Scaling laws for amplitude surrogates

Joaquín Iturriza Ramirez

In collaboration with Henning Bahl, Víctor Bresó-Pla and Anja Butter

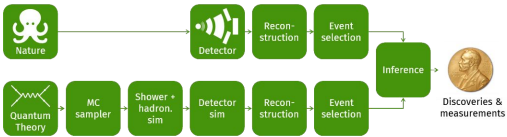
LPNHE, Sorbonne Université / CNRS-IN2P3

arXiv:2601.13308

ML4Jets 2026, Vienna

Amplitudes sit at the start of every simulated event

- $|\mathcal{A}|^2$ maps particle momenta to the probability of an outcome. Every simulated event starts from it.
- Evaluation gets slow with more final-state particles and higher orders.
- Standard fix: a **surrogate**. Evaluate the exact calculation on a sample, fit it, then evaluate for almost free.
- Classical interpolation does not survive in high dimension and is hard to extend with new data. Neural networks handle both.
- Open question: **how much data and compute** does a given accuracy cost?



Top: data. Bottom: simulation. Amplitudes seed the lower chain.

Scaling laws: error falls as a power law

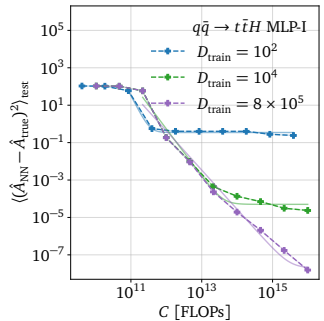
Grow dataset size D, compute C or parameter count N, and the test error drops as a power law until it hits a plateau set by the other two:

$$L(X, Y, Z) = (X_c/X)^{\alpha_X} + K(Y, Z)$$

Seen in language models, vision and jet tagging. **Do amplitudes behave the same way?**

What we scan:

- **11 processes**: $q\bar{q} \to t\bar{t}H$, $q\bar{q} \to Z{+}ng$, $q\bar{q} \to WZ{+}ng$, $q\bar{q} \to WWZ$, $gg \to \gamma\gamma{+}ng$. LO, MadGraph.
- **Architectures**: MLP on Lorentz invariants (MLP-I) and a Lorentz-equivariant LLoCa transformer.
- **Losses**: MSE and a heteroscedastic loss.

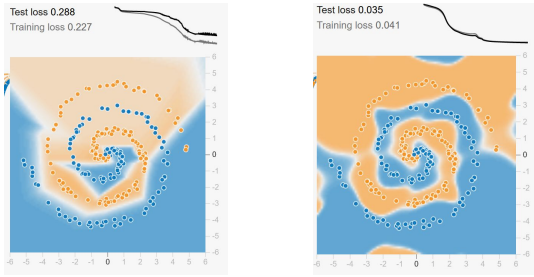


$q\bar{q} \to t\bar{t}H$: error against compute, one curve per D.

If the laws hold, we can cheaply estimate the resources we need.

Putting the symmetry into the network

- A network spends capacity learning the structure of its inputs. Structure supplied for free is structure it does not have to learn.
- Particle physics respects Lorentz symmetry. Two ways to use it:
  - **MLP-I:** feed only Lorentz invariants, $z_{ij} = \log(p_i \cdot p_j)$.
  - **LLoCa transformer:** a Lorentz-equivariant network.



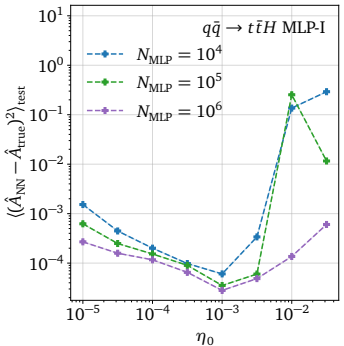
Toy: same net, same spiral data. Raw inputs (x_1, x_2) on the left against $(\sin x_1, \sin x_2)$ on the right, test loss drops about $8\times$.

playground.tensorflow.org

μP, so hyperparameters transfer across widths

- Given D, C and N, in order to meaningfully talk about scaling laws we need to make sure we are reasonably close to the optimal hyperparameters for that configuration.
- μP scales the initialisation and the per-layer learning rates with width, so activations and updates keep the same size as the network grows and the optimum stays put.
- Many hyperparameters transfer across widths, so they can be optimised cheaply in small models.
- In practice, the rest of the hyperparameter optimisation is made simpler, and results are comparable or better than with the regular parametrization.

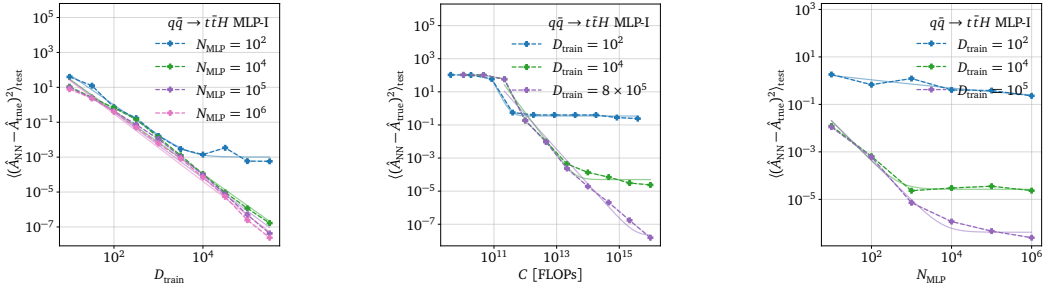
arXiv:2203.03466



$q\bar{q} \to t\bar{t}H$, MLP-I under μP. Test error against the initial learning rate for three widths.

Results

Clean power laws in all three knobs

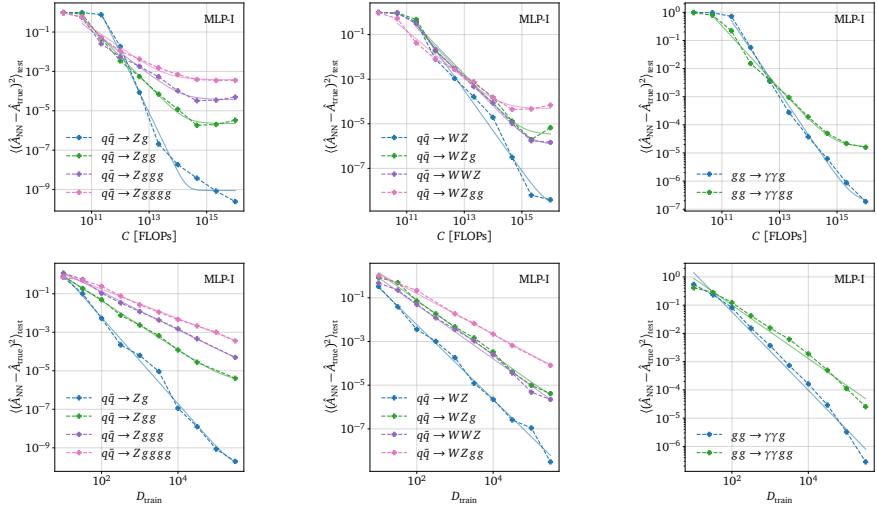


$q\bar{q} \to t\bar{t}H$, MLP-I. Error against D, C and N. Solid lines are the fitted power laws.

- Power laws hold over many orders of magnitude, with consistent slopes.
- **Dataset size is the bottleneck.** Starve D and everything plateaus. Past $\sim 10^4$ parameters, model size barely matters.

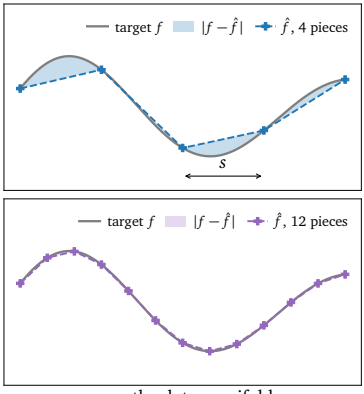
Amplitude surrogates scale as predictably as anything else in deep learning.

The same picture across process families



Where the exponent comes from

- The data live on a manifold of intrinsic dimension d, not in the full input space.
- A ReLU network with N parameters fits a piecewise linear function with about N pieces on that manifold, so each piece has side $s \propto N^{-1/d}$.
- Inside a piece the network gets the linear part right and the leftover error is quadratic in s. The MSE then scales as s^4,
$$L \propto N^{-4/d}.$$
- The same counting with the spacing between training points, $r \propto D^{-1/d}$, gives $L \propto D^{-4/d}$, and since $C \propto N\, D\, N_{\text{epochs}}$ the compute exponent follows: $\alpha_N,\ \alpha_D,\ \alpha_C \gtrsim 4/d$.
- It is a lower bound. A smoother target is learned faster, and the network can pick up some curvature inside a piece.



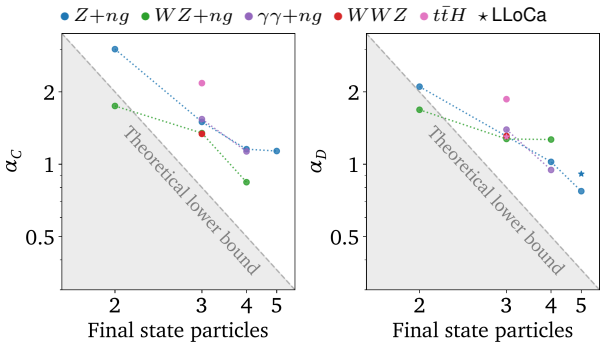
arXiv:2004.10802

The exponent is set by the final state

- The exponent is bounded by the intrinsic dimension d of the data manifold, $\alpha \gtrsim 4/d$.
- For a given process d is **known before training**: the final-state degrees of freedom,

$$d = 3\,n_f - 4.$$

- More final-state particles means a smaller exponent, so a harder surrogate, and we can say by how much in advance.
- Fitted exponents follow that trend across all 11 processes and stay above the bound. Processes with different physics but the same multiplicity land close together.

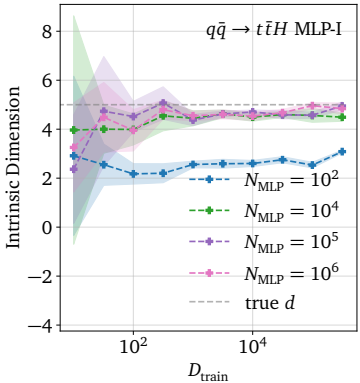


Fitted α_C (left) and α_D (right) against the number of final-state particles. Shaded: the bound $\alpha \gtrsim 4/d$.

$d = 3n_f - 4$ is known before training and puts a lower bound on the exponent.

Measuring the intrinsic dimension

- The bound $\alpha \gtrsim 4/d$ refers to the dimension of the manifold the data lives on, not the dimension of the input vector.
- We estimate it directly from the network with TwoNN, applied to the activations of the second-to-last layer.
- The estimate tracks $3n_f - 4$ once the network has enough data to actually learn the manifold, which is a consistency check rather than a new input.



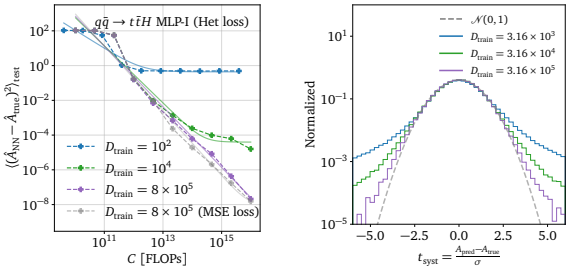
Estimated intrinsic dimension of the learned representation, $q\bar{q} \to t\bar{t}H$.

The same laws hold with calibrated uncertainties

- Replace MSE by a heteroscedastic loss: the network predicts the amplitude $\bar{A}$ *and* a per-point σ,

$$\mathcal{L}_{\text{het}} = \left\langle \frac{(A_{\text{true}} - \bar{A})^2}{2\sigma^2} + \log \sigma \right\rangle$$

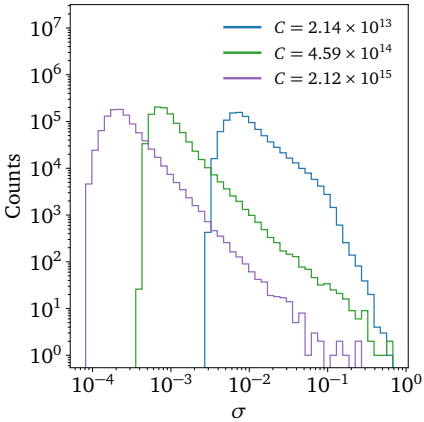
- Same scaling as MSE, at a small cost in accuracy.
- The pull $t = (A_{\text{NN}} - A_{\text{true}})/\sigma$ follows $\mathcal{N}(0,1)$ once the dataset is large enough, so the error bar can be used.
- σ shrinks with D and C along with the error itself (next slide).

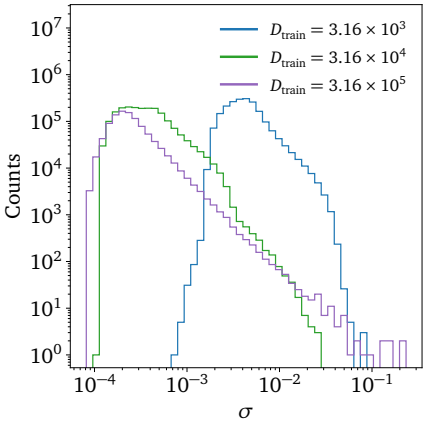


Left: heteroscedastic training, error against compute, grey is the MSE reference. Right: pulls across the dataset-size scan.

An uncertainty-aware surrogate costs one extra output and scales the same way.

The predicted uncertainty shrinks as resources grow





Distribution of the learned σ over the test set as compute (left) and dataset size (right) grow. The whole distribution shifts down, so the network reports a smaller uncertainty exactly where it has become more accurate.

Summary

- Amplitude surrogates follow clean power laws in D, C and N, across 11 processes, two architectures and two losses.
- **Dataset size is what limits you**, not model size.
- The exponent tracks the final-state multiplicity through $d = 3n_f - 4$ and respects the $\alpha \gtrsim 4/d$ bound, so the difficulty of a new process is known in advance.
- Heteroscedastic training gives calibrated uncertainties and the same scaling.

In practice

- Estimate the scaling behaviour from a few cheap runs and know the budget you need for your desired precision.

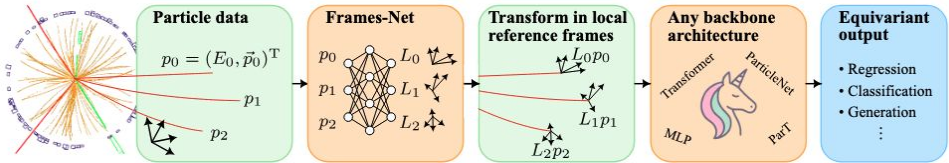
Thank you

arXiv:2601.13308

Questions welcome.

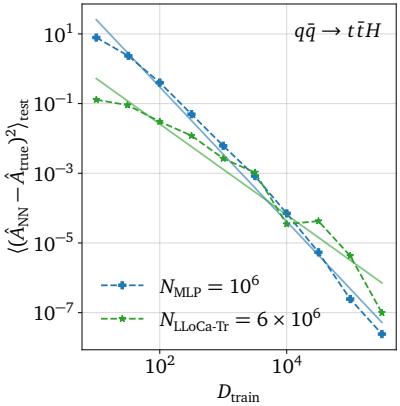
Backup

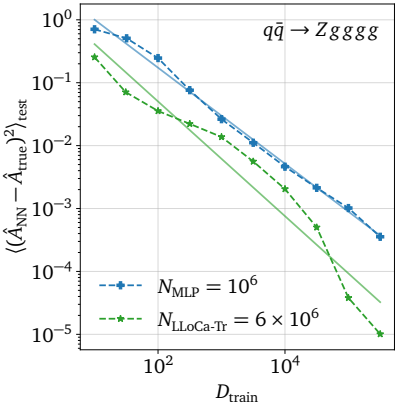
Backup: LLoCa, Lorentz local canonicalization



A small network learns a local reference frame L_i per particle. Vectors are transformed into those frames before message passing, so any backbone becomes Lorentz equivariant: $\mathcal{N}(\mathcal{G}x) = \mathcal{G}\mathcal{N}(x)$. arXiv:2505.20280

Backup: LLoCa transformer against MLP-I





Same exponent for both architectures. The equivariant model is slightly behind on the simpler process (left) and uniformly ahead on the many-gluon one (right), where the permutation and Lorentz structure matters most. Hyperparameter tuning for LLoCa was considerably harder.

Backup: the heteroscedastic loss

MSE: $\mathcal{L}_{\mathrm{MSE}} = \left\langle (A_{\mathrm{true}}(x) - A_{\mathrm{NN}}(x))^2 \right\rangle_{x \sim D_{\mathrm{train}}}$

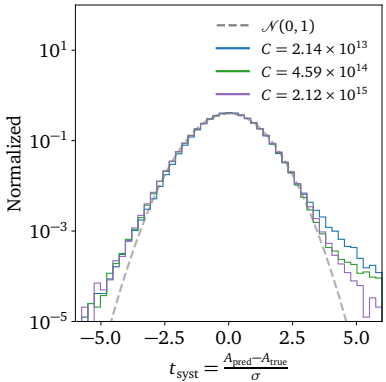
Heteroscedastic: assume $p(A|x) = \mathcal{N}(A \,|\, \bar{A}(x), \sigma^2(x))$ and minimise the negative log-likelihood,

$$\mathcal{L}_{\mathrm{het}} = \left\langle \frac{(A_{\mathrm{true}}(x) - \bar{A}(x))^2}{2\sigma^2(x)} + \log \sigma(x) \right\rangle_{x \sim D_{\mathrm{train}}}$$

The network predicts two outputs, $\bar{A}(x)$ and $\sigma(x)$. If it is calibrated,

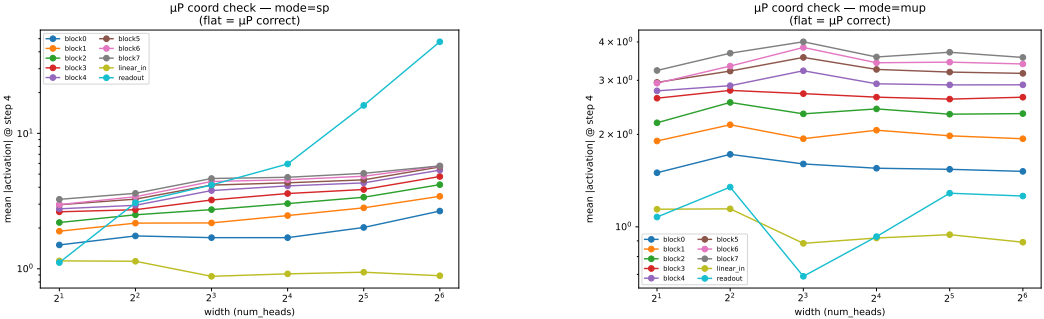
$$t(x) = \frac{A_{\mathrm{NN}}(x) - A_{\mathrm{true}}(x)}{\sigma(x)} \quad \text{follows} \quad \mathcal{N}(0, 1).$$

Backup: calibration against compute



The same pulls across the compute scan rather than the dataset-size scan. They stay on $\mathcal{N}(0,1)$ over orders of magnitude in C.

Backup: μP coordinate check



Coordinate check. Under the standard parametrization (left) activations drift with width, so every width needs its own learning-rate scan. Under μP (right) they stay put and the optimum transfers, which is what makes a scan over N affordable. arXiv:2203.03466