \, ||_{L_2}$$

Conditional Flow Matching

Jet diffusion versus JetGPT



$$\mathscr{L}_{\text{CFM}} = \left\langle \left[v_\phi((1-t)x_0 + t\epsilon, t) - (\epsilon - x_0) \right]^2 \right\rangle_{U(0,1),\mathcal{N},p_{data}} .$$

Sampling $\longrightarrow$ solve the differential equation numerically: $x(t=0) = x(t=1) - \int_0^1 v_\phi(x,t)dt$

Learning normalised showers

Learning normalised showers

normalized showers

$$u_0 = \frac{\sum_i E_i}{f\,E_{inc}} \qquad \text{and} \qquad u_i = \frac{E_i}{\sum_{j\geq i} E_j}\,,$$

logit

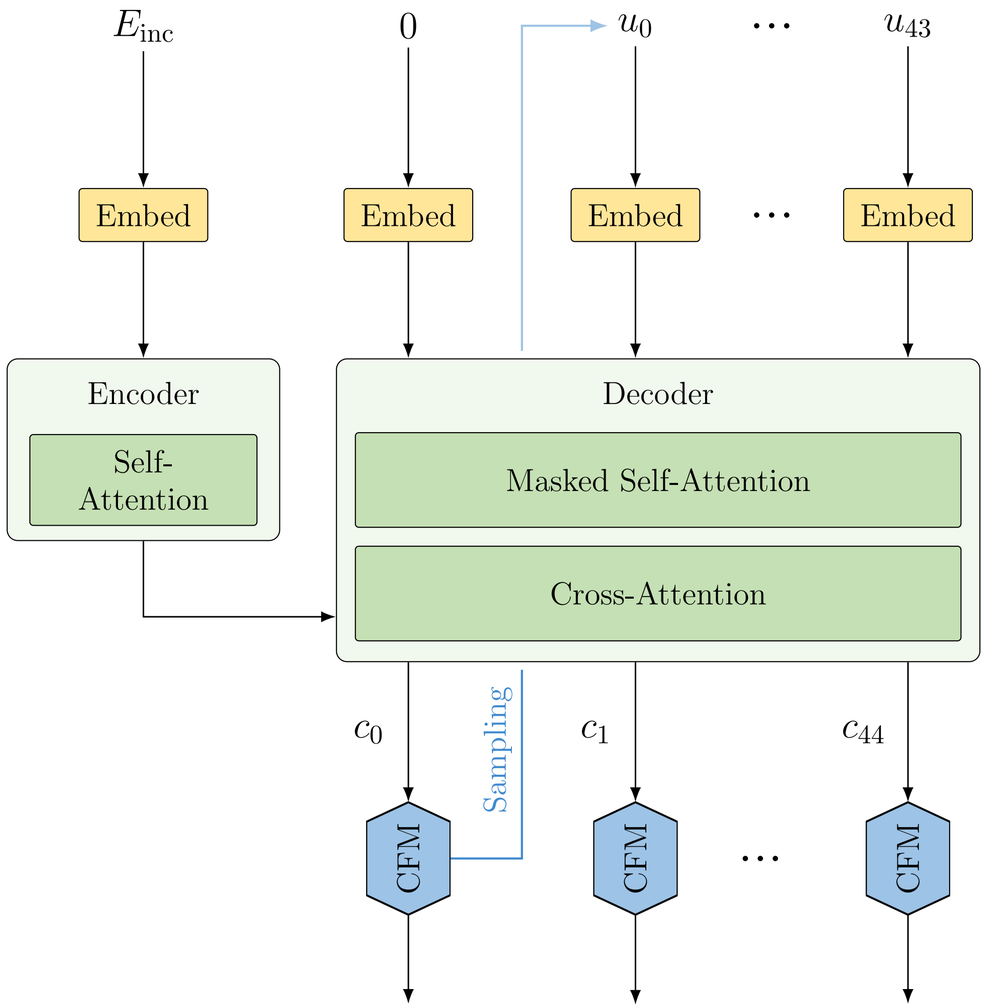
$$x_\alpha = (1-2\alpha)x + \alpha \in [\alpha, 1-\alpha] \qquad \text{with} \quad \alpha = 10^{-6}$$

$$x' = \log\frac{x_\alpha}{1-x_\alpha}\,.$$

Factorise the problem into:

- learn the energy distribution, $p(u\,|\,E_{inc})$
- learn the normalised voxels $p(x\,|\,u, E_{inc})$

Learning layer energies

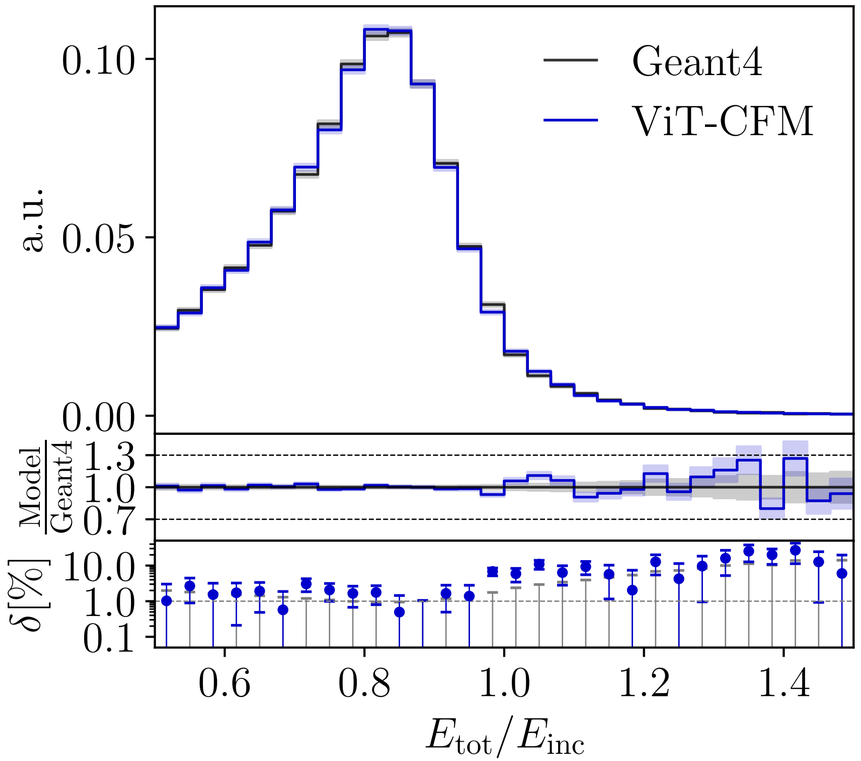


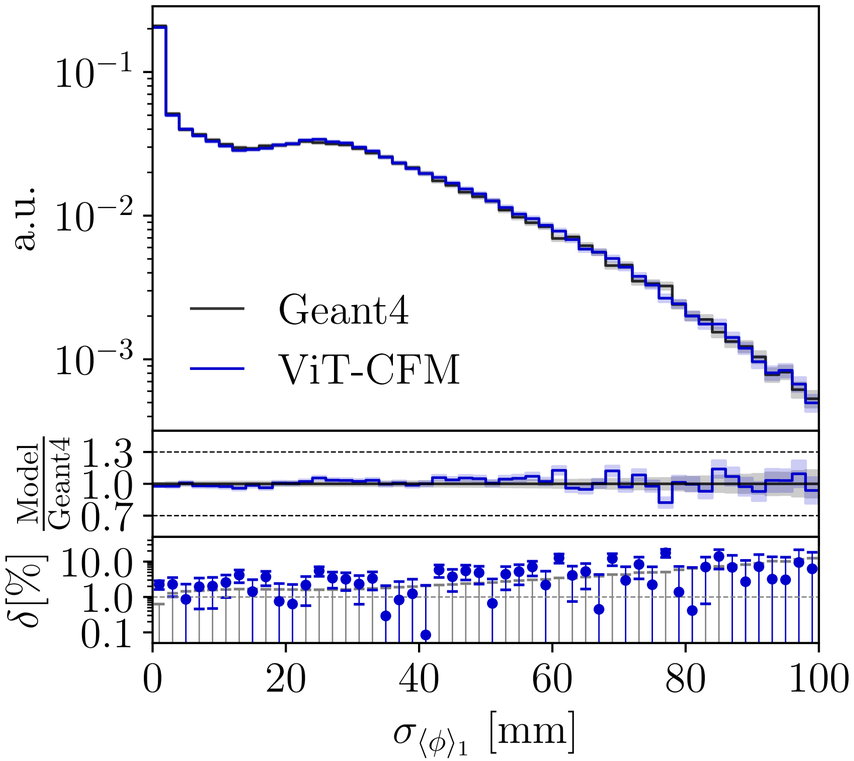
$$\nu_{\text{full}}(t, u; E_{\text{inc}}) = \Big(\nu_\theta(t, u_0; c_0), \quad \nu_\theta(t, u_1; c_1), \cdots, \nu_\theta(t, u_{44}; c_{44}) \Big)$$

High-Level Features

Hadronic showers

Dataset 1 — π^+





Weight distributions