Top Reconstruction with LGATR

Jorn Bach

DESY CMS Top Reconstruction Discussion

01.07.2025

Contents

1. Introduction to Transformers

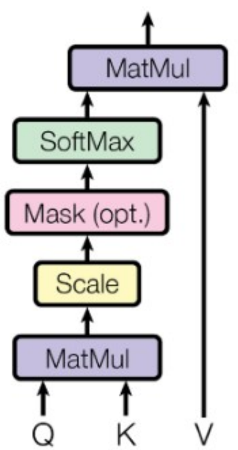
- Transformers have emerged as well-scaling, powerful tools in NLP
- Based on the **Attention Mechanism** to learn relationships in data

Attention mechanism as scaled dot-product:

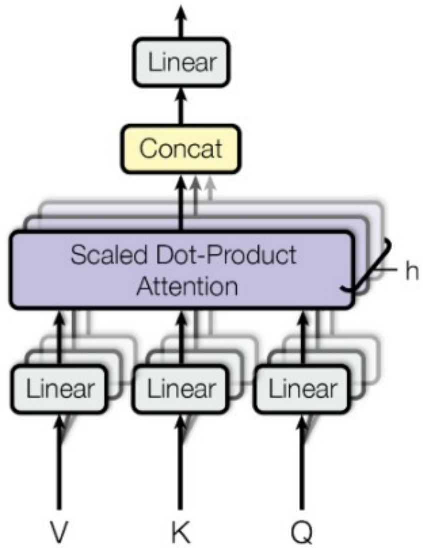
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

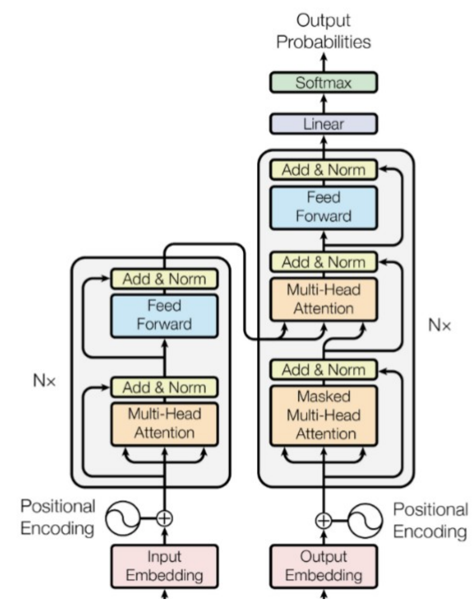
- $\alpha_{1,i} = q_1 k_i / \sqrt{d_K}$, $\hat{\alpha}_{1,i} = e^{\alpha_{1,i}} / \sum_j e^{\alpha_{1,j}}$, $b_1 = \sum_i \hat{\alpha}_{1,i} v_i$
- Q (queries): quantify attention to be paid to other tokens
- K (keys): reference that each token provides for answering the query
- V (values): information to use when a token is selected

Scaled Dot-Product Attention



Multi-Head Attention





1. Introduction to LGATR

- https://arxiv.org/abs/2405.14806, https://arxiv.org/abs/2411.00446, https://github.com/heidelberg-hepml/lorentz-gatr
- Now take the allmighty TRANSFORMER but make it physics (lorentz equivariant)
- In fact: make it structurally physics: Everything it computes, inputs and outputs is a *Lorentz Algebra Object:*

- 16-dimensional space
- Equipped with a geometric product $< \cdot, \cdot >$ generating 'grades':
  - Grade 0: scalars (e.g., particle type)
  - Grade 1: four-vectors (e.g., $(E, \vec{p})$)
  - Higher grades: antisymmetric tensors for increased expressivity
- General multivector:

$$x = x^S \mathbf{1} + x^V_\mu \gamma^\mu + x^B_{\mu\nu} \sigma^{\mu\nu} + x^A_\mu \gamma^\mu \gamma^5 + x^P \gamma^5, \quad \begin{pmatrix} x^S \\ x^V_\mu \\ x^B_{\mu\nu} \\ x^A_\mu \\ x^P \end{pmatrix} \in \mathbb{R}^{16}$$

Each layer $f(x)$ is equivariant w.r.t. Lorentz transformations Λ:

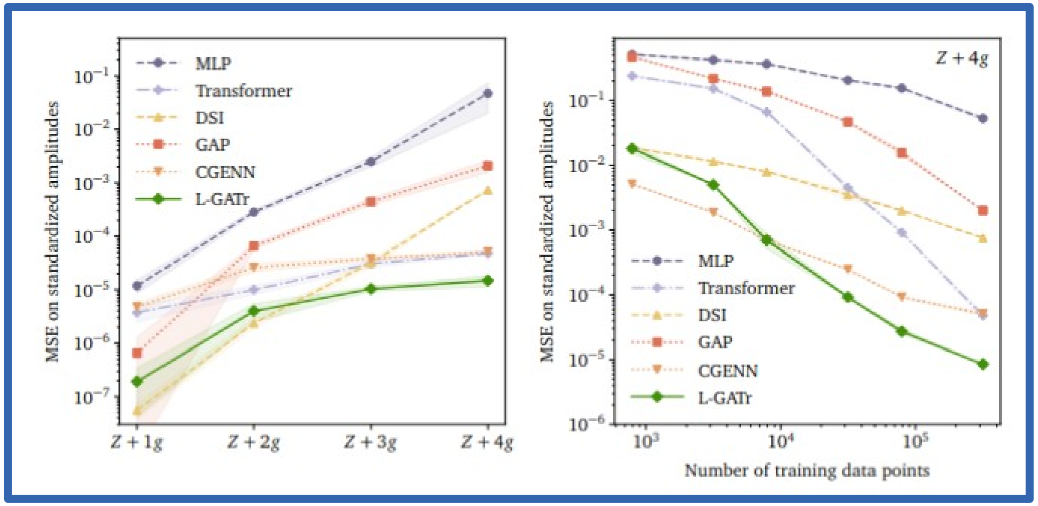
- $f(\Lambda(x)) = \Lambda(f(x))$

Layer type	Transformer	L-GATr
Linear(x)	$vx + w$	$\sum_{k=0}^4 v_k \langle x \rangle_k$
Attention$(q,k,v)_{ic}$	$\sum_{j=1}^{n_t} \text{Softmax}_j \left(\sum_{c'=1}^{n_c} \frac{q_{ic'} k_{jc'}}{\sqrt{n_c}} \right) v_{jc}$	$\sum_{j=1}^{n_t} \text{Softmax}_j \left(\sum_{c'=1}^{n_c} \frac{\langle q_{ic'}, k_{jc'} \rangle}{\sqrt{16 n_c}} \right) v_{jc}$
LayerNorm(x)	$x \left[\frac{1}{n_c} \sum_{c=1}^{n_c} x_c^2 + \epsilon \right]^{-1/2}$	$x \left[\frac{1}{n_c} \sum_{c=1}^{n_c} \sum_{k=0}^4 \left| \left\langle \langle x_c \rangle_k, \langle x_c \rangle_k \right\rangle \right| + \epsilon \right]^{-1/2}$

- Token (i.e., particle): $x_i = \{x_{ic} : c = 1, \ldots, n_c\}$
- $x_{ic} = (x^S_{ic}, x^V_{\mu,ic}, x^B_{\mu\nu,ic}, x^A_{\mu,ic}, x^P_{ic}) \in \mathbb{R}^{16} \quad i = 1, \ldots, n_t \quad c = 1, \ldots, n_c$
- n_c: number of multivector channels
- $< x >_k$: multivector projection, sets all non-grade-k elements to zero

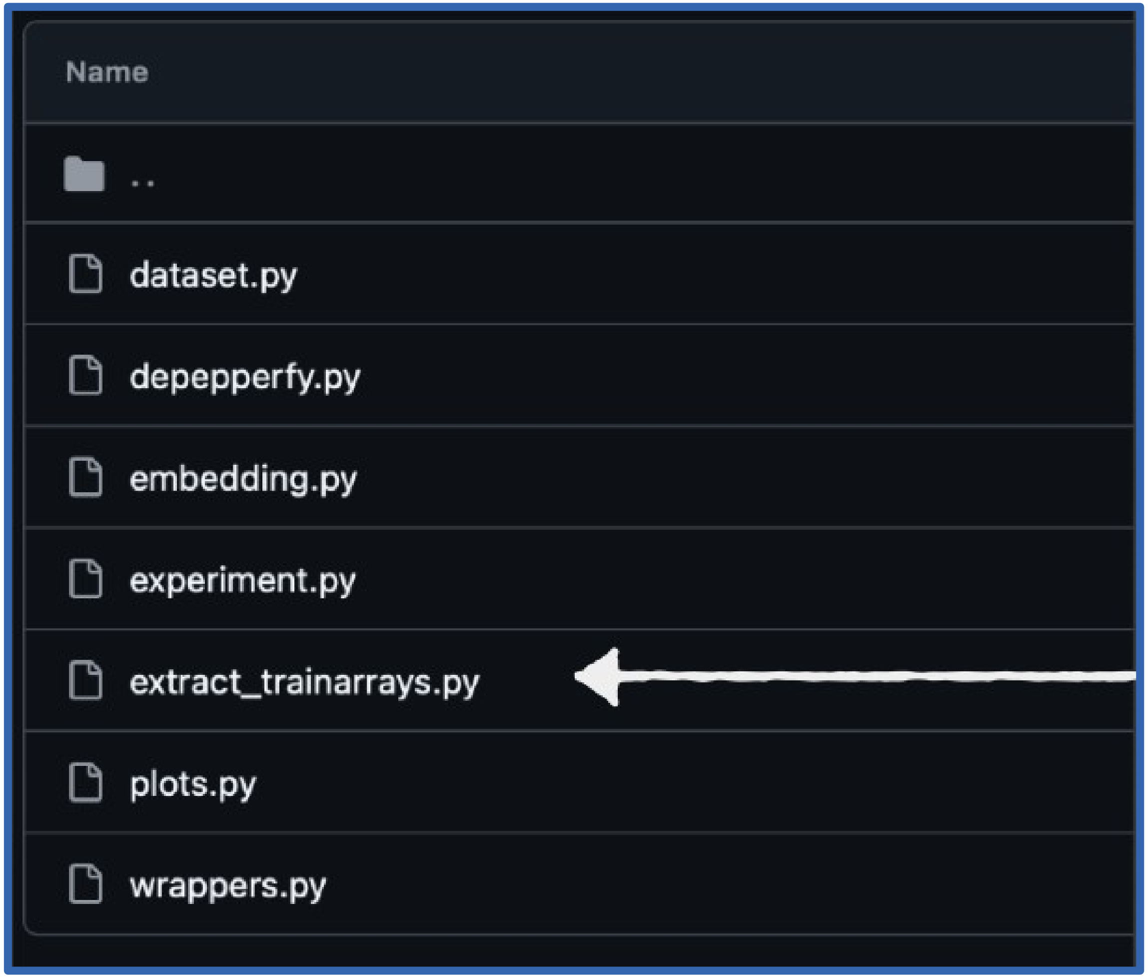
LGATR – WHY?

- Imposed Lorentz Structure means more efficient learning
- Their paper shows competitive/superior performance on: Amplitude Regression, Jet Tagging, Event Generation
- Scaling with data and network size outperforms regular transformers!
- Their code is really good: MLFLOW for logging, config management via hydra
- Latest code revision 3 weeks ago – maintained!



LGATR for Top Reconstruction

- Lives in my github: https://github.com/jobach18/lorentz-gatr-ttbarreco
- experiments/top-reco/ houses some script to put pepper output into place, experiment code (training, validation, plotting) and some wrapper.py to get a suitable lgatr instance



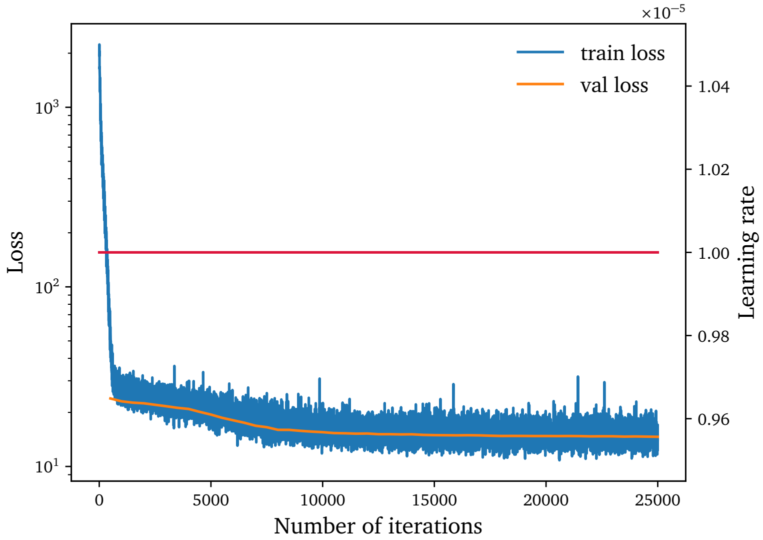
Dilepton Reconstruction Setup

- Pepper output compiled by Dominic on A/H selection from last year
- On **detector level objects:**
  - Input: 7 jets, lepton, antilepton, bottom, antibottom, MET
  - Bottoms are already mlb matched, should not matter because LGATR could learn the matching method and “rematch” if needed
  - For particles: (t, x, y ,z) for MET (pt, x,y,phi) (this might be wrong?)
  - Output: top, antitop (from GEN)
- Output aggregation: (just pick one element vs. mean or sum)

```
def extract_from_ga(self, multivector):
    #summed_mv = sum_along_dimension(multivector)
    #print('before extraction the output reads')
    #print(multivector.shape)
    ## (1,batch_size * seq_len, 1, 16)
    tokens_per_item = 14  # or however many tokens per item
    out = multivector.view(1, int(multivector.shape[1]/tokens_per_item), tokens_per_item, 2, 16)  # [1, 256, 14, 2, 16]
    # Select the first token for each item
    out_reduced = out[:, :, 0, :, :]  # [1, 256, 2, 16]
    reshape_out = True
    if reshape_out:
        outputs =  extract_vector(out_reduced)
    else:
        outputs = extract_vector(multivector[0,::14,:,:]) # just pick every 14th object.
    return outputs
```

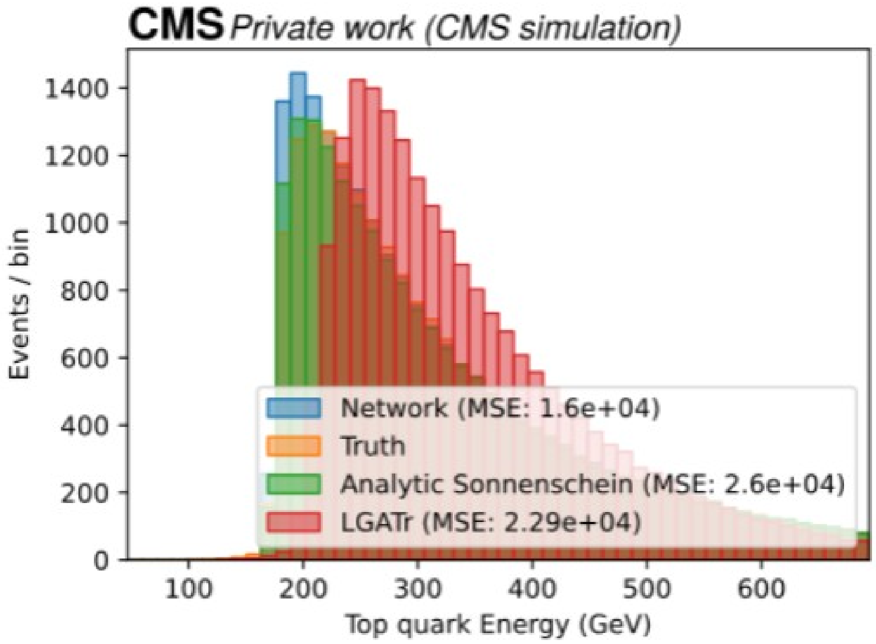
Experiments

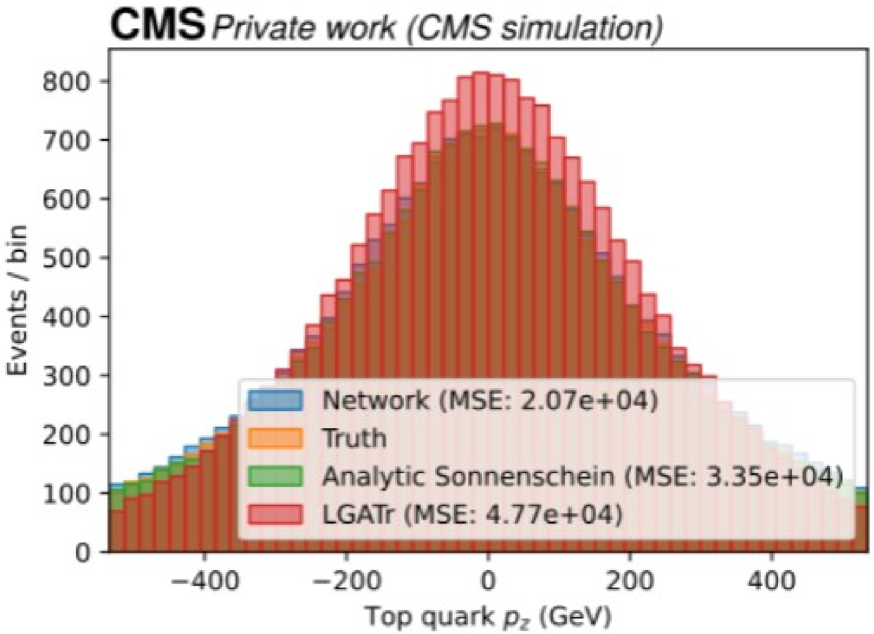
- Credits to Oscar for keeping up with my bullying
- Checked many different configs, training configurations, output aggregation, spurion symmetry breaking, input scaling

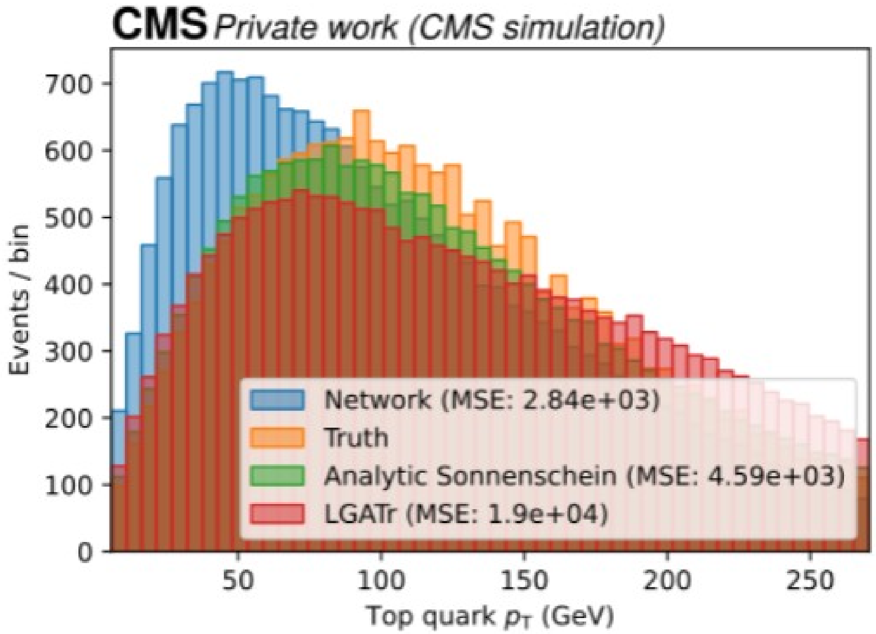


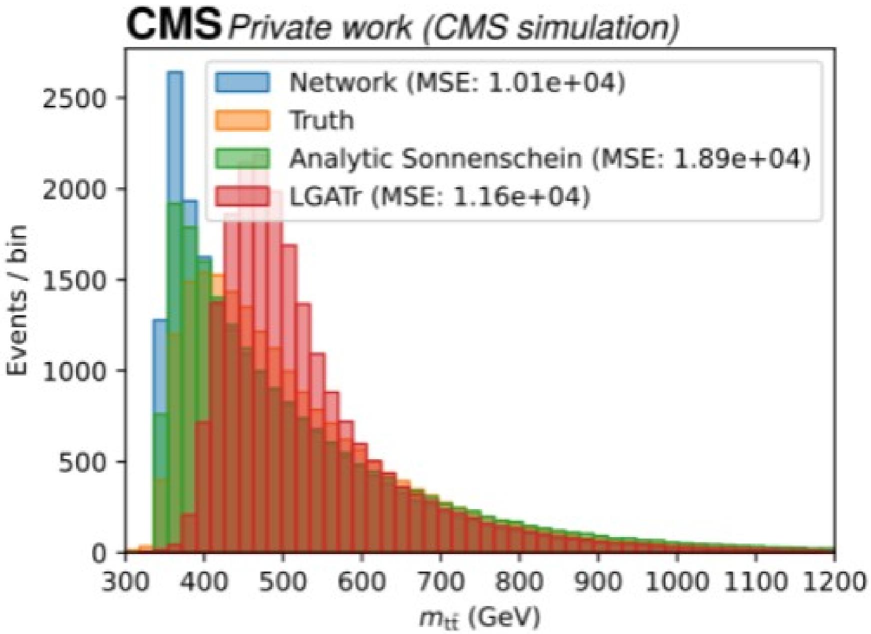
- **Key Findings:**
  - Input needs to be proper 4 vectors, "normalization" can only be done Lorentz Invariant! (think: subtracting 4-vector norm)
  - Performance is very sensitive to Symmetry Breaking (no breaking vs. breaking with x-y plane, breaking with a z-direction beam token)
  - Training Hyperparameter set identified that works reliably
  - Small LGATR configurations work already well, experiments with many parameters outstanding (rn: ~2-4h of training, could scale that!)

Results


CMS Private work (CMS simulation)
Events / bin
Top quark Energy (GeV)
Network (MSE: 1.6e+04)
Truth
Analytic Sonnenschein (MSE: 2.6e+04)
LGATr (MSE: 2.29e+04)


CMS Private work (CMS simulation)
Events / bin
Top quark p_z (GeV)
Network (MSE: 2.07e+04)
Truth
Analytic Sonnenschein (MSE: 3.35e+04)
LGATr (MSE: 4.77e+04)

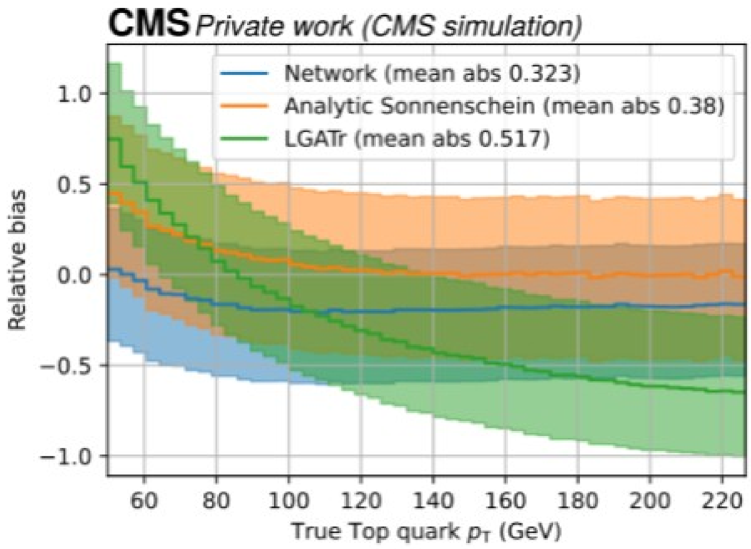

CMS Private work (CMS simulation)
Events / bin
Top quark p_T (GeV)
Network (MSE: 2.84e+03)
Truth
Analytic Sonnenschein (MSE: 4.59e+03)
LGATr (MSE: 1.9e+04)

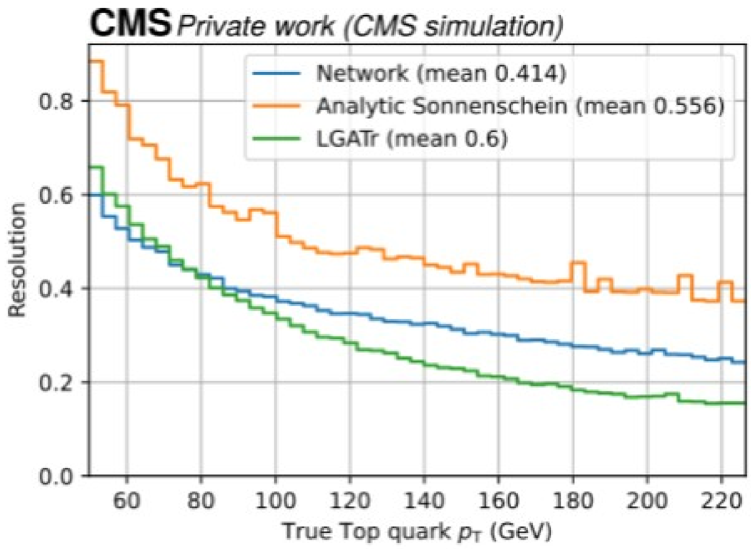

CMS Private work (CMS simulation)
Events / bin
$m_{t\bar{t}}$ (GeV)
Network (MSE: 1.01e+04)
Truth
Analytic Sonnenschein (MSE: 1.89e+04)
LGATr (MSE: 1.16e+04)

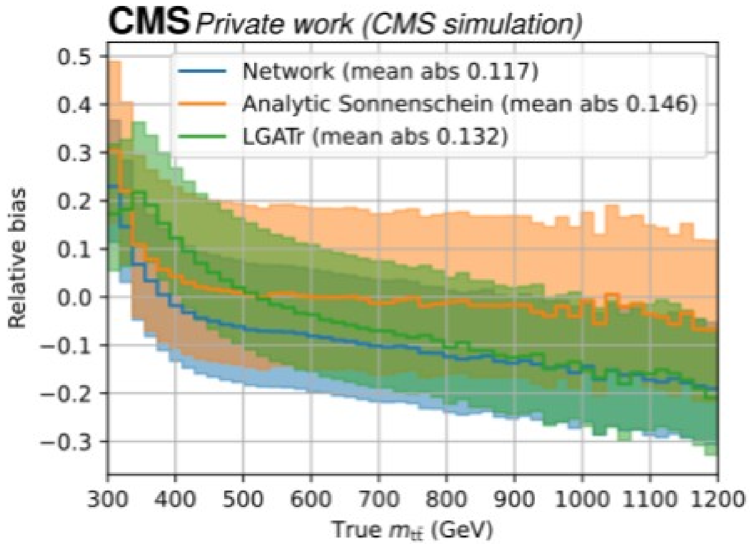
Results

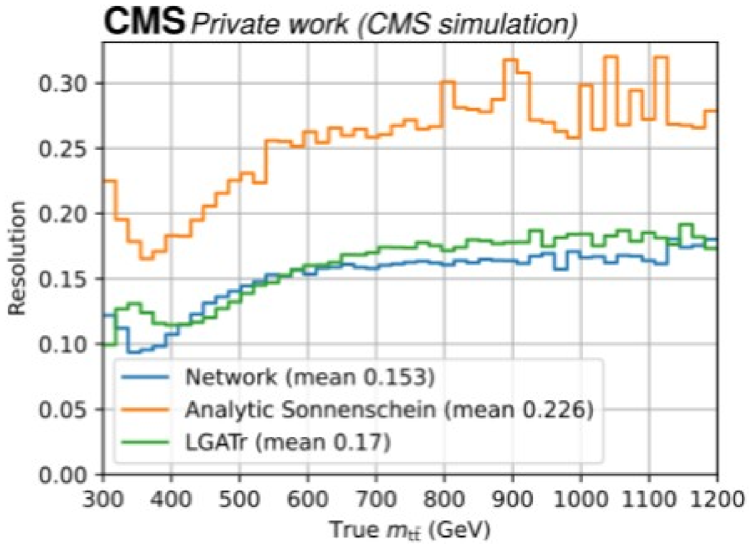
$$\mathrm{RB} = \frac{1}{N}\sum_{i=1}^{N}\frac{p_i - t_i}{t_i}$$

$$\mathrm{Res} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{p_i - t_i}{t_i}\right)^2 - \mathrm{RB}^2}$$









Conclusion

- All the cool kids use transformers
- The even cooler ones use physics informed transformers
- We can get away with very small models (10k-200k parameters) and get very good performance already → more optimization might lead to further advancements
- Have a pipeline for validation and visualization and could also plug the trained model into pepper

Further Ideas

- Implement b-quark scalar token to force the network to match b-quarks correctly
- Add auxiliary terms to loss for important variables to model them better (m_tt)
- Larger networks, longer training   (Jonas’s NNs had 5-10 Mio parameters!)