Refining fast simulation using machine learning

Samuel Bein[1], *Patrick* Connor[1,2], *Kevin* Pedro[3], *Peter* Schleper[1], and *Moritz* Wolf[1,*]
(on behalf of the CMS Collaboration)

[1]University of Hamburg, Institut für Experimentalphysik, Germany
[2]Center for Data and Computing in Natural Sciences, Hamburg, Germany
[3]Fermi National Accelerator Laboratory, Batavia, IL, USA

Abstract. At the CMS experiment, a growing reliance on the fast Monte Carlo application (FastSim) will accompany the high luminosity and detector granularity expected in Phase 2. The FastSim chain is roughly 10 times faster than the application based on the GEANT4 detector simulation and full reconstruction referred to as FullSim. However, this advantage comes at the price of decreased accuracy in some of the final analysis observables. In this contribution, a machine learning-based technique to refine those observables is presented. We employ a regression neural network trained with a sophisticated combination of multiple loss functions to provide post-hoc corrections to samples produced by the FastSim chain. The results show considerably improved agreement with the FullSim output and an improvement in correlations among output observables and external parameters. This technique is a promising replacement for existing correction factors, providing higher accuracy and thus contributing to the wider usage of FastSim.

1 Introduction

Simulating particle collisions, the subsequent detector response, and the reconstruction of the final state are crucial for modern high energy physics. For the purpose of simulating events in the CMS detector [1], the collaboration largely relies on a simulation chain based on GEANT4 [2, 3], referred to as FullSim, which gives an accurate representation of the truth [4, 5]. However, this requires a considerable amount of computing power. Therefore, another simulation chain has been established, which uses approximations to speed up the process by roughly a factor of 10 [6–8]. This application, known as FastSim, provides output with the same format and structure as FullSim. Looking towards the future with higher LHC luminosity and increased CMS detector granularity [9], FastSim will only gain in importance as the collaboration strives to keep the computing needs within budget in Phase 2 [10, 11].

The output of the FastSim chain is generally in good agreement with the FullSim chain, but discrepancies on the order of up to 20% are observed in some analysis observables. Traditionally, differences between simulation samples (or differences between simulation and data) are treated with dedicated correction factors or weights that are derived either by physics object groups or by individual users carrying out analyses. These corrections or weights are applied to events or physics objects to correct certain biases, for example in the transverse

*e-mail: moritz.wolf@cern.ch

momentum or selection efficiency of jets, photons, or leptons. One approach that goes beyond these traditional corrections in terms of sophistication and accuracy is the application of weights that are derived using machine learning-based methods, such as the DCTR (deep neural networks using classification for tuning and reweighting) approach [12]. While the accuracy of simulated variables, as well as correlations among the variables, is improved compared to the unweighted sample, the use of weights reduces the statistical power, undermining the advantage of fast simulation applications. In contrast to reweighting, the aim of our method is to change the values of the sample to reach better agreement with the target sample, without the need for weights. Such a *refinement* has been studied and tested in air shower images using a Wasserstein GAN [13], demonstrating the possibility of improving the accuracy of fast particle reconstruction. Lower-level refinement of fast detector simulations, at the level of simulated energy deposits or detector hits, has also been demonstrated [14, 15]. We have developed a distinct method that operates on the level of final analysis observables and employs a simple regression neural network. This approach may be considered a natural replacement for the traditional correction factors currently applied to FastSim.

2 Data sample

For the purpose of this study, the pair production of supersymmetric partners of gluons, namely, gluinos, is simulated. Each gluino subsequently decays to a top quark pair and a neutralino (supersymmetric simplified model T1tttt [16]). Events are simulated with both the FullSim and the FastSim chain up to and including the step producing NanoAOD output [17], which consists of high-level observables for various physics objects. Both cases share the same event generation using PYTHIA8, referred to as the GEN step. Therefore, after the detector response and the reconstruction have been simulated, a matching can be established between a true generator-level jet and its reconstructed counterparts in the FullSim and FastSim samples; all jets are clustered using the anti-k_T algorithm [18, 19] with a distance parameter of 0.4. The matching is achieved with a distance matching criterion of $\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2} < 0.2$. In this way, jet triplets are constructed in the form (GEN, FullSim, FastSim). All jets need to fulfill the requirement that no neighboring jet is closer than $\Delta R = 0.5$ avoiding overlapping jets. Otherwise, no additional selection is applied. Jets have transverse momenta as low as $p_T = 15\,\text{GeV}$ and the range of pseudorapidity is fully inclusive. The final data sample used for training consists of roughly 6 million jet triplets.

3 Method

A vector of analysis observables simulated by the FastSim chain is defined as $\vec{x}^{\text{Fast}}$ and can be compared to $\vec{x}^{\text{Full}}$, the corresponding FullSim output. A fully-connected feed-forward neural network is trained to provide an output $\vec{x}^{\text{Refined}}$ for a given input vector $\vec{x}^{\text{Fast}}$. The aim is to establish a *refined* version of the FastSim data sample, which is more similar to the FullSim output, i.e., more accurate.

The analysis observables used in this study are four jet flavor tagging observables available in the CMS NanoAOD data analysis format:

$$\vec{x} = \begin{pmatrix} \text{B} & \text{CvB} & \text{CvL} & \text{QG} \end{pmatrix}^{\text{T}} . \tag{1}$$

They are calculated from the output of the DeepJet algorithm [20], a multiclass neural network with six output nodes, activated with a softmax function. The nodes correspond to jets containing hadronically (leptonically) decaying b hadrons (labeled b (lepb)), jets containing

two b hadrons (bb), and jets from c quarks (c), light quarks (uds), and gluons (g). From those values, the four discriminator observables are calculated as

$$\mathrm{B} = \mathrm{b} + \mathrm{bb} + \mathrm{lepb}\,,\ \mathrm{CvB} = \frac{\mathrm{c}}{\mathrm{c} + \mathrm{b} + \mathrm{bb} + \mathrm{lepb}}\,,\ \mathrm{CvL} = \frac{\mathrm{c}}{\mathrm{c} + \mathrm{uds} + \mathrm{g}}\,,\ \mathrm{QG} = \frac{\mathrm{g}}{\mathrm{g} + \mathrm{uds}}\,. \tag{2}$$

The refinement network is parametrized by a vector of parameters $\vec{y}$, including the true transverse momentum of the jet $p_{\mathrm{T}}^{\mathrm{GEN}}$, the true pseudorapidity η^{GEN}, and the true hadron flavor.

3.1 Network architecture

The neural network is composed of five residual blocks, which each consist of two linear layers with 1024 nodes and a skip connection adding the input of the first layer back to the output of the second layer. This ResNet-like architecture [21] is motivated by the fact that the FastSim output is already a good approximation of the FullSim output and we only want to apply a residual correction. The Leaky ReLU activation function (with a slope of 0.01 for negative input values) is used between layers whereas no activation function is applied to the output of the very last layer. During training, dropout with a rate of 50% is used to avoid overtraining.

Additionally, the input variables and parameters are transformed before they are passed to the network. The logit transformation is used for the DeepJet discriminators. For the true transverse momentum $p_{\mathrm{T}}^{\mathrm{GEN}}$, values are first scaled to the interval $(0, 1)$ by applying the transformation $\tanh \frac{p_{\mathrm{T}}^{\mathrm{GEN}}}{200}$ and then also logit-transformed. The true hadron flavor is one-hot-encoded and the true pseudorapidity η^{GEN} is not transformed.

A postprocessing layer is appended to the network to apply the inverse transformations and to enforce the constraint that the refined DeepJet discriminators (B, CvB, CvL, QG) have to be constructed such that the sum of the original DeepJet output nodes (b, bb, lepb, c, uds, g) is equal to unity. This is done by analytically calculating the values of the output nodes according to Eq. (2) and then dividing each value by the sum. The network is implemented using the PyTorch package [22] and its architecture is summarized in Fig. 1.

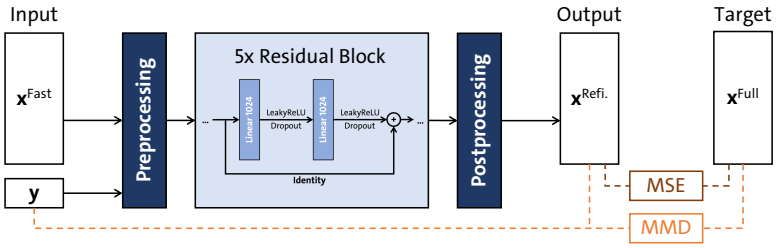

Figure 1: The refinement network is built from residual blocks embedded in pre- and post-processing layers. The inputs to the loss functions are indicated by the dashed lines.

3.2 Loss terms

The most widely used loss function for regression tasks is the mean-squared-error (L2) loss, defined for a batch of n output-target pairs as:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\vec{x}_i^{\,\mathrm{Refined}} - \vec{x}_i^{\,\mathrm{Full}}\right)^2 . \tag{3}$$

To devalue outliers while keeping a smooth behavior around 0, we use the Huber loss function, which is a combination of the mean-squared-error and mean-absolute-error:

$$\text{Huber} = \frac{1}{n}\sum_{i=1}^{n} h_i\,, \tag{4}$$

$$h_i = \begin{cases} 0.5\left(\vec{x}_i^{\,\text{Refined}} - \vec{x}_i^{\,\text{Full}}\right)^2 & \text{if } \left|\vec{x}_i^{\,\text{Refined}} - \vec{x}_i^{\,\text{Full}}\right| < \delta \\ \delta\left(\left|\vec{x}_i^{\,\text{Refined}} - \vec{x}_i^{\,\text{Full}}\right| - 0.5\delta\right) & \text{otherwise} \end{cases} \qquad \text{with } \delta = 0.1\,. \tag{5}$$

Both of these functions measure distances between fixed output-target pairs and thus serve to render each FastSim jet more similar to the corresponding FullSim jet. While this addresses deterministic biases, the independent stochasticity in both simulation chains prevents the refinement of full distributions. Training only with such a loss term consistently leads to a regression to the mean as fluctuations towards small (large) values are systematically corrected to larger (smaller) values. This problem is overcome by using an ensemble-based loss function, the maximum mean discrepancy (MMD) [23].

The sample estimate of the MMD is calculated by evaluating a kernel function for all possible pairs of output-output, target-target, and output-target samples defined by $\vec{x}$. The parameter vector $\vec{y}$ is concatenated to the vectors $\vec{x}$ to include correlations between the variables and parameters:

$$\begin{aligned} \text{MMD}_\sigma =& \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k_\sigma\left(\begin{pmatrix}\vec{x}_i^{\,\text{Refined}}\\ \vec{y}_i\end{pmatrix},\begin{pmatrix}\vec{x}_j^{\,\text{Refined}}\\ \vec{y}_j\end{pmatrix}\right) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k_\sigma\left(\begin{pmatrix}\vec{x}_i^{\,\text{Full}}\\ \vec{y}_i\end{pmatrix},\begin{pmatrix}\vec{x}_j^{\,\text{Full}}\\ \vec{y}_j\end{pmatrix}\right) \\ & - \frac{2}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k_\sigma\left(\begin{pmatrix}\vec{x}_i^{\,\text{Refined}}\\ \vec{y}_i\end{pmatrix},\begin{pmatrix}\vec{x}_j^{\,\text{Full}}\\ \vec{y}_j\end{pmatrix}\right). \end{aligned} \tag{6}$$

We use a Gaussian kernel function $k_\sigma(x_1, x_2) = \exp\left(-\frac{1}{\sigma}\|x_1 - x_2\|^2\right)$ with bandwidth σ. The values for MMD_σ with five different bandwidths are added to make up the MMD loss:

$$\text{MMD} = \sum\nolimits_{\sigma\,\in\,\{\frac{\sigma_0}{100},\,\frac{\sigma_0}{10},\,\sigma_0,\,10\sigma_0,\,100\sigma_0\}} \text{MMD}_\sigma\,, \tag{7}$$

where the central bandwidth is the mean L2 distance between all considered pairs:

$$\begin{aligned} \sigma_0 =& \frac{1}{4}\left(\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left\|\begin{pmatrix}\vec{x}_i^{\,\text{Refined}}\\ \vec{y}_i\end{pmatrix} - \begin{pmatrix}\vec{x}_j^{\,\text{Refined}}\\ \vec{y}_j\end{pmatrix}\right\|^2 + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left\|\begin{pmatrix}\vec{x}_i^{\,\text{Full}}\\ \vec{y}_i\end{pmatrix} - \begin{pmatrix}\vec{x}_j^{\,\text{Full}}\\ \vec{y}_j\end{pmatrix}\right\|^2 \right. \\ & \left. + \frac{2}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left\|\begin{pmatrix}\vec{x}_i^{\,\text{Refined}}\\ \vec{y}_i\end{pmatrix} - \begin{pmatrix}\vec{x}_j^{\,\text{Full}}\\ \vec{y}_j\end{pmatrix}\right\|^2\right). \end{aligned} \tag{8}$$

3.3 MDMM algorithm

The above two losses are partially correlated, but cannot, in general, be extremized simultaneously. To arrive at an optimal set of influences from each loss, we use an algorithm called the Modified Differential Method of Multipliers (MDMM), introduced in Ref. [24]. The MDMM reformulates the training as a Lagrangian optimization process. Identifying a primary loss $f(\theta)$ and an additional loss $g(\theta)$, which both depend on the network parameters θ, the Lagrangian becomes

$$\mathcal{L}(\theta, \lambda) = f(\theta) - \lambda\left(\varepsilon - g(\theta)\right)\,. \tag{9}$$

Hence, the objective is to minimize the primary loss subject to the constraint $g(\theta) = \varepsilon$. The algorithm uses stochastic gradient descent for the network parameters θ but gradient ascent for the Lagrange multiplier λ, meaning that the weight of the additional loss is dynamically updated during the training. Additionally, a damping term $0.5\,(\varepsilon - g(\theta))^2$ is added to the Lagrangian to ensure convergence. The implementation of the MDMM algorithm is adapted from the mdmm package [25].

3.4 Training

The network is trained with 500 batches of 4096 jet triplets for 100 epochs. An additional 1000 batches of 4096 jet triplets are used as validation and test datasets with a 50/50 split. Different loss schemes are investigated both with and without the MDMM algorithm: training versions without MDMM are performed with only the MMD loss, only the Huber loss, and with a simple sum of the two. For the training versions with MDMM, MMD is taken as the primary loss while Huber is the additional loss, and different values of ε are explored. The MDMM algorithm uses the Adam optimizer with a learning rate of 0.0001. Fig. 2 shows the convergence of all training versions in the plane of the two loss functions. The value of MMD(Refined, FullSim) is normalized to the baseline value of MMD(FastSim, FullSim), meaning that the starting points are all very close to 1 and values smaller than 1 indicate improved agreement between the distributions.

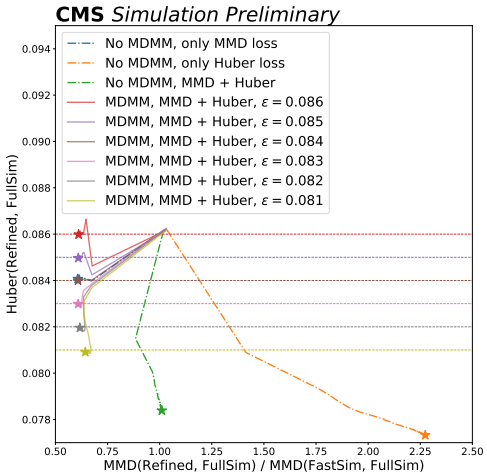

Figure 2: Convergence of different network trainings with and without the MDMM algorithm shown in the plane of the two loss terms, Huber and MMD. The training endpoints are marked with stars and for trainings using the MDMM algorithm, the values of ε are indicated by horizontal lines.

The version with only the Huber loss, while improving the agreement between the fixed corresponding jet-jet pairs (the value on the y-axis), drastically worsens the agreement between the ensembles (given by the value on the x-axis) demonstrating regression to the mean as expected. Also, a naive addition of MMD and Huber does not lead to a better distribution-level agreement. If, however, the network is trained using only the MMD loss, an improvement on both axes can be observed. Yet, the point on the Pareto front (the set of all optimal solutions) to which the training converges cannot be chosen directly in such a setup. This

could be achieved indirectly by experimenting with different fixed weights in the sum of the two loss terms, but when using the MDMM algorithm, the Pareto front can be scanned directly by choosing different values for ε. We observe a convex Pareto front indicating the tradeoff between the two loss terms. In the following section, the results are shown for the training with MDMM using $\varepsilon = 0.084$, which converges close to the MMD-only training.

Training the network takes approximately five hours on a NVIDIA Tesla V100 GPU and the inference within the complete FastSim chain increases the total processing time by 0.2–0.9%, depending on the number of threads.

4 Results

Fig. 3 shows the distributions of the four DeepJet discriminators for the three datasets: FullSim, FastSim, and refined FastSim. It can be seen that the refinement improves the accuracy of FastSim, as quantified by the agreement with the FullSim output. Furthermore, Fig. 4 shows that correlations within the considered variables and between the variables and parameters have better agreement after refinement.

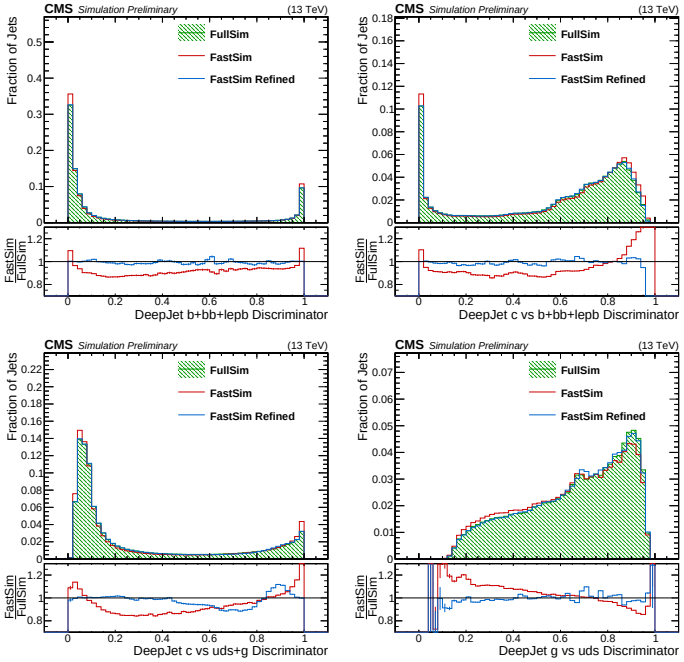

Figure 3: The distributions of the four DeepJet discriminators B (upper left), CvB (upper right), CvL (lower left), and QG (lower right) for FullSim, FastSim, and the refined version of FastSim.

We also compute values for the Fréchet Physics Distance (FPD) and Kernel Physics Distance (KPD) introduced in Ref. [26] and calculated with the JetNet package [27, 28], shown in Table 1. The metrics are computed in the four-dimensional space of the original (not logit-transformed) DeepJet discriminators. The metric values for refined FastSim are similar in magnitude to a baseline comparison of FullSim to itself, an order of magnitude improvement

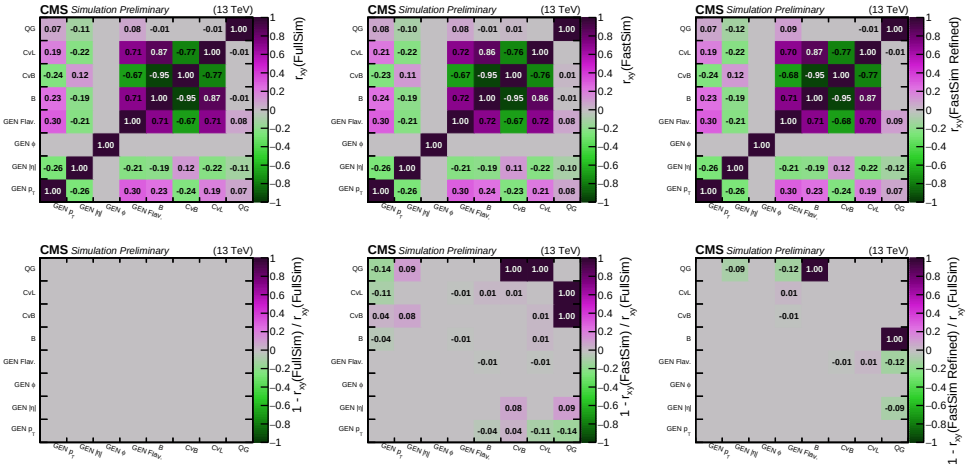

Figure 4: Upper: the Pearson correlation coefficients (rounded to two digits) for all pairs of variables and parameters for FullSim (left), FastSim (center), and refined FastSim (right). Lower: the relative difference between the correlation factors for each simulation and the respective FullSim value with the same axes. All entries in the bottom left plot are zero by construction and a relative difference of 100% indicates cases where the correlation factor for FullSim or for (refined) FastSim in the upper plot is zero.

compared to unrefined FastSim. The trained network is also evaluated on a sample of 6 million jet triplets from simulated top quark pair production ($t\bar{t}$) events and the corresponding metric values are stated in Table 1. This indicates that the method generalizes to different event topologies.

Table 1: The Fréchet and Kernel Physics Distance metrics computed using the DeepJet discriminators to quantify the agreement between the different simulation approaches for multiple physics processes. The last row serves as a baseline, computed by comparing the first half of the FullSim sample to the second half.

FullSim vs.	FPD $\times 10^3$	KPD $\times 10^3$	FPD $\times 10^3$ ($t\bar{t}$)	KPD $\times 10^3$ ($t\bar{t}$)
FastSim	0.801 ± 0.046	1.07 ± 0.58	0.540 ± 0.036	0.927 ± 0.448
FastSim Refined	0.071 ± 0.025	0.083 ± 0.418	0.065 ± 0.025	-0.127 ± 0.164
FullSim	0.061 ± 0.029	-0.024 ± 0.250	0.061 ± 0.024	-0.119 ± 0.167

5 Conclusions

We have introduced a novel approach to refine the output of the CMS FastSim chain to better match the FullSim chain. A regression neural network is trained with the maximum mean discrepancy (MMD) as the primary loss, formulating an ensemble-based training objective not relying on fixed output-target pairs. Additionally, we incorporate the pair-based Huber loss as a constraint on the training via the modified differential method of multipliers. We apply the method to a sample of jets simulated with both FastSim and FullSim, attempting the

refinement of four DeepJet flavor-tagging discriminators. The results show a clear improvement in agreement with the FullSim output in one-dimensional projections, linear correlation coefficients, and dedicated metrics. Besides the improved accuracy, another advantage of this method is the absence of weights, which would reduce the statistical power of the simulated sample. Stability and convergence were observed for all network trainings. The method can be straightforwardly extended to other variables, and a refinement of simulation directly to data is possible using the MMD loss, which does not require labeled pairs of events or objects.

References

[1] CMS Collaboration, JINST **3**, S08004 (2008)

[2] S. Agostinelli et al., Nucl. Instrum. Meth. A **506**, 250 (2003)

[3] J. Allison et al., Nucl. Instrum. Meth. A **835**, 186 (2016)

[4] D.J. Lange, M. Hildreth, V.N. Ivantchenko, I. Osborne (CMS), J. Phys. Conf. Ser. **608**, 012056 (2015)

[5] M. Hildreth, V.N. Ivanchenko, D.J. Lange (CMS), J. Phys. Conf. Ser. **898**, 042040 (2017)

[6] S. Abdullin, P. Azzi, F. Beaudette, P. Janot, A. Perrotta, J. Phys. Conf. Ser. **331**, 032049 (2011)

[7] A. Giammanco, J. Phys. Conf. Ser. **513**, 022012 (2014)

[8] S. Sekmen (CMS), PoS **ICHEP2016**, 181 (2016), 1701.03850

[9] CMS Collaboration, CMS Technical Design Report CERN-LHCC-2017-023, CMS-TDR-019, CERN (2017), https://cds.cern.ch/record/2293646

[10] K. Pedro (CMS), Eur. Phys. J. Web Conf. **245**, 02020 (2020), 2004.02327

[11] CMS Offline Software and Computing, Tech. rep., CERN, Geneva (2022), https://cds.cern.ch/record/2815292

[12] A. Andreassen, B. Nachman, Phys. Rev. D **101**, 091901 (2020)

[13] M. Erdmann, L. Geiger, J. Glombitza, D. Schmidt, Comput. Softw. Big Sci. **2**, 4 (2018), 1802.03325

[14] S. Banerjee, B.C. Rodriguez, L. Franklin, H.G. De La Cruz, T. Leininger, S. Norberg, K. Pedro, A. Rosado Trinidad, Y. Ye, J. Phys. Conf. Ser. **2438**, 012079 (2023), 2202.05320

[15] S. Diefenbacher, V. Mikuni, B. Nachman (2023), 2308.12339

[16] D. Alves (LHC New Physics Working Group), J. Phys. G **39**, 105005 (2012), 1105.2838

[17] M. Peruzzi, G. Petrucciani, A. Rizzi, for the CMS Collaboration, J. Phys. Conf. Ser. **1525**, 012038 (2020)

[18] M. Cacciari, G.P. Salam, G. Soyez, JHEP **04**, 063 (2008), 0802.1189

[19] M. Cacciari, G.P. Salam, G. Soyez, Eur. Phys. J. C **72**, 1896 (2012), 1111.6097

[20] E. Bols, J. Kieseler, M. Verzetti, M. Stoye, A. Stakia, JINST **15**, P12012 (2020)

[21] K. He, X. Zhang, S. Ren, J. Sun, *Deep Residual Learning for Image Recognition*, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., in *Advances in Neural Information Processing Systems 32* (Curran Associates, Inc., 2019), pp. 8024–8035

[23] A. Gretton, K.M. Borgwardt, M.J. Rasch, B. Schölkopf, A. Smola, JMLR **13**, 723 (2012)

[24] J. Platt, A. Barr, *Constrained Differential Optimization*, in *Neural Information Processing Systems*, edited by D. Anderson (American Institute of Physics, 1987), Vol. 0

[25] K. Crowson, *mdmm*, [software] (2021), version 0.1.3, https://github.com/crowsonkb/mdmm

[26] R. Kansal, A. Li, J. Duarte, N. Chernyavskaya, M. Pierini, B. Orzari, T. Tomei, Phys. Rev. D **107**, 076017 (2023), 2211.10295

[27] R. Kansal, J. Duarte, H. Su, B. Orzari, T. Tomei, M. Pierini, M. Touranakou, J.R. Vlimant, D. Gunopulos, *Particle Cloud Generation with Message Passing Generative Adversarial Networks*, in *35th Conference on Neural Information Processing Systems* (2021), 2106.11535

[28] R. Kansal, J. Duarte, C. Pareja, L. Action, Z. Hao, mova, *jet-net/jetnet: v0.2.3.post3* (2023), https://doi.org/10.5281/zenodo.7778868