Extrapolating Jet Radiation with Autoregressive Transformers

Anja Butter[1,2], François Charton[3], Javier Mariño Villadamigo[1],
Ayodele Ore[1], Tilman Plehn[1,4], Jonas Spinner[1]
[1]ITP Heidelberg, [2]LPNHE Paris, [3]Meta FAIR, [4]IWR Heidelberg

arXiv:2412.12074

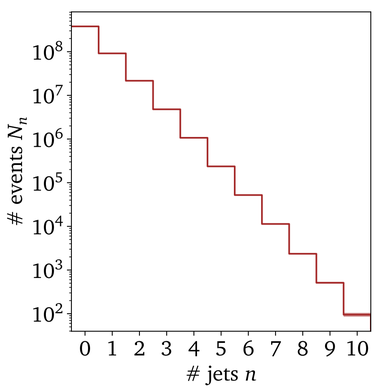
UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386
SPONSORED BY THE
Federal Ministry of Education and Research

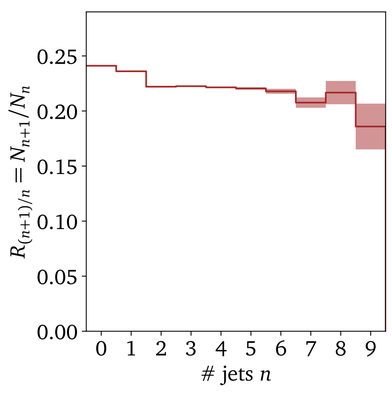
Autoregressive QCD Jet Radiation

Collinear splittings are described by universal collinear splitting kernels $P(z)$, allowing the generation of $n+1$ final-state particles from n final-state particles

$$\sigma_{n+1} \sim \int \frac{dp}{p} dz \frac{\alpha_s}{2\pi} P(z) \sigma_n \, .$$

This universality is exact in the leading approximation. The full simulation, however, includes additional effects that break the universality. There still remain distinctive patterns, Poisson scaling and staircase scaling. The standard example for staircase scaling at the LHC is $pp \to Z + n$ jets.

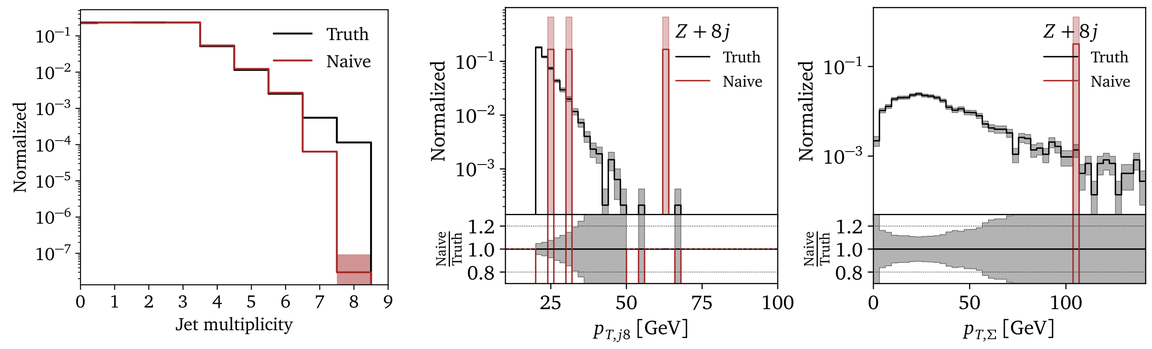




Naive Extrapolation

If the generative network has correctly the universal features of jet radiation, it should be able to extrapolate beyond the training dataset. To this end, we train the network on small multiplicities $n = 0,1,2,3,4,5,6$, then generate events and extract only the events with 8 jets.

Out of the box, the generative network only generates few $n = 8$ events, making it impossible to study their kinematic distribution. This happens because the network also learns the multiplicity distribution in the training data, which vanishes for $n = 8$.



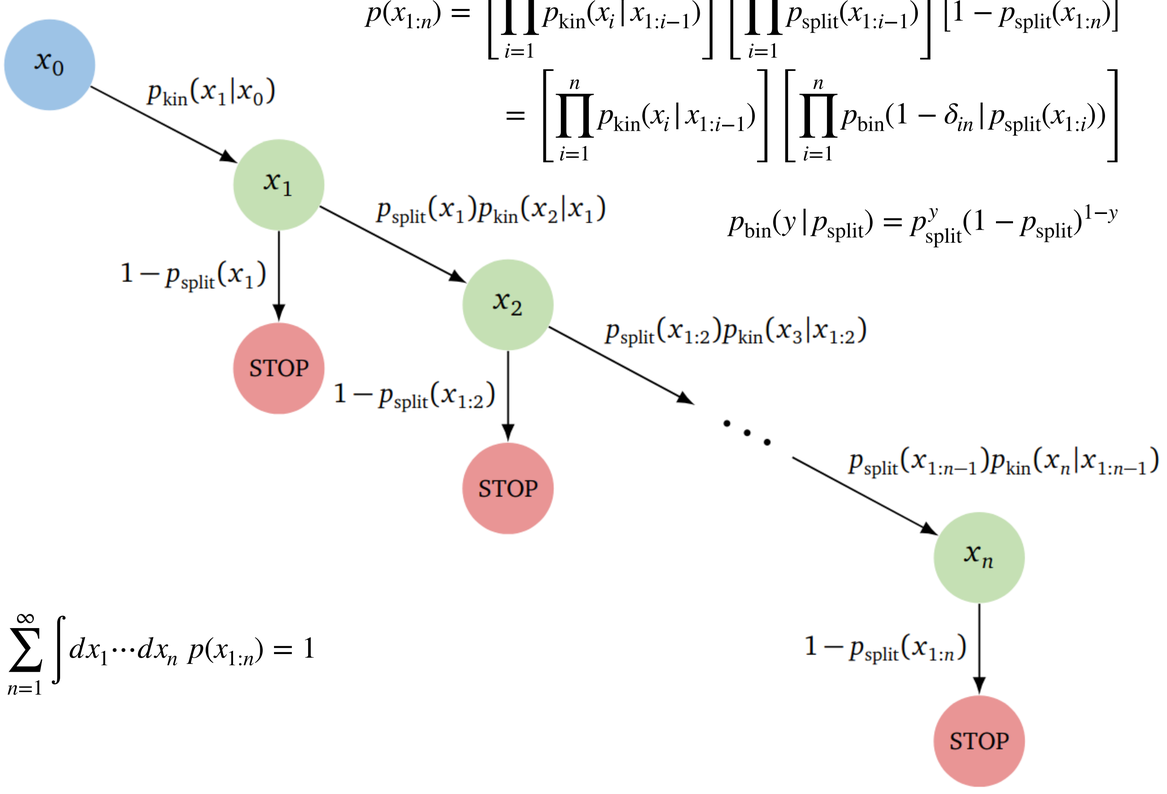
How can we get the generative network to extrapolate beyond the training data?

Factorized Probability

We want to describe the sequential structure of QCD jet radiation using a generative neural network. Inspired by the elementary splitting processes, it should generate jet kinematics $x_1, \ldots x_n$ one after the other and decide dynamically when to terminate the sequence. To this end, we decompose the joint density of jet kinematics $p(x_{1:n}) = p(x_1, \ldots x_n)$ as an autoregressive product of conditional distributions

$$p(x_{1:n}) = \left[\prod_{i=1}^{n} p_{\text{kin}}(x_i \,|\, x_{1:i-1})\right] \left[\prod_{i=1}^{n} p_{\text{split}}(x_{1:i-1})\right] \left[1 - p_{\text{split}}(x_{1:n})\right]$$
$$= \left[\prod_{i=1}^{n} p_{\text{kin}}(x_i \,|\, x_{1:i-1})\right] \left[\prod_{i=1}^{n} p_{\text{bin}}(1 - \delta_{in} \,|\, p_{\text{split}}(x_{1:i}))\right]$$

$$p_{\text{bin}}(y \,|\, p_{\text{split}}) = p_{\text{split}}^{y} (1 - p_{\text{split}})^{1-y}$$



$$\sum_{n=1}^{\infty} \int dx_1 \cdots dx_n \; p(x_{1:n}) = 1$$

Similarly to the autoregressive decomposition of the density over different jets, we factorise the density describing the kinematics (p_T, ϕ, η, m) of an individual jet
We use Gaussian mixture models p_{GM} to parametric the one-dimensional densities,

$$p_{\text{kin}}(x_{i+1} \,|\, x_{1:i}) = p_{\text{GM}}(p_{T,i+1} \,|\, x_{1:i}) \, p_{\text{vMM}}(\phi_{i+1} \,|\, x_{1:i}, p_{T,i+1})$$
$$\times p_{\text{GM}}(\eta_{i+1} \,|\, x_{1:i}, p_{T,i+1}, \phi_{i+1}) \, p_{\text{GM}}(m_{i+1} \,|\, x_{1:i}, p_{T,i+1}, \phi_{i+1}, \eta_{i+1})$$

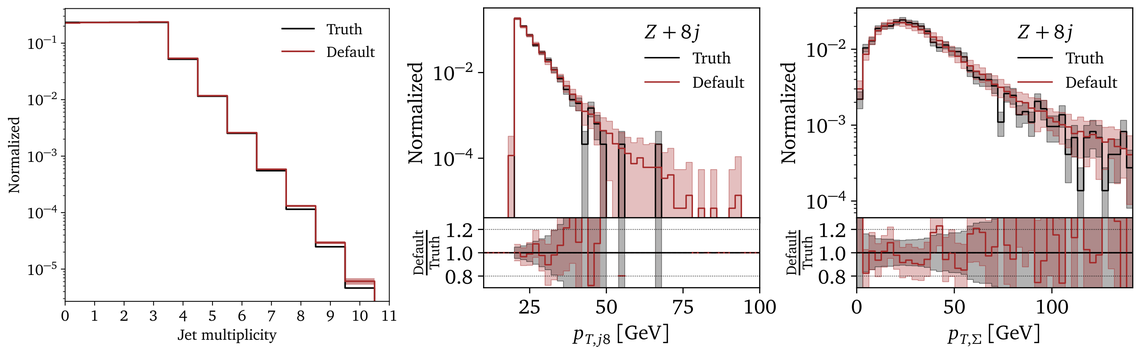
and von-Mises mixture models p_{vMM} for the periodic variable ϕ.

We use two transformer encoders, one for correlating jet properties and one for correlating jet kinematic properties. The conditional structure is implemented with causal masks. The whole network is trained on the log-likelihood objective

$$\mathscr{L}_{\text{like}} = -\left\langle \sum_{j=1}^{n} \log p_{\text{kin}}(x_i \,|\, x_{1:i-1}) + \sum_{i=0}^{n} \log p_{\text{bin}}(1 - \delta_{in} \,|\, p_{\text{split}}(x_{1:i}) \right\rangle$$

We use Bayesian neural networks to capture uncertainties from the training process.

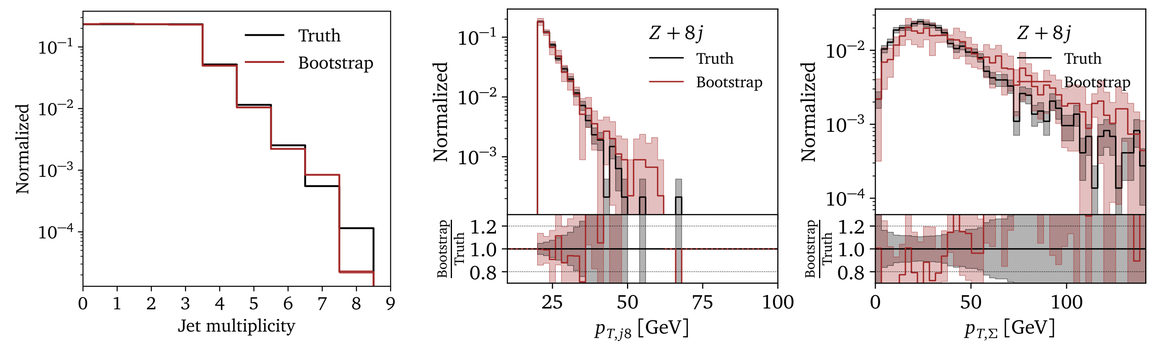
Finally, we evaluate network performance when trained on the $Z + n$ jets dataset with $n = 0 \ldots 10$, with capped event counts for $n = 0,1,2$:



Bootstrap Extrapolation

A straight-forward method to modify the learned multiplicity distribution is to generate $n = 7$ events and eventually $n = 8$ alongside the training and append them to the training dataset in the required ratios.

In practice this approach requires careful tuning: If bootstrapped events are included too early in the training, the network did not have the chance to learn their kinematics to high precision, and the additional high-multiplicity events will be of poor quality. On the other hand, once the generative network has learned the kinematic distributions to high precision, it has also learned the multiplicity distribution well, and bootstrapping becomes inefficient.

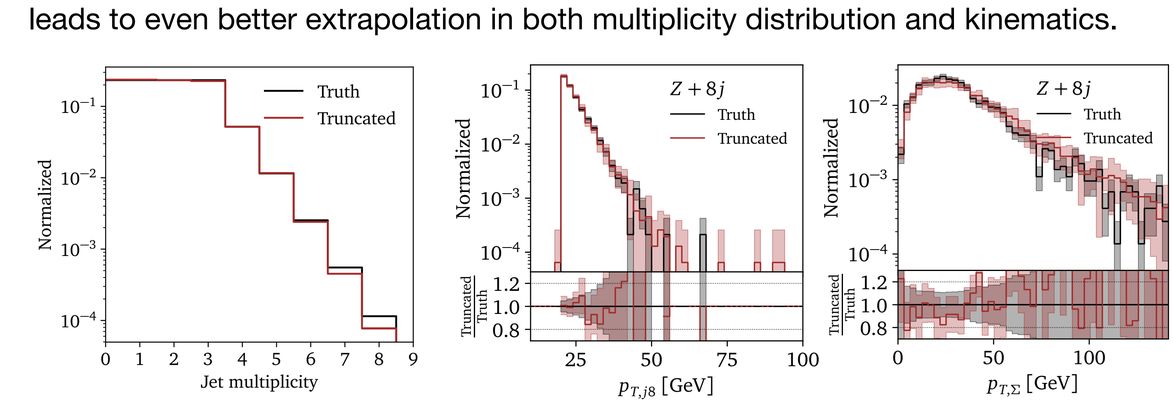


Truncated Loss Extrapolation

Instead of modifying the training data, one can modify the loss function to inform the network about the need for high-multiplicity events. The problem of standard log-likelihood training is that the network is forced to always predict $p_{\text{split}}(x_{1:n_{\text{max}}}) = 0$ after generating the last jet, $n_{\text{max}} = 6$. This constraint can be removed by removing this loss term, leading to the *truncated* loss

$$\mathscr{L}_{\text{trunc}} = -\left\langle \sum_{j=1}^{n} \log p_{\text{kin}}(x_i \,|\, x_{1:i-1}) + \sum_{i=0}^{n} (1 - \delta_{in_{\text{max}}}) \log p_{\text{bin}}(1 - \delta_{in} \,|\, p_{\text{split}}(x_{1:i}) \right\rangle$$

In contrast to the bootstrapping approach, this method does not introduce any additional hyperparameters. This makes the training procedure simple and stable, and leads to even better extrapolation in both multiplicity distribution and kinematics.



Nett architecture.
But have you tried "pip install lgatr"?

ITP
Baden-Württemberg
THE LÄND