The MadNIS reloaded

Theo Heimel[1], Nathan Huetsch[1], Fabio Maltoni[2,3], Olivier Mattelaer[2], Tilman Plehn[1] and Ramon Winterhalder[2]

1 Institut für Theoretische Physik, Universität Heidelberg, Germany
2 CP3, Université catholique de Louvain, Louvain-la-Neuve, Belgium
3 Dipartimento di Fisica e Astronomia, Universitá di Bologna, Italy

Abstract

In pursuit of precise and fast theory predictions for the LHC, we present an implementation of the MadNIS method in the MadGraph event generator. A series of improvements in MadNIS further enhance its efficiency and speed. We validate this implementation for realistic partonic processes and find significant gains from using modern machine learning in event generators.

Received 17-11-2023
Accepted 19-06-2024
Published 29-07-2024
doi:10.21468/SciPostPhys.17.1.023

Contents

1 Introduction

One of the defining aspects of the precision-LHC program is that we compare vast amounts of scattering data with first-principle predictions. These predictions come from multi-purpose LHC event generators, for instance PYTHIA8 [1], MG5AMC [2], or SHERPA [3]. They start from a fundamental Lagrangian and rely on perturbative quantum field theory to provide the key part of a comprehensive simulation and inference chain. LHC simulations, including event generation, are a challenging numerical task, so we expect improvements from modern machine learning [4,5]. These improvements cover precision and speed, which can at least in part be interchanged, on track to meet the challenges by the upcoming HL-LHC program.

Following the modular structure of event generators, we will use modern neural networks to speed up expensive scattering amplitude evaluations [6–12]. Given these amplitudes, we need to improve the phase-space integration and sampling [13–22]. Here, precise generative networks are extremely useful. Their advantage in the LHC context is that they cover interpretable physics phase spaces, for example scattering events [23–29], parton showers [30–37], and detector simulations [38–61].

Generative networks for LHC physics can be trained on first-principle simulations and are easy to ship, powerful in amplifying the training data [62, 63], and — most importantly — controlled and precise [28, 64–67]. Subsequent tasks for these generative networks include event subtraction [68], event unweighting [69,70], or super-resolution enhancement [71,72]. Their conditional versions enable new analysis methods, like probabilistic unfolding [73–82], inference [83–85], or anomaly detection [86–91].

Every LHC event generator covers phase space using a combination of importance sampling and a multi-channel factorization. To improve phase-space sampling through modern machine learning, we need to tackle both of these technical ingredients together. This observation is the basis of the MADNIS approach [20], which has independently been proposed by Refs. [16,17]. Going beyond this proof of principle, we show in this paper how MADNIS can be implemented within MG5AMC and by how much a classic event generator setup can be improved. This is the logical next step after employing MADNIS within the phase-space generator Chili [22] and is similar to previous applications of neural importance sampling (NIS) in Sherpa [18]. In Sec. 2 we describe the updated MADNIS framework, including an improved loss for the network training in Sec 2.2, a fast initialization using classic importance sampling in Sec. 2.3, and an efficient training strategy in Sec. 2.4. In Sec. 3.2 we then provide detailed performance tests for a range of reference processes, and always compared to an optimized MG5AMC setup. In Sec. 3.3, we show how the MADNIS performance gains can be understood in terms of amplitude patterns and symmetries, and in Sec. 3.4 we push our implementation to its limits. As a spoiler, we will find that for challenging LHC processes, MADNIS can improve the unweighting efficiencies by an order of magnitude, a performance metric which can be translated directly into a speed gain. This gain is orthogonal to the development of faster amplitude evaluations and hardware acceleration using highly parallelized GPU farms [92–94].

2 Improving MADNIS

Before we discuss our improved ML-implementations, we briefly review the basics of multi-channel Monte Carlo. We integrate a function $f \sim |\mathcal{M}|^2$ over phase space

$$I[f] = \int \mathrm{d}^D x\, f(x) \qquad\qquad x \in \mathbb{R}^D \,. \tag{1}$$

This integral can be decomposed introducing local channel weights $\alpha_i(x)$ [95,96]

$$f(x) = \sum_{i=1}^{n_c} \alpha_i(x)\, f(x) \qquad \text{with} \quad \sum_{i=1}^{n_c} \alpha_i(x) = 1 \quad \text{and} \quad \alpha_i(x) \geq 0\,, \tag{2}$$

This MG5AMC decomposition is different from the original multi-channel method [97,98], which is used by SHERPA [3], or WHIZARD [99], as discussed in Ref. [20]. The phase-space integral now reads

$$I[f] = \sum_{i=1}^{n_c} I_i[f] = \sum_{i=1}^{n_c} \int \mathrm{d}^D x\; \alpha_i(x) f(x)\,. \tag{3}$$

Next, we introduce a set of channel-dependent phase-space mappings

$$x \in \mathbb{R}^D \quad \underset{\leftarrow \overline{G}_i(z)}{\overset{G_i(x)\rightarrow}{\longleftrightarrow}} \quad z \in [0,1]^D\,, \tag{4}$$

which parametrize properly normalized densities

$$g_i(x) = \left| \frac{\partial G_i(x)}{\partial x} \right| \qquad \text{with} \quad \int \mathrm{d}^D x\; g_i(x) = 1\,. \tag{5}$$

The phase-space integral now covers the D-dimensional unit cube and can be sampled as

$$\begin{aligned} I[f] &= \sum_{i=1}^{n_c} \int \frac{\mathrm{d}^D z}{g_i(x)}\; \alpha_i(x) f(x) \Big|_{x=\overline{G}_i(z)} \\ &= \sum_{i=1}^{n_c} \int \mathrm{d}^D x\; g_i(x)\, \frac{\alpha_i(x) f(x)}{g_i(x)} \equiv \sum_{i=1}^{n_c} \left\langle \frac{\alpha_i(x) f(x)}{g_i(x)} \right\rangle_{x\sim g_i(x)}\,. \end{aligned} \tag{6}$$

Typically, physics-inspired channels are initialized with analytic mappings and then refined by an adaptive algorithm, for instance VEGAS [100–104]. It exploits the fact that the $I_i[f]$ are unchanged under the mappings G_i, but the variance

$$\begin{aligned} \sigma_i^2 \equiv \sigma_i^2\left[\frac{\alpha_i f}{g_i}\right] &= \left\langle \left(\frac{\alpha_i(x) f(x)}{g_i(x)} - I_i[f] \right)^2 \right\rangle_{x\sim g_i(x)} \\ &= \left\langle \frac{\alpha_i(x)^2 f(x)^2}{g_i(x)^2} \right\rangle_{x\sim g_i(x)} - I_i[f]^2 \end{aligned} \tag{7}$$

is minimized by the optimal mapping

$$g_i(x)\big|_{\text{opt}} = \frac{\alpha_i(x) f(x)}{I_i[f]}\,, \tag{8}$$

so the optimal form of $g_i(x)$ is only defined in relation to the choice of $\alpha_i(x)$ and, consequently, $I_i[f]$.

We compute the Monte Carlo estimate of each integral using a discrete set of points,

$$I_i[f] = \left\langle \frac{\alpha_i(x) f(x)}{g_i(x)} \right\rangle_{x\sim g_i(x)} \approx \frac{1}{N_i} \sum_{k=1}^{N_i} \frac{\alpha_i(x_k) f(x_k)}{g_i(x_k)} \Bigg|_{x_k \sim g_i(x)}\,. \tag{9}$$

In analogy, the variance gives us an estimate of the uncertainty [98],

$$\Delta^2_{i,N_i} = \frac{\sigma_i^2}{N_i} = \frac{1}{N_i} \left[\left\langle \frac{\alpha_i(x)^2 f(x)^2}{g_i(x)\, q_i(x)} \right\rangle_{x\sim q_i(x)} - \left\langle \frac{\alpha_i(x) f(x)}{q_i(x)} \right\rangle^2_{x\sim q_i(x)} \right]. \tag{10}$$

Here, we rely on a different sampling density $q_i(x) \neq g_i(x)$. This appears ad-hoc, but is needed to define a properly differentiable loss function during training and will improve the performance [20]. When the variance is estimated from a finite sample, a correction factor $N_i/(N_i-1)$ has to be introduced to ensure an unbiased estimator. As the complete integral $I[f]$ is the sum of statistically independent $I_i[f]$, its uncertainty and variance are

$$\Delta_N^2 = \sum_{i=1}^{n_c} \Delta_{i,N_i}^2 = \sum_{i=1}^{n_c} \frac{\sigma_i^2}{N_i} \qquad \text{and} \qquad \sigma_{\text{tot}}^2 = N\Delta_N^2 = \sum_{i=1}^{n_c} \frac{N}{N_i}\sigma_i^2 \,. \tag{11}$$

2.1 ML implementation

Starting from Eq.(6), MadNIS encodes the multi-channel weight $\alpha_i(x)$ and the channel mappings $G_i(x)$ in neural networks.

Neural channel weights

First, MadNIS employs a channel-weight neural network (CWnet) to encode the local multi-channel weights defined in Eq.(2),

$$\alpha_i(x) \equiv \alpha_{i\xi}(x)\,, \tag{12}$$

where ξ denotes the network parameters. The best way to implement the normalization condition from Eq.(2) in the architecture is [20]

$$\alpha_{i\xi}(x) = \text{Softmax} A_{i\xi}(x) \equiv \frac{\exp A_{i\xi}(x)}{\sum_j \exp A_{j\xi}(x)}\,, \tag{13}$$

where $A_i(x)$ is the unnormalized network output. As the channel weights vary strongly over phase space, it helps the performance to learn them as a correction to a physically motivated prior assumption. Two well-motivated and normalized priors in MG5aMC are

$$\begin{aligned} \alpha_i^{\text{MG}}(x) &= \frac{|\mathcal{M}_i(x)|^2}{\sum_j |\mathcal{M}_j(x)|^2} \\ \alpha_i^{\text{MG}}(x) &= \frac{P_i(x)}{\sum_j P_j(x)} \quad \text{with} \quad P_i(x) = \prod_{k\in\text{prop}} \frac{1}{|p_k(x)^2 - m_k^2 - \mathrm{i} m_k \Gamma_k|^2}\,. \end{aligned} \tag{14}$$

They define the single-diagram enhanced multi-channel method [95,96]. Relative to either of them we only learn a correction to the prior, i.e.

$$\alpha_{i\xi}(x) = \text{Softmax}\big[\log \alpha_i^{\text{MG}}(x) + A_{i\xi}(x)\big] = \frac{\alpha_i^{\text{MG}}(x)\, \exp A_{i\xi}(x)}{\sum_j [\alpha_j^{\text{MG}}(x)\, \exp A_{j\xi}(x)]}\,. \tag{15}$$

We initialize the network parameters ξ_0 such that $A_{i\xi_0}(x) = 0$.

Neural importance sampling

Second, we combine analytic channel mappings and a normalizing flow for a mapping from a latent space z to the phase space x,

$$x \in \mathbb{R}^D \xleftrightarrow{\text{analytic}} y \in [0,1]^D \xleftrightarrow{\text{INN}} z \in [0,1]^D\,. \tag{16}$$

This chain replaces VEGAS with an invertible neural network (INN) [105], an incarnation of a normalizing flow equally fast in both directions. This INN is used for latent space refinement, where the INN links two unit hypercubes $[0,1]^D$. MADNIS improves the physics-inspired phase-space mappings by training an INN as part of

$$z = G_{i\theta}(x) \qquad \text{or} \qquad x = \overline{G}_{i\theta}(z)\,, \tag{17}$$

with the network weights θ. Similar to the multi-channel weights, we use physics knowledge to simplify the task of the normalizing flow and improve its efficiency.

A crucial condition for using a normalizing flow for phase-space integration is its equally fast evaluation in both directions [20]. The building blocks of the best-suited flows are coupling layers. The INN task in Eq.(16) can, in principle, be realized with any invertible coupling layer and an additional sigmoid layer. Coupling layers [105–107] split the input y into two parts, $y = (y^A, y^B)$, and define the forward and inverse passes as

$$\begin{pmatrix} z_r^A \\ z_s^B \end{pmatrix} = \begin{pmatrix} y_r^A \\ C(y_s^B; u_{s\theta}(y^A)) \end{pmatrix} \qquad \Leftrightarrow \qquad \begin{pmatrix} y_r^A \\ y_s^B \end{pmatrix} = \begin{pmatrix} z_r^A \\ C^{-1}(z_s^B; u_{s\theta}(z^A)) \end{pmatrix}, \tag{18}$$

The component-wise coupling transform C acts on the learned function u_θ, is invertible and has a tractable Jacobian. The Jacobian of the full mapping is

$$g(y) = \prod_s \frac{\partial C(y_s^B; u_{s\theta}(y^A))}{\partial y_s^B}\,. \tag{19}$$

For a stable numerical performance we use rational-quadratic spline (RQS) blocks [108], where each bin is a monotonically-increasing rational-quadratic (RQ) function. They provide superior expressivity and are naturally defined on a compact interval $[a,b]$. We split the unit interval $[0,1]$ in K bins with $K+1$ boundaries or knots $(y_s^{(k)}, z_s^{(k)})$ $(k = 0,\dots,K)$, with fixed endpoints $(0,0)$ and $(1,1)$. The K widths w_s and heights h_s of the bins and the $K+1$ derivatives d_s at the boundaries are constructed from the output of the network,

$$u_{s\theta}(x^A) = (\Theta_s^w, \Theta_s^h, \Theta_s^d)\,. \tag{20}$$

They are normalized as

$$\begin{aligned} w_s &= \text{Softmax}\,\Theta_s^w \\ h_s &= \text{Softmax}\,\Theta_s^h \\ d_s &= \frac{\text{Softplus}\,\Theta_s^d}{\log 2} \equiv \frac{\log\left(1+\exp\Theta_s^d\right)}{\log 2}\,. \end{aligned} \tag{21}$$

In contrast to the original Ref. [108], we add learnable derivatives at the boundaries to increase expressivity, and introduce the normalization of d_s such that $\Theta_s^d = 0$ is associated with a unit derivative $d_s = 1$. The w, h and d are then used to construct the RQ function C and its derivative [108].

In Eq.(18) we see that the coupling layer describes each dimension z_s^B using a learned function of all $y^A = z^A$. This way, it cannot describe correlations between different directions z_s^B. To encode correlations between all dimensions, we stack multiple coupling layers and permute the elements between them. The permutation changes which components are combined within y^A and y^B. We use a deterministic set of permutations based on a logarithmic decomposition of the integral dimension [16, 20], which ensures that any element is conditioned on any other element at least once. The number of coupling blocks then scales with $\log D$.

2.2 Multi-channel loss

After encoding channel weights and importance sampling as neural networks,

$$\alpha_i(x) \equiv \alpha_{i\xi}(x) \qquad \text{and} \qquad g_i(x) \equiv g_{i\theta}(x)\,, \tag{22}$$

the integral in Eq.(6) is estimated as

$$I[f] = \sum_{i=1}^{n_c} \left\langle \frac{\alpha_{i\xi}(x) f(x)}{g_{i\theta}(x)} \right\rangle_{x\sim g_{i\theta}(x)} . \tag{23}$$

The common optimization task for both networks is given by Eq.(8). The relative contribution per sub-integral changes when we train the two networks, ruling out the use of any f-divergence [109] when optimizing the channel weights $\alpha_{i\xi}(x)$.

As the optimization is performed on batches with size $b \ll N$, we use a loss function that does not scale with b and is independent of the number of sampled points. Moreover, the sampling density in the loss function $x \sim q_i(x)$ cannot coincide with the learned mapping [20]. We can directly minimize the variance as the loss function

$$\begin{aligned} \mathcal{L}_{\text{variance}} &= \sum_{i=1}^{n_c} \frac{N}{N_i} \sigma_i^2 \\ &= \sum_{i=1}^{n_c} \frac{N}{N_i} \left(\left\langle \frac{\alpha_{i\xi}(x)^2 f(x)^2}{g_{i\theta}(x) q_i(x)} \right\rangle_{x\sim q_i(x)} - \left\langle \frac{\alpha_{i\xi}(x) f(x)}{q_i(x)} \right\rangle^2_{x\sim q_i(x)} \right) . \end{aligned} \tag{24}$$

In practice, $q_i(x) \simeq g_{i\theta}(x)$ allows us to compute the loss as precisely as possible and stabilizes the combined online [110] and buffered training [20].

A critical hyperparameter in the variance loss is the distribution of sample points, N_i, during training and integral evaluation. Its optimal choice depends on the σ_i^2, and with that on α_i and g_i. While the latter are only known numerically, the optimal choice of N_i can be computed by minimizing the loss with respect to N_i, given $N = \sum_i N_i$ [98, 111]

$$N_i = N \frac{\sigma_i}{\sum_j \sigma_j} \qquad \Leftrightarrow \qquad \frac{N}{N_i} = \frac{\sum_j \sigma_j}{\sigma_i} . \tag{25}$$

This known result from stratified sampling defines the improved MadNIS loss

$$\begin{aligned} \mathcal{L}_{\text{MadNIS}} &= \sum_{i=1}^{n_c} \left(\sum_{j=1}^{n_c} \sigma_j \right) \sigma_i = \left[\sum_{i=1}^{n_c} \sigma_i \right]^2 \\ &= \left[\sum_{i=1}^{n_c} \left(\left\langle \frac{\alpha_{i\xi}(x)^2 f(x)^2}{g_{i\theta}(x) q_i(x)} \right\rangle_{x\sim q_i(x)} - \left\langle \frac{\alpha_{i\xi}(x) f(x)}{q_i(x)} \right\rangle^2_{x\sim q_i(x)} \right)^{1/2} \right]^2 . \end{aligned} \tag{26}$$

The physics information used to construct a set of integration channels is usually extracted from Feynman diagrams. MG5aMC automatically groups Feynman diagrams or channels, which are linked by permutations of the final-state particles. Because of the complications through the parton densities it does not consider permutations in the initial state. Following the MG5aMC approach, symmetry-related MadNIS channels share the same phase space mapping, but the channel-weight network has separate outputs for each channel. The σ_i in the loss can be defined for individual channels or for channel groups. For processes with many channels we find that using sets of channels stabilizes the training, so we resort to this assignment as our default.

2.3 VEGAS initialization

VEGAS [100–102] is the classic algorithm for importance sampling to compute high-dimensional integrals. For approximately factorizing integrands VEGAS is extremely efficient and converges much faster than factorized neural importance sampling, so we use use a VEGAS pre-training to initialize our normalizing flow.

The standard VEGAS implementation integrates the unit cube, $z_s \in [0,1]$ and relies on an assumed factorization of the phase space dimensions, made explicit for the phase space mapping and the corresponding sampling distribution in the second step of Eq.(16)

$$g(y) = \prod_{s=1}^{D} g(y_s) \,. \tag{27}$$

In analogy to Eq.(4) it encodes sampling distributions g for each dimension by dividing the unit interval into K bins of equal probability but different widths w_k with $\sum_k w_k = 1$. In each interval,

$$y \in [W_{k-1}, W_k] \qquad \text{with} \qquad W_k = \sum_{m=1}^{k} w_m \,, \tag{28}$$

VEGAS samples uniformly,

$$g(y) = \frac{1}{K w_k} \qquad \text{for} \qquad y \in [W_{k-1}, W_k] \,, \tag{29}$$

such that each bin integrates to $1/K$, and their sum from Eq.(5) to

$$\int_0^1 \mathrm{d}y \; g(y) = \sum_{k=1}^{K} w_k \, \frac{1}{K w_k} = 1 \,. \tag{30}$$

In one dimension, the integrated mapping $G(x)$ from Eq.(4) is a piece-wise linear function, or linear spline,

$$G(y) = (k-1)\,\frac{1}{K} + \frac{y - W_{k-1}}{K w_k} \qquad \text{for} \qquad y \in [W_{k-1}, W_k] \,. \tag{31}$$

The VEGAS algorithm iteratively adapts the bin widths w_k such that the distribution $g(y)$ matches the integrand. We illustrate g and G with $K = 20$ bins for a Gaussian mixture model in the upper panels of Fig. 1.

If we want to use VEGAS to pre-train an INN-mapping of two unit cubes, we need to relate the VEGAS grid to the RQ splines introduced in Sec. 2.1. Following Eq.(21), we need to extract the bin widths w, the bin heights h, and the derivatives on the bin edges d from a VEGAS grid. The width in y are the same as the VEGAS bin sizes, the heights are equal for all bins, and the derivatives can be estimated as the ratio of the bin heights and widths,

$$\begin{aligned}
\text{widths:} \quad & w_k && && \text{for} \quad k = 1, \dots, K \\
\text{heights:} \quad & h_k = \frac{1}{K} && && \text{for} \quad k = 1, \dots, K \\
\text{endpoints:} \quad & d_0 = \frac{h_1}{w_1} = \frac{1}{K w_1} && d_K = \frac{h_K}{w_K} = \frac{1}{K w_K} \\
\text{internal points:} \quad & d_k = \frac{h_{k+1} + h_k}{w_{k+1} + w_k} = \frac{2}{K(w_{k+1} + w_k)} && && \text{for} \quad k = 1, \dots, K-1 \,.
\end{aligned} \tag{32}$$

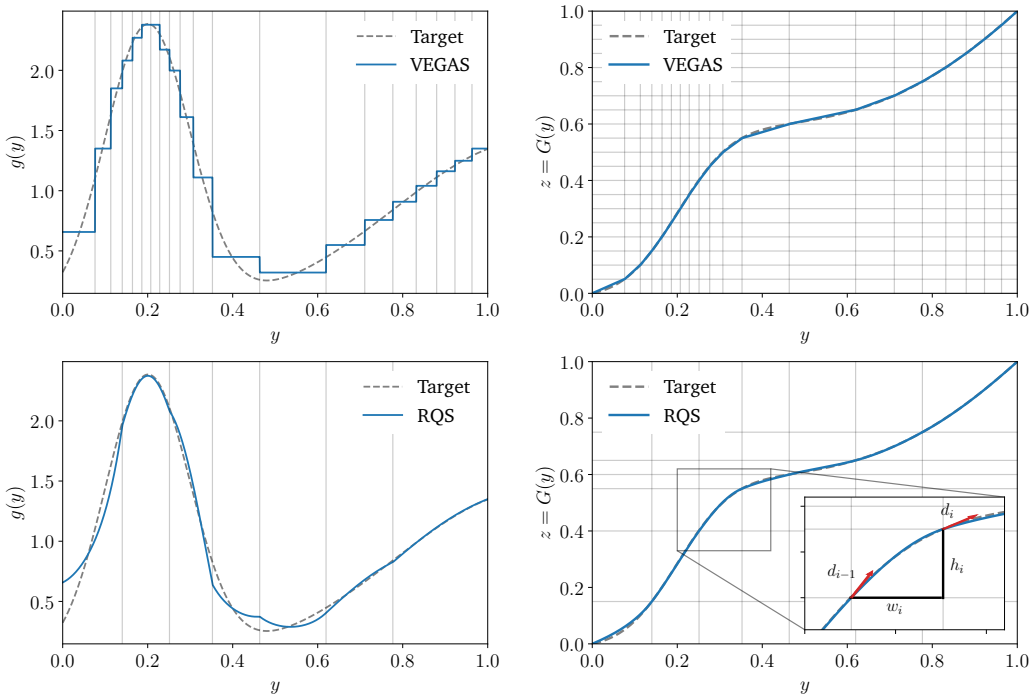

Figure 1: Top: learned VEGAS density $g(y)$ with 20 bins (left) and its transformation $G(y)$ (right). Bottom: the RQS density $g(y)$ after embedding the VEGAS grid and performing bin reduction to 7 bins (left). The right plot shows the corresponding mapping $G(y)$ including a zoom-in box illustrating the definition of the widths w_i, heights h_i of the bins and the derivatives d_i on the bin edges.

Because of the higher expressivity of the RQ splines, they need much fewer bins than VEGAS. For instance, all bins in VEGAS have equal probability, meaning that more bins than necessary are used in regions of high probability but with little change in the derivative of the transformation. We therefore reduce the number of bins by repeatedly merging adjacent bins until the desired number of target bins is reached:

1. Calculate the absolute difference between the slopes of adjacent bins,

$$\Delta_k = \left| \frac{w_k}{h_k} - \frac{w_{k+1}}{h_{k+1}} \right| . \tag{33}$$

2. Choose the lowest Δ_k, i.e. the two bins k and $k+1$ with the most similar average slope. We introduce an additional cutoff, to prevent very large bins, unless all smaller bins are already merged.
3. Reduce the number of bins by one

$$\begin{aligned} w &\leftarrow (w_1, \ldots, w_{k-1}, w_k + w_{k+1}, w_{k+2}, \ldots, w_K) , \\ h &\leftarrow (h_1, \ldots, h_{k-1}, h_k + h_{k+1}, h_{k+2}, \ldots, h_K) , \\ d &\leftarrow (d_0, \ldots, d_{k-1}, d_{k+1}, \ldots, d_K) , \\ K &\leftarrow K - 1 . \end{aligned} \tag{34}$$

The lower panels of Fig. 1 show g and G for the RQ spline after applying the bin reduction algorithm to reduce the number of bins to $K = 7$. We show w, h and d for one bin.

As shown in Eq.(18), a RQS spline block includes a trainable sub-network. It encodes the bin widths w, heights h and derivatives d. By inverting the normalization from Eq.(21), setting the weight matrix of the final sub-network layer to zero and assigning our results for Θ^w, Θ^h

and Θ^d to the bias vector, we can initialize the spline block to behave like the VEGAS grid. This has to be done for the final two coupling blocks (in the sampling direction), one for each half of the dimensions.

2.4 Training strategies

Online & buffered training

MADNIS supports two training modes [20]. First, in the online training mode generated events are immediately used to update the network weights. Their integrands, channel weights, and Jacobians are stored for later use in the second, buffered training mode. For that, only the weight update is performed. Online training is typically more stable, especially in the early stages, while buffered training is much faster in cases where the integrand is computationally expensive. We alternate between online and buffered training and keep the buffer at a fixed size, so the oldest sample points are replaced through online training. The impact of the buffering is given by $R_@$, the ratio of the number of weight updates from online training to all weight updates. For expensive integrands, $R_@$ determines the speed gain relative to online training only. For the results shown in Sec. 3 we use a buffer size of 1000 batches and start with online training until the buffer is filled. We then train alternately on the full buffer and online, where we aim to replace either a quarter or half of the buffer, corresponding to gain factors $R_@ = 5$ and $R_@ = 3$.

Learning and correcting the channel weights changes the normalization for each channel. Since this normalization is a function of all channel weights, the complete set has to be stored as part of the buffered training. Storing $\mathcal{O}(1M)$ phase space points for a process with $\mathcal{O}(1k)$ channels requires several gigabytes, so this becomes prohibitive. On the other hand, we typically find that for a given sampling only a few channels contribute significantly. Instead of storing all weights we then store the indices and weights (i, α_i) of the $m < n_c$ channels with the largest weights. The channel that was used to generate the sample always has to be included in this list, even if it is not among the m largest weights. To optimize the network on such a buffered sample, we set the other α_i to zero and adjust the normalization accordingly. Because this approximation is only used during buffered training epochs, but not during online training, integration or sampling, it does not introduce a bias.

Stratified training & channel dropping

The variance of a channel scales with its cross section, and even if different channels contribute similarly to the total cross section, their variances can still be very imbalanced. The variance loss will be dominated by channels with a large variance, leading to large and noisy training gradients if the number of points in these channels is not sufficiently large. We improve our training by adjusting the number of points per channel and allow for channels to be dropped altogether.

Because the variance of the channels changes during the training of the channel weights and during the training of the importance sampling, we track running means of the channel variances and use them to adjust the number of points per channel during online training. To ensure a stable training we first distribute a fraction r of points evenly among channels. The remaining part of the sample is distributed according to the channel variances, following the stratified sampling defined in Eq.(25). This means r interpolates between uniform sampling ($r = 1$) and stratified sampling ($r = 0$). Note that in the presence of phase space cuts we cannot be sure that there will be points with non-vanishing weights in a given channel. For the results shown in Sec. 3 we ensure a stable estimate of the channel variance by first training with uniform N_i for 1000 batches and then compute the variance with a running mean over the last 1000 batches.

We also go a step further and drop channels with a very small contribution to the cross section. For this, we track the running mean over the I_i per channel and allow for channels to be dropped after every training epoch. We define a fraction of the total cross section, typically $10^{-3} \times I$, which can be neglected by dropping the corresponding channels. Next we sort all channels by I_i and drop channels, starting from the smallest contributions, until this fraction is reached. If we drop a channel we also remove its contribution to the buffered sample and adjust the normalization of the channel weights. This way no training time is invested into dropped channels and the result remains unbiased.

3 Implementation and benchmarks

We call MG5AMC from MADNIS to compute the matrix element and parton densities, the initial phase space mapping, and the initial channel weights used in Eq.(15). For each event the inputs to this MG5AMC call are a vector of random numbers r_s with $s = 1 \dots D$ and the index i of the channel used to sample the event. The MG5AMC output is the concatenation of all 4-momenta p, the event weight w, and the set of channel weights α_j^{MG} with $j = 1 \dots n_c$. All MADNIS hyperparameters used for our benchmark study are given in the Appendix, Tab. 2. We note, while trained on significantly less training points than the flow, VEGAS reaches its optimum faster due to its simpler optimization and construction. We further emphasize, that VEGAS is trained much longer and on more points than usually done in MG5AMC, guaranteeing it has reached its optimum. In detail, as indicated in Tab. 2, we train VEGAS for 7 iterations (default is 3) using 50k samples per iteration. We verified that increasing this number by a factor of 10 did not further improve the performance of VEGAS. Currently, our interface is not fully optimized and is still a speed bottleneck. Furthermore, our setup does not support multiple partonic processes yet. Hence, we will not show run time comparisons in this work and will use processes with a fixed partonic initial state.

3.1 Reference processes

To benchmark the MADNIS performance we choose a set of realistic and challenging hard processes,

$$\begin{aligned}
&\text{Triple-W} && \mathrm{u\bar{d}} \to \mathrm{W^+W^+W^-} && && \\
&\text{VBS} && \mathrm{uc} \to \mathrm{W^+W^+\,ds} && && \\
&\text{W+jets} && \mathrm{gg} \to \mathrm{W^+d\bar{u}} && \mathrm{gg} \to \mathrm{W^+d\bar{u}g} && \mathrm{gg} \to \mathrm{W^+d\bar{u}gg} \\
&\text{t}\bar{\text{t}}\text{+jets} && \mathrm{gg} \to \mathrm{t\bar{t}} + \mathrm{g} && \mathrm{gg} \to \mathrm{t\bar{t}} + \mathrm{gg} && \mathrm{gg} \to \mathrm{t\bar{t}} + \mathrm{ggg}\,.
\end{aligned} \tag{35}$$

The produced heavy particles are assumed to be stable. The motivation for VBS and Triple-W production is that they are key to understand electroweak symmetry breaking, and that their large number of gauge-related Feynman diagrams will challenge our framework with potentially large interference. Next, we choose W+jets and t$\bar{\text{t}}$+jets production to study the scaling with additional jets. In Tab. 1 we show the number of Feynman diagrams and the number of MG5AMC channels for all processes. For instance, MG5AMC does not construct a separate channel for four-point vertices, so the number of channels is smaller than the number of diagrams. Channels which only differ by a permutation of the final-state momenta are combined into groups, further reducing the numbers of mappings. Finally, we show the number of channels that remain active after a MADNIS training, as discussed in Sec. 3.3.

Table 1: Number of Feynman diagrams, channels and channel groups after accounting for symmetries. The last column shows the number of channels that remain active after MadNIS channel dropping. Its range reflects ten independent trainings.

Process		# diagrams	# channels	# channel groups	# active channels
Triple-W	$u\bar{d} \to W^+W^+W^-$	17	16	8	2 ... 4
VBS	$uc \to W^+W^+ ds$	51	30	15	4 ... 6
W+jets	$gg \to W^+ d\bar{u}$	8	8	4	6
	$gg \to W^+ d\bar{u}g$	50	48	24	12 ... 16
	$gg \to W^+ d\bar{u}gg$	428	384	108	28 ... 51
$t\bar{t}$+jets	$gg \to t\bar{t} + g$	16	15	9	4 ... 6
	$gg \to t\bar{t} + gg$	123	105	35	12
	$gg \to t\bar{t} + ggg$	1240	945	119	60 ... 72

3.2 Benchmarking MadNIS features

To see the gain from each of the MadNIS features introduced in Sec. 2 we apply them to our reference processes one by one. Our baseline is Vegas, combined with MG5aMC channel mappings and channel weights. Again, for a fair comparison, we also optimize Vegas to the best of knowledge and work with more iterations and larger samples than the MG5aMC default. The Vegas optimization is still significantly faster than MadNIS, but our goal is to provide a pre-trained MadNIS generator and the training time is amortized for generating large numbers of events, so the generation time is our only criterion.

We use two metrics for our comparison: (i) the relative standard deviation σ/I minimized by stratified sampling, as is it independent from the sampling statistics and cross section; (ii) the unweighting efficiency ϵ [18], where the maximum weight is determined by bootstrapping and taking the median. For each, we use ten independent trainings and compute the means and standard deviations as an estimate of the stability of the training. We note that all obtained results, both integrated as well as differential cross-sections, agree with the default MG5aMC output and yield the correct result.

In Fig. 2 we show the results of our benchmarking for the four different processes with gauge bosons. For the more time-consuming top pair processes without any specific challenges we will provide the final improvements below.

First, we show the results from a simple setup where only the channel mappings are trained, but the channel weights are kept fixed, Sec. 2.1. Already here we see a sizeable gain over the standard method. Next, we also train the channel weights, and observe a further significant gain. For instance, the unweighting efficiency for VBS now reaches 20%, up from a few per-cent from the standard method and by more than a factor ten.

Next, we stick to the trainings without and with adaptive channel weights, but combine them with the Vegas-initialization from Sec. 2.3. For all processes, the gain compared to the standard Vegas method stabilizes, albeit without major improvements. This changes when we include stratified training, as introduced in Sec. 2.4. For all processes, stratified sampling with trained channel weights lead to another significant performance gain, up to a factor 15 for the VBS process.

Finally, we add channel dropping and buffered training, to reduce the training time. For time-wise gain factors $R_@ = 3$ and $R_@ = 5$ the improvement of the full MadNIS setup over the standard Vegas and MG5aMC integration remains stable. For processes with a large number of channels, channel dropping leads to a more stable training, while providing an equally good performance for the processes shown in Fig. 2. The fact that the performance gain for the unweighting efficiency is much larger than for the relative standard deviation reflects the sensitivity of the unweighting efficiency to the far tails of the weight distribution.

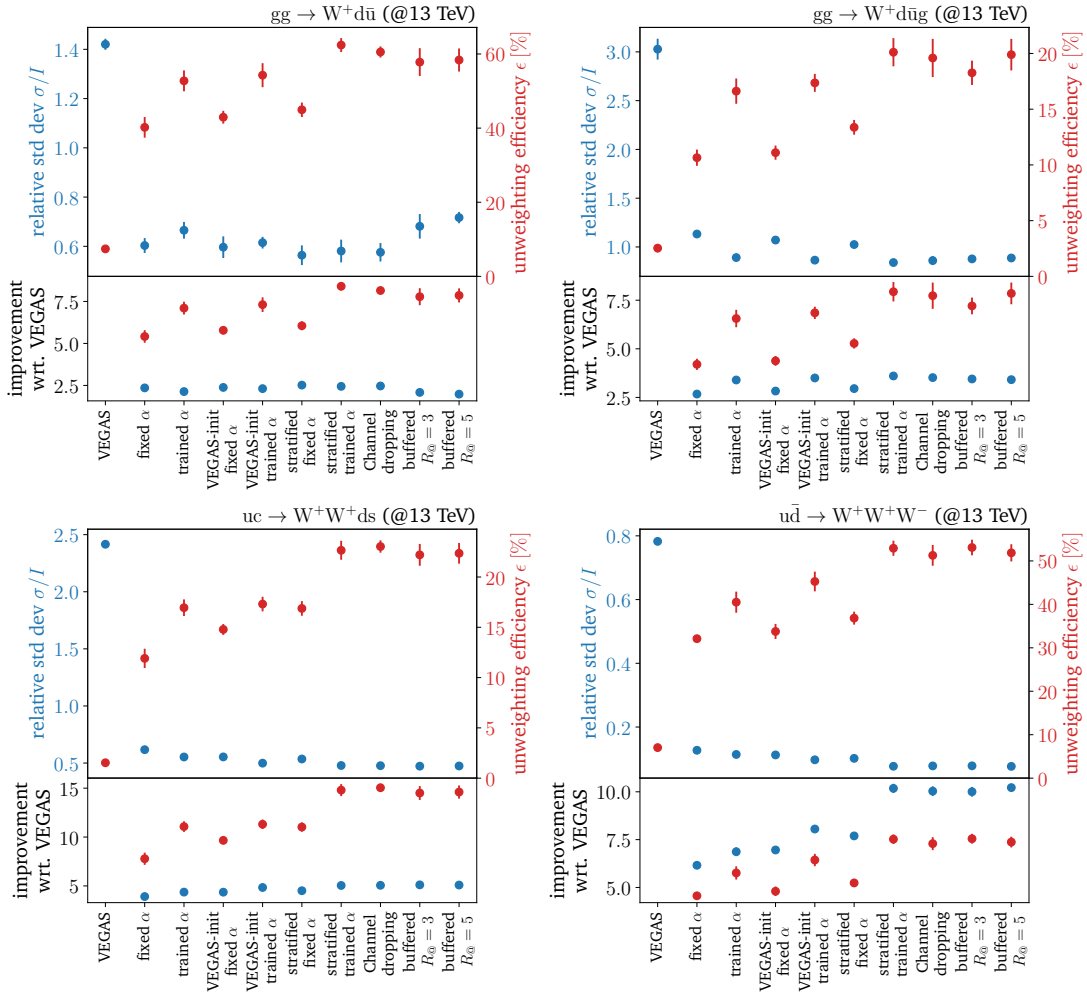

Figure 2: Relative standard deviation and unweighting efficiency for W+2jets, W+3jets, VBS and Triple-W for various combinations of MadNIS features.

3.3 Learning from channel weights

It turns our that we can also extract information from the learned channel weights. We look at the contribution of each channel to the total cross section,

$$\frac{I_i}{I} = \frac{\int \mathrm{d}^D x \; \alpha_i(x) f(x)}{\int \mathrm{d}^D x \; f(x)} \,. \tag{36}$$

We find that MadNIS changes the contribution of channel groups coherently, so for W+3jets and Triple-W production we look at the contributions given by the sum over the channels in the group.

In Fig. 3, we show the contribution of MadNIS channels or channel groups, compared to the initial MG5aMC assignments. We mark dropped channels with empty circles, the number of remaining active channels corresponds to Tab. 1. We see that MadNIS prefers much fewer channels, illustrating the benefit of our channel dropping feature.

For VBS and Triple-W production, MadNIS adapts the channel weights in a way that the integrand is almost completely made up from a single group of symmetry-related channels. The general behaviour and the specific choice of channels is consistent between repetitions of the training. The Feynman diagrams corresponding to these channels are shown in Fig. 4. For VBS five channel groups significantly contribute to the integral in MG5aMC, all of them

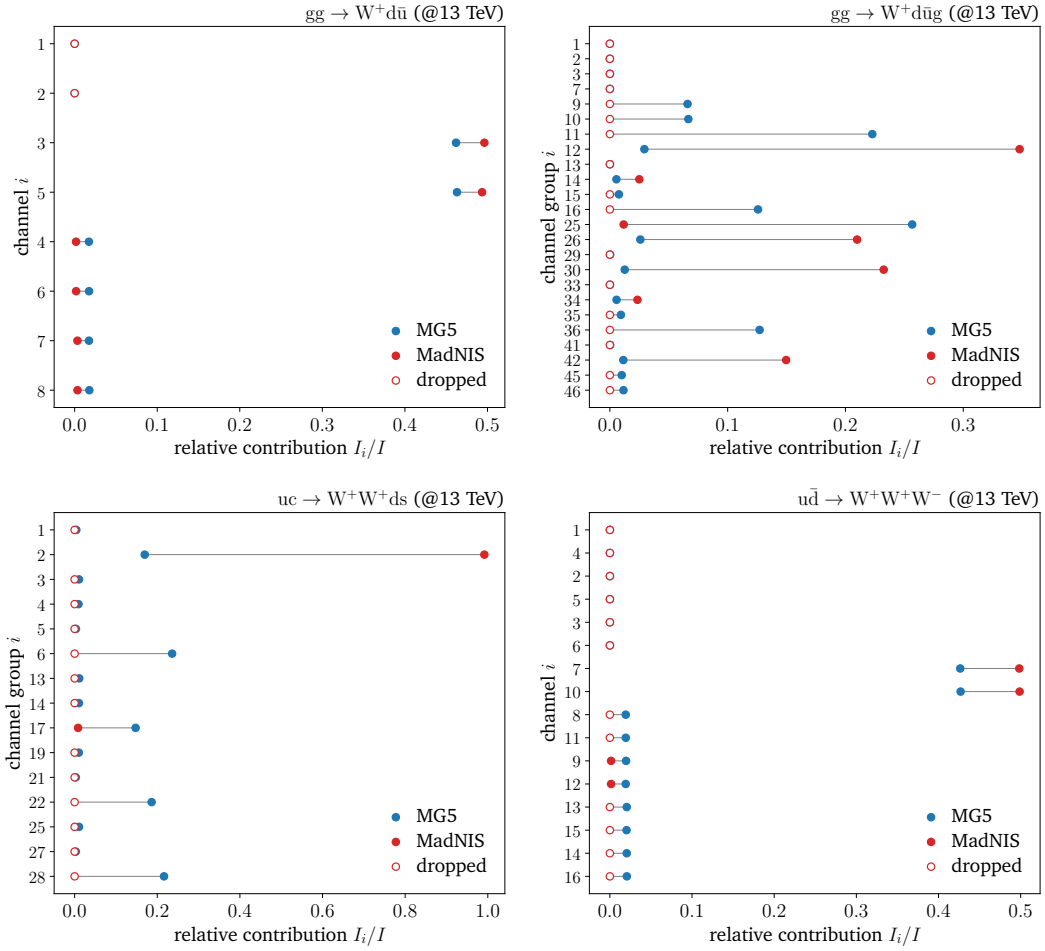

Figure 3: Relative contributions of the channels for W+2jets and Triple-W production, and for the channel groups for W+3jets and VBS. The channel weights are defined by MG5aMC, their weights are learned by MadNIS. An empty circle indicates a dropped channel.

with a t-channel gluon or photon. Of those, MadNIS enhances the QCD contribution $\mathcal{O}(\alpha_s^2\alpha^2)$ without an s-channel quark propagator. For Triple-W production, one channel group already dominates the integral in MG5aMC, and it is further enhanced by MadNIS. We give an example for the distribution learned by the learned channel weights as a function of phase space in Appendix B.

3.4 Scaling with jet multiplicity

The last challenge of modern event generation MadNIS needs to meet is large number of additional jets. We study the scaling of the MadNIS performance with the number of gluons in the final state for W+jets and $t\bar{t}$+jets production. Again, we use the relative standard deviation σ/I and the unweighting efficiency ϵ as performance metrics. As for the final result in Fig. 2, we train MadNIS with all features, including buffered training with $R_{@} = 5$. The results are shown in Fig. 5. While the unweighting efficiency decreases and standard deviation increases towards higher multiplicities, the gain over Vegas and MG5aMC remains roughly constant for W+jets production. For the even more challenging $t\bar{t}$+jets production the gain decreases for three jets, defining a remaining task for the final, public release of MadNIS.

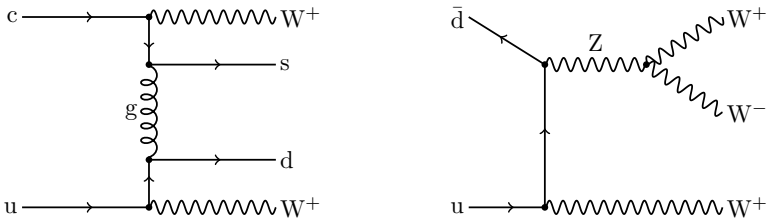

Figure 4: Feynman diagrams corresponding to the dominant channels after training MADNIS for VBS (left) and Triple-W production (right).

4 Outlook

We have, for the first time, shown that modern machine learning leads to a significant speed gain in MG5AMC. We have improved the MADNIS method [20] and implemented it in MG5AMC, to be able to quantify the performance gain from a modern ML-treatment of phase space sampling. This implementation will allow us to use the entire MG5AMC functionality while developing an ultra-fast event generator for the HL-LHC.

Starting from a combined training of a learnable phase space mapping encoded in an INN and learnable channel weights encoded in a simple regression network, we have added a series of new features, including an improved loss function and a fast VEGAS initialization. The combined online and buffered training has been further enhanced by stratified sampling and channel dropping. The basic structure behind the phase space mapping is still a state-of-the-art INN with rational quadratic spline coupling layers, rather than more expressive diffusion networks [29], because event generation requires the mapping to be fast in both directions.

The processes we used to benchmark the MADNIS performance gain are W+2,3,4 jets, VBS, Triple-W, and $t\bar{t}$+1,2,3 jets. They combine large numbers of gauge-related Feynman diagrams with a large number of particles in the final state. Through MG5AMC, these key processes can easily be expanded to any other partonic production or decay process. For the electroweak reference processes we have shown the performance gain, in terms of the integration error and the unweighting efficiency, for each of our new features separately. The performance benchmark was a tuned MG5AMC implementation. It turned out that learned phase-space weights, the VEGAS initialization, and the stratified training each contribute to a significant performance gain. Channel dropping and buffered training stabilize the training and limit the computational cost of the network training. The MADNIS gain in the unweighting efficiency ranges between a factor 5 and a factor 15, the latter realized for the notorious VBS process.

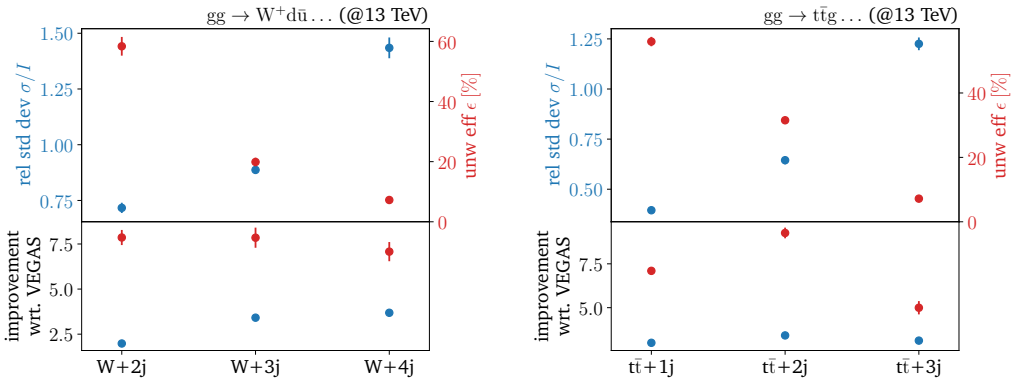

Figure 5: Relative standard deviation and unweighting efficiency for W+jets and $t\bar{t}$+jets with different numbers of gluons in the final state. The final MADNIS performance gain is illustrated in the lower panels, just as in Fig. 2.

Acknowledgements

OM, FM and RW acknowledge support by FRS-FNRS (Belgian National Scientific Research Fund) IISN projects 4.4503.16. TP would like to thank the Baden-Württemberg-Stiftung for financing through the program *Internationale Spitzenforschung*, project *Uncertainties — Teaching AI its Limits* (BWST_IF2020-010). TP is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant 396021762 – TRR 257 *Particle Physics Phenomenology after the Higgs Discovery*. TH is funded by the Carl-Zeiss-Stiftung through the project *Model-Based AI: Physical Models and Deep Learning for Imaging and Cancer Treatment*.

Funding information The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG (bwForCluster NEMO). Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

A Hyperparameters

Table 2: MadNIS hyperparameters. For the Vegas parameters, the first value is used for the pre-training and the value in parentheses for the remaining runs.

Parameter	Value
Optimizer	Adam [112]
Learning rate	0.001
LR schedule	Inverse decay
Final learning rate	0.0001
Batch size	$\min(200 \cdot n_c^{0.8}, 10000)$
Training length	88k batches
Permutations	Logarithmic decomposition [16]
Number of coupling blocks	$2\lceil \log_2 D \rceil$
Coupling transformation	RQ splines [108]
Subnet hidden nodes	32
Subnet depth	3
CWnet parametrization	$(\log p_T, \eta, \phi)$
CWnet hidden nodes	64
CWnet depth	3
Activation function	leaky ReLU
Max. # of buffered channel weights	75
Buffer size	1000 batches
Channel dropping cutoff	0.001
Uniform training fraction r	0.1
Vegas iterations	7 (7)
Vegas bins	64 (128)
Vegas samples per iteration	20k (50k)
Vegas damping α	0.7 (0.5)

B Channel-weight kinematics

To illustrate the learned channel weights we use an example with only three channels,

$$gg \to gg\,. \tag{B.1}$$

This process has four Feynman diagrams, but MG5AMC only constructs mappings corresponding to the s, t and u-channel Feynman diagrams. The latter two are related by an exchange of the two final-state gluons and will therefore share the same normalizing flow in MADNIS. We run a MADNIS training and use it to generate a set of weighted events. We keep the channel weights α^{MG} provided by MG5AMC and the α obtained from the CWnet. In Fig. 6, we show a stacked histogram of the z component of the momentum of the first gluon, because the asymmetric structure of the t and u channels is best visible for this observable. Furthermore, we show the average α^{MG} and α in every bin of this histogram. In this example we see that MADNIS does not significantly change the functional form of the channel weights over phase space, but enhances the contribution of the s channel, while decreasing the contributions of the t and u channels. This might be because MADNIS prefers channels that cover the entire phase space instead of more specialized channels for simple processes like this one.

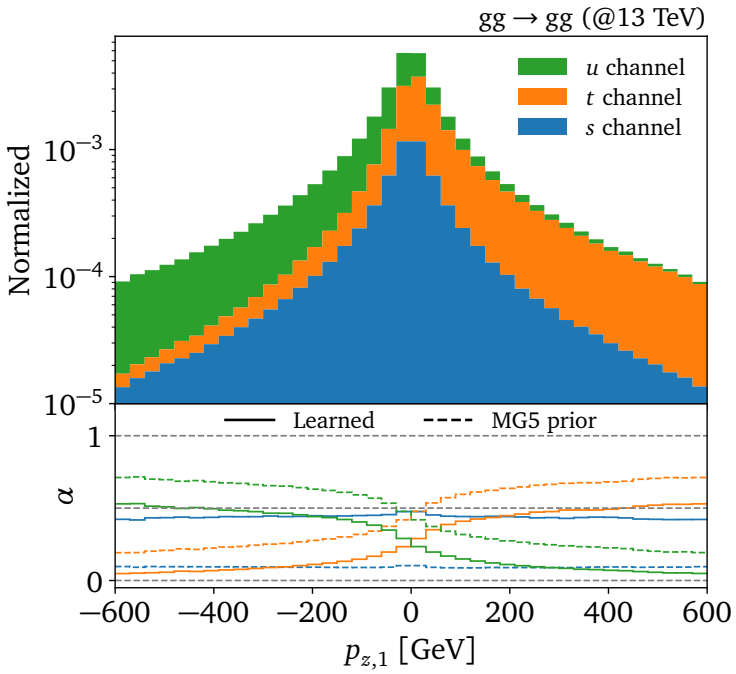

Figure 6: Stacked histogram of z-component of the first gluon momentum for the process gg → gg generated by MADNIS. The lower panel shows the bin-wise averages of the channels weights as obtained from MG5AMC and the learned channel weights.

References

[1] T. Sjöstrand et al., *An introduction to PYTHIA 8.2*, Comput. Phys. Commun. **191**, 159 (2015), doi:10.1016/j.cpc.2015.01.024.

[2] J. Alwall et al., *The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations*, J. High Energy Phys. **07**, 079 (2014), doi:10.1007/JHEP07(2014)079.

[3] E. Bothmann et al., *Event generation with Sherpa 2.2*, SciPost Phys. **7**, 034 (2019), doi:10.21468/SciPostPhys.7.3.034.

[4] A. Butter et al., *Machine learning and LHC event generation*, SciPost Phys. **14**, 079 (2023), doi:10.21468/SciPostPhys.14.4.079.

[5] T. Plehn, A. Butter, B. Dillon, T. Heimel, C. Krause and R. Winterhalder, *Modern machine learning for LHC physicists*, (arXiv preprint) doi:10.48550/arXiv.2211.01421.

[6] F. Bishara and M. Montull, *(Machine) learning amplitudes for faster event generation*, (arXiv preprint) doi:10.48550/arXiv.1912.11055.

[7] S. Badger and J. Bullock, *Using neural networks for efficient evaluation of high multiplicity scattering amplitudes*, J. High Energy Phys. **06**, 114 (2020), doi:10.1007/JHEP06(2020)114.

[8] J. Aylett-Bullock, S. Badger and R. Moodie, *Optimising simulations for diphoton production at hadron colliders using amplitude neural networks*, J. High Energy Phys. **08**, 066 (2021), doi:10.1007/JHEP08(2021)066.

[9] D. Maître and H. Truong, *A factorisation-aware Matrix element emulator*, J. High Energy Phys. **11**, 066 (2021), doi:10.1007/JHEP11(2021)066.

[10] R. Winterhalder, V. Magerya, E. Villa, S. Jones, M. Kerner, A. Butter, G. Heinrich and T. Plehn, *Targeting multi-loop integrals with neural networks*, SciPost Phys. **12**, 129 (2022), doi:10.21468/SciPostPhys.12.4.129.

[11] S. Badger, A. Butter, M. Luchmann, S. Pitz and T. Plehn, *Loop amplitudes from precision networks*, SciPost Phys. Core **6**, 034 (2023), doi:10.21468/SciPostPhysCore.6.2.034.

[12] D. Maître and H. Truong, *One-loop matrix element emulation with factorisation awareness*, J. High Energy Phys. **05**, 159 (2023), doi:10.1007/JHEP05(2023)159.

[13] J. Bendavid, *Efficient Monte Carlo integration using boosted decision trees and generative deep neural networks*, (arXiv preprint) doi:10.48550/arXiv.1707.00028.

[14] M. Klimek and M. Perelstein, *Neural network-based approach to phase space integration*, SciPost Phys. **9**, 053 (2020), doi:10.21468/SciPostPhys.9.4.053.

[15] I.-K. Chen, M. Klimek and M. Perelstein, *Improved neural network Monte Carlo simulation*, SciPost Phys. **10**, 023 (2021), doi:10.21468/SciPostPhys.10.1.023.

[16] C. Gao, J. Isaacson and C. Krause, *<tt>i- flow</tt>: High-dimensional integration and sampling with normalizing flows*, Mach. Learn.: Sci. Technol. **1**, 045023 (2020), doi:10.1088/2632-2153/abab62.

[17] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale and S. Schumann, *Exploring phase space with neural importance sampling*, SciPost Phys. **8**, 069 (2020), doi:10.21468/SciPostPhys.8.4.069.

[18] C. Gao, S. Höche, J. Isaacson, C. Krause and H. Schulz, *Event generation with normalizing flows*, Phys. Rev. D **101**, 076002 (2020), doi:10.1103/PhysRevD.101.076002.

[19] K. Danziger, T. Janßen, S. Schumann and F. Siegert, *Accelerating Monte Carlo event generation – rejection sampling using neural network event-weight estimates*, SciPost Phys. **12**, 164 (2022), doi:10.21468/SciPostPhys.12.5.164.

[20] T. Heimel, R. Winterhalder, A. Butter, J. Isaacson, C. Krause, F. Maltoni, O. Mattelaer and T. Plehn, *MadNIS - Neural multi-channel importance sampling*, SciPost Phys. **15**, 141 (2023), doi:10.21468/SciPostPhys.15.4.141.

[21] T. Janßen, D. Maître, S. Schumann, F. Siegert and H. Truong, *Unweighting multijet event generation using factorisation-aware neural networks*, SciPost Phys. **15**, 107 (2023), doi:10.21468/SciPostPhys.15.3.107.

[22] E. Bothmann, T. Childers, W. Giele, F. Herren, S. Höche, J. Isaacson, M. Knobbe and R. Wang, *Efficient phase-space generation for hadron collider event simulation*, SciPost Phys. **15**, 169 (2023), doi:10.21468/SciPostPhys.15.4.169.

[23] S. Otten, S. Caron, W. de Swart, M. van Beekveld, L. Hendriks, C. van Leeuwen, D. Podareanu, R. Ruiz de Austri and R. Verheyen, *Event generation and statistical sampling for physics with deep generative models and a density information buffer*, Nat. Commun. **12**, 2985 (2021), doi:10.1038/s41467-021-22616-z.

[24] B. Hashemi, N. Amin, K. Datta, D. Olivito and M. Pierini, *LHC analysis-specific datasets with generative adversarial networks*, (arXiv preprint) doi:10.48550/arXiv.1901.05282.

[25] R. Di Sipio, M. Faucci Giannelli, S. Ketabchi Haghighat and S. Palazzo, *DijetGAN: A generative-adversarial network approach for the simulation of QCD dijet events at the LHC*, J. High Energy Phys. **08**, 110 (2019), doi:10.1007/JHEP08(2019)110.

[26] A. Butter, T. Plehn and R. Winterhalder, *How to GAN LHC events*, SciPost Phys. **7**, 075 (2019), doi:10.21468/SciPostPhys.7.6.075.

[27] Y. Alanazi et al., *Simulation of electron-proton scattering events by a feature-augmented and transformed generative adversarial network (FAT-GAN)*, Proc. Thirtieth Int. Jt. Conf. Artif. Intell. (2021), doi:10.24963/ijcai.2021/293.

[28] A. Butter, T. Heimel, S. Hummerich, T. Krebs, T. Plehn, A. Rousselot and S. Vent, *Generative networks for precision enthusiasts*, SciPost Phys. **14**, 078 (2023), doi:10.21468/SciPostPhys.14.4.078.

[29] A. Butter, N. Huetsch, S. Palacios Schweitzer, T. Plehn, P. Sorrenson and J. Spinner, *Jet diffusion versus JetGPT – Modern networks for the LHC*, (arXiv preprint) doi:10.48550/arXiv.2305.10475.

[30] L. de Oliveira, M. Paganini and B. Nachman, *Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis*, Comput. Softw. Big Sci. **1**, 4 (2017), doi:10.1007/s41781-017-0004-6.

[31] A. Andreassen, I. Feige, C. Frye and M. D. Schwartz, *JUNIPR: a framework for unsupervised machine learning in particle physics*, Eur. Phys. J. C **79**, 102 (2019), doi:10.1140/epjc/s10052-019-6607-9.

[32] E. Bothmann and L. Del Debbio, *Reweighting a parton shower using a neural network: the final-state case*, J. High Energy Phys. **01**, 033 (2019), doi:10.1007/JHEP01(2019)033.

[33] K. Dohi, *Variational autoencoders for jet simulation*, (arXiv preprint) doi:10.48550/arXiv.2009.04842.

[34] E. Buhmann, G. Kasieczka and J. Thaler, *EPiC-GAN: Equivariant point cloud generation for particle jets*, SciPost Phys. **15**, 130 (2023), doi:10.21468/SciPostPhys.15.4.130.

[35] M. Leigh, D. Sengupta, G. Quétant, J. Andrew Raine, K. Zoch and T. Golling, *PC-JeDi: Diffusion for particle cloud generation in high energy physics*, SciPost Phys. **16**, 018 (2024), doi:10.21468/SciPostPhys.16.1.018.

[36] V. Mikuni, B. Nachman and M. Pettee, *Fast point cloud generation with diffusion models in high energy physics*, Phys. Rev. D **108**, 036025 (2023), doi:10.1103/PhysRevD.108.036025.

[37] E. Buhmann et al., *EPiC-ly fast particle cloud generation with flow-matching and diffusion*, (arXiv preprint) doi:10.48550/arXiv.2310.00049.

[38] M. Paganini, L. de Oliveira and B. Nachman, *Accelerating science with generative adversarial networks: An application to 3D particle showers in multilayer calorimeters*, Phys. Rev. Lett. **120**, 042003 (2018), doi:10.1103/PhysRevLett.120.042003.

[39] L. de Oliveira, M. Paganini and B. Nachman, *Controlling physical attributes in GAN-accelerated simulation of electromagnetic calorimeters*, J. Phys.: Conf. Ser. **1085**, 042017 (2018), doi:10.1088/1742-6596/1085/4/042017.

[40] M. Paganini, L. de Oliveira and B. Nachman, *CaloGAN: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks*, Phys. Rev. D **97**, 014021 (2018), doi:10.1103/PhysRevD.97.014021.

[41] M. Erdmann, L. Geiger, J. Glombitza and D. Schmidt, *Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks*, Comput. Softw. Big Sci. **2**, 4 (2018), doi:10.1007/s41781-018-0008-x.

[42] M. Erdmann, J. Glombitza and T. Quast, *Precise simulation of electromagnetic calorimeter showers using a Wasserstein generative adversarial network*, Comput. Softw. Big Sci. **3**, 4 (2019), doi:10.1007/s41781-018-0019-7.

[43] D. Belayneh et al., *Calorimetry with deep learning: particle simulation and reconstruction for collider physics*, Eur. Phys. J. C **80**, 688 (2020), doi:10.1140/epjc/s10052-020-8251-9.

[44] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol and K. Krüger, *Getting high: High fidelity simulation of high granularity calorimeters with high speed*, Comput. Softw. Big Sci. **5**, 13 (2021), doi:10.1007/s41781-021-00056-0.

[45] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol and K. Krüger, *Decoding photons: Physics in the latent space of a BIB-AE generative network*, EPJ Web Conf. **251**, 03003 (2021), doi:10.1051/epjconf/202125103003.

[46] C. Krause and D. Shih, *Fast and accurate simulations of calorimeter showers with normalizing flows*, Phys. Rev. D **107**, 113003 (2023), doi:10.1103/PhysRevD.107.113003.

[47] ATLAS Collaboration, *AtlFast3: The next generation of fast simulation in ATLAS*, Comput. Softw. Big Sci. **6**, 7 (2022), doi:10.1007/s41781-021-00079-7.

[48] C. Krause and D. Shih, *CaloFlow II: Even faster and still accurate generation of calorimeter showers with normalizing flows*, (arXiv preprint) doi:10.48550/arXiv.2110.11377.

[49] E. Buhmann et al., *Hadrons, better, faster, stronger*, Mach. Learn.: Sci. Technol. **3**, 025014 (2022), doi:10.1088/2632-2153/ac7848.

[50] C. Chen, O. Cerri, T. Q. Nguyen, J. R. Vlimant and M. Pierini, *Analysis-specific fast simulation at the LHC with deep learning*, Comput. Softw. Big Sci. **5**, 15 (2021), doi:10.1007/s41781-021-00060-4.

[51] V. Mikuni and B. Nachman, *Score-based generative models for calorimeter shower simulation*, Phys. Rev. D **106**, 092009 (2022), doi:10.1103/PhysRevD.106.092009.

[52] A. Collaboration, *Deep generative models for fast photon shower simulation in ATLAS*, Comput. Softw. Big Sci. **8**, 7 (2024), doi:10.1007/s41781-023-00106-9.

[53] C. Krause, I. Pang and D. Shih, *CaloFlow for CaloChallenge dataset 1*, SciPost Phys. **16**, 126 (2024), doi:10.21468/SciPostPhys.16.5.126.

[54] J. C. Cresswell, B. Leigh Ross, G. Loaiza-Ganem, H. Reyes-Gonzalez, M. Letizia and A. L. Caterini, *CaloMan: Fast generation of calorimeter showers with density estimation on learned manifolds*, (arXiv preprint) doi:10.48550/arXiv.2211.15380.

[55] S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, C. Krause, I. Shekhzadeh and D. Shih, *L2LFlows: generating high-fidelity 3D calorimeter images*, J. Instrum. **18**, P10017 (2023), doi:10.1088/1748-0221/18/10/P10017.

[56] B. Hashemi, N. Hartmann, S. Sharifzadeh, J. Kahn and T. Kuhr, *Ultra-high-resolution detector simulation with intra-event aware GAN and self-supervised relational reasoning*, (arXiv preprint) doi:10.48550/arXiv.2303.08046.

[57] A. Xu, S. Han, X. Ju and H. Wang, *Generative machine learning for detector response modeling with a conditional normalizing flow*, (arXiv preprint) doi:10.48550/arXiv.2303.10148.

[58] S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, K. Krüger, P. McKeown and L. Rustige, *New angles on fast calorimeter shower simulation*, (arXiv preprint) doi:10.48550/arXiv.2303.18150.

[59] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasicezka, A. Korol, W. Korcari, K. Krüger and P. McKeown, *CaloClouds: Fast geometry-independent highly-granular calorimeter simulation*, J. Instrum. **18**, P11025 (2023), doi:10.1088/1748-0221/18/11/P11025.

[60] M. R. Buckley, I. Pang, D. Shih and C. Krause, *Inductive simulation of calorimeter showers with normalizing flows*, Phys. Rev. D **109**, 033006 (2024), doi:10.1103/PhysRevD.109.033006.

[61] S. Diefenbacher, V. Mikuni and B. Nachman, *Refining fast calorimeter simulations with a Schrödinger bridge*, (arXiv preprint) doi:10.48550/arXiv.2308.12339.

[62] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman and T. Plehn, *GANplifying event samples*, SciPost Phys. **10**, 139 (2021), doi:10.21468/SciPostPhys.10.6.139.

[63] S. Bieringer et al., *Calomplification — the power of generative calorimeter models*, J. Instrum. **17**, P09028 (2022), doi:10.1088/1748-0221/17/09/P09028.

[64] R. Winterhalder, M. Bellagente and B. Nachman, *Latent space refinement for deep generative models*, (arXiv preprint) doi:10.48550/arXiv.2106.00792.

[65] B. Nachman and R. Winterhalder, *Elsa: Enhanced latent spaces for improved collider simulations*, Eur. Phys. J. C **83**, 843 (2023), doi:10.1140/epjc/s10052-023-11989-8.

[66] M. Leigh, D. Sengupta, J. A. Raine, G. Quétant and T. Golling, *Faster diffusion model with improved quality for particle cloud generation*, Phys. Rev. D **109**, 012010 (2024), doi:10.1103/PhysRevD.109.012010.

[67] R. Das, L. Favaro, T. Heimel, C. Krause, T. Plehn and D. Shih, *How to understand limitations of generative networks*, SciPost Phys. **16**, 031 (2024), doi:10.21468/SciPostPhys.16.1.031.

[68] A. Butter, T. Plehn and R. Winterhalder, *How to GAN event subtraction*, SciPost Phys. Core **3**, 009 (2020), doi:10.21468/SciPostPhysCore.3.2.009.

[69] B. Stienen and R. Verheyen, *Phase space sampling and inference from weighted events with autoregressive flows*, SciPost Phys. **10**, 038 (2021), doi:10.21468/SciPostPhys.10.2.038.

[70] M. Backes, A. Butter, T. Plehn and R. Winterhalder, *How to GAN event unweighting*, SciPost Phys. **10**, 089 (2021), doi:10.21468/SciPostPhys.10.4.089.

[71] F. Armando Di Bello, S. Ganguly, E. Gross, M. Kado, M. Pitt, L. Santi and J. Shlomi, *Towards a computer vision particle flow*, Eur. Phys. J. C **81**, 107 (2021), doi:10.1140/epjc/s10052-021-08897-0.

[72] P. Baldi, L. Blecher, A. Butter, J. Collado, J. N. Howard, F. Keilbach, T. Plehn, G. Kasieczka and D. Whiteson, *How to GAN higher jet resolution*, SciPost Phys. **13**, 064 (2022), doi:10.21468/SciPostPhys.13.3.064.

[73] K. Datta, D. Kar and D. Roy, *Unfolding with generative adversarial networks*, (arXiv preprint) doi:10.48550/arXiv.1806.00433.

[74] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn and R. Winterhalder, *How to GAN away detector effects*, SciPost Phys. **8**, 070 (2020), doi:10.21468/SciPostPhys.8.4.070.

[75] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman and J. Thaler, *OmniFold: A method to simultaneously unfold all observables*, Phys. Rev. Lett. **124**, 182001 (2020), doi:10.1103/PhysRevLett.124.182001.

[76] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, A. Rousselot, R. Winterhalder, L. Ardizzone and U. Köthe, *Invertible networks or partons to detector and back again*, SciPost Phys. **9**, 074 (2020), doi:10.21468/SciPostPhys.9.5.074.

[77] M. Backes, A. Butter, M. Dunford and B. Malaescu, *An unfolding method based on conditional Invertible Neural Networks (cINN) using iterative training*, (arXiv preprint) doi:10.48550/arXiv.2212.08674.

[78] M. Leigh, J. A. Raine, K. Zoch and T. Golling, *ν-flows: Conditional neutrino regression*, SciPost Phys. **14**, 159 (2023), doi:10.21468/SciPostPhys.14.6.159.

[79] J. A. Raine, M. Leigh, K. Zoch and T. Golling, *Fast and improved neutrino reconstruction in multineutrino final states with conditional normalizing flows*, Phys. Rev. D **109**, 012005 (2024), doi:10.1103/PhysRevD.109.012005.

[80] A. Shmakov, K. Greif, M. Fenton, A. Ghosh, P. Baldi and D. Whiteson, *End-to-end latent variational diffusion models for inverse problems in high energy physics*, (arXiv preprint) doi:10.48550/arXiv.2305.10399.

[81] J. Ackerschott, R. K. Barman, D. Gonçalves, T. Heimel and T. Plehn, *Returning CP-observables to the frames they belong*, SciPost Phys. **17**, 001 (2024), doi:10.21468/SciPostPhys.17.1.001.

[82] S. Diefenbacher, G.-H. Liu, V. Mikuni, B. Nachman and W. Nie, *Improving generative model-based unfolding with Schrödinger bridges*, (arXiv preprint) doi:10.48550/arXiv.2308.12351.

[83] S. Bieringer, A. Butter, T. Heimel, S. Höche, U. Köthe, T. Plehn and S. T. Radev, *Measuring QCD splittings with invertible networks*, SciPost Phys. **10**, 126 (2021), doi:10.21468/SciPostPhys.10.6.126.

[84] A. Butter, T. Heimel, T. Martini, S. Peitzsch and T. Plehn, *Two invertible networks for the matrix element method*, SciPost Phys. **15**, 094 (2023), doi:10.21468/SciPostPhys.15.3.094.

[85] T. Heimel, N. Huetsch, R. Winterhalder, T. Plehn and A. Butter, *Precision-machine learning for the matrix element method*, (arXiv preprint) doi:10.48550/arXiv.2310.07752.

[86] B. Nachman and D. Shih, *Anomaly detection with density estimation*, Phys. Rev. D **101**, 075042 (2020), doi:10.1103/PhysRevD.101.075042.

[87] A. Hallin, J. Isaacson, G. Kasieczka, C. Krause, B. Nachman, T. Quadfasel, M. Schlaffer, D. Shih and M. Sommerhalder, *Classifying anomalies through outer density estimation*, Phys. Rev. D **106**, 055006 (2022), doi:10.1103/PhysRevD.106.055006.

[88] J. Andrew Raine, S. Klein, D. Sengupta and T. Golling, *CURTAINs for your sliding window: Constructing unobserved regions by transforming adjacent intervals*, Front. Big Data **6**, 899345 (2023), doi:10.3389/fdata.2023.899345.

[89] A. Hallin, G. Kasieczka, T. Quadfasel, D. Shih and M. Sommerhalder, *Resonant anomaly detection without background sculpting*, Phys. Rev. D **107**, 114012 (2023), doi:10.1103/PhysRevD.107.114012.

[90] T. Golling, S. Klein, R. Mastandrea and B. Nachman, *Flow-enhanced transportation for anomaly detection*, Phys. Rev. D **107**, 096025 (2023), doi:10.1103/PhysRevD.107.096025.

[91] D. Sengupta, S. Klein, J. A. Raine and T. Golling, *CURTAINs flows for flows: Constructing unobserved regions with maximum likelihood estimation*, (arXiv preprint) doi:10.48550/arXiv.2305.04646.

[92] A. Valassi, S. Roiser, O. Mattelaer and S. Hageboeck, *Design and engineering of a simplified workflow execution for the MG5aMC event generator on GPUs and vector CPUs*, EPJ Web Conf. **251**, 03045 (2021), doi:10.1051/epjconf/202125103045.

[93] A. Valassi, T. Childers, L. Field, S. Hageboeck, W. Hopkins, O. Mattelaer, N. Nichols, S. ROISER and D. Smith, *Developments in performance and portability for MadGraph5_aMC@NLO*, Proc. 41st Int. Conf. High Energy phys. **ICHEP2022**, 212 (2022), doi:10.22323/1.414.0212.

[94] E. Bothmann, T. Childers, W. Giele, S. Höche, J. Isaacson and M. Knobbe, *A portable parton-level event generator for the high-luminosity LHC*, (arXiv preprint) doi:10.48550/arXiv.2311.06198.

[95] F. Maltoni and T. Stelzer, *MadEvent: Automatic event generation with MadGraph*, J. High Energy Phys. **02**, 027 (2003), doi:10.1088/1126-6708/2003/02/027.

[96] O. Mattelaer and K. Ostrolenk, *Speeding up MadGraph5_aMC@NLO*, Eur. Phys. J. C **81**, 435 (2021), doi:10.1140/epjc/s10052-021-09204-7.

[97] R. Kleiss and R. Pittau, *Weight optimization in multichannel Monte Carlo*, Comput. Phys. Commun. **83**, 141 (1994), doi:10.1016/0010-4655(94)90043-4.

[98] S. Weinzierl, *Introduction to Monte Carlo methods*, (arXiv preprint) doi:10.48550/arXiv.hep-ph/0006269.

[99] W. Kilian, T. Ohl and J. Reuter, *WHIZARD—simulating multi-particle processes at LHC and ILC*, Eur. Phys. J. C **71**, 1742 (2011), doi:10.1140/epjc/s10052-011-1742-y.

[100] G. P. Lepage, *A new algorithm for adaptive multidimensional integration*, J. Comput. Phys. **27**, 192 (1978), doi:10.1016/0021-9991(78)90004-9.

[101] G. P. Lepage, *VEGAS: An adaptive multidimensional integration program*, CLNS-80/447 (1980).

[102] G. P. Lepage, *Adaptive multidimensional integration: VEGAS enhanced*, J. Comput. Phys. **439**, 110386 (2021), doi:10.1016/j.jcp.2021.110386.

[103] T. Ohl, *Vegas revisited: Adaptive Monte Carlo integration beyond factorization*, Comput. Phys. Commun. **120**, 13 (1999), doi:10.1016/S0010-4655(99)00209-X.

[104] S. Brass, W. Kilian and J. Reuter, *Parallel adaptive Monte Carlo integration with the event generator WHIZARD*, Eur. Phys. J. C **79**, 344 (2019), doi:10.1140/epjc/s10052-019-6840-2.

[105] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother and U. Köthe, *Analyzing inverse problems with invertible neural networks*, (arXiv preprint) doi:10.48550/arXiv.1808.04730.

[106] L. Dinh, D. Krueger and Y. Bengio, *NICE: Non-linear independent components estimation*, (arXiv preprint) doi:10.48550/arXiv.1410.8516.

[107] L. Dinh, J. Sohl-Dickstein and S. Bengio, *Density estimation using Real NVP*, (arXiv preprint) doi:10.48550/arXiv.1605.08803.

[108] C. Durkan, A. Bekasov, I. Murray and G. Papamakarios, *Neural spline flows*, (arXiv preprint) doi:10.48550/arXiv.1906.04032.

[109] F. Nielsen and R. Nock, *On the chi square and higher-order chi distances for approximating f-divergences*, IEEE Signal Process. Lett. **21**, 10 (2014), doi:10.1109/lsp.2013.2288355.

[110] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman, T. Plehn, D. Shih and R. Winterhalder, *Ephemeral learning - Augmenting triggers with online-trained normalizing flows*, SciPost Phys. **13**, 087 (2022), doi:10.21468/SciPostPhys.13.4.087.

[111] W. H. Press and G. R. Farrar, *Recursive stratified sampling for multidimensional Monte Carlo integration*, Comput. Phys. **4**, 190 (1990), doi:10.1063/1.4822899.

[112] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, (arXiv preprint) doi:10.48550/arXiv.1412.6980.