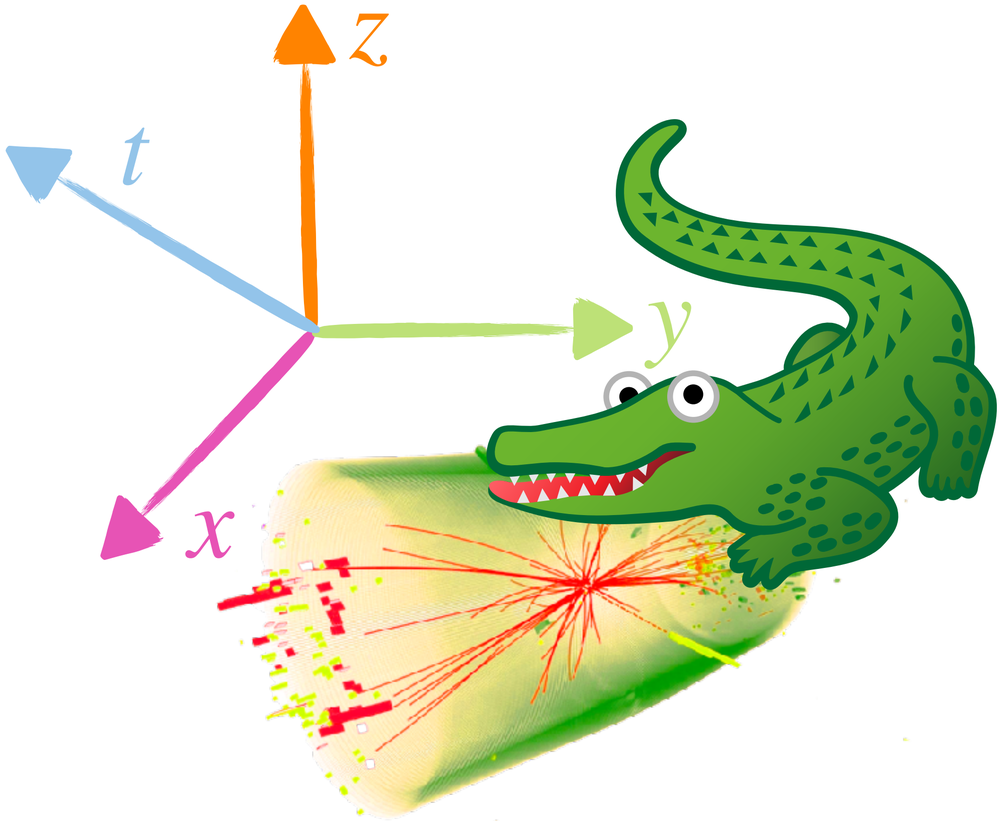
Transformers and Symmetries



Jonas Spinner

in collaboration with Johann Brehmer, Víctor Bresó, Luigi Favaro, Gerrit Gerhartz, Pim de Haan, Fred A. Hamprecht, Peter Lippmann, Sebastian Pitz, Tilman Plehn, Huilin Qu and Jesse Thaler

Build Big or Build Smart Workshop
05.09.2025

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Symmetries in LHC physics

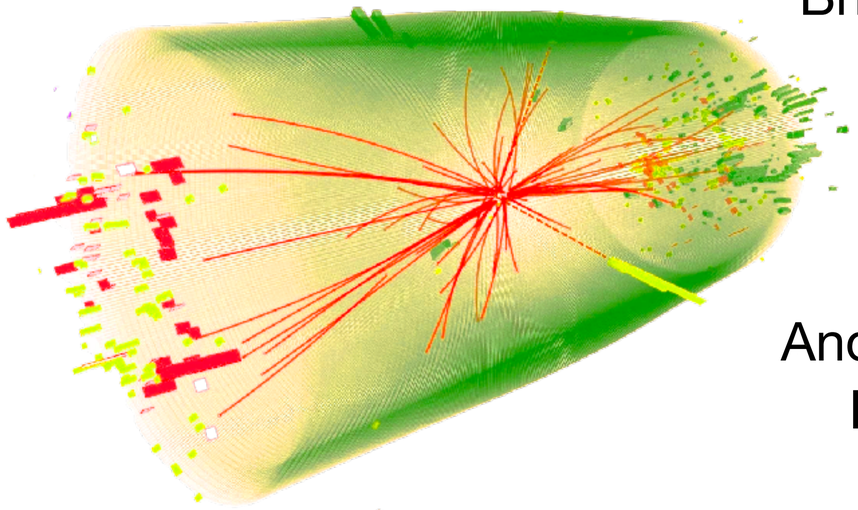
Permutation symmetry

Rotations and translations in (η, ϕ)

Dillon et al, arXiv:2108.04253

Favaro et al, arXiv:2312.03067

Bhardwaj et al, arXiv:2402.12449



Lorentz symmetry

Gong et al, arXiv:2201.08187

Bogatskiy et al, arXiv:2211.00454

Hao et al, arXiv:2212.07347

Ruhe et al, arXiv:2305.11141

Spinner et al, arXiv:2405.14806

Spinner et al, arXiv:2505.20280

Universal splitting kernels

Andreassen et al, arXiv:1804.09720

Butter et al, arXiv:2412.12074

Invariance under re-simulation

Harris et al, arXiv:1804.09720

More…

Permutation symmetry

Message-passing Graph Networks

✓ Formally more expressive

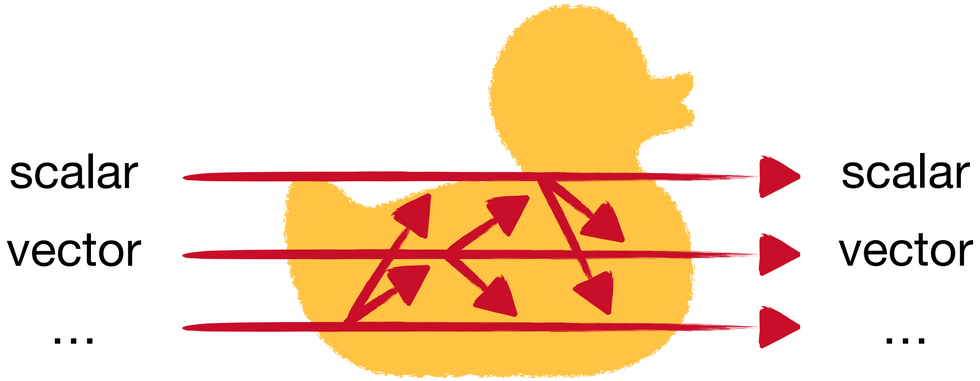
vs

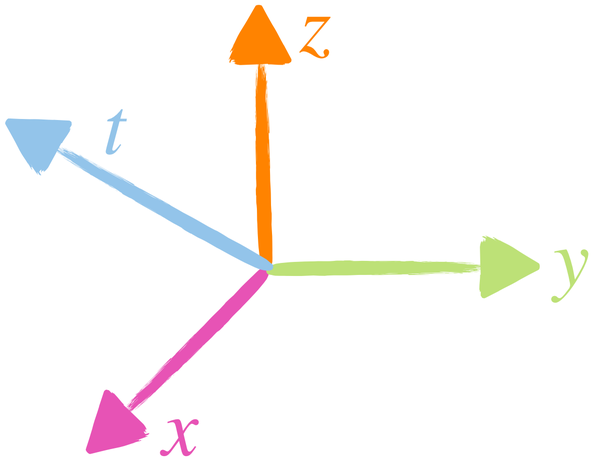
Transformers

✓ Better at scale

Lorentz symmetry

Specialised layers vs Canonicalization





PELICAN

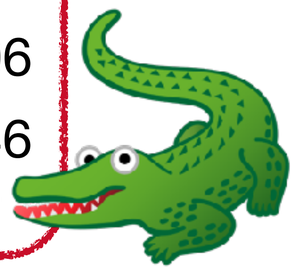
arXiv:2211.00454
arXiv:2307.16506

LorentzNet

arXiv:2201.08187

L-GATr

arXiv:2405.14806
arXiv:2411.00446
pip install lgatr

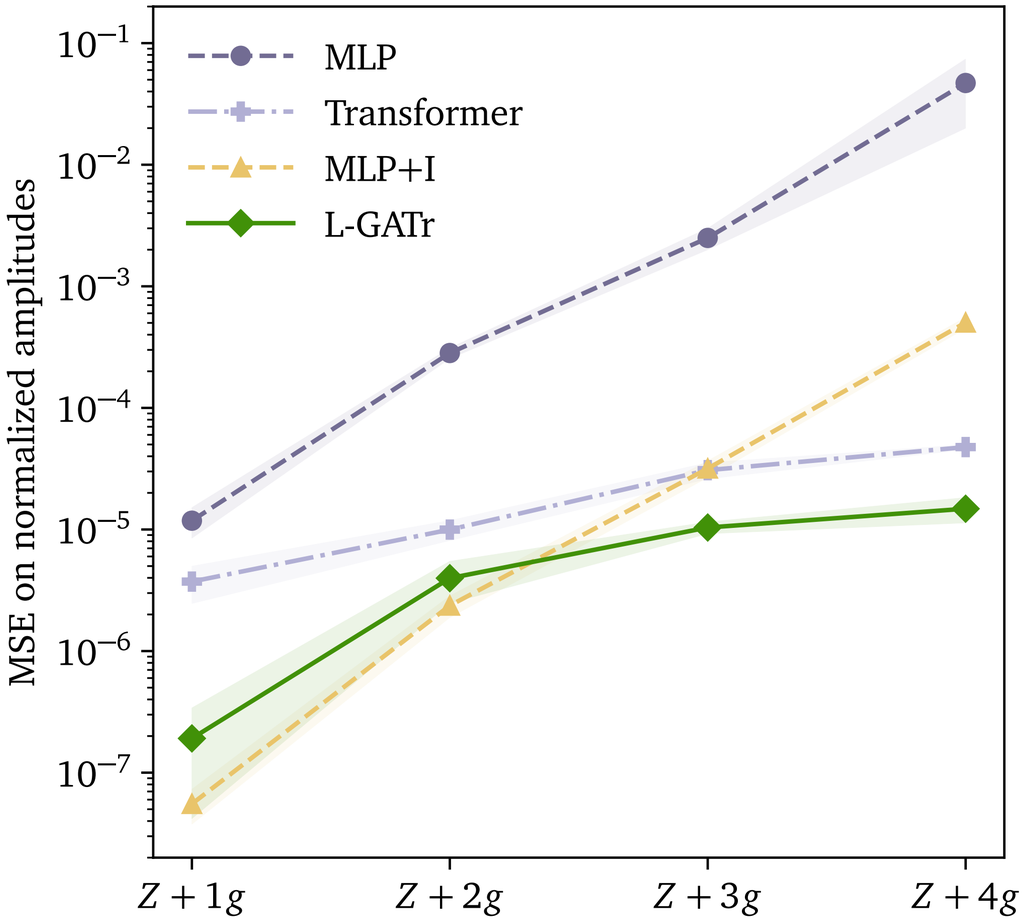


LLoCa

arXiv:2505.20280
arXiv:2508.14898

Amplitude regression

Gain from Lorentz and Permutation symmetry



Naive

$$\begin{pmatrix} E_1 \\ p_{x,1} \\ p_{y,1} \\ p_{z,1} \\ E_2 \\ p_{x,2} \\ p_{y,2} \\ p_{z,2} \\ \vdots \end{pmatrix}$$

MLP

Permutation symmetry

$$\begin{pmatrix} E_1 \\ p_{x,1} \\ p_{y,1} \\ p_{z,1} \\ \mathrm{PID}_1 \end{pmatrix} \quad \begin{pmatrix} E_2 \\ p_{x,2} \\ p_{y,2} \\ p_{z,2} \\ \mathrm{PID}_2 \end{pmatrix} \quad \vdots$$

Transformer

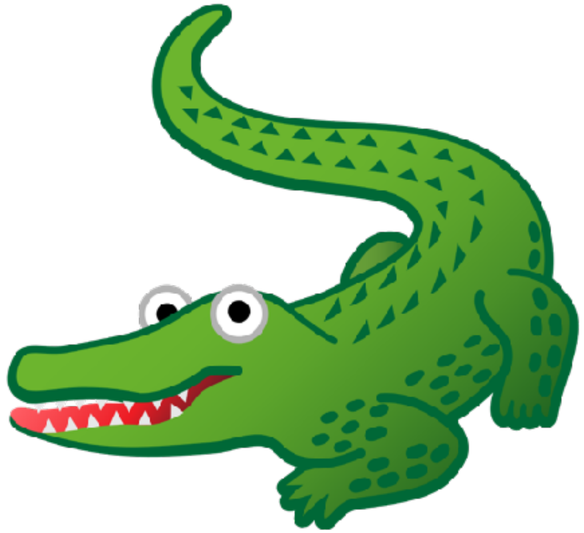
Lorentz invariants

$$\begin{pmatrix} p_1^\mu p_{1,\mu} \\ p_1^\mu p_{2,\mu} \\ p_1^\mu p_{3,\mu} \\ \vdots \\ p_2^\mu p_{2,\mu} \\ p_2^\mu p_{3,\mu} \\ \vdots \end{pmatrix}$$

MLP+I

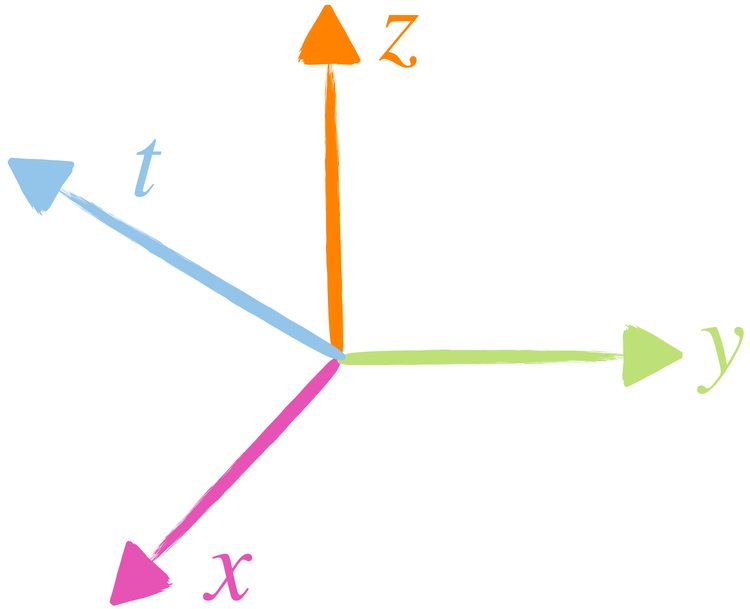
Permutation + Lorentz

$$\begin{pmatrix} p_1^\mu \\ \mathrm{PID}_1 \end{pmatrix} \quad \begin{pmatrix} p_2^\mu \\ \mathrm{PID}_1 \end{pmatrix} \quad \vdots$$



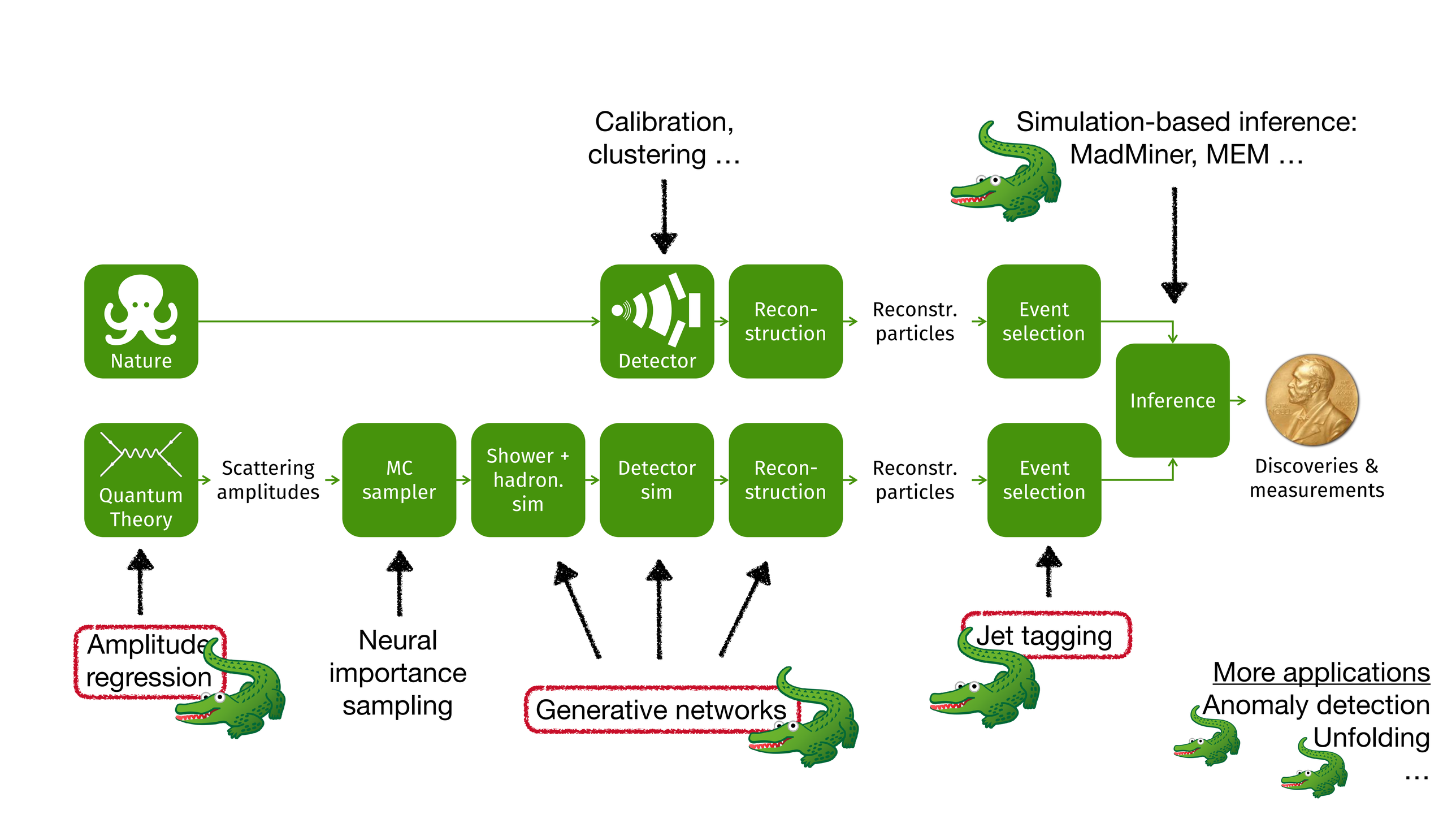
Lorentz-Equivariant
Geometric **A**lgebra
Transformer

- ✓ `pip install lgatr`
- ✓ Easy to use



Lorentz **Lo**cal
Canonicalization

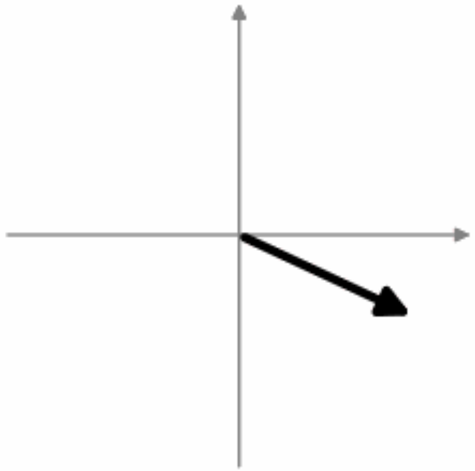
- ✓ Resource efficient
- ✓ Flexible


Calibration,
clustering …
Simulation-based inference:
MadMiner, MEM …
Nature
Detector
Recon-
struction
Reconstr.
particles
Event
selection
Inference
Discoveries &
measurements
Quantum
Theory
Scattering
amplitudes
MC
sampler
Shower +
hadron.
sim
Detector
sim
Recon-
struction
Reconstr.
particles
Event
selection
Amplitude
regression
Neural
importance
sampling
Generative networks
Jet tagging
More applications
Anomaly detection
Unfolding
…

Symmetries

Invariance

Input



Output
(not invariant)

0.8 ✗

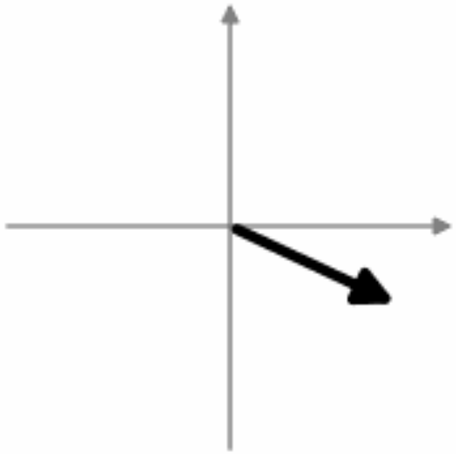
Output
(invariant)

0.8 ✓

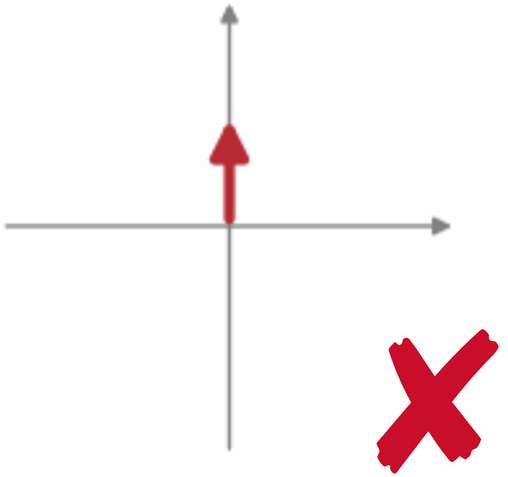
Symmetries

Equivariance = Covariance

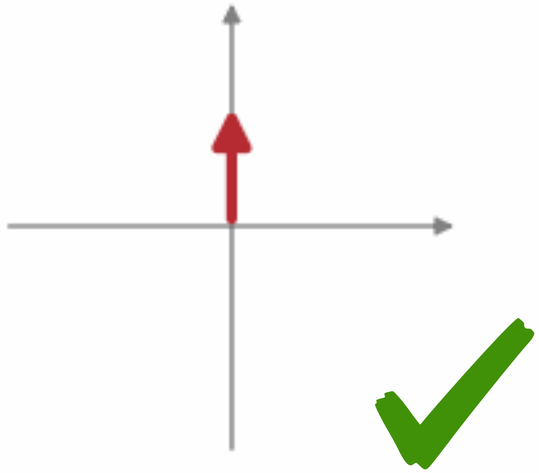
Input



Output
(not equivariant)



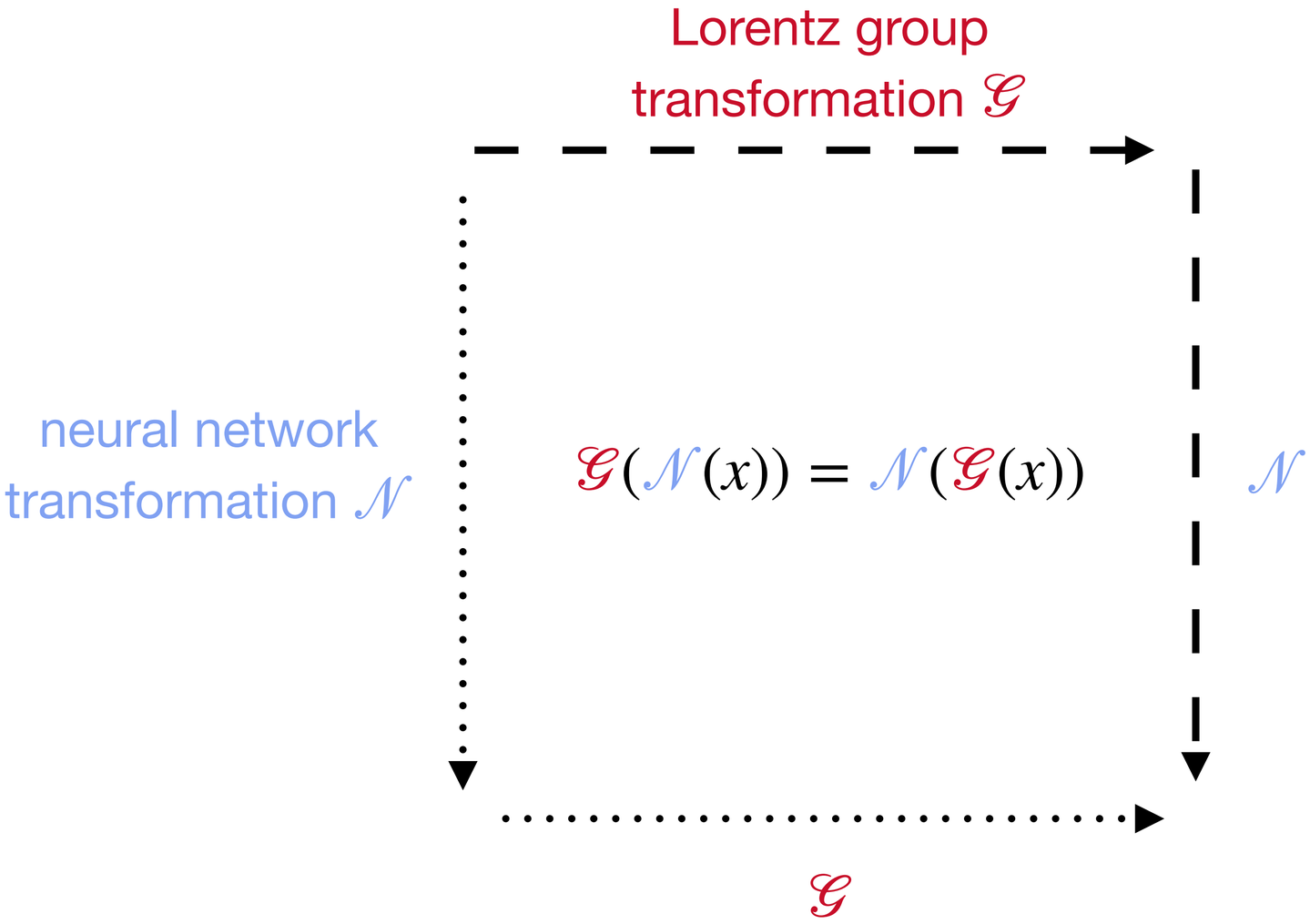
Output
(equivariant)

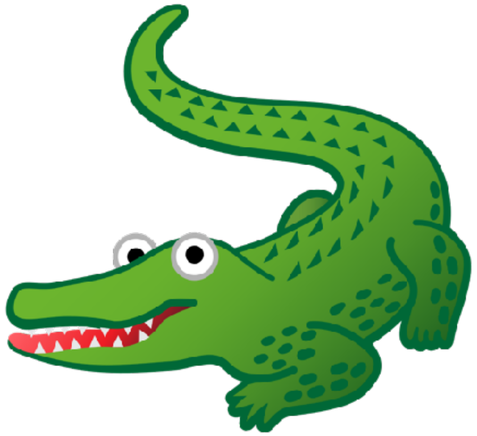


Invariance = Equivariance with scalar output

Symmetries

Equivariance - formalised





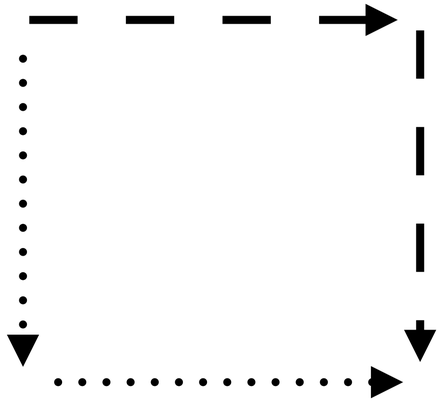
Lorentz-Equivariant
Geometric **A**lgebra
Transformer

=

$$\begin{pmatrix} 1 \\ \gamma^{\mu} \\ \sigma^{\mu\nu} \\ \gamma^{\mu}\gamma_5 \\ \gamma_5 \end{pmatrix}$$

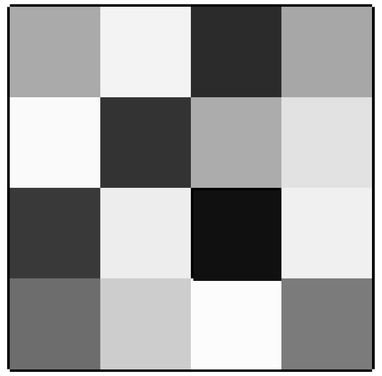
Geometric algebra
representations

+



Lorentz-Equivariant
layers

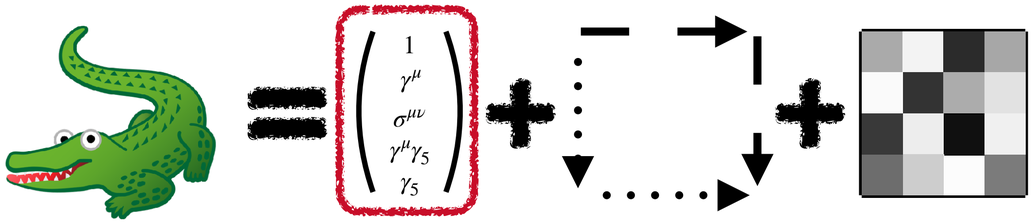
+



Transformer
architecture

GATr was originally
developed for E(3)
arXiv:2305.18415

L-GATr



Geometric algebra = Clifford algebra

Geometric algebra

= Vector space + geometric product $xy = \frac{\{x,y\}}{2} + \frac{[x,y]}{2}$

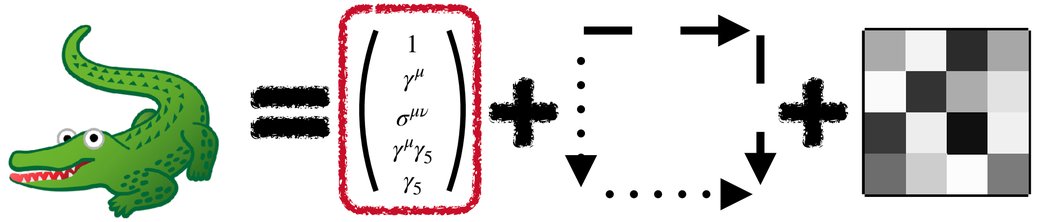
Spacetime geometric algebra

= Minkowski space + $\{\gamma^\mu, \gamma^\nu\} = 2g^{\mu\nu}1$ over $\mathbb{R}$

≠ Dirac algebra (uses $\mathbb{C}$)

L-GATr

Building the algebra



$$\{\gamma^\mu, \gamma^\nu\} = 2g^{\mu\nu}1$$

Linear combinations
+ Geometric product

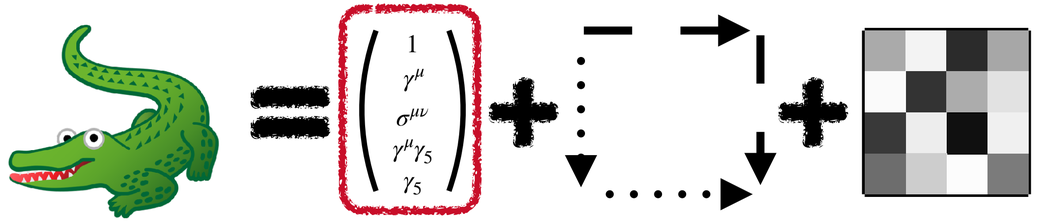
$$x = x^S 1 + x^V_\mu \gamma^\mu + x^B_{\mu\nu}\sigma^{\mu\nu} + x^A_\mu \gamma^\mu\gamma^5 + x^P\gamma^5$$

Multivector representation: $(x^S, x^V_\mu, x^B_{\mu\nu}, x^A_\mu, x^P) \in \mathbb{R}^{16}$

Particle: $x^V = (E, p_x, p_y, p_z), \quad x^S = \text{PID}, \quad x^{\text{others}} = 0$

L-GATr

Geometric algebra representations



$$x^S \in \mathbb{R}$$

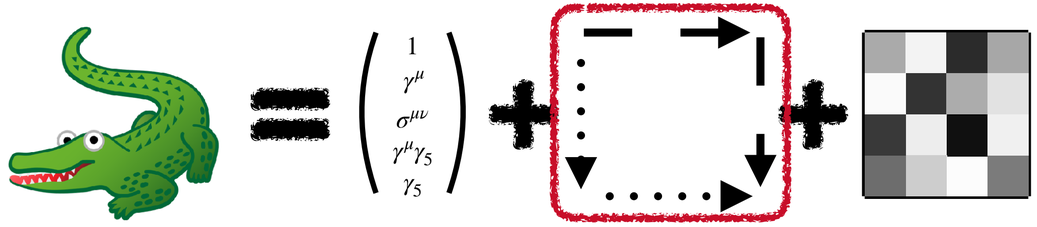
$$\begin{pmatrix} x^S \\ x^V_\mu \\ x^B_{\mu\nu} \\ x^A_\mu \\ x^P \end{pmatrix} \in \mathbb{R}^{16}$$

scalar channels

multivector channels

L-GATr

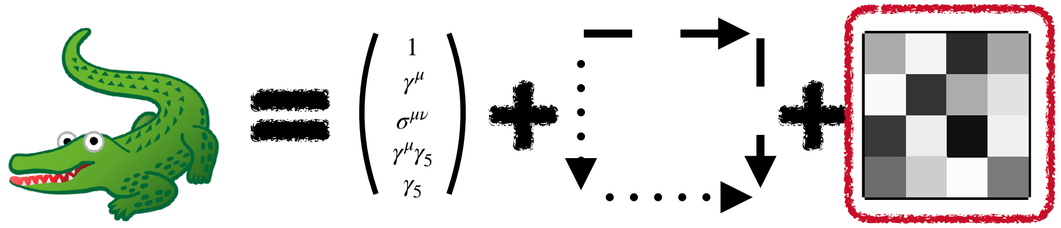
L-GATr-ing all transformer layers



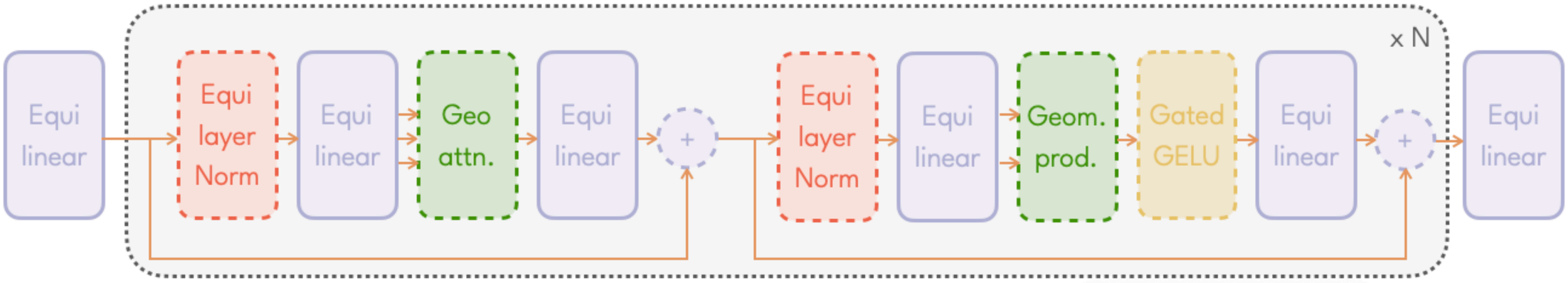
Layer type	Transformer	L-GATr
Linear(x)	$vx + w$	$\sum_{k=0}^{4} v_k \langle x \rangle_k \left(+ \sum_{k=0}^{4} w_k \gamma^5 \langle x \rangle_k \right)$
Attention$(q, k, v)_{ic}$	$\sum_{j=1}^{n_t} \mathrm{Softmax}_j \left(\sum_{c'=1}^{n_c} \frac{q_{ic'} k_{jc'}}{\sqrt{n_c}} \right) v_{jc}$	$\sum_{j=1}^{n_t} \mathrm{Softmax}_j \left(\sum_{c'=1}^{n_c} \frac{\langle q_{ic'}, k_{jc'} \rangle}{\sqrt{16 n_c}} \right) v_{jc}$
LayerNorm(x)	$x \left[\frac{1}{n_c} \sum_{c=1}^{n_c} x_c^2 + \epsilon \right]^{-1/2}$	$\mathbf{x} \left[\frac{1}{n_c} \sum_{c=1}^{n_c} \sum_{k=0}^{4} \left\lvert \left\langle \langle x_c \rangle_k, \langle x_c \rangle_k \right\rangle \right\rvert + \epsilon \right]^{-1/2}$
Activation(x)	GELU(x)	GELU$(\langle x \rangle_0) x$
GP(x, y)	—	xy

L-GATr

Full architecture



L-GATr blocks
can be stacked
to large depth



Equivariant **Self-Attention**

Equivariant **MLP**

Quickstart

This page sets you up for building and running L-GATr models.

Installation

Before using the package, install it via pip:

```
pip install lgatr
```

Alternatively, if you're developing locally:

```
git clone https://github.com/heidelberg-hepml/lgatr.git
cd lgatr
pip install -e .
```

Building L-GATr

You can construct a simple `LGATr` model as follows:

```
from lgatr import LGATr

attention = dict(num_heads=2)
mlp = dict()
lgatr = LGATr(
    in_mv_channels=1,
    out_mv_channels=1,
    hidden_mv_channels=8,
    in_s_channels=0,
    out_s_channels=0,
    hidden_s_channels=16,
    attention=attention,
    mlp=mlp,
    num_blocks=2,
)
```

```
from lgatr.interface import embed_vector, extract_scalar
multivector = embed_vector(p)
print(multivector.shape) # torch.Size([128, 20, 1, 16])
```

```
output_mv, output_s = lgatr(multivectors=multivector, scalars=None)
out = extract_scalar(output_mv)
print(out.shape) # torch.Size([128, 20, 1, 1])
```

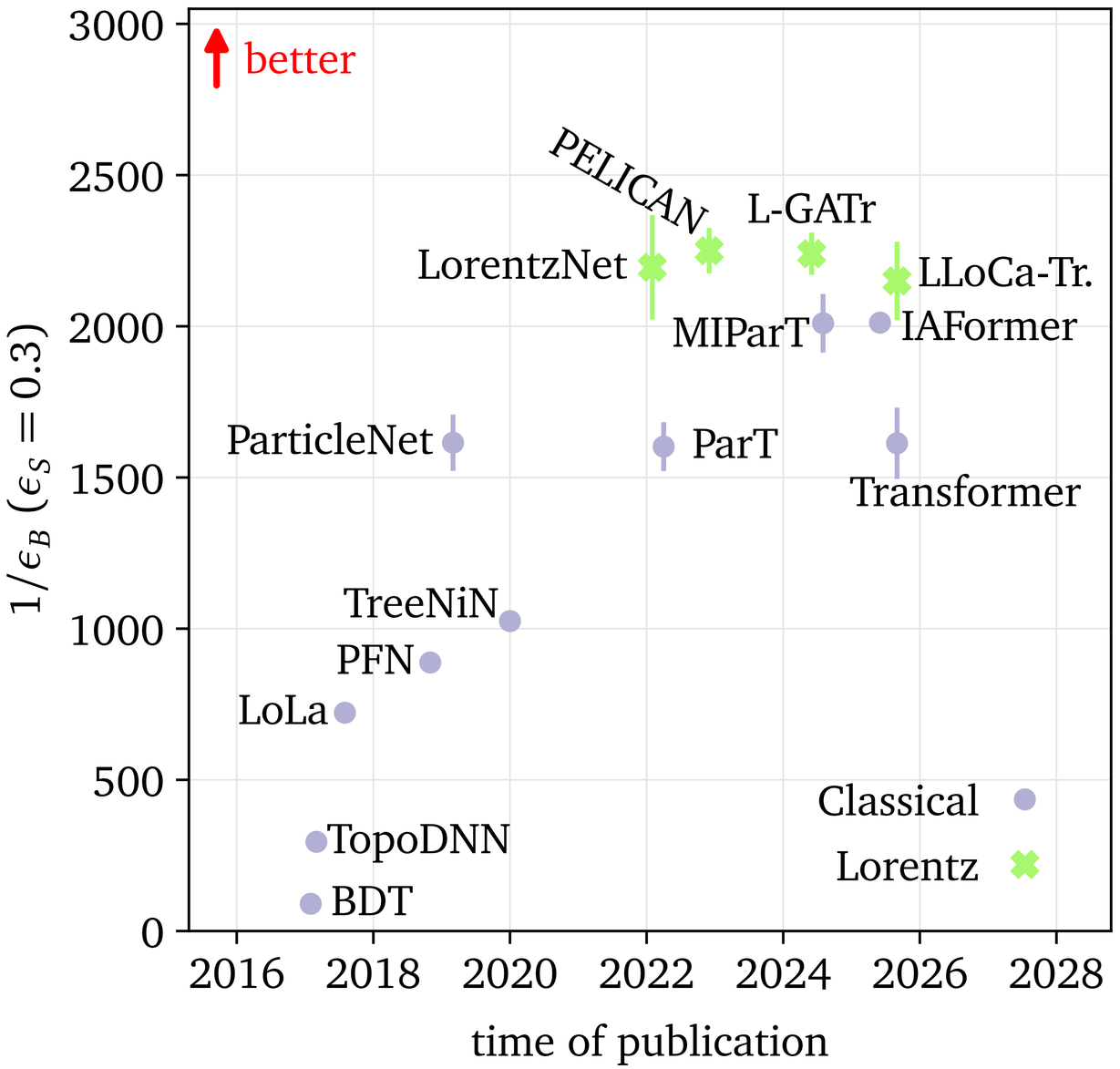
heidelberg-hepml.github.io/lgatr/

Jet Tagging

The history of top tagging

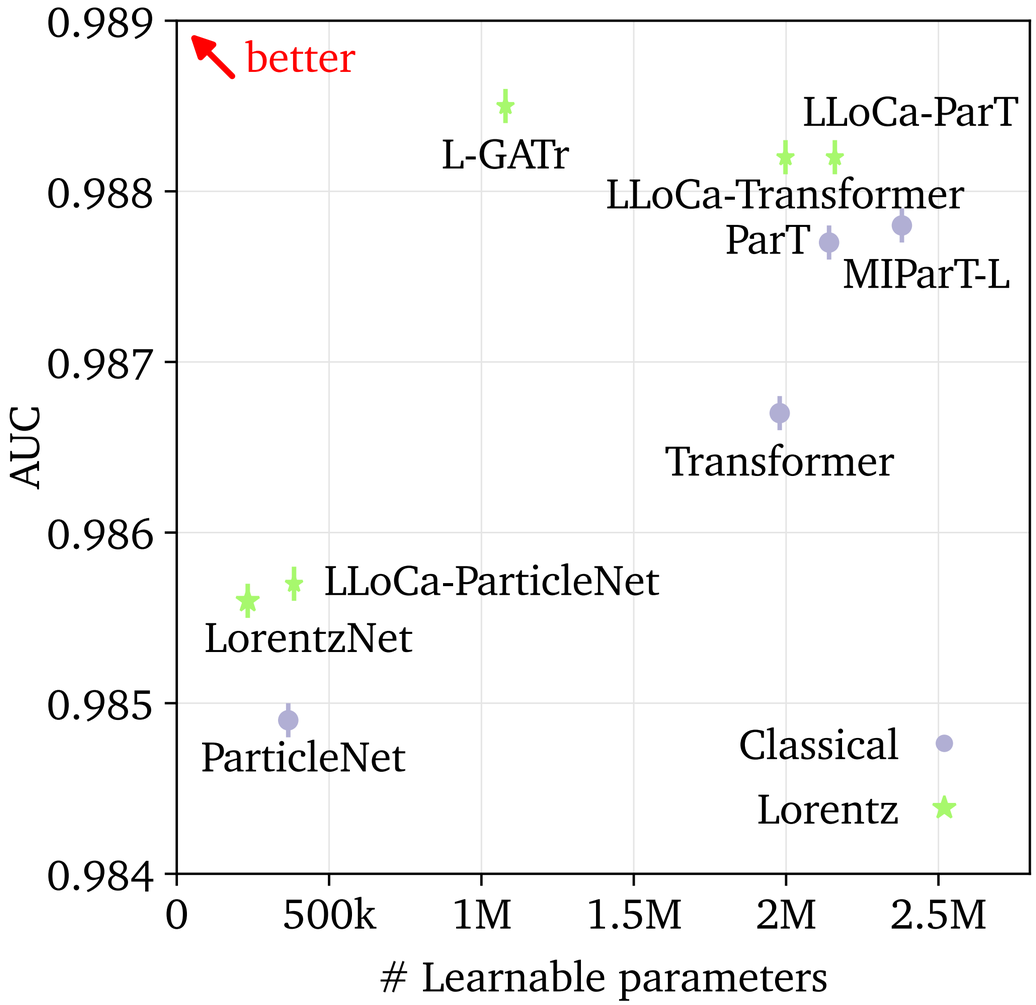
All top-performing taggers
are Lorentz-equivariant

Graph networks and transformers
perform similarly well



Jet Tagging

Training on 100M jets



Transformers outscale graph networks on large datasets

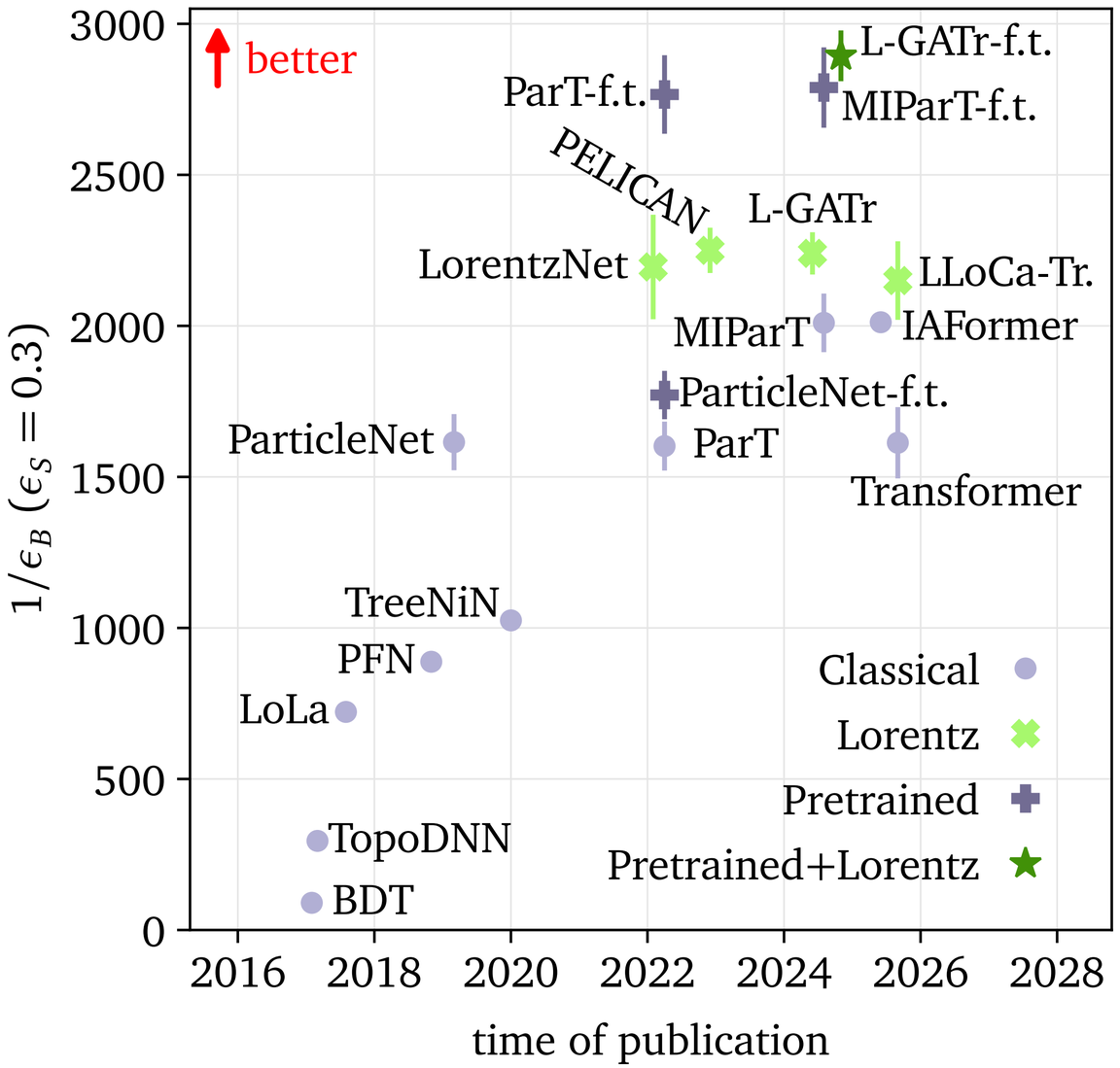
Lorentz-equivariance improves performance also at scale

Jet Tagging

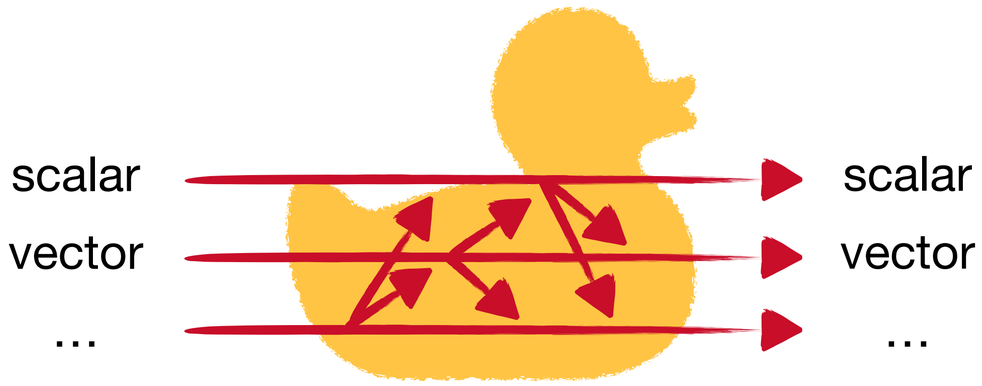
Transfer learning

L-GATr-f.t. combines the benefits of pretraining and Lorentz equivariance

You can go for big + smart!



Beyond specialised layers



Efficient Lorentz-equivariance?

How to make my **domain-specific** architecture Lorentz-equivariant?

What about **higher-order** internal representations?

PELICAN

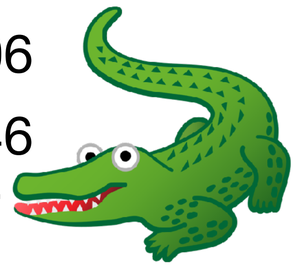
arXiv:2211.00454
arXiv:2307.16506

LorentzNet

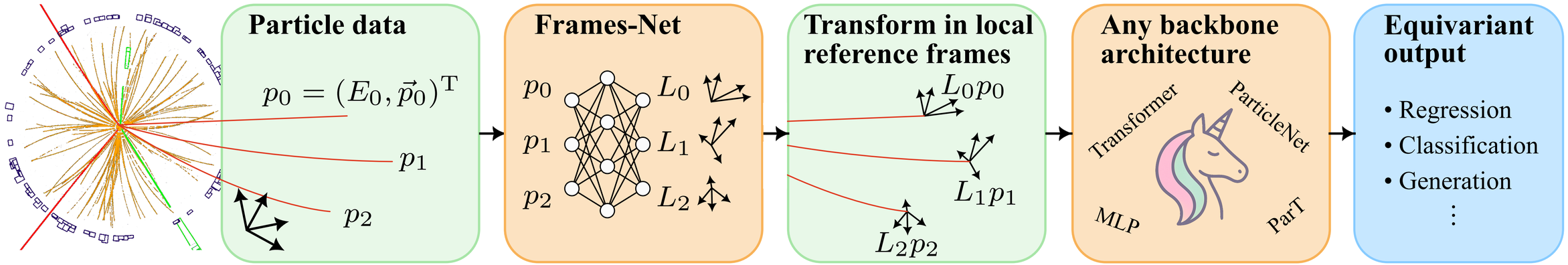
arXiv:2201.08187

L-GATr

arXiv:2405.14806
arXiv:2411.00446
pip install lgatr



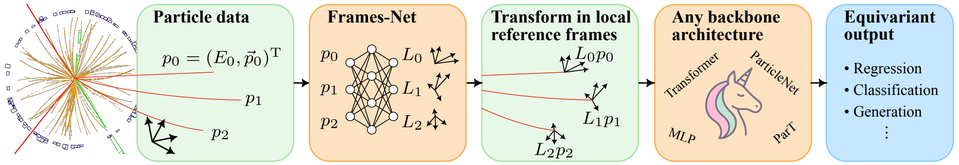
How to make any network Lorentz-equivariant:



Lorentz Local Canonicalization

LLoCa

Local frames

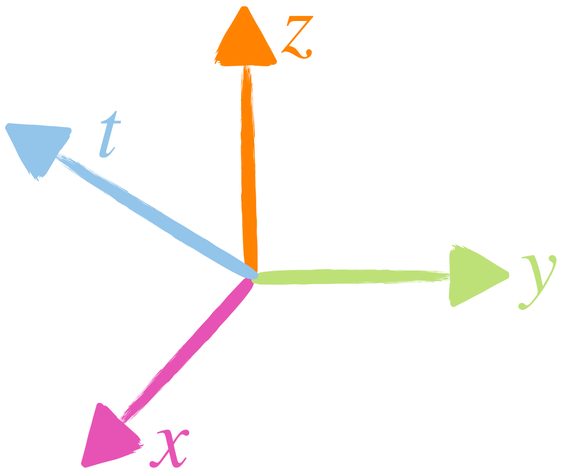


Features $x_L = Lx$ in local frame are **invariant** by definition:

$$x_L = Lx \xrightarrow{\Lambda} L'x' = L'\Lambda x \overset{!}{=} x_L \qquad \Rightarrow \qquad L' = L\Lambda^{-1}$$

Achieve this by stacking **Lorentz vectors** $u_a{}^\mu$, $a = 0,1,2,3$:

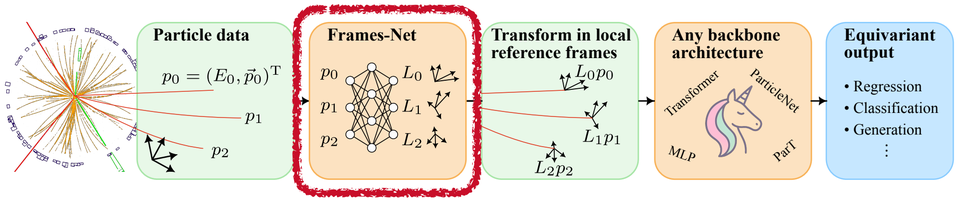
$$L = \begin{pmatrix} — \; u_0^T g \; — \\ — \; u_1^T g \; — \\ — \; u_2^T g \; — \\ — \; u_3^T g \; — \end{pmatrix} \xrightarrow{\Lambda} L' = \begin{pmatrix} — \; u_0^T \Lambda^T g \; — \\ — \; u_1^T \Lambda^T g \; — \\ — \; u_2^T \Lambda^T g \; — \\ — \; u_3^T \Lambda^T g \; — \end{pmatrix} = L\Lambda^{-1}$$



$$\sum_{\mu,\nu} u_a{}^\mu u_b{}^\nu g_{\mu\nu} = g_{ab}$$

LLoCa

How to construct local frames



1) Predict 3 vectors $v_{i,0}, v_{i,1}, v_{i,2}$ with simple **Lorentz-equivariant network**

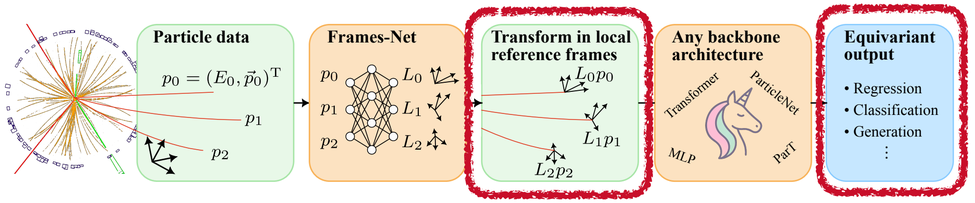
$$v_{i,k} = \sum_{j=1}^{N} \text{softmax}\Big(\varphi_k(s_i, s_j, \langle p_i, p_j \rangle)\Big)(p_i + p_j) \qquad \text{for } k = 0,1,2.$$

Frames-Net (MLP)

2) Orthonormalize $v_{i,0}, v_{i,1}, v_{i,2}$ with **Gram-Schmidt algorithm** to obtain $u_{i,a}$

LLoCa

Equivariance from canonicalisation



Latent features in the local frame $f_L = \rho(L)f$ are **invariant**

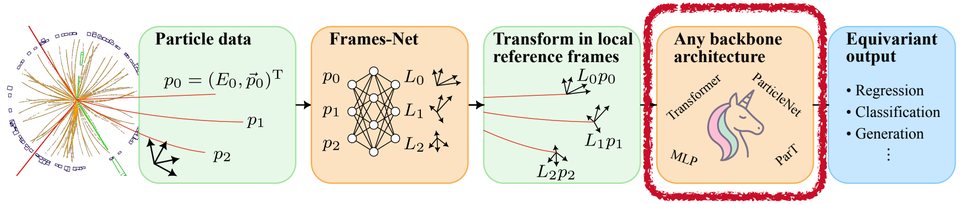
$$f_L \xrightarrow{\Lambda} f'_L = \rho(L')f' = \rho(L\Lambda^{-1})\rho(\Lambda)f = \rho(L\Lambda^{-1}\Lambda)f = \rho(L)f = f_L$$

Output features in the global frame $y = \rho(L^{-1})y_L$ are **equivariant**

$$y \xrightarrow{\Lambda} y' = \rho(L'^{-1})y'_L = \rho(\Lambda L^{-1})y_L = \rho(\Lambda)\rho(L^{-1})y_L = \rho(\Lambda)y$$

LLoCa

Message-passing between local frames



Standard message-passing → **Tensorial message-passing**
between different local frames

$$f_i^{\text{updated}} = \psi\left(f_i, \bigoplus_{j=1}^{N} \phi(m_j)\right)$$

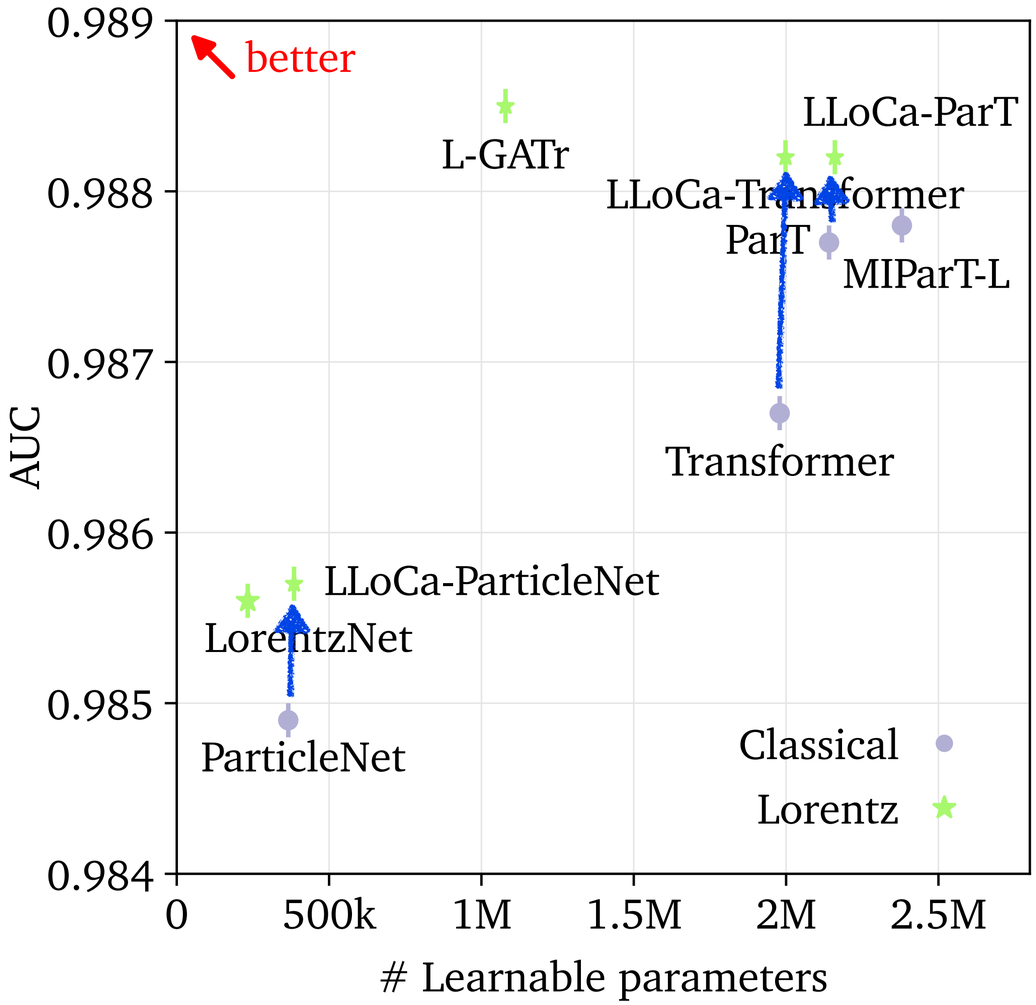
$$f_{i,L_i}^{\text{updated}} = \psi\left(f_{i,L_i}, \bigoplus_{j=1}^{N} \phi\left(\rho_m(L_i L_j^{-1}) m_{j,L_j}\right)\right)$$

Message representation ρ_m is a hyperparameter

Invariant frame transformation matrix $L_i L_j^{-1}$

Jet Tagging

Make any network Lorentz-equivariant

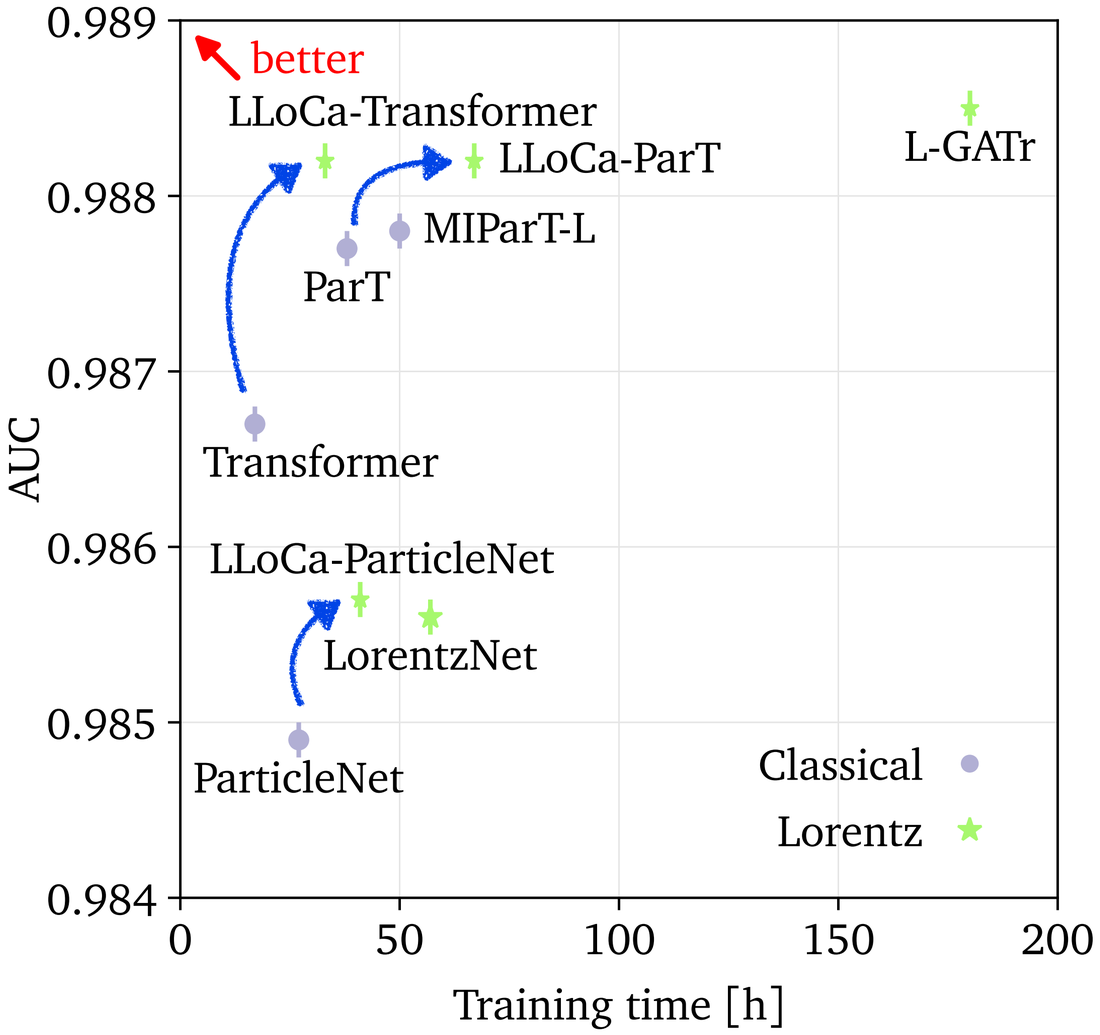


LLoCa upgrades established taggers to be Lorentz-equivariant

LLoCa-ParT is not better than LLoCa-Transformer

Jet Tagging

Efficient Lorentz-equivariance

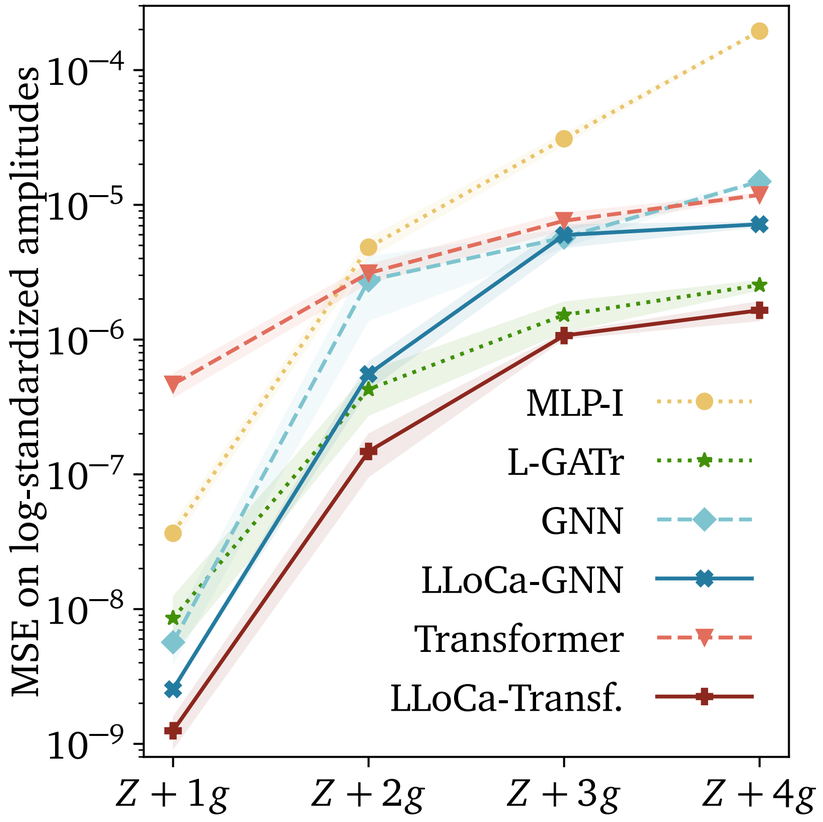


LLoCa costs only ~2x extra training time

LLoCa-Transformer is better and faster than Part

Amplitude regression

Efficient Lorentz-equivariance



Model	MSE$\times 10^{-6}$ (↓)	FLOPs	Time
MLP-I [38]	195.0 ± 2	0.1M	0.6h
LGATr [38]	2.5 ± 0.2	1160.0M	19.5h
GNN	14.9 ± 0.3	9.9M	1.3h
LLoCa-GNN	7.2 ± 0.3	10.6M	2.3h
Transformer	11.9 ± 0.5	10.6M	3.5h
LLoCa-Transformer	**1.5** ± 0.1	11.3M	4.5h

L-GATr and LLoCa-Transformer achieve similar performance

… but LLoCa-Transformer is 4x faster and requires 100x less FLOPs!

Amplitude regression

Arbitrary internal representations ρ_m

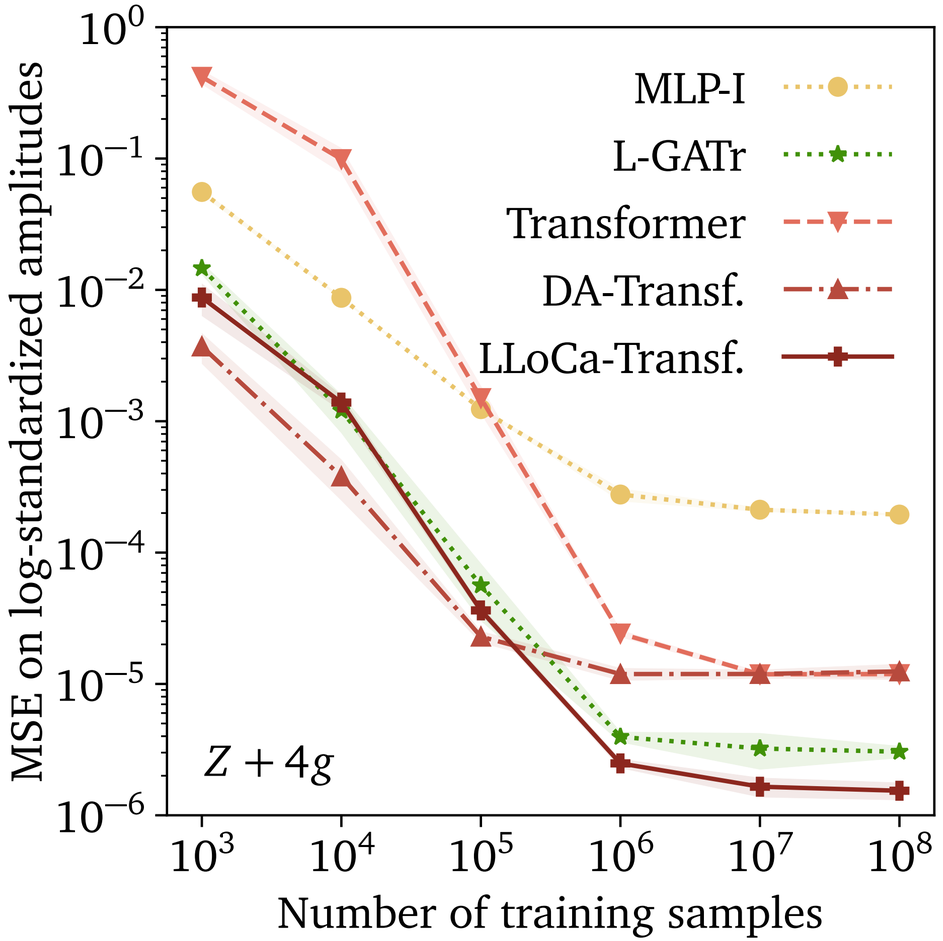
Method	MSE ($\times 10^{-6}$)
Non-equivariant	11.9 ± 0.8
Global canonical.	5.9 ± 0.4
LLoCa (16 scalars)	54.0 ± 3
LLoCa (single 2-tensor)	2.4 ± 0.3
LLoCa (4 vectors)	2.0 ± 0.2
LLoCa (8 scalars, 2 vectors)	**1.5** ± 0.1

LLoCa handles arbitrary tensorial representations

Across our experiments, a mix of scalar and vector representations works best.

Amplitude regression

Equivariance vs data augmentation



Data augmentation (DA) emerges from LLoCa as random global frames

Data augmentation beats Lorentz-equivariance in the limit of little training data

Symmetry breaking

Architecture vs input level

The jet tagging score is
not Lorentz-equivariant

Symmetry breaking at the
architecture level

- Hard to do with specialised layers
- Possible in LLoCa by fixing $v_{i,0}$

Symmetry breaking at the
input level with **reference vectors**

- Beam reference vector:
  $x^V = (0,0,0,1)$
- Time reference vector:
  $x^V = (1,0,0,0)$

All Lorentz-equivariant taggers
use symmetry breaking

Jet Tagging

Lorentz symmetry breaking

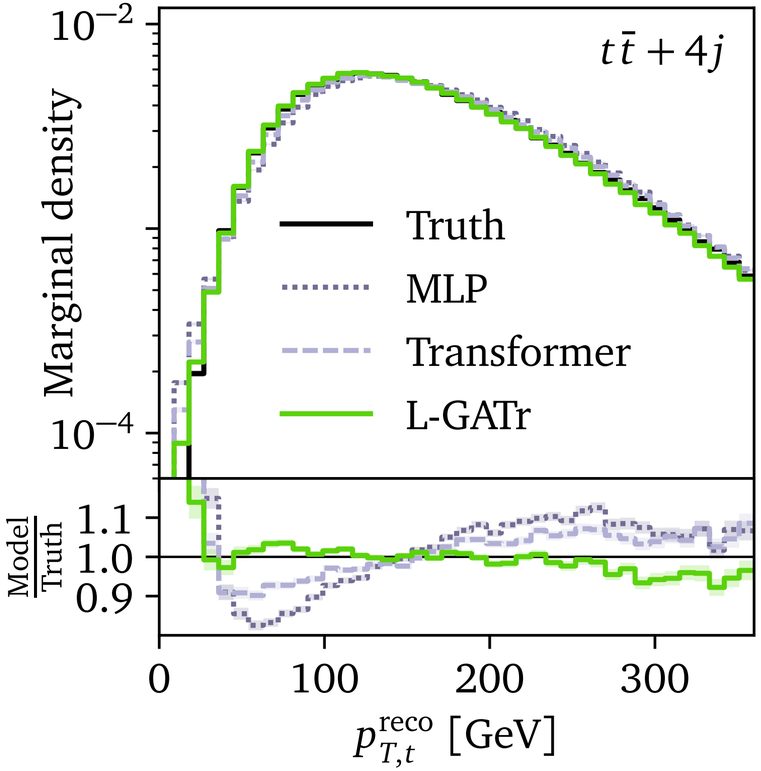
Symmetry breaking	Architecture		Input	
Network	Accuracy	AUC	Accuracy	AUC
Non-equivariant	0.855	0.9867	0.864	0.9882
SO(2)-equivariant	0.862	0.9878	0.864	0.9882
Lorentz-equivariant	0.856	0.9870	0.856	0.9870

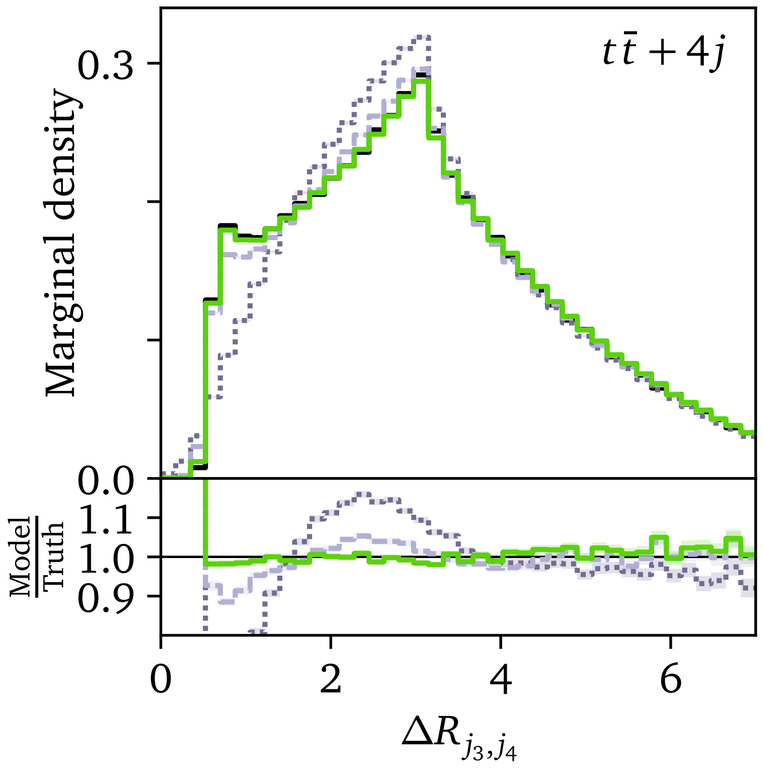
= LLoCa-Transformer

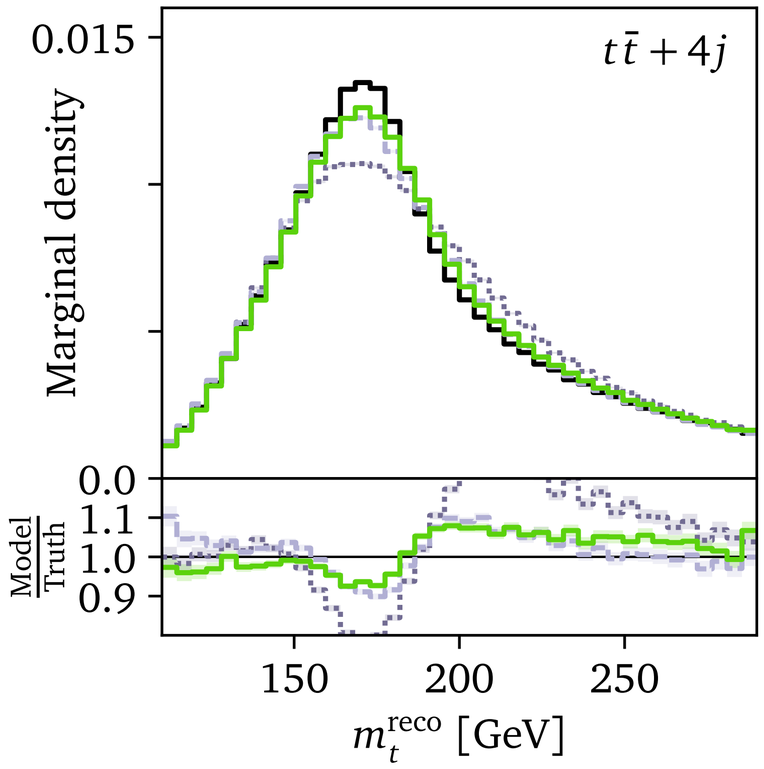
Lorentz-equivariance + symmetry breaking
outperforms SO(2)-equivariance!

Event generation

Kinematic distributions



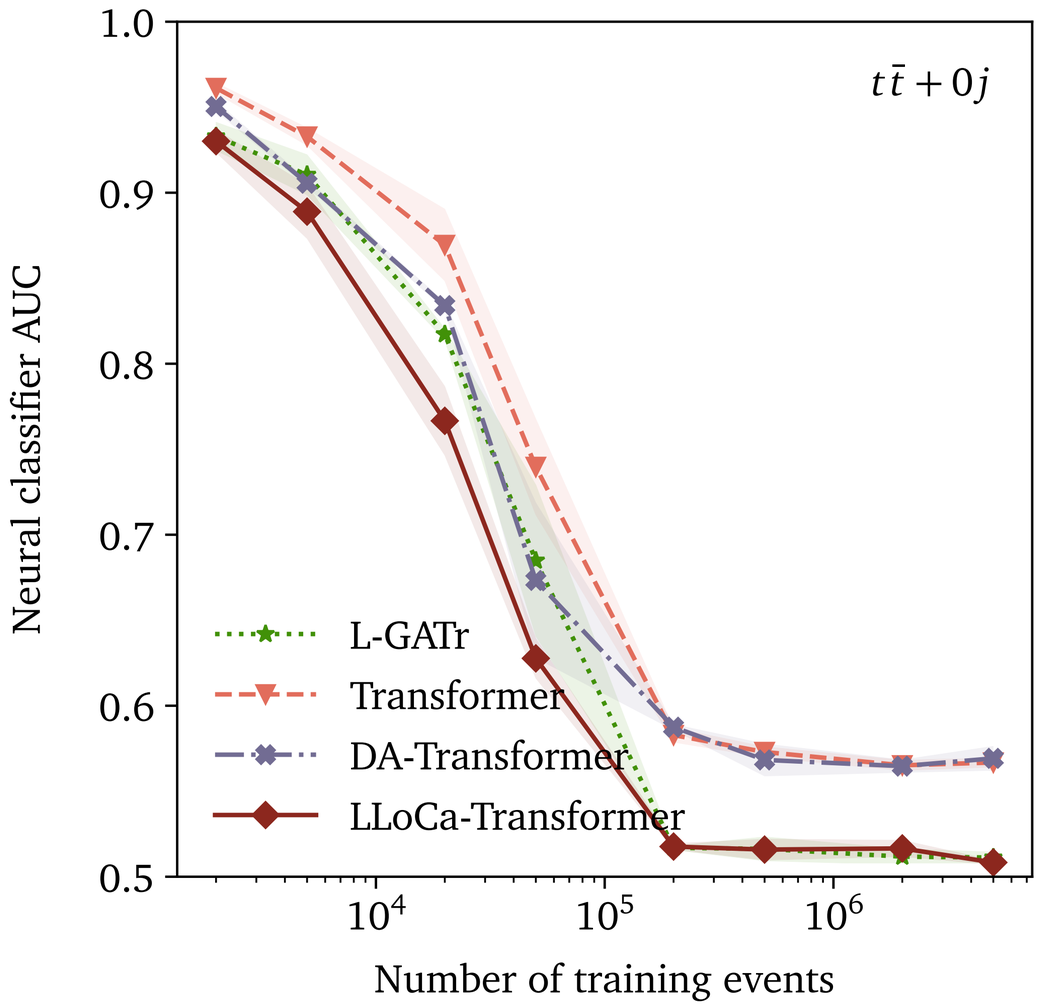




Equivariance helps,
especially for angular correlations

Event generation

Neural classifier metric

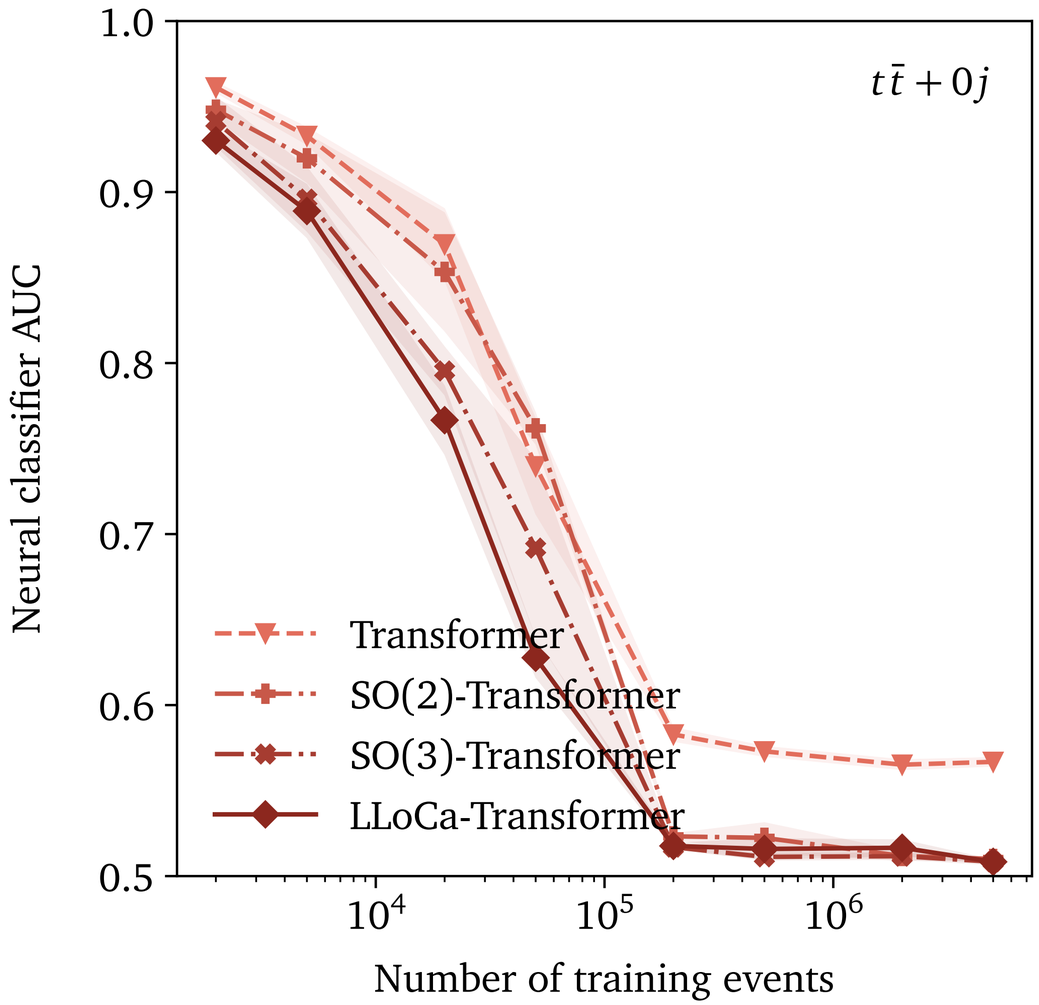


L-GATr and LLoCa-Transformer generate samples that challenge the classifier

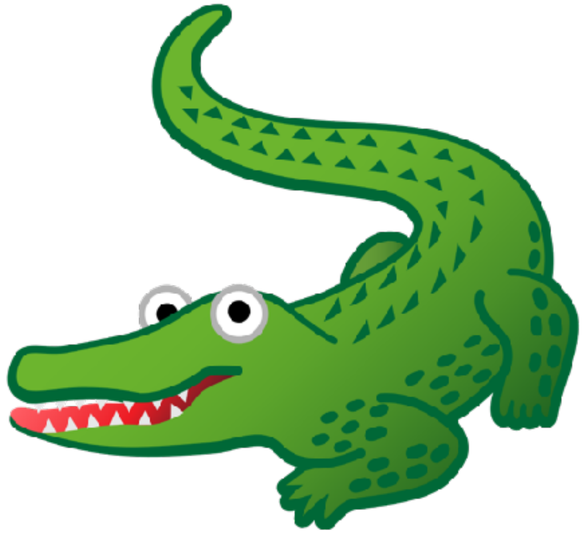
Data augmentation helps for small datasets, but not at scale

Event generation

Symmetry breaking

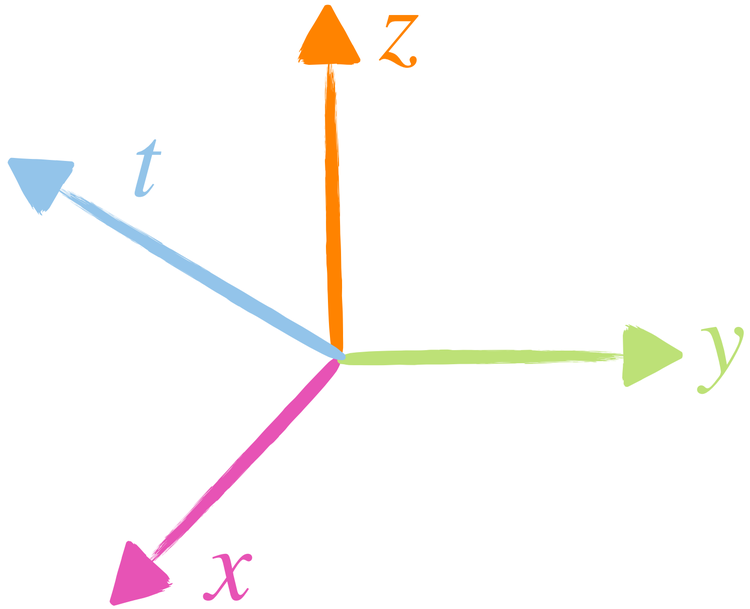


Symmetry breaking at the input and architecture level yield the same result



Lorentz-Equivariant
Geometric **A**lgebra
Transformer

- ✓ `pip install lgatr`
- ✓ Easy to use



Lorentz **Lo**cal
Canonicalization

- ✓ Resource efficient
- ✓ Flexible


Victor Bresó


Pim de Haan


Tilman Plehn


Huilin Qu


Jesse Thaler


Johann Brehmer

Lorentz-Equivariant Geometric Algebra Transformers for High-Energy Physics
NeurIPS 2024, arXiv:2405.14806

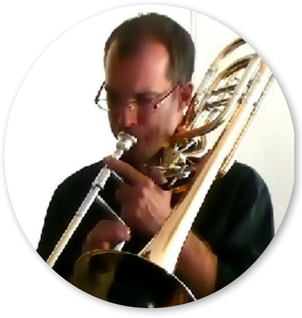
A Lorentz-Equivariant Transformer for all of the LHC
SciPost Physics 2025, arXiv:2411.00446


Luigi Favaro


Peter Lippmann


Sebastian Pitz


Gerrit Gerhartz


Tilman Plehn


Huilin Qu


Fred Hamprecht

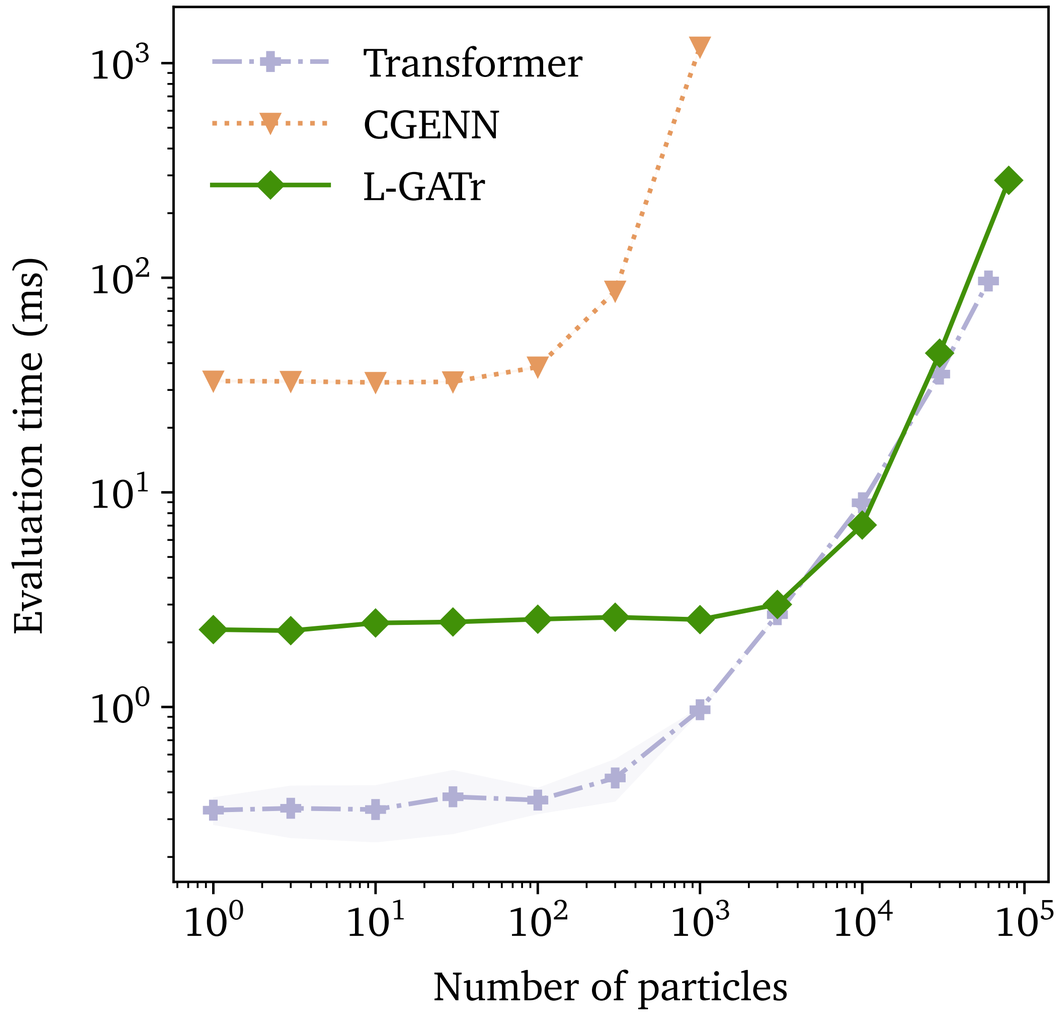
Lorentz Local Canonicalization:
How to make any network Lorentz-equivariant
Under review at NeurIPS 2025, arXiv:2505.20280

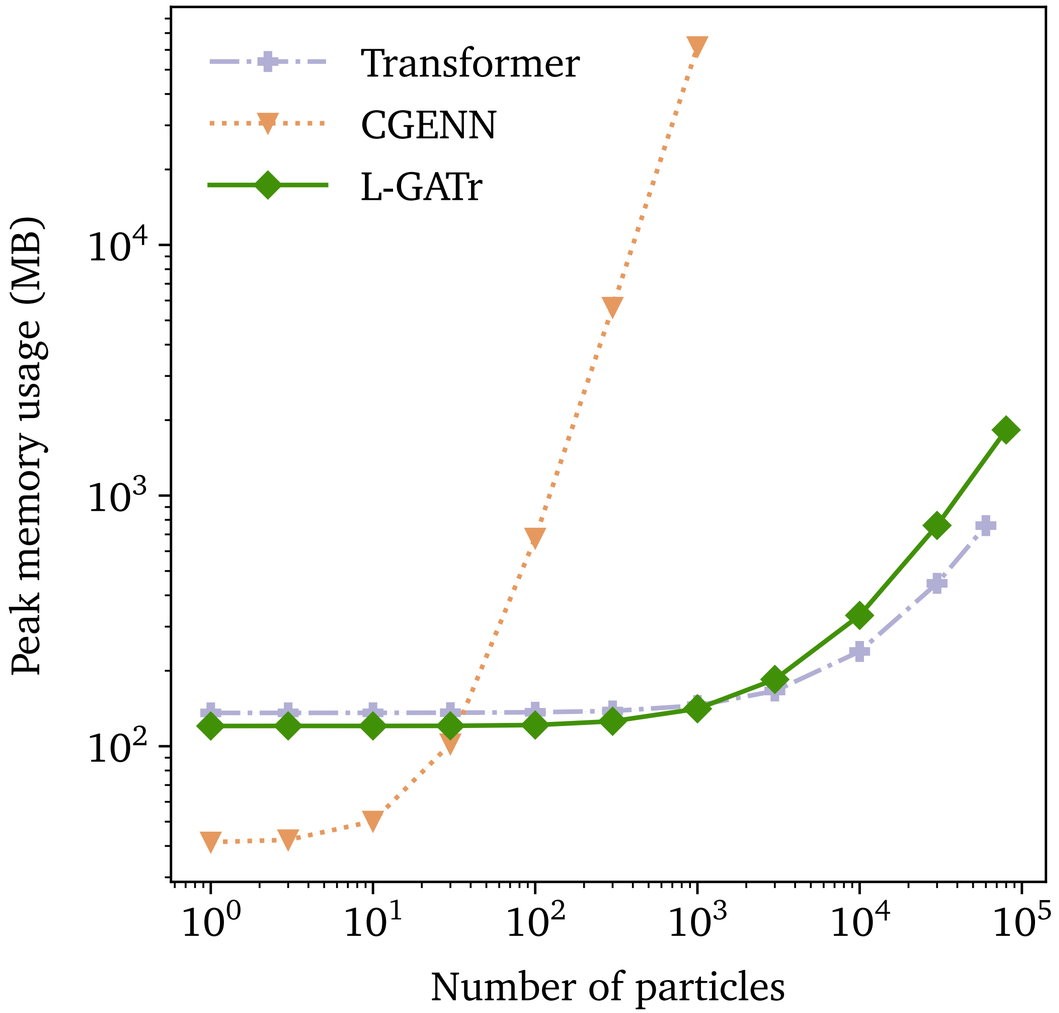
Lorentz-Equivariance without Limitations
arXiv:2508.14898

Bonus material

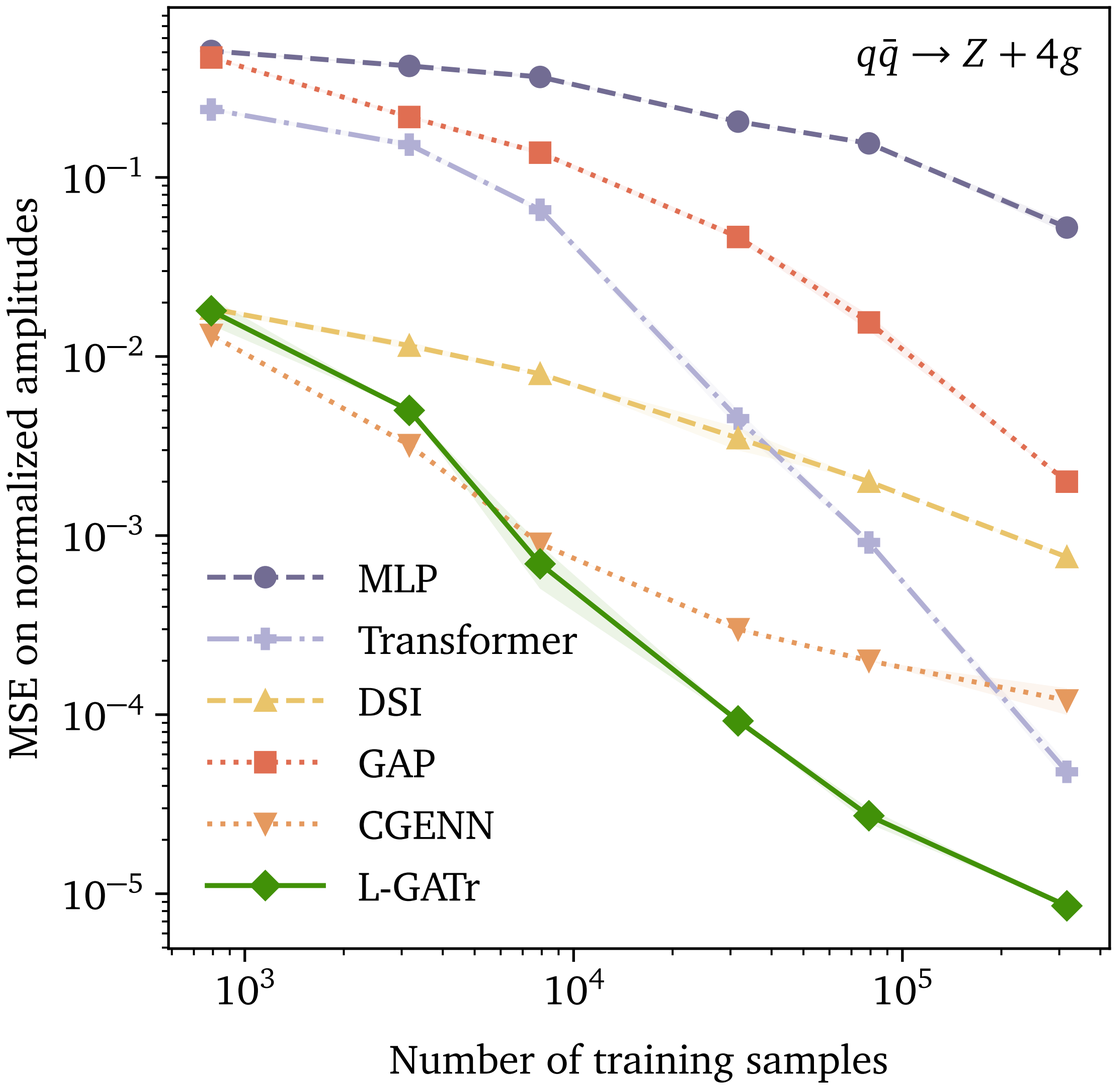
L-GATr

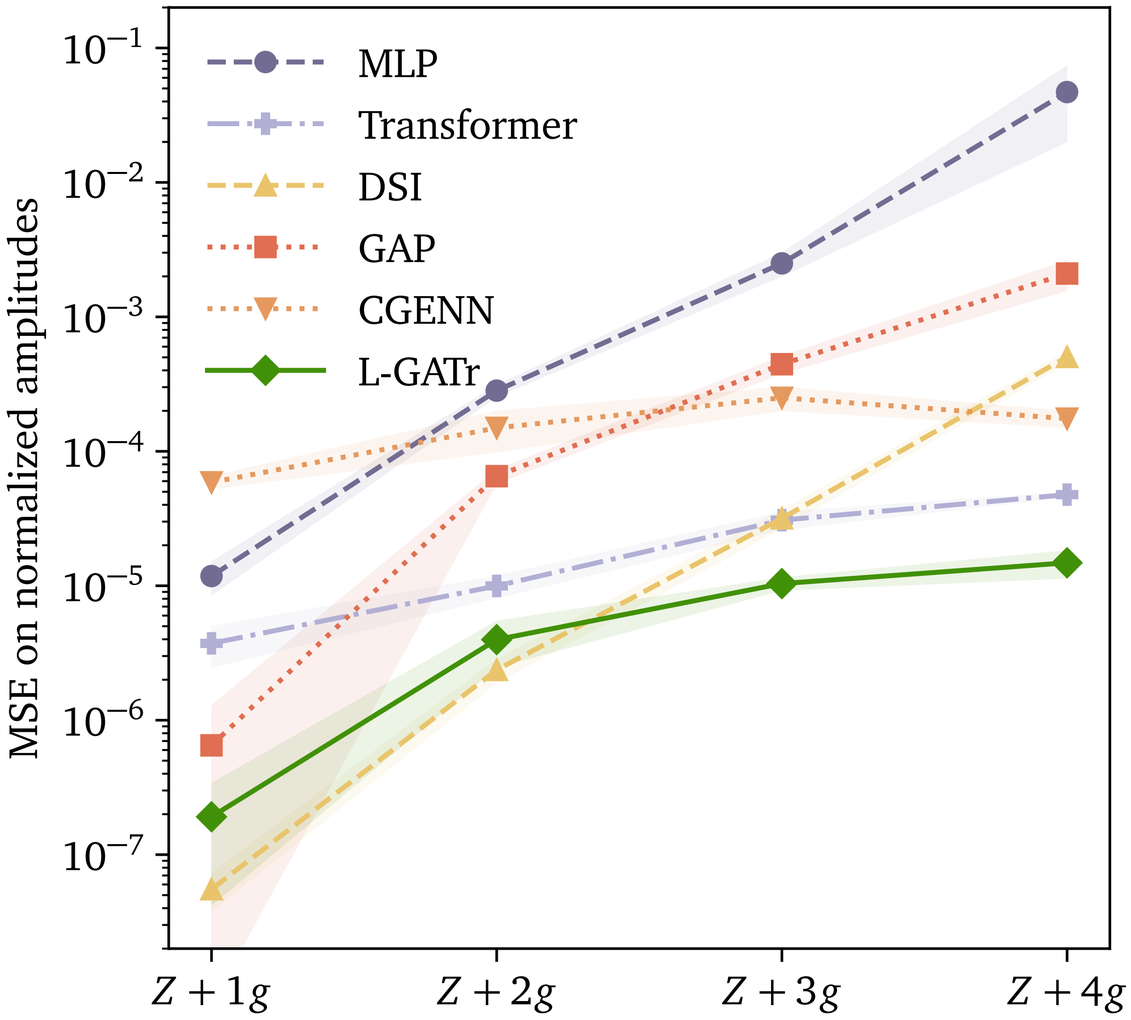
Transformers scale better than graphs





Amplitude regression



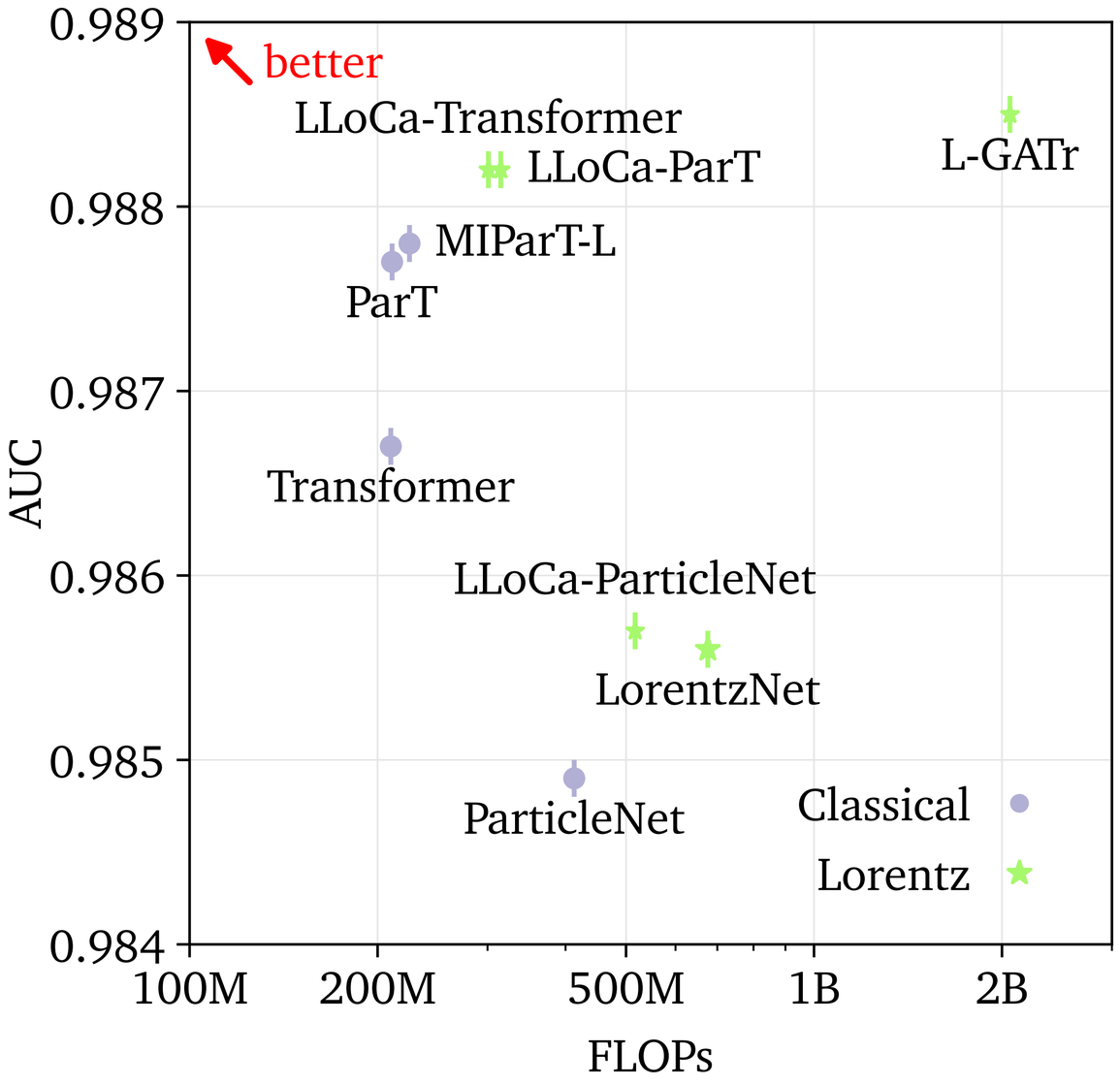


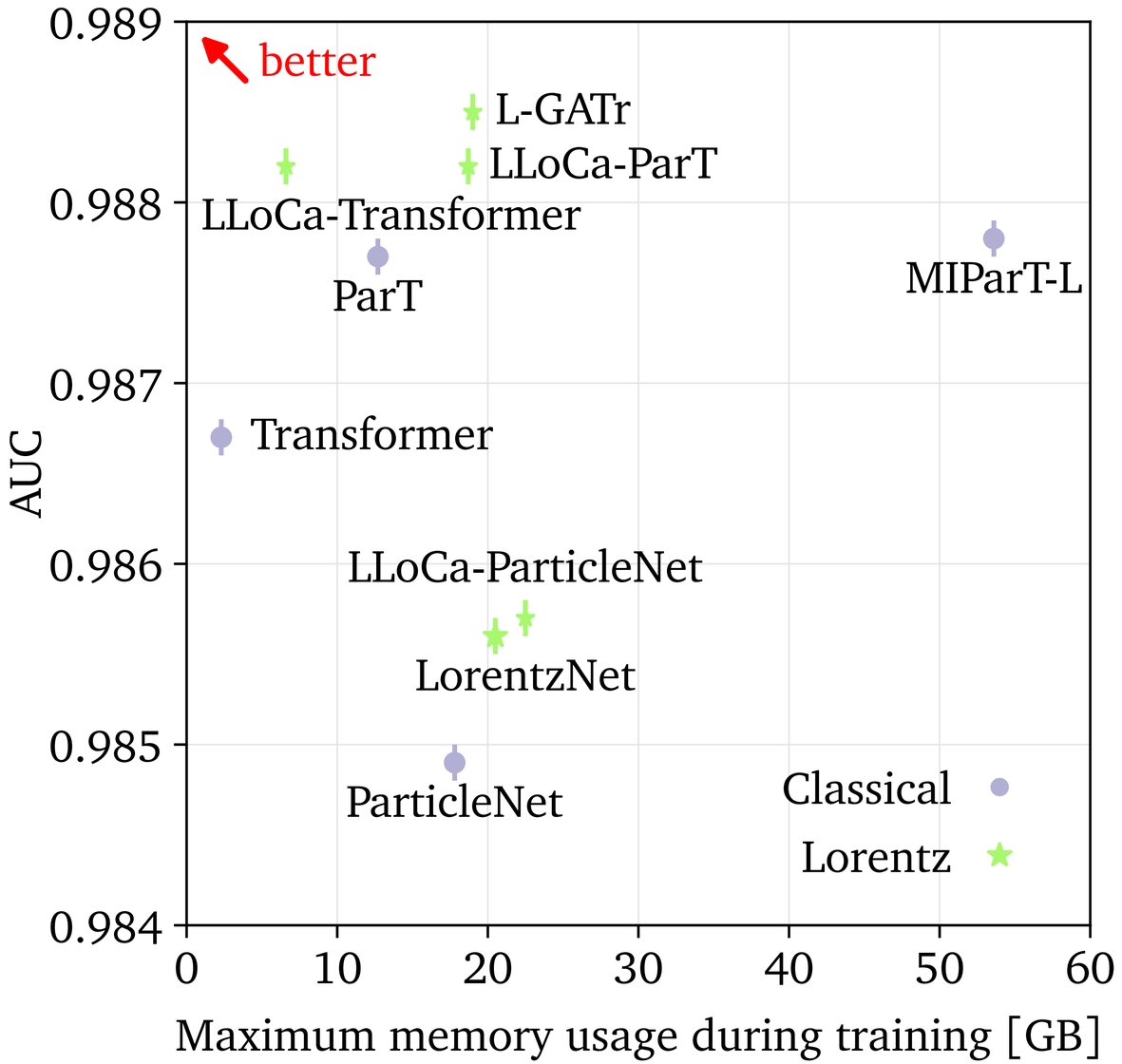
Network	Accuracy	AUC	$1/\epsilon_B$ ($\epsilon_S = 0.5$)	$1/\epsilon_B$ ($\epsilon_S = 0.3$)
TopoDNN [54]	0.916	0.972	–	295 ± 5
LoLa [9]	0.929	0.980	–	722 ± 17
N-subjettiness [55]	0.929	0.981	–	867 ± 15
PFN [56]	0.932	0.9819	247 ± 3	888 ± 17
TreeNiN [57]	0.933	0.982	–	1025 ± 11
ParticleNet [58]	0.940	0.9858	397 ± 7	1615 ± 93
ParT [59]	0.940	0.9858	413 ± 16	1602 ± 81
MIParT [60]	0.942	0.9868	505 ± 8	2010 ± 97
LorentzNet* [10]	0.942	0.9868	498 ± 18	2195 ± 173
CGENN* [14]	0.942	0.9869	500	2172
PELICAN* [42]	0.9426 ± 0.0002	0.9870 ± 0.0001	–	2250 ± 75
L-GATr* [35]	0.9423 ± 0.0002	0.9870 ± 0.0001	540 ± 20	2240 ± 70
ParticleNet-f.t. [60]	0.942	0.9866	487 ± 9	1771 ± 80
ParT-f.t. [60]	0.944	0.9877	691 ± 15	2766 ± 130
MIParT-f.t. [60]	0.944	0.9878	640 ± 10	2789 ± 133
L-GATr-f.t.* (new)	0.9442 ± 0.0002	0.98792 ± 0.00004	661 ± 24	3005 ± 186

	All classes		$H \to b\bar{b}$	$H \to c\bar{c}$	$H \to gg$	$H \to 4q$	$H \to l\nu q\bar{q}'$	$t \to bq\bar{q}'$	$t \to bl\nu$	$W \to q\bar{q}'$	$Z \to q\bar{q}$
	Accuracy	AUC	$\text{Rej}_{50\%}$	$\text{Rej}_{50\%}$	$\text{Rej}_{50\%}$	$\text{Rej}_{50\%}$	$\text{Rej}_{99\%}$	$\text{Rej}_{50\%}$	$\text{Rej}_{99.5\%}$	$\text{Rej}_{50\%}$	$\text{Rej}_{50\%}$
ParticleNet [58]	0.844	0.9849	7634	2475	104	954	3339	10526	11173	347	283
ParT [59]	0.861	0.9877	10638	4149	123	1864	5479	32787	15873	543	402
MIParT [60]	0.861	0.9878	10753	4202	123	1927	5450	31250	16807	542	402
L-GATr	0.865	0.9884	12195	4819	128	2304	5764	37736	19231	580	427

Model	Accuracy (↑)	AUC (↑)	Time	FLOPs
PFN [25]	0.772	0.9714	3h	3M
P-CNN [38]	0.809	0.9789	3h	12M
MIParT [45]	0.861	0.9878	–	–
LorentzNet* [19]	0.847	0.9856	57h	676M
L-GATr* [9]	**0.866**	**0.9885**	180h	2060M
ParticleNet [34]	0.844	0.9849	27h	413M
LLoCa-ParticleNet*	<u>0.848</u>	<u>0.9857</u>	41h	517M
ParT [35]	0.861	0.9877	38h	211M
LLoCa-ParT*	<u>0.864</u>	<u>0.9882</u>	67h	315M
Transformer	0.855	0.9867	17h	210M
LLoCa-Transformer*	<u>0.864</u>	<u>0.9882</u>	33h	301M

Beam	Time	Embedding	AUC	$1/\epsilon_B$ ($\epsilon_S = 0.3$)	
–	–	Token	✗	0.9846 ± 0.0002	1494 ± 62
$x_3^V = \pm 1$	–	Token	✗	0.9854 ± 0.0005	1684 ± 221
$x_{12}^B = x_{13}^B = x_{23}^B = 1$	$x_0^V = 1$	Token	✗	0.9864 ± 0.0002	2076 ± 75
–	$x_0^V = 1$	Token	✗	0.9865 ± 0.0001	2179 ± 123
$x_{12}^B = 1$	$x_0^V = 1$	Channel	✗	0.9866 ± 0.0001	2150 ± 82
$x_0^V = 1, x_3^V = \pm 1$	$x_0^V = 1$	Token	✗	0.9869 ± 0.0001	2282 ± 174
$x_3^V = \pm 1$	$x_0^V = 1$	Token	✗	0.9869 ± 0.0001	2293 ± 141
–	–	Token	✓	0.9869 ± 0.0001	2179 ± 156
$x_{12}^B = 1$	$x_0^V = 1$	Token	✓	0.9870 ± 0.0001	2173 ± 83
$x_{12}^B = 1$	$x_0^V = 1$	Token	✗	0.9870 ± 0.0001	2240 ± 70


better
LLoCa-Transformer
LLoCa-ParT
L-GATr
MIParT-L
ParT
Transformer
LLoCa-ParticleNet
LorentzNet
ParticleNet
Classical
Lorentz
AUC
0.989
0.988
0.987
0.986
0.985
0.984
100M
200M
500M
1B
2B
FLOPs

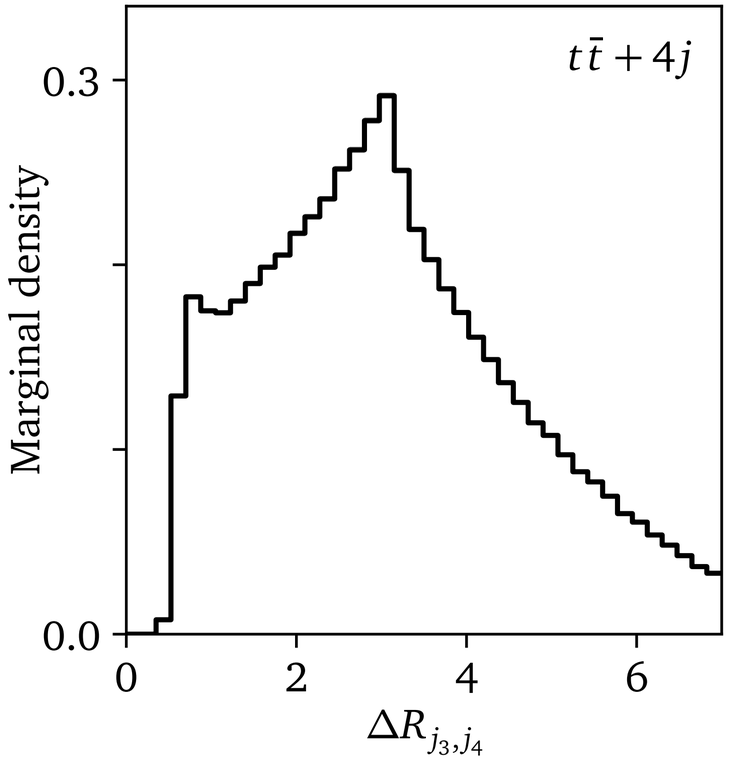

better
L-GATr
LLoCa-ParT
LLoCa-Transformer
ParT
MIParT-L
Transformer
LLoCa-ParticleNet
LorentzNet
ParticleNet
Classical
Lorentz
AUC
0.989
0.988
0.987
0.986
0.985
0.984
0
10
20
30
40
50
60
Maximum memory usage during training [GB]

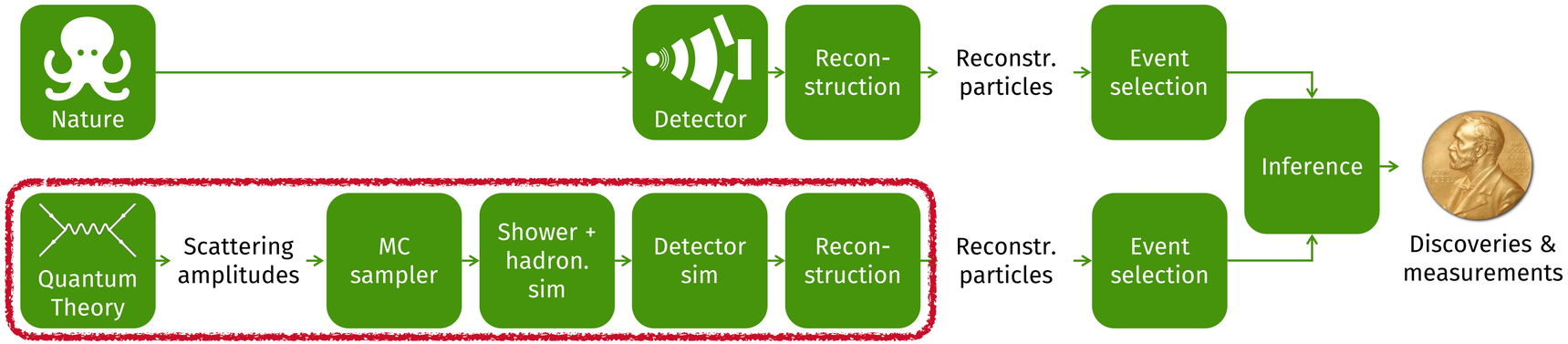
Event generation

Task

Dataset: $pp \to t_h \bar{t}_h + n\,j, \;\; n = 0 \ldots 4$

- MadGraph + Pythia + Delphes + Reconstruction
- Challenging features: $m_t, m_W, \Delta R_{jj} > 0.5$
- Symmetry breaking with reference multivectors required at reconstruction level





Event generation

Conditional Flow Matching

Continuous normalising flows (CNFs)
connect a simple base density
to a complex target density
through a neural differential equation

$$\frac{d}{dt}x = v_t(x)$$

arXiv:1806.07366

Conditional flow matching (CFM)
is a simple way to train CNFs
by comparing the learned velocity $v_t(x)$
to a conditional target velocity $u_t(x \,|\, x_1)$

$$\mathscr{L} = \left\langle (v_t(x) - u_t(x \,|\, x_1))^2 \right\rangle$$

arXiv:2210.02747

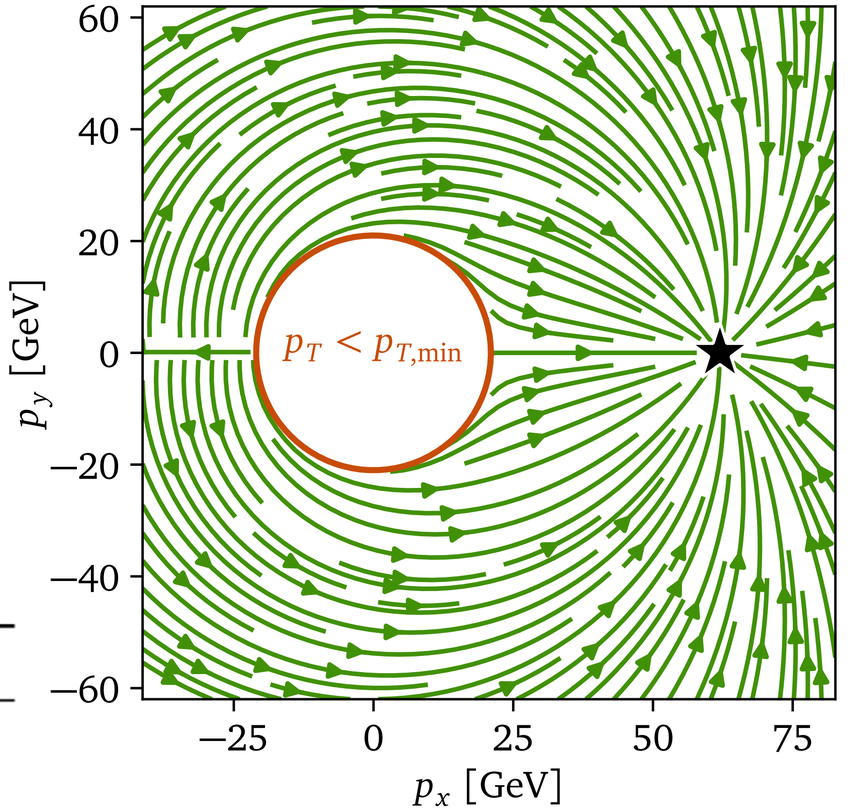
How to pick the target velocity $u_t(x \,|\, x_1)$?

Event generation

Physics-inspired target trajectories

Straight trajectories in ‘modified jet momenta’ x:

$$p = \begin{pmatrix} E \\ p_x \\ p_y \\ p_z \end{pmatrix} \quad \rightarrow \quad f^{-1}(p) = x = \begin{pmatrix} x_p \\ x_m \\ x_\eta \\ x_\phi \end{pmatrix} \equiv \begin{pmatrix} \log(p_T - p_{T,\text{min}}) \\ \log m^2 \\ \eta \\ \phi \end{pmatrix}$$



Data	Architecture	Base distribution	Periodic	Neg. log-likelihood	AUC
p	L-GATr	rejection sampling	✓	-30.80 ± 0.17	0.945 ± 0.004
x	MLP	rejection sampling	✓	-32.13 ± 0.05	0.780 ± 0.003
x	L-GATr	rejection sampling	✗	-32.57 ± 0.05	0.530 ± 0.017
x	L-GATr	no rejection sampling	✓	-32.58 ± 0.04	0.523 ± 0.014
x	L-GATr	rejection sampling	✓	-32.65 ± 0.04	0.515 ± 0.009

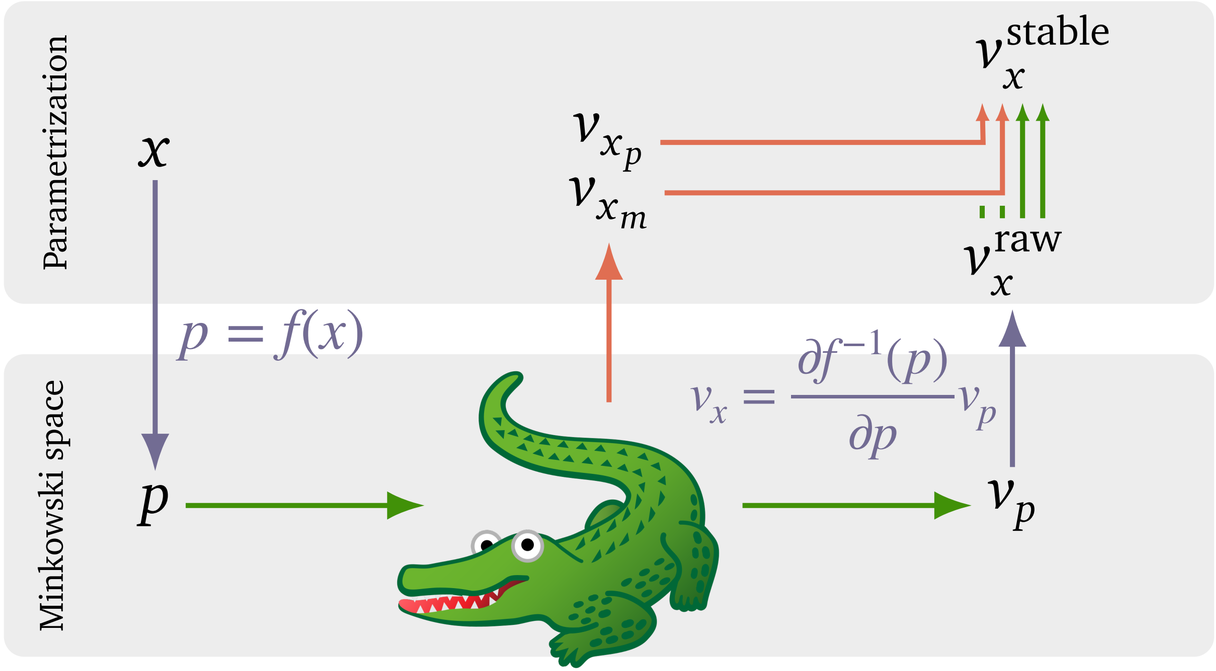
→ default

Event generation

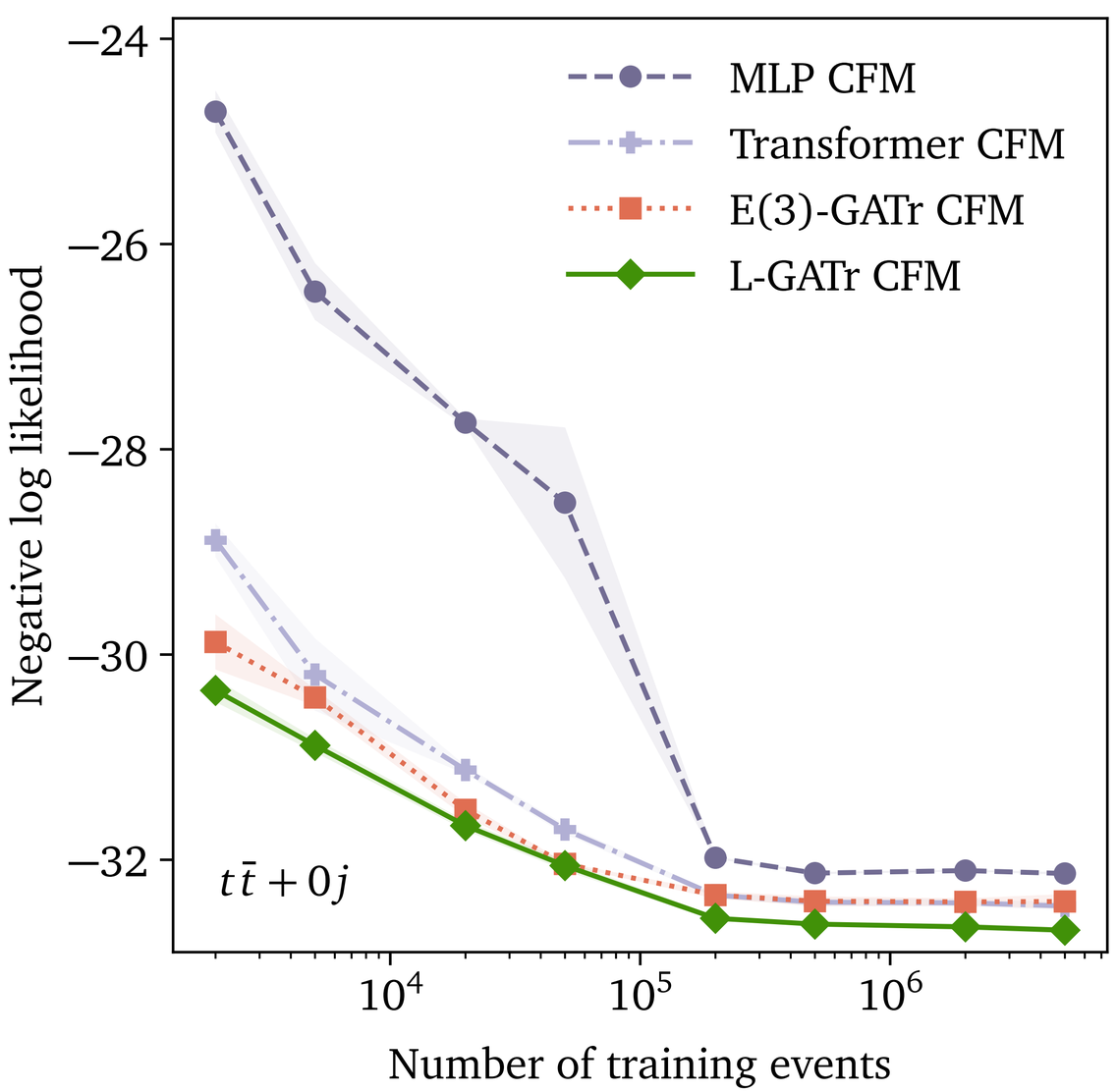
How to build an equivariant CFM field $v_x(x)$?

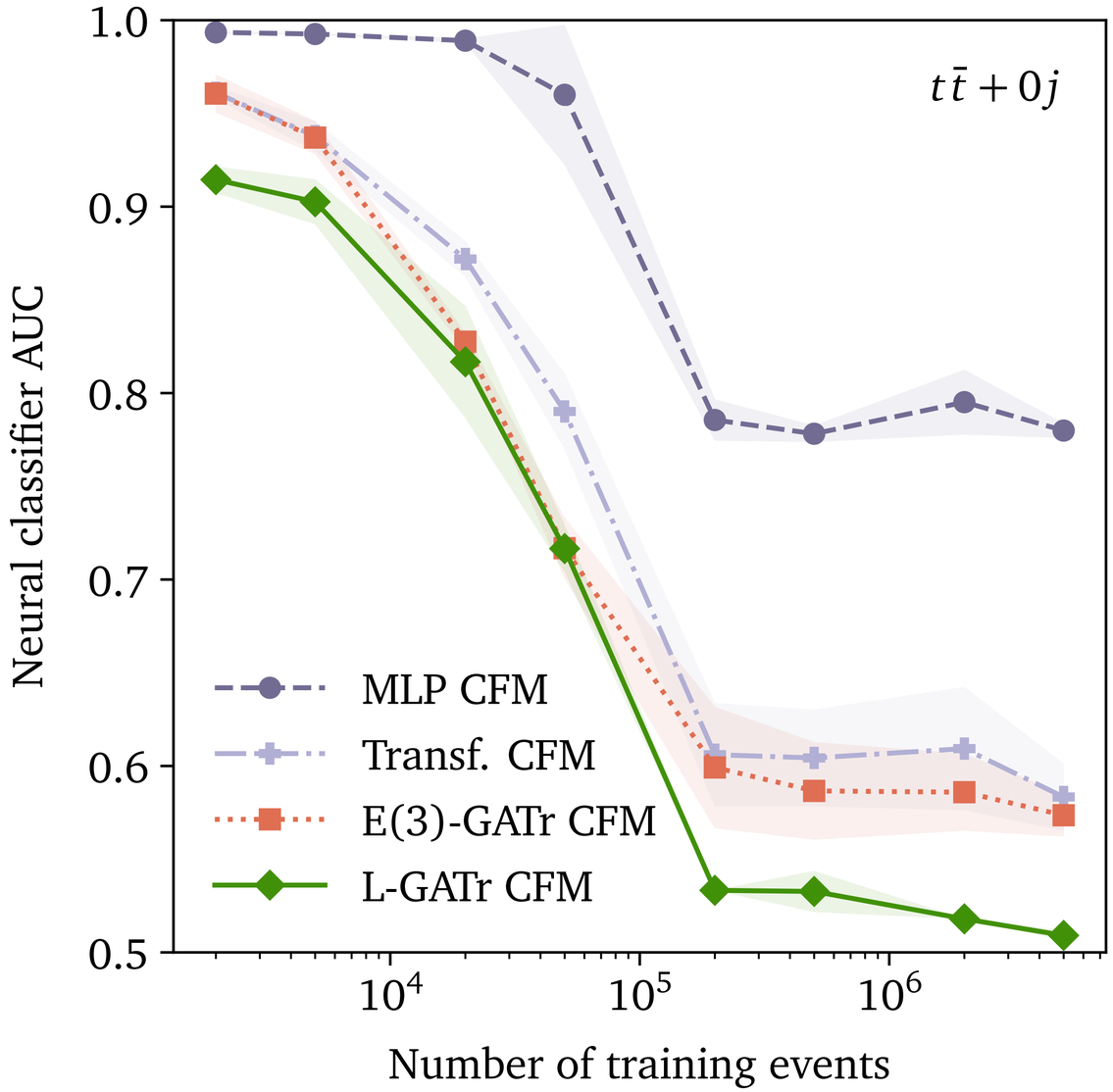
Extend standard CFM workflow with L-GATr:

- Transformations $f(x)$ between Minkowski space p and the parametrization x
- Equivariant L-GATr operations using multivectors
- Symmetry-breaking operations using scalars (required for numerical stability)

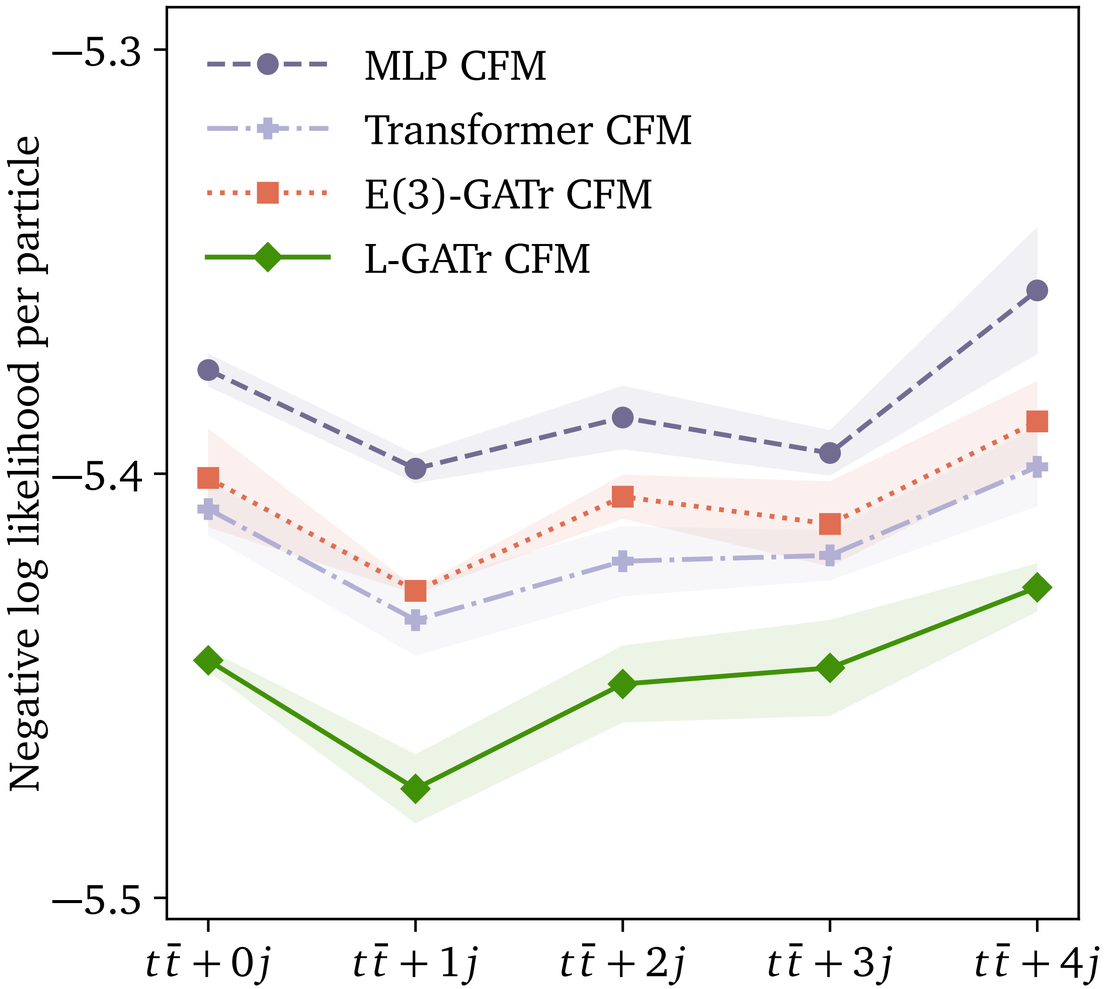


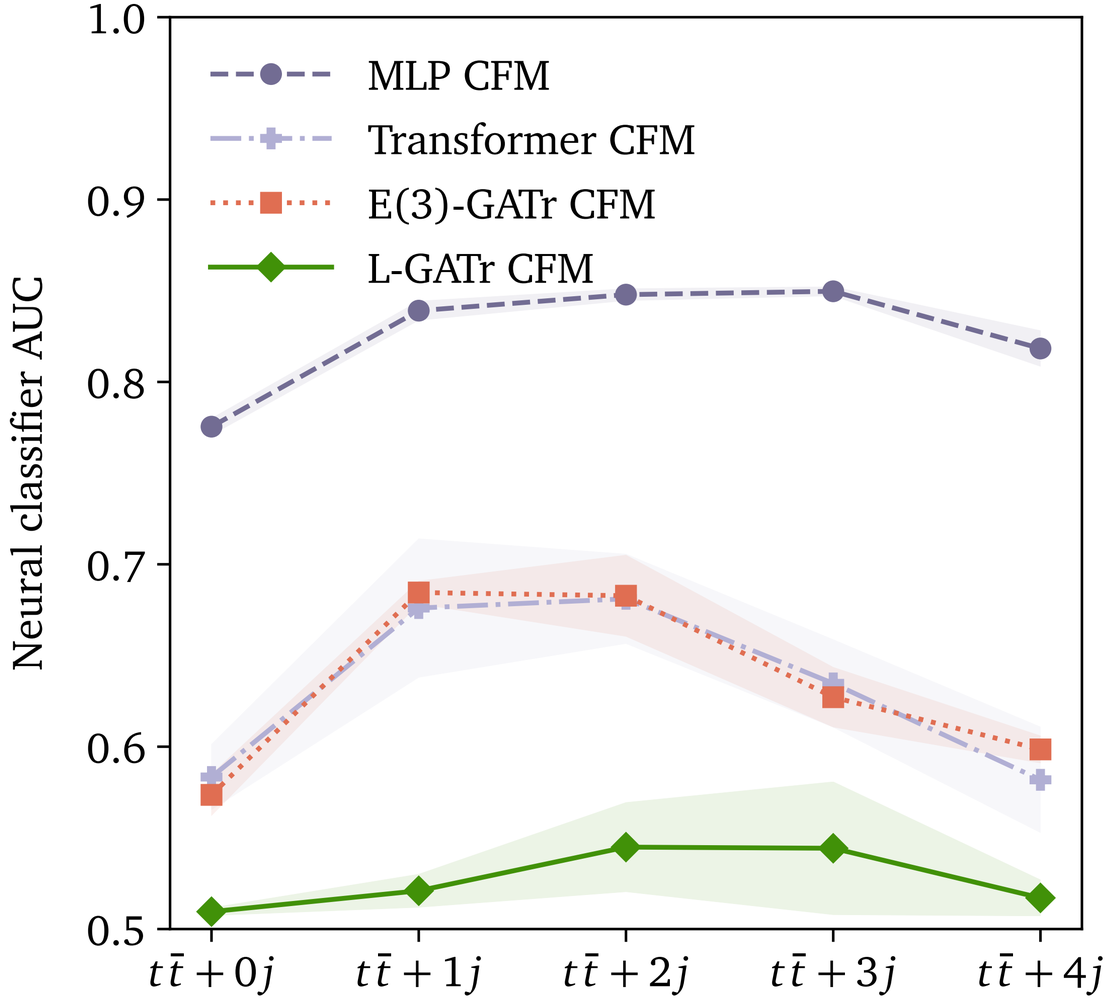
Event generation



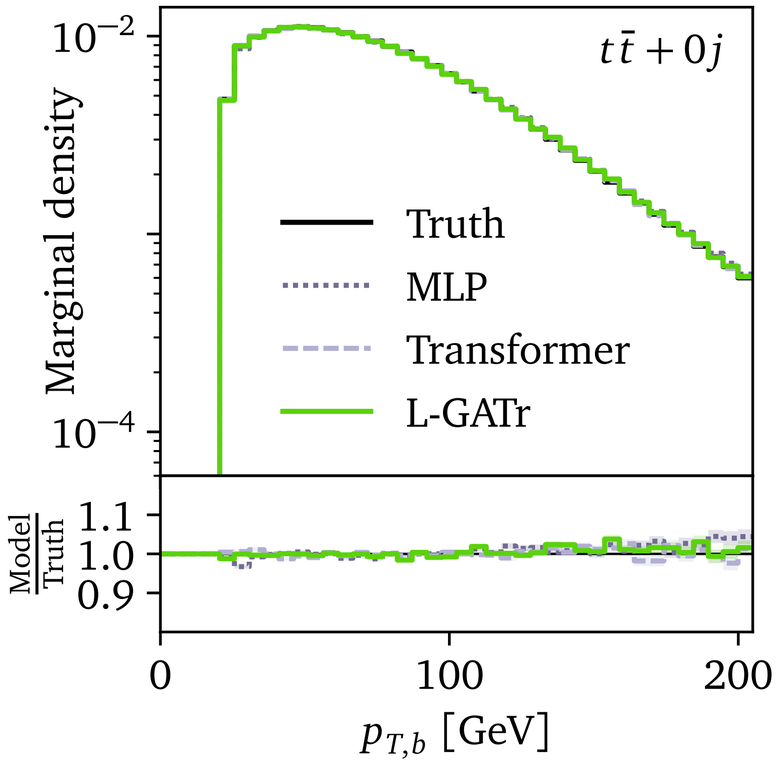


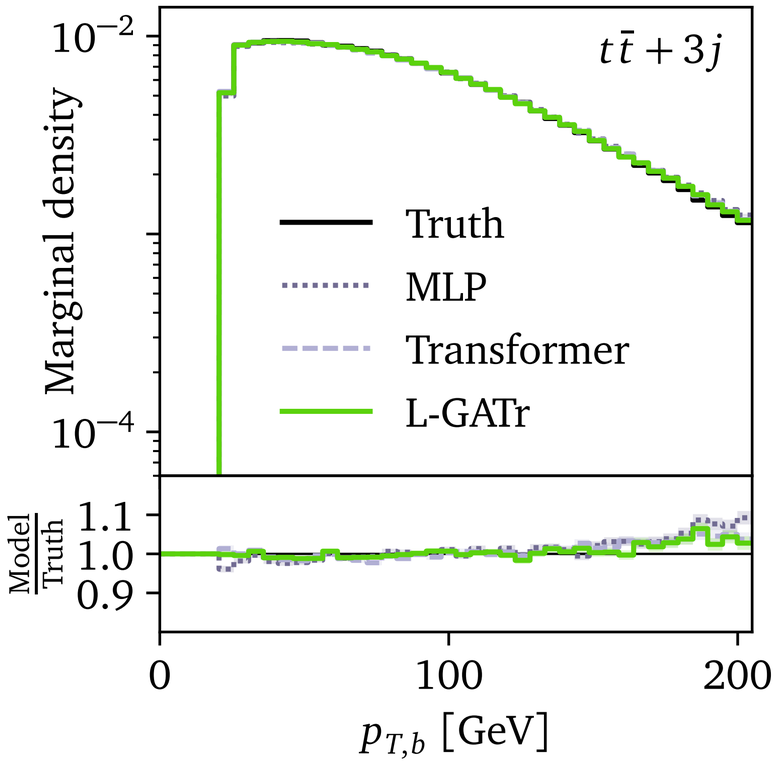
Event generation

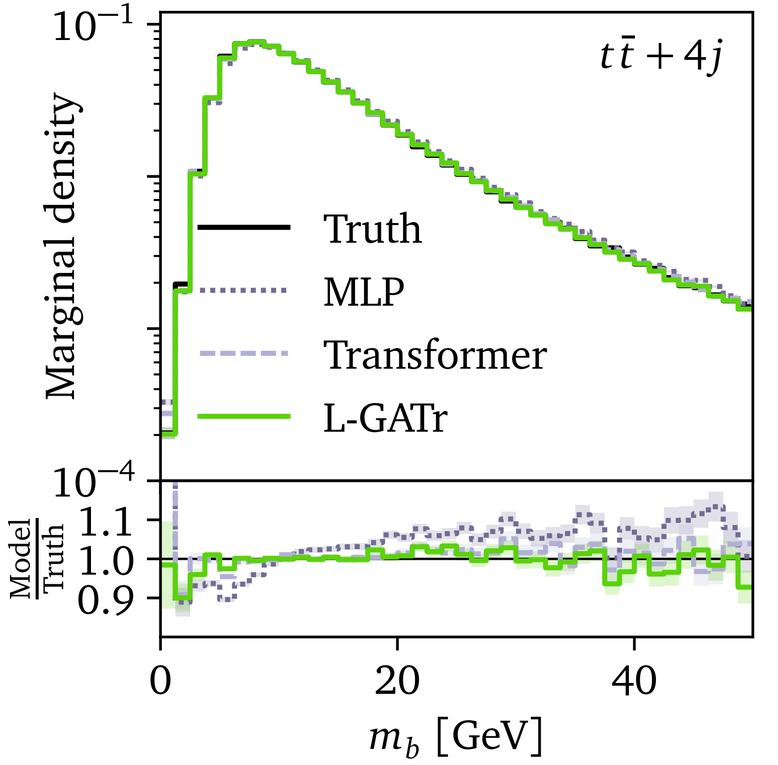




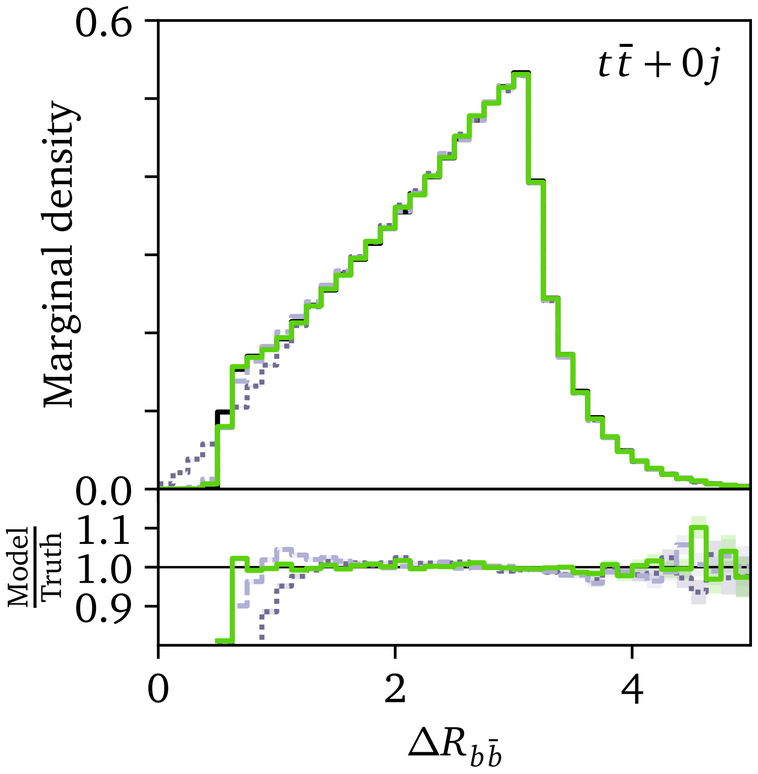
Event generation

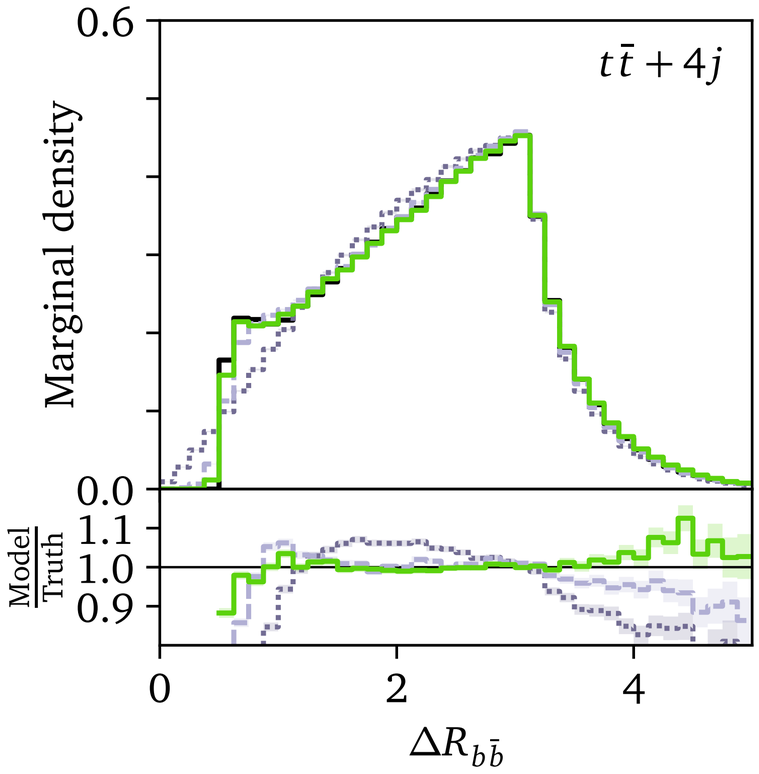


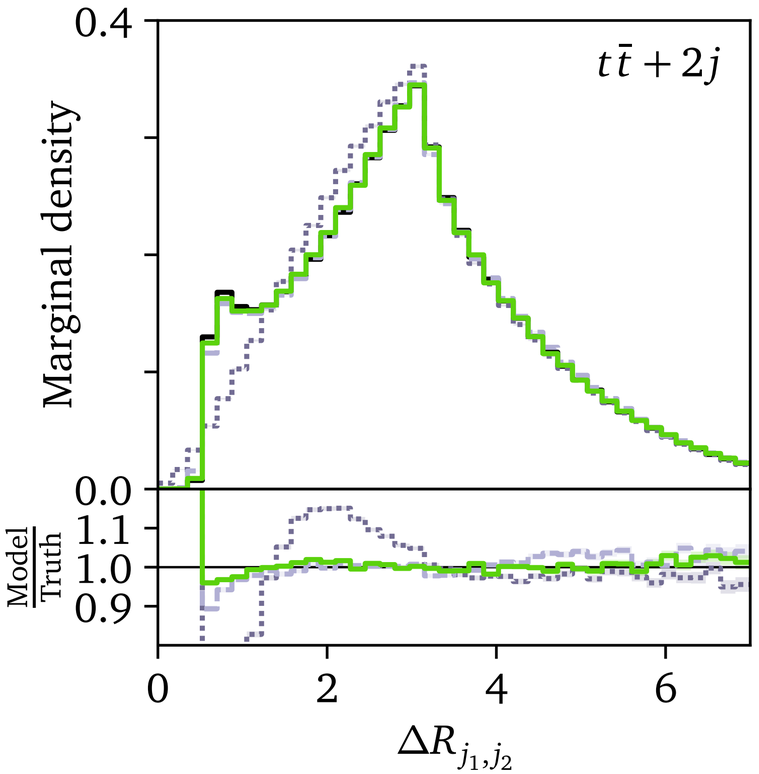




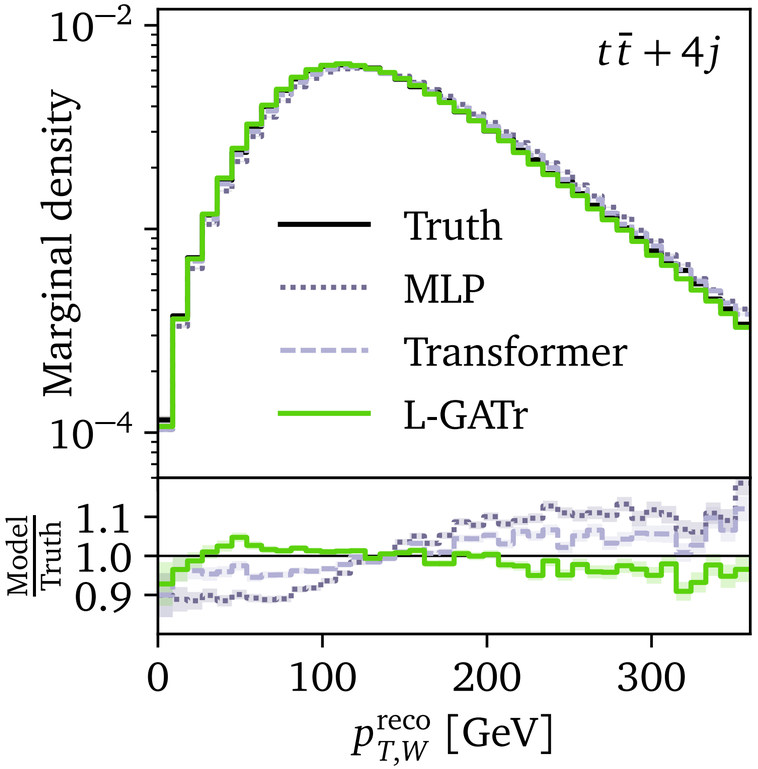
Event generation

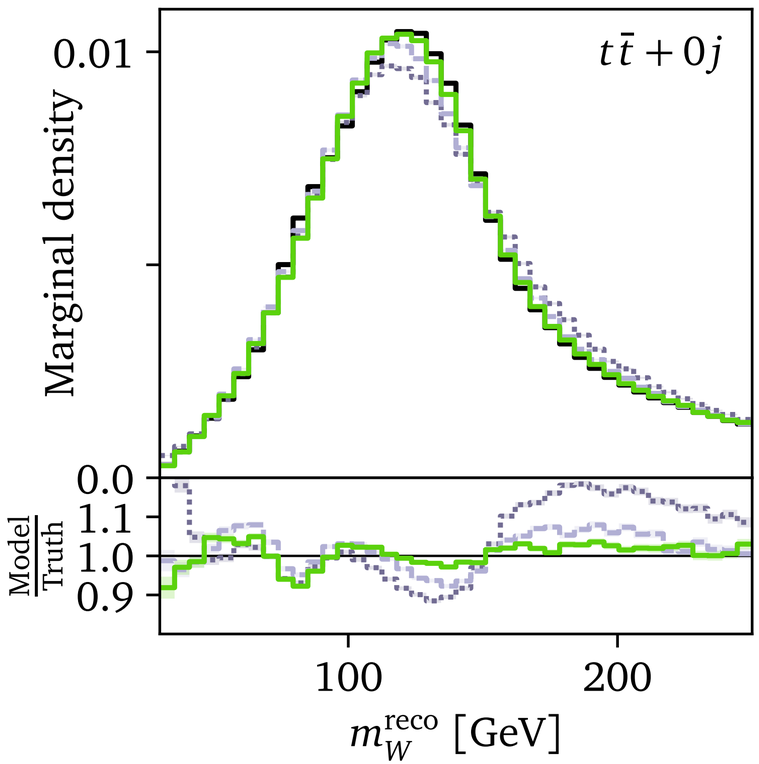


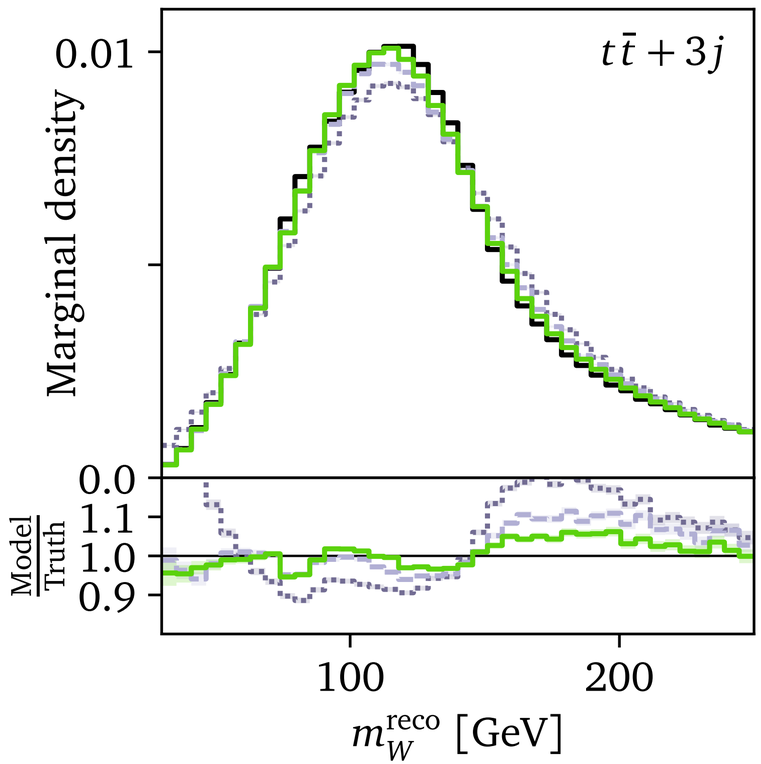




Event generation







Event generation