Parameter estimation using neural networks in the presence of detector effects

Anders Andreassen[,1,*] Shih-Chieh Hsu,[2,†] Benjamin Nachman[,3,4,‡] Natchanon Suaysom,[2,§] and Adi Suresh[3,5,‖]

[1]*Google, Mountain View, California 94043, USA*
[2]*Department of Physics, University of Washington, Seattle, Washington 98195, USA*
[3]*Physics Division, Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA*
[4]*Berkeley Institute for Data Science, University of California, Berkeley, California 94720, USA*
[5]*Department of Physics, University of California, Berkeley, California 94720, USA*

Histogram-based template fits are the main technique used for estimating parameters of high energy physics Monte Carlo generators. Parametrized neural network reweighting can be used to extend this fitting procedure to many dimensions and does not require binning. If the fit is to be performed using reconstructed data, then expensive detector simulations must be used for training the neural networks. We introduce a new two-level fitting approach that only requires one dataset with detector simulation and then a set of additional generation-level datasets without detector effects included. This simulation-level fit based on reweighting generator-level events with neural networks (SRGN) is demonstrated using simulated datasets for a variety of examples including a simple Gaussian random variable, parton shower tuning, and the top quark mass extraction.

I. INTRODUCTION

Synthetic data produced from Monte Carlo (MC) generators are a key tool for statistical analysis in high energy particle physics. These MC generators have a number of parameters that can be measured by producing multiple synthetic datasets and finding the one that agrees best with data. This procedure can be computationally expensive, especially when detector simulations are involved. In some cases, one can avoid detector simulations by using unfolded data for parameter estimation. Until recently [1,2], unfolding methods were only available for low-dimensional or binned data. Even with the advent of new methods, one can achieve a higher precision with folding instead of unfolding. For example, template-based fitting is the standard approach for extracting the top quark mass [3], one of the most precisely measured quantities at hadron colliders.[1]

---
*ajandreassen@google.com
†schsu@uw.edu
‡bpnachman@lbl.gov
§nsuaysom@uw.edu
‖adisurtya@berkeley.edu
[1]Even with ∼1 GeV theoretical ambiguity [4,5], the uncertainty is still at the 0.5% level.

Machine learning may provide a solution to the simulation challenge. One possibility is to replace or augment synthetic data from slow physics-based generators with synthetic data generated from neural generative models [2,6–36]. This requires neural networks to learn $p(\text{data}|\text{parameters})$ accurately, which is a difficult task. An alternative solution is to instead learn the ratio $p(\text{data}|\text{parameters})/p(\text{data}|\text{reference})$, where the reference may be from a particular synthetic dataset generated with a fixed set of parameters. It is well known [37,38] (also in high energy physics [1,39–49]) that a suitably structured and trained neural network-based classifier learns to approximate this likelihood ratio, so one can turn the difficult problem of probability density estimation into the relatively easier task of classification. Applying this idea to full phase space reweighting and parameter estimation was recently proposed with the deep neural networks using classification for tuning and reweighting (DCTR) protocol [39]. When used to perform an unbinned fit, the original DCTR algorithm first learns a parametrized reweighting function and then continuously (and differentially) modifies the MC generator parameters until the classifier loss used to define the reweighting function is minimized.

The DCTR fitting protocol is effective because it factorizes the reweighting and fitting steps. Furthermore, the fit can be performed with gradient-based methods due to the differentiability of neural networks. However, a key challenge with this approach is that one must train the

reweighting function using data of the same type as the data that are used in the fit. In other words, if the fit is performed with data at detector level, the reweighting function must be trained with a large number of synthetic data examples that include detector effects. As detector simulations can be computationally expensive, this can be a significant challenge.

We propose a new approach whereby only one synthetic dataset with detector effects ("simulation") is required, and all of the reweighting is performed at particle level ("generation") (following the nomenclature from Ref. [1]). This new simulation-level fit based on reweighting generator-level events with neural networks (SRGN) approach still factorizes the problem into a reweighting step and a fitting step, except that now each step includes training classifiers: one at generator level and one at simulation level, respectively. This approach is the same as DCTR in the reweighting step but differs in the fitting step. In the form proposed in this paper, the fitting step is not differentiable, but it is amenable to nongradient-based optimization procedures. Given the computational efficiency of particle-level generation compared with detector-level simulation, this approach will enable new fitting strategies for analyses like the top quark mass measurement, related tasks at the Large Hadron Collider (LHC), and beyond.

Computationally, this new approach requires an up-front cost to train a continuous reweighting function and then a per-evaluation cost to train a classifier to determine a goodness-of-fit. The fitting step could use a computationally cheaper and more standard goodness-of-fit metric at the cost of precision. This is more computationally effective than the traditional approach, which requires many simulated samples at detector level.

This paper is organized as follows. Section II reviews neutral network reweighting and introduces the new two-level approach for incorporating detector effects. A variety of numerical results are presented in Sec. III. In particular, (1) a simple Gaussian example is used to first demonstrate the salient features of the new approach, then (2) parton shower tuning provides a high-dimensional example without detector effects, and finally (3) the top quark mass measurement is deployed for a multidimensional use case including detector effects. The paper ends with conclusions and outlook in Sec. IV.

II. NEURAL NETWORK REWEIGHTING AND DETECTOR EFFECTS

Suppose that features $X \in \mathbb{R}^N$ follow a probability density $p(x|\theta)$, where θ are parameters of the model. A reweighting function $w_{\theta_0}(x, \theta)$ is designed so that a sample drawn from $p(x|\theta_0)$ weighted by w is statistically identical to a sample drawn from $p(x|\theta)$. The ideal reweighting function is $w_{\theta_0}(x, \theta) = p(x|\theta)/p(x|\theta_0)$. One strategy for constructing w is to model the probability density $p(x|\theta)$

and then take the ratio.[2] Density estimation is a significant challenge, especially in the case of collision events where X is a variable and high-dimensional object and $p(x)$ has significant symmetry. One solution is to turn the challenge of density estimation into the relatively easier task of classification. Suppose that f is a neural network trained to distinguish between a sample of events $\boldsymbol{\theta}$ drawn from $p(x|\theta)$ and a sample of events $\boldsymbol{\theta_0}$ drawn from $p(x|\theta_0)$. If f is trained using the binary cross entropy loss function:

$$\text{Loss}(f(x)) = -\sum_{x_i \in \boldsymbol{\theta}} \log(f(x_i)) - \sum_{x_i \in \boldsymbol{\theta_0}} \log(1 - f(x_i)), \quad (1)$$

then with a flexible enough architecture, an effective training protocol, and sufficient training data,

$$\frac{f(x)}{1 - f(x)} \propto \frac{p(x|\theta)}{p(x|\theta_0)}. \quad (2)$$

Therefore, one can construct w using f. Furthermore, if the training of f includes a variety of values of θ, then it will naturally learn to interpolate and become $f(x, \theta)$; consequently, w becomes a parametrized reweighting function.

The original DCTR fitting protocol is expressed symbolically as

$$\theta^*_{\text{DCTR}} = \underset{\theta'}{\text{argmax}} \sum_{x_i \in \boldsymbol{\theta_?}} \log(f(x_i, \theta'))$$
$$+ \sum_{x_i \in \boldsymbol{\theta_0}} \log(1 - f(x_i, \theta')), \quad (3)$$

where $\theta_?$ is not known. If f is the optimal classifier, then $\theta^*_{\text{DCTR}} = \theta_?$.

Detector effects distort the feature space. Let $X_S|X_G \in \mathbb{R}^M$ represent simulation-level features given generator-level features X_G. In synthetic samples, we have the corresponding pairs of X_G and X_S for every collision event. However, X_G is not known for real data. Therefore, it would be ideal to do the fit using X_S, but perform the reweighting using X_G, as reweighting only requires events from generation.

The SRGN protocol is a two-step procedure as illustrated in Fig. 1. First, a reweighting function is trained. Then, a classifier is trained to distinguish the target data from the reweighted simulation. As this classifier is trained, the parameters θ are also modified. When the classifier is unable to distinguish the target data from the reweighted simulation, then the current parameters are the fitted parameters.

Symbolically, suppose that $w_{\theta_0}(x_G, \theta)$ is a reweighting function learned at generator level, where θ_0 represents the

---

[2]This was used in a recently proposed parametric unfolding method in Ref. [50].
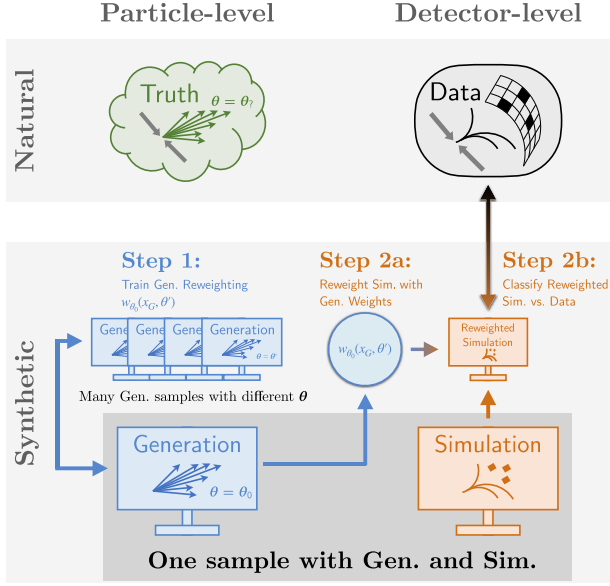
FIG. 1. An illustration of SRGN, applied to a set of synthetic and natural data. There is one synthetic dataset where the particle-level data (generation) is passed through a detector emulation (simulation). SRGN is a two-step process. First, a parametrized reweighting function is learnt using the generation dataset and a set of additional synthetic generator-level datasets. Second, the synthetic simulation is reweighted and compared with real data, iterated to converge on the parameters $\theta_?$. Illustration is inspired by Ref. [1].

nominal parameters for the synthetic sample. Furthermore, suppose that g is a neural network defined as follows:

$$g_{\theta'}(x_S) = \operatorname*{argmax}_{g} \sum_{x_{S,i} \in \boldsymbol{\theta}_?} \log(g_{\theta'}(x_{S,i}))$$

$$+ \sum_{(x_{G,i}, x_{S,i}) \in \boldsymbol{\theta}_0} w_{\theta_0}(x_{G,i}, \theta') \log(1 - g_{\theta'}(x_{S,i})). \quad (4)$$

Then,

$$\theta^*_{\mathrm{SRGN}} \equiv \operatorname*{argmin}_{\theta'} [\mathrm{Pr}(g_{\theta_?}(x_S) > g_{\theta'}(x_S))], \quad (5)$$

where the quantity in $[\cdot]$ is typically called the area under the receiver operating characteristic curve or AUC. We calculate the AUC between g's predictions on events from the unknown sample and g's predictions on reweighted events from the synthetic sample; effectively, if we reweight events from the synthetic sample $\boldsymbol{\theta}_0$ to events from $\boldsymbol{\theta}'$, then we calculate the AUC between g's predictions on $\boldsymbol{\theta}'$ and g's predictions on $\boldsymbol{\theta}_?$.

In analogy to Eq. (3), one might think to define θ^*_{SRGN} as the value of θ' that maximizes the loss in Eq. (4). This would make the SRGN procedure differentiable in contrast to Eq. (5) (the AUC is not generically differentiable). However, one can show that (see the Appendix A)

$$g_{\theta'}(x_S) \approx \frac{p}{\mathbb{E}[w_{\theta_0}(x_G, \theta')|x_S](1 - p) + p}, \quad (6)$$

where $p = \mathrm{Pr}(\theta = \theta_?|x_S)$. When $w = 1$, Eq. (6) is the usual result that the classifier is simply the probability of the target class given the features. Plugging Eq. (6) into Eq. (4) and optimizing with respect to θ' does not generally result in $\theta^* = \theta_?$ (see Appendix B).

The SRGN result defined by Eq. (5) achieves $\theta^*_{\mathrm{SRGN}} = \theta_?$ when the features x_G include the *full phase space*, defined below.

The probability density of the features x_S weighted by $w_{\theta_0}(x_G, \theta)$ is given by

$$p_{\mathrm{weighted}}(x_S|\theta, \theta_0) \equiv \int p(x_S, x_G|\theta_0) w_{\theta_0}(x_G, \theta) dx_G \quad (7)$$

$$\approx \int p(x_S, x_G|\theta_0) \frac{p(x_G|\theta)}{p(x_G|\theta_0)} dx_G \quad (8)$$

$$= \int p(x_S|x_G, \theta_0) p(x_G|\theta) dx_G, \quad (9)$$

where the approximation in Eq. (8) depends on the fidelity of the neural network optimization. Equation (9) is equal to $p(x_S|\theta)$ if $p(x_S|x_G, \theta_0) = p(x_S|x_G, \theta)$. In this case $\theta^*_{\mathrm{SRGN}} = \theta_?$. The equality $p(x_S|x_G, \theta_0) = p(x_S|x_G, \theta)$ holds if x_G contains all of the relevant information about the detector response so that changing θ has no impact on the resolution. In this case, the feature space is said to contain the full phase space (later denoted Ω). Note that it is common in experimental analyses to perform generator-level reweighting for estimating theoretical modeling uncertainties. These reweighting schemes typically use histograms and, therefore, are constrained to one or two-dimensional feature spaces. The above calculation suggests[3] that this is likely insufficient for an unbiased estimate of the impact on simulation-level quantities.

The various properties of the SRGN method will be illustrated in the next section with a variety of examples.

III. RESULTS

Three sets of examples are used to illustrate various aspects of the SRGN method. First, simple Gaussian examples are used, where the probability density is known, and thus the reweighting function can be computed analytically. The features of SRGN described in the previous section are explored with these examples. The parton shower examples from Ref. [39] are used as a second example. These examples show how the new method can be effective

---

[3]We have only shown that if x_G is full phase space, then the procedure is unbiased. However, it could happen that x_G could be less than full phase space, but $p(x_S|x_G, \theta) = p(x_S|x_G, \theta')$ still holds.
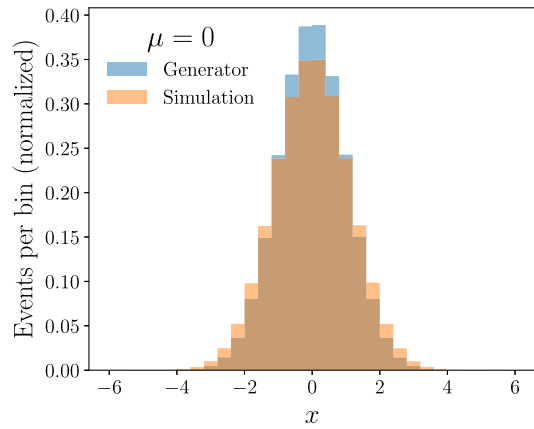
FIG. 2. A histogram of the Gaussian random variable x for $\mu = 0$ at geneator level and simulation level.

with high-dimensional features but do not incorporate detector effects. A measurement of the top quark mass is used as a third example to demonstrate both multivariate fitting and detector effects.

The SRGN protocol calls for two neural networks: one called f that is used to construct the reweighting function w and another called g that is used to perform the fit. These neural networks are implemented using KERAS [51] with the TENSORFLOW backend [52] and optimized with ADAM [53]. Networks are trained using the binary cross entropy loss function. The network architectures vary by example and are described below.

A. Gaussian example

The generator-level feature space is one dimensional and follows a Gaussian distribution: $X_G \sim \mathcal{N}(\mu, \sigma^2)$. Detector effects are modeled as independent Gaussian noise: $X_S = X_G + Z$ where $Z \sim \mathcal{N}(0, \epsilon^2)$. The detector smearing ϵ and the generator width $\sigma = 1$ are known, but μ is not known. In this case, the reweighting function can be computed analytically:

$$w_{\mu_0}(x_G, \mu) = \exp\left(\frac{(x_G - \mu_0)^2 - (x_G - \mu)^2}{2}\right). \quad (10)$$

The parametrized reweighting is trained with μ values sampled uniformly at random in the range $[-2, 2]$. One million examples are used for both data and the nominal synthetic dataset, and $\epsilon = 0.5$. These data for $\mu = 0$ are presented in Fig. 2.

A reweighting function is derived using a neural network with three hidden layers using 50 nodes each. Rectified linear units (ReLU) connect the intermediate layers and the output is Softmax. The training/validation split is 50/50. The network is trained for 200 epochs with early stopping using a patience of 10, and the training time is about 5 seconds per epoch on an NVIDIA Tesla V100 ("Volta") Graphics Processing Unit (GPU). The batch size is 10^5. A
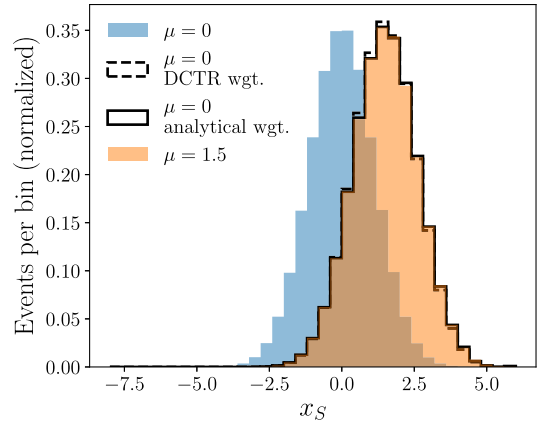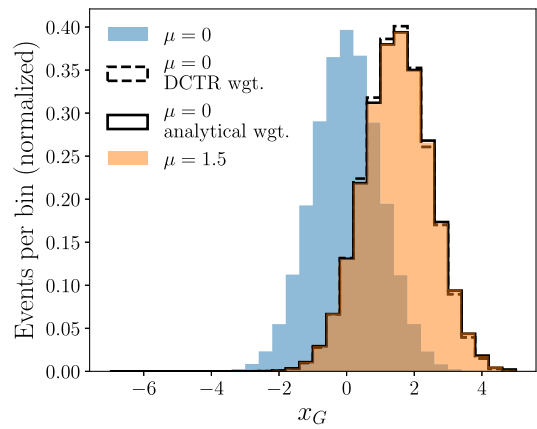

FIG. 3. A demonstration of reweighting for generation (top) and simulation (bottom).

comparison of the analytical [Eq. (10)] and learned reweighting is shown in Fig. 3 using weights based on generator level in both cases. The reweighting effectively morphs the $\mu = 0$ distribution to one that has $\mu = 1.5$.

The goal of SRGN is to use simulated features with reweighting based on generator level. This is explored in Fig. 4. In order to show that the features need not be the same at generator level and simulation level, X_G is two-dimensional. Then, we construct the detector response such that the simulation-level observable X_S depends explicitly on the primary generator-level feature, but its detector smearing depends on the secondary generator-level feature. That is, detector effects are nonuniform and are dependent on the generator-level parameter(s). In particular, we choose the primary generator-level feature $X_{G,0} \sim \mathcal{N}(\mu, 1)$ and the secondary generator-level feature $X_{G,1} \sim \mathcal{N}(0, \nu^2)$, where $\nu = (\omega_0 + \omega_1\mu)^2$ for two constants ω_0 and ω_1. (Specifically, we choose $\omega_0 = 0.7$ and $\omega_1 = 0.2$ for this example.) Then, on a per-event basis, detector effects are emulated by $X_S = X_{G,0} + Z$, where $Z \sim \mathcal{N}(4|x_{G,1}|, (x_{G,1})^4)$, with $4|x_{G,1}|$ representing a net shift bias and $(x_{G,1})^2$ representing a smearing bias. Importantly, the resolution depends on the secondary generator-level feature.

Figure 4 shows the result of a reweighting derived on generator level for ten million events, using the same architecture and training procedure as the previous example. By construction, both the smearing and the shifting are more intense for the $\mu = 1.5$ simulation-level distribution. When using both generator-level features (constituting the full phase space Ω), reweighting is successful. However, if only the primary generator-level feature is used for w, then the reweighting fails to reproduce the simulated-level probability density.

So far, the results have only illustrated the efficacy of reweighting–the rest of the plots in this section demonstrate how the reweighting can be used for fitting. To begin, the one-dimensional generator-level setup is used for the fit. The fitting data consist of one million events with $\epsilon = 0.5$ for detector effects. Then, a classifier is trained with different values of μ to distinguish the unknown dataset from the reweighted synthetic dataset, and the AUC from Eq. (5) is plotted as a function of μ for a fit at both generator level and simulation level. The architecture of this neural network consists of two hidden layers using 128 nodes each. Rectified linear units (ReLU) connect the intermediate layers, and the output is a sigmoid. The training/validation split is 50/50. The network is trained for 200 epochs with early stopping using a patience of 5 and the training time is about 5 seconds per epoch on an NVIDIA Tesla V100 GPU. The batch size is 1000. In both cases, the reweighting is performed at generator level. Figure 5 illustrates several aspects of the proposed fitting method with SRGN. First, the minimum of the AUC is 0.5 and occurs at $\mu = 1$ in both cases, which is the correct value. Second, the rise in the AUC function away from the minimum is steeper at generator level than simulation level, as expected given the loss of statistical power from detector smearing. In addition to showing the AUC function, the values of fits using a nondifferentiable optimizer are also presented as markers in Fig. 5. At both generator level and simulation level, the fit correctly identified $\mu_? = 1$. The numerical results of the one-dimensional Gaussian fit are
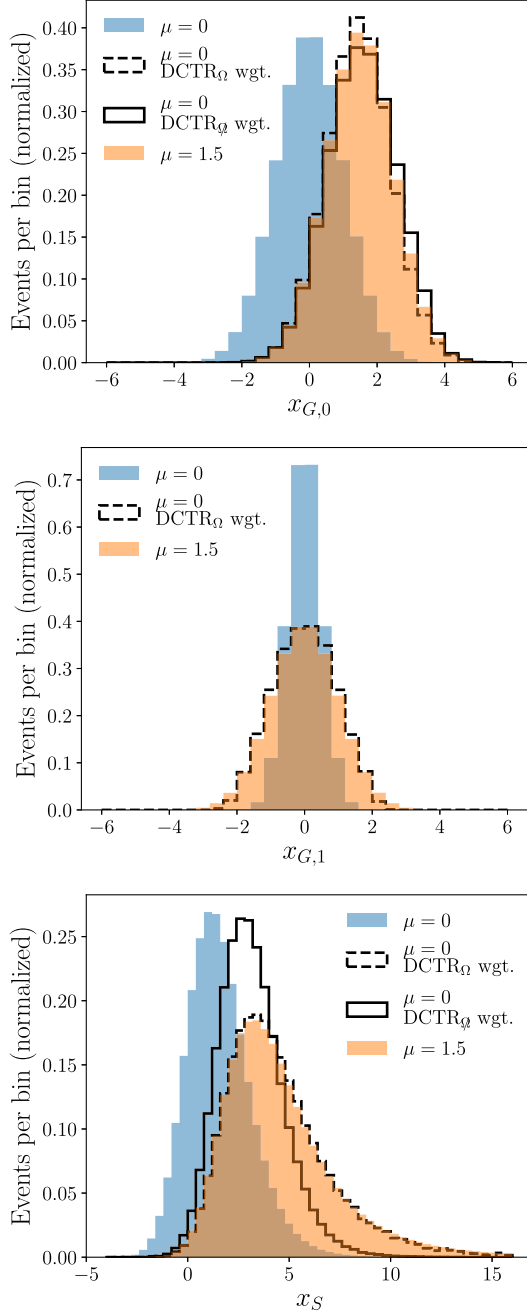

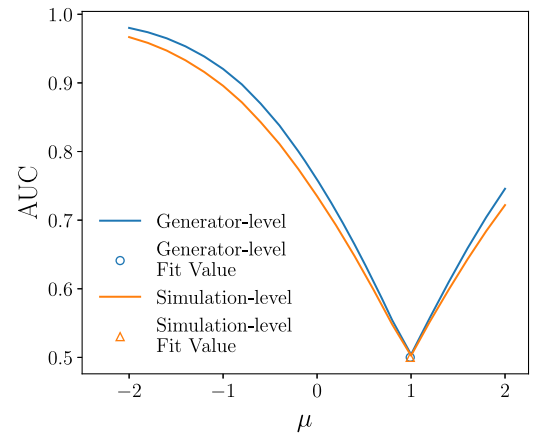FIG. 4. A demonstration of reweighting derived at generator level for the primary generator-level feature (top), the secondary generator-level feature (middle), and simulation level (bottom). In the top and bottom plots, a reweighting using only the primary generator feature is also shown (labeled DCTR$_\phi$ wgt., where "wgt." stands for "reweighted distribution".)

FIG. 5. For an individual run, the AUC versus μ for generator-level and simulation-level features. Also shown are the fitted values at both levels using SCIPY 1.5.2's POWELL nondifferentiable optimizer fitting with 0.0001 as the relative error in both the solution μ_{SRGN} and the function value AUC(μ_{SRGN}) acceptable for convergence, within the bounds [-2, 2] (the same range that the reweighting function is parametrized in), and optimizing for a maximum of 100 iterations.

TABLE I. Numerical results for the one-dimensional Gaussian fit. The reported values and errors represent the mean and standard deviation over the 40 runs (with outliers removed), employing the same nondifferentiable optimization scheme described in Fig 5.

Parameter	Target	Level	Fit value
μ	1.000	Generator	0.998 ± 0.013
		Simulation	1.000 ± 0.017

presented in Table I, and the reported measurements are the averages and standard deviations over 40 runs using a nondifferentiable optimizer. A small number of runs resulted in defective optimization terminating at the fitting bounds; these were identified and removed by examining values outside a 2σ window around the mean, as well as all values within 5% of the range of the bounds.

As a next illustration, a fit is performed for both μ and σ. A two-dimensional reweighting function is parametrized in the one-dimensional Gaussian mean and standard deviation within the ranges [-2, 2] and [0.25, 4], respectively. The reweighting function can still be computed analytically and is given by

$$w_{(\mu_0,\sigma_0)}(x_G,(\mu,\sigma))$$

$$= \frac{\sigma_0}{\sigma}\exp\left(\frac{1}{2}\left(\left(\frac{x_G-\mu_0}{\sigma_0}\right)^2 - \left(\frac{x_G-\mu}{\sigma}\right)^2\right)\right), \quad (11)$$

where μ_0 and σ_0 denote the nominal values for the Gaussian distribution. As before, one million events are used for the fit, and detector effects are modeled with $\epsilon = 0.5$. The efficacy of a two-dimensional reweighting function is presented in Fig. 6 for a case with $\mu_0 = 0$ and $\sigma_0 = 1$. The neural network weights are just as effective as the analytic weights to morph the default distribution into a distribution with $\mu = 1$ and $\sigma = 1.25$.

A two-dimensional fit to μ and σ is demonstrated in Fig. 7. The AUC function is minimized at the correct values of $\mu = -1$ and $\sigma = 0.75$ for both generator level and simulation level for a reweighting function derived at generator level in both cases. The contours in Fig. 7 indicate that the AUC function rises more steeply away from the minimum at generator level, as would be expected of the enhanced statistical power of the dataset without
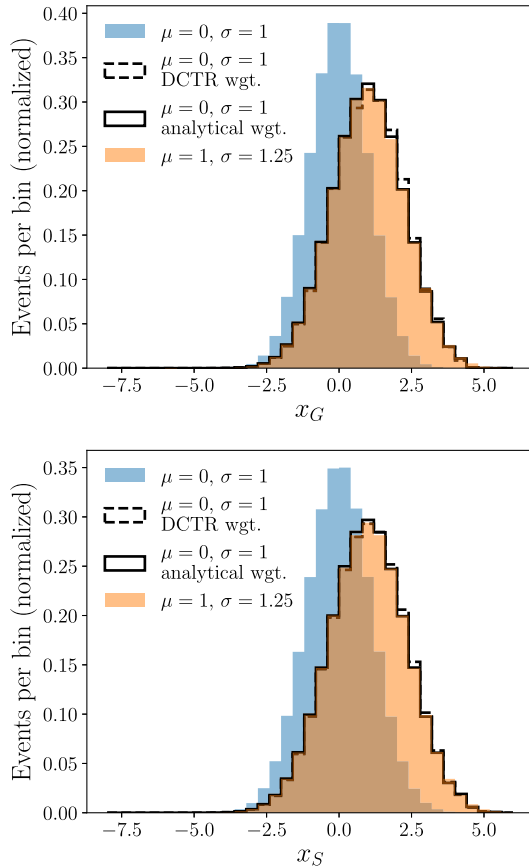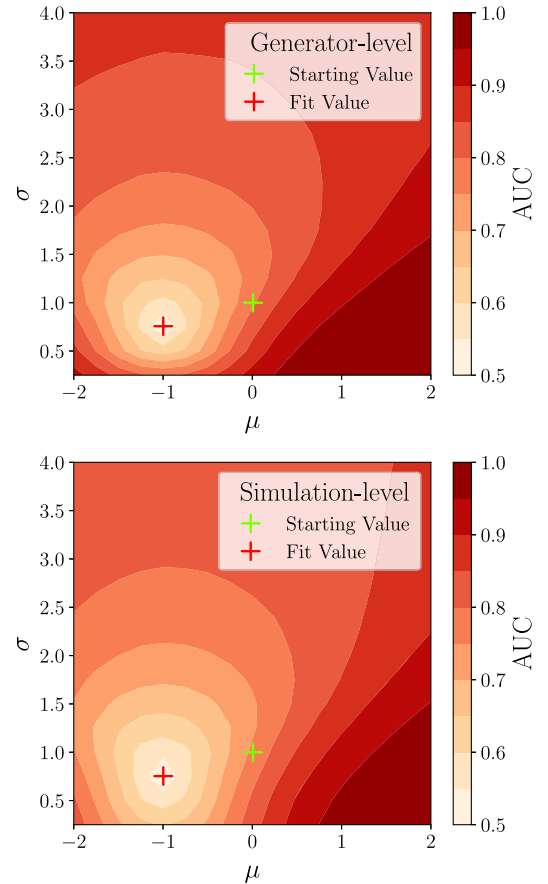

FIG. 6. An illustration of two-dimensional reweighting on generator level (top) and simulation level (bottom). For comparison, the results with analytical weights from Eq. (11) and the results with neural network-based weights are both plotted.

FIG. 7. For an individual run, the AUC as a function of μ and σ. The true values are $\mu = -1.000$ and $\sigma = 0.750$. The values of the nominal synthetic dataset are indicated by a green cross and the nondifferentiable optimizer's fitted values are represented by a red cross.

TABLE II.   Numerical results for the two-dimensional Gaussian fit. The reported values and errors represent the mean and standard deviation over 40 runs (with outliers removed), employing the same nondifferentiable optimization scheme described in Fig. 5 with the addition of fitting σ within the bounds [0.25, 4].

Parameter	Target	Level	Fit value
μ	−1.000	Generator	−0.994 ± 0.014
σ	0.750		0.746 ± 0.013
μ	−1.000	Simulation	−0.997 ± 0.017
σ	0.750		0.747 ± 0.017

detector effects. The numerical results of the two-dimensional Gaussian fit are presented in Table II. The reported measurements and uncertainties are again the averages and standard deviations over 40 runs using a nondifferentiable optimizer. The same procedure for identifying and removing outliers as the one-dimensional fit was employed.

B. Parton shower Monte Carlo tuning

The parton shower tuning examples from Ref. [39] are presented in this section. There are no detector effects, but we show that the new fitting methodology works with high-dimensional features and in particular can be integrated with particle flow networks [54], which are based on deep sets [55]. The event generator details can be found in Ref. [39] and are briefly reviewed here. In particular, $e^+e^- \to Z \to$ dijets are generated using PYTHIA 8.230 [56,57] and anti-k_t [58] $R = 0.8$ jets are clustered using FastJet 3.03 [59,60]. The jets are presented to the neural network for training, with each jet constituent represented by (p_T, η, ϕ, particle type, θ), where θ are the generator parameters to be determined. The neural network setup is the same as in Ref. [39], which uses the default particle flow network parameters from Ref. [54]. Training time was about 20 seconds per epoch on an NVIDIA Tesla V100 GPU.

The default generator parameters follow the Monash tune [61]. Three representative generator parameters are used here to illustrate the SRGN fitting procedure. First, `TimeShower:alphaSvalue` is varied to illustrate a parameter that has a significant impact on the entire phase space and is thus relatively easy to tune. Second, `StringZ:aLund` is a parameter that also impacts the entire phase space but to a lesser extent than the strong coupling constant used in final state radiation. Finally, `StringFlav:probStoUD` is a parameter that has a large impact on a narrow region of phase space. The Monash tune values of the three parameters are 0.1365, 0.68, and 0.217, respectively. For `TimeShower: alphaSvalue` and `StringFlav:probStoUD`, two nearly sufficient one-dimensional statistics are known: the number of particles inside the jets and the number of strange hadrons, respectively. Fits using these simple observables will be compared with the full phase space fit below.

Generator-level features illustrating variations in each of the three parameters are shown in Figure 9. The full phase space will be used in the fit, but these are representative features to illustrate the effects of parameter variations. These features are the same as used in Ref. [39] and are the number of particles inside the jet (multiplicity), the number of kaons inside the jet, an n-subjettiness ratio τ_2/τ_1 [62,63], and a four-point energy correlation function using angular exponent $\beta = 4$ [64] ECF($N = 3, \beta = 4$). As advertised, the final state shower α_s and hadronization parameters affect all four observables, with a bigger shift from α_s. In
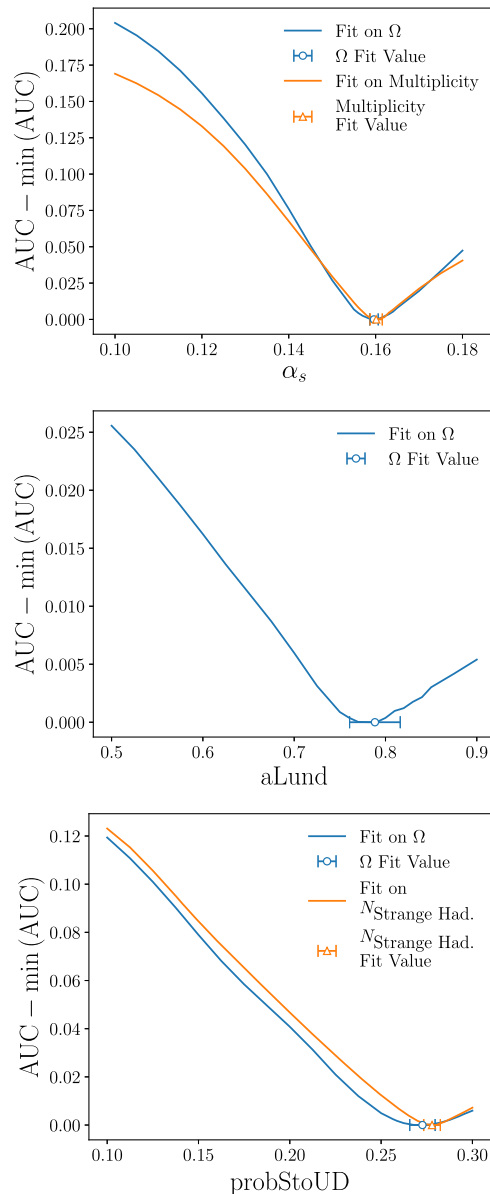

FIG. 8.   One-dimensional fits for each of the three parton shower parameters. The vertical axes show the increase of the AUC for the classifier g from its minimum value. For comparison, the α_s and strangeness plots also show fits using SRGN with only the inclusive or strange hadron multiplicity, respectively.

contrast, the strangeness parameter only affects the number of kaons and has no impact on the other observables. To perform a given fit, we scan for the AUC as a function of the parameter to search for the minimum; the step sizes are 0.001, 0.01, and 0.005 for `TimeShower: alphaSvalue`, `StringZ:aLund`, and `StringFlav: probStoUD`, respectively.

One-dimensional fits to each of the three parton shower parameters are shown in Fig. 8. Since `TimeShower: alphaSvalue` has such a large effect on the phase space, it is the most precisely measured parameter as indicated by the steepness of the AUC curve near the minimum. The steepness of the full phase space fit also shows that there is slightly more information with respect to
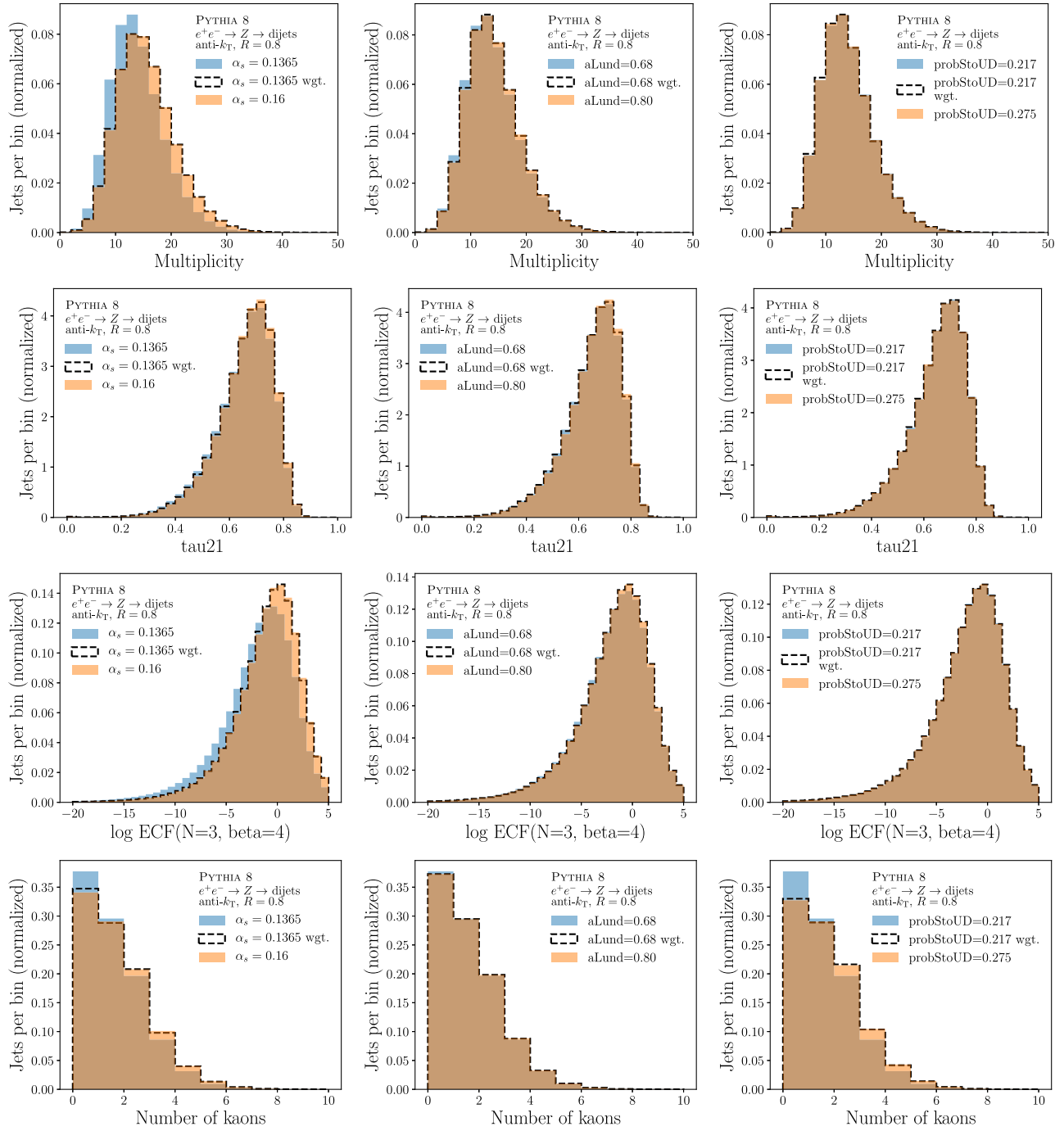

FIG. 9. Features used to show the impact of generator parameter variations for the parton shower dataset. Variations in `TimeShower:alphaSvalue`, `StringZ:aLund`, and `StringFlav:probStoUD` are presented in the first, second, and third columns, respectively. Each row represents a different observable. Reweighted distributions are plotted over an average of 40 reweightings.

TABLE III.   Numerical results for the parton shower parameter fits. The errors represent the standard deviation over 40 runs (with outliers removed). Note that Ω denotes the full phase space.

Parameter	Target	Input	Fit value
TimeShower: alphaSvalue	0.1600	Ω Multiplicity	0.1596 ± 0.0010 0.1601 ± 0.0014
StringZ:aLund	0.8000	Ω	0.7884 ± 0.0277
StringFlav: probStoUD	0.2750	Ω $N_{\text{Strange Had.}}$	0.2726 ± 0.0070 0.2779 ± 0.0045

multiplicity alone. The `StringZ:aLund` parameter has the smallest effect on the phase space of all three parameters and thus is the least precisely measured parameter. `StringFlav:probStoUD` primarily has an effect on the number of strange particles, and thus the full phase space does not offer much more information than only the number of strange hadrons, so the precision is comparable for both approaches. The reported measurements and plots are the averages and standard deviations over 40 runs, each with a different reweighting function and classifier that differed only in their random initialization. A small number of the runs resulted in reweighting functions that were defective and these were identified and removed by examining the runs with fitted values outside a 2σ window around the mean. Across the 40 runs, most of the results clustered around the mean, and so the outliers look systematically different than the fits with effective reweighting functions.

The numerical results of the three fits are presented in Table III. The fitted values are statistically consistent with the target values and the uncertainties are generally comparable to or smaller than the values from the original DCTR protocol [39].

C. Top quark mass

Top quark pair production is generated using PYTHIA 8.230 [56,57] and detector effects are modeled using DELPHES 3.4.1 [65–67] using the default compact muon solenoid (CMS) run card. One of the W bosons is forced to decay to $\mu + \nu_\mu$ while the other W boson decays hadronically. Each event is recorded as a variable-length set of objects, consisting of jets, muons, and neutrinos. At simulation level, the neutrino is replaced with missing transverse momentum. Generator-level and simulation-level jets are clustered with the anti-k_t algorithm using $R = 0.4$ and are labeled as b tagged if the highest energy parton inside the jet cone ($\Delta R < 0.5$) is a b quark. Jets are required to have $p_T > 20$ GeV, and they can only be b tagged if $|\eta| < 2.5$. Furthermore, jets overlapping with the muon are removed.

Events are only saved if they have at least two b-tagged jets and at least two additional non-b-tagged jets. Four observables are formed for performing the top quark mass extraction. First, the b jet closest to the muon is labeled b_1.
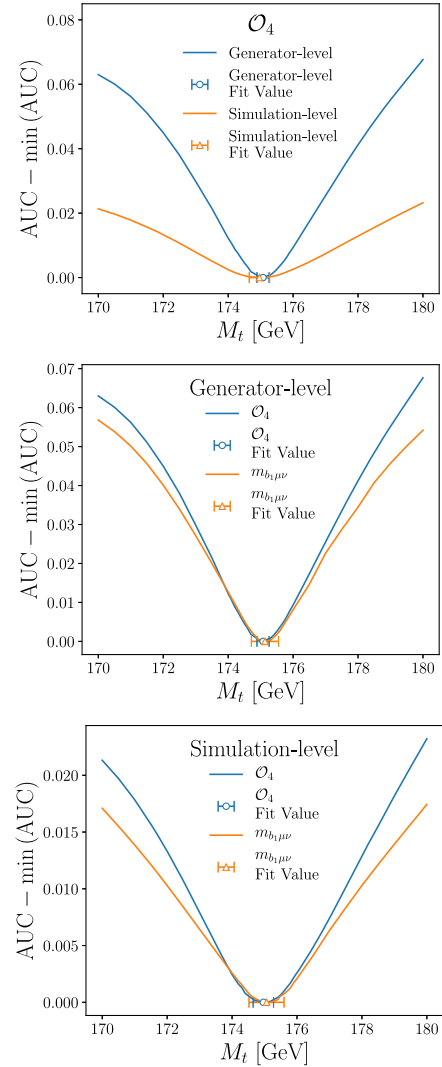

FIG. 10.   One-dimensional fits to the top quark mass. The vertical axes show the increase of the AUC for the classifier g from its minimum value. The top plot uses all four observables and compares the fit at generator level to the fit at simulation level. The middle (bottom) is at generator level (simulation level) and compares the fit with all four observables to the fit with only $m_{b_1\mu\nu}$.

Of the remaining b-tagged jets, the highest p_T one is labeled b_2. The highest two[4] p_T non-b-tagged jets are labeled j_1 and j_2. The four observables are given by: $m_{b_1\mu\nu}$, $m_{b_2\mu\nu}$, $m_{b_1j_1j_2}$, and $m_{b_2j_1j_2}$, where the four-momentum of the detector-level neutrino is determined by solving the quadratic equation for the W boson mass.[5]

Histograms of the four observables for generator level and simulation level are presented in Fig. 11. On both

---

[4]This has a higher efficiency than imposing a W mass constraint, but there is no significant impact on the results.

[5]If there is no solution, then the mass is set to zero. If there are two real solutions, then the one with the smaller $|p_z|$ is selected.

particle and detector levels, one can see that varying the top quark mass M_t has the greatest effect on $m_{b_1\mu\nu}$ and $m_{b_2\mu\nu}$, as opposed to $m_{b_2j_1j_2}$ and $m_{b_1j_1j_2}$. However, the latter two still have some visible dependence on M_t. Therefore, it is expected that fitting on all four observables (denoted $\mathcal{O}_4 = \{m_{b_1\mu\nu}, m_{b_2\mu\nu}, m_{b_2j_1j_2}, m_{b_1j_1j_2}\}$) should yield a more precise fit than fitting on any single one.

The application of the SRGN technique to the top quark mass fit is presented in Fig. 10. Both neural networks used for reweighting and classifying are implemented identically to the Gaussian example, with the exception of increasing early stopping patience to 20; the training time is again about 5 seconds per epoch on an NVIDIA Tesla V100 GPU. The parametrized reweighting is trained with 5.5×10^6
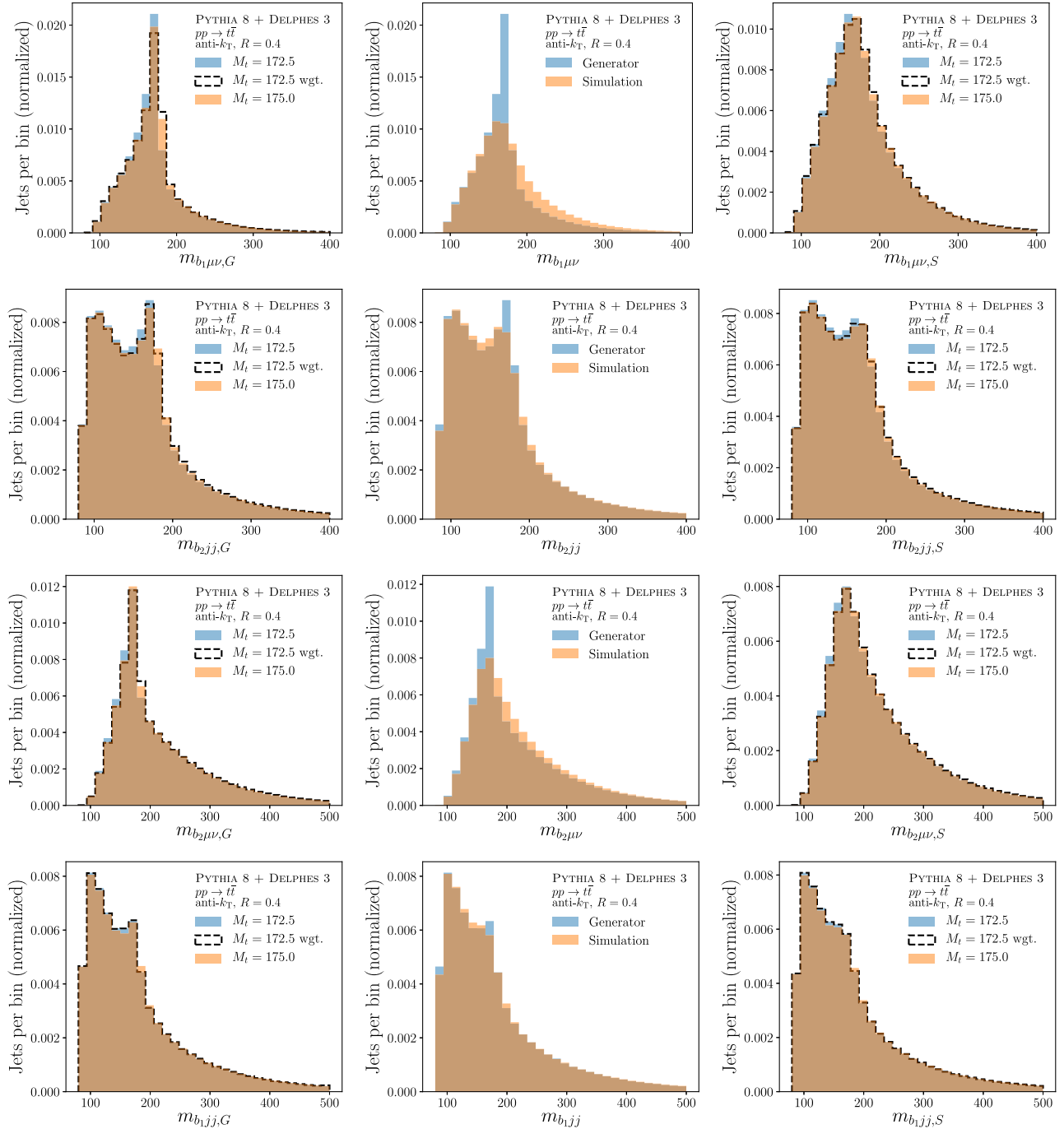

FIG. 11. Histograms for the four observables for (left column) generator level for two different top quark masses, (middle column) a comparison of generator level and simulation level for a fixed top quark mass ($M_t = 172.5$ GeV), and (right column) simulation level for two different top quark masses. Reweighted distributions are plotted over an average of 40 reweightings.

TABLE IV. Numerical results for the top quark mass fit. The reported values and errors represent the mean and standard deviation over the 40 runs (with outliers removed).

Parameter	Target [GeV]	Input	Level	Fit value [GeV]
M_t	175.00	$\mathcal{O}_4$	Generator	175.07 ± 0.19
			Simulation	174.96 ± 0.31
		$m_{bl\nu}$	Generator	175.13 ± 0.42
			Simulation	175.06 ± 0.54

events with M_t values sampled uniformly at random in the range [170 GeV, 180 GeV] against 1.4×10^6 events sampled with $M_t = 172.5$ GeV. To perform a given fit, we scan for the AUC as a function of the top quark mass with a step size of 0.1 GeV to search for the minimum. There are 1.4×10^6 events in the nominal synthetic sample and 1.5×10^6 events in the fitting data. In all cases, the fitted value agrees with the correct mass, $M_t = 175$ GeV. The top plot in Fig. 10 shows that the generator-level fit is much more precise than the simulation-level fit, based on the curvature of the AUC near the minimum. The other two plots in the figure demonstrate a superior precision for the four-dimensional fit compared with the one-dimensional fit. The same ensembling and outlier removal procedure is applied here as in the previous section. Horizontal error bars are the standard deviation across 40 runs (outliers removed) with different random initializations.

Numerical values for the top quark mass fit are presented in Table IV.

IV. CONCLUSIONS AND OUTLOOK

This paper addresses a key challenge with simulation-based inference in the presence of detector effects. In particular, detector simulations are computationally expensive, so it is desirable to construct a method that uses as little detector simulation as possible. We have introduced the SRGN approach that only requires one synthetic event sample with a detector simulation, and all other synthetic event samples need only be known at the generator level. A variety of similar methods have been proposed in Ref. [39,43–46], but they typically require many synthetic event samples with detector simulation.

The SRGN protocol is unbinned and can process multidimensional feature spaces and parameter spaces. In its current form, there is a nondifferentiable step required to optimize the area under the receiver operating characteristic curve. Future refinements of this method may result in a fully differentiable pipeline.

The code for this paper can be found at [68]. The synthetic data used for the numerical examples can be found at [69] for the parton shower fits and at [70] for the top quark mass fits.

APPENDIX A: WEIGHTED LOSS OPTIMIZATION

This section derives Eq. (6), which is the optimal classifier function g, if using the weighted binary cross entropy loss function. Given features (X_G, X_S), labels $Y \in \{0, 1\}$, weighting function w, and function g, a common loss functional to determine g is the binary cross entropy:

$$
\mathcal{L}[g] = -Y \log(g(X_S)) \\
- (1 - Y) w_{\theta_0}(X_G, \theta') \log(1 - g(X_S)). \quad (A1)
$$

Conditioned on $X_S = x_S$, the expected loss is given by

$$
\mathbb{E}[\mathcal{L}|X_S = x_s] = -\mathbb{E}[Y|X_S = x_s] \log(g(x_S)) \\
- \mathbb{E}[(1 - Y) w_{\theta_0}(x_G, \theta')|X_S = x_S] \\
\times \log(1 - g(x_S)). \quad (A2)
$$

In general, Y and $w(X_G)$ are not independent given X_S. However, we are assuming that $p(X_S|X_G)$ is the same in data and in simulation, so these two quantities should be approximately independent so long as the X_G probability density is similar in data and in simulation. With this approximation,

$$
\mathbb{E}[\mathcal{L}|X_S = x_S] \approx -\mathbb{E}[Y|X_S = x_S] \log(g(x_S)) \\
- \mathbb{E}[1 - Y|X_S = x_s] \\
\times \mathbb{E}[w_{\theta_0}(x_G, \theta')|X_S = x_S] \log(1 - g(x_S)). \quad (A3)
$$

By taking the derivative of Eq. (A3) with respect to $g(x)$ and setting it equal to zero, one finds that

$$g^*(x_S) \approx \frac{p}{\mathbb{E}[w_{\theta_0}(X_G, \theta')|X_S = x_S](1-p)+p}, \quad \text{(A4)}$$

where since Y is binary, $p \equiv \mathbb{E}[Y|X_S = x_S] = \Pr(Y = 1| X_S = x_S)$.

Furthermore, note that if $X_S = X_G$ (no detector effects), it is still the case that using $g^*(x)$ will generally not result in $\theta^* = \theta_?$. However, if there is a loss function for g such that when g^* is inserted into the total loss, the result is Eq. (3), then $\theta^* = \theta_?$ (proven in Ref. [39]). We do not know if such a function exists, in general, but when $X_S \neq X_G$, this is irrelevant because g^* cannot depend on X_G as we do not have access to this information for the data.

APPENDIX B: LOSS VS AUC

As noted earlier, one may want to define

$$\theta^*_{\text{SRGN}} = \text{argmin}_{\theta'} \max_g \left(\sum_{i \in \theta_0} \log g(x_{S,i}) \right.$$

$$\left. + \sum_{i \in \theta_?} \frac{f(x_{G,i}, \theta')}{(1 - f(x_{G,i}, \theta'))} \log(1 - g(x_{G,i})) \right). \quad \text{(B1)}$$

However, this generally does not reduce to $\theta^* = \theta_?$. AUC still appears to be a more precise metric for parameter estimation even in the case where loss is employable, as illustrated in Fig. 12. Furthermore, AUC is robust, whereas loss is unpredictable and unreliable for other parameters, as seen in Fig. 13.
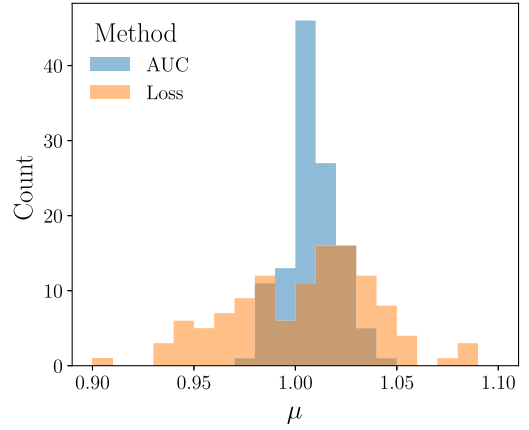

FIG. 12. The distribution of fitted values μ_{SRGN} for the simple one-dimensional Gaussian case over 120 runs, comparing the methods of maximizing loss and minimizing AUC.
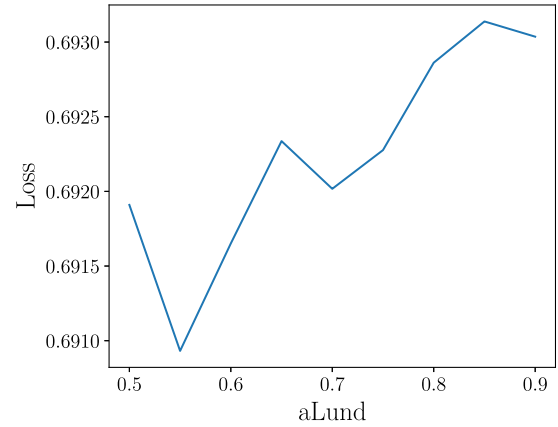

FIG. 13. Plotted here are the fully trained loss values of the classifying step for various values of `StringZ:aLund`, ensembled over 40 runs. It is clear that the loss is not maximized for the correct value of `StringZ:aLund`, 0.8000; conversely, AUC is (in comparison) smoothly minimized at the correct value (Fig. 8).

[1] 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).

[2] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, and R. Winterhalder, How to GAN away detector effects, SciPost Phys. **8**, 070 (2020).

[3] First combination of Tevatron and LHC measurements of the top-quark mass, arXiv:1403.4427.

[4] A. H. Hoang, What is the top quark mass?, Annu. Rev. Nucl. Part. Sci. **70**, 225 (2020).

[5] G. Corcella, The top-quark mass: Challenges in definition and determination, Front. Phys. **7**, 54 (2019).

[6] 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).

[7] 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).

[8] S. Vallecorsa, F. Carminati, and G. Khattak, 3D convolutional GAN for fast simulation, EPJ Web Conf. **214**, 02010 (2019).

[9] V. Chekalina, E. Orlova, F. Ratnikov, D. Ulyanov, A. Ustyuzhanin, and E. Zakharov, Generative models for fast calorimeter simulation, EPJ Web Conf. **214**, 02034 (2019).

[10] ATLAS Collaboration, Deep generative models for fast shower simulation in ATLAS, CERN Report No. ATL-SOFT-PUB-2018-001, 2018.

[11] F. Carminati, A. Gheata, G. Khattak, P. Mendez Lorenzo, S. Sharan, and S. Vallecorsa, Three dimensional generative adversarial networks for fast simulation, J. Phys. Conf. Ser. **1085,** 032016 (2018).

[12] S. Vallecorsa, Generative models for fast simulation, J. Phys. Conf. Ser. **1085,** 022005 (2018).

[13] P. Musella and F. Pandolfi, Fast and accurate simulation of particle detectors using generative adversarial networks, Comput. Software Big Sci. **2,** 8 (2018).

[14] M. Erdmann, L. Geiger, J. Glombitza, and D. Schmidt, Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks, Comput. Software Big Sci. **2,** 4 (2018).

[15] M. Erdmann, J. Glombitza, and T. Quast, Precise simulation of electromagnetic calorimeter showers using a Wasserstein generative adversarial network, Comput. Software Big Sci. **3,** 4 (2019).

[16] L. de Oliveira, M. Paganini, and B. Nachman, Tips and tricks for training GANs with physics constraints (2017).

[17] 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).

[18] B. Hooberman, A. Farbin, G. Khattak, V. Pacela, M. Pierini, J.-R. Vlimant, M. Spiropulu, W. Wei, M. Zhang, and S. Vallecorsa, Calorimetry with deep learning: Particle classification, energy regression, and simulation for high-energy physics (2017).

[19] D. Belayneh *et al.*, Calorimetry with deep learning: Particle simulation and reconstruction for collider physics, Eur. Phys. J. C **80,** 688 (2020).

[20] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, and K. Krger, Getting high: High fidelity simulation of high granularity calorimeters with high speed, arXiv:2005.05334.

[21] L. de Oliveira, M. Paganini, and B. Nachman, Learning particle physics by example: Location-aware generative adversarial networks for physics synthesis, Comput. Software Big Sci. **1,** 4 (2017).

[22] A. Butter, T. Plehn, and R. Winterhalder, How to GAN event subtraction, SciPost Phys. Core **3,** 009 (2020).

[23] J. A. Martinez, T. Q. Nguyen, M. Pierini, M. Spiropulu, and J.-R. Vlimant, Particle generative adversarial networks for full-event simulation at the LHC and their application to pileup description, J. Phys. Conf. Ser. **1525,** 012081 (2020).

[24] C. Ahdida *et al.* (SHiP Collaboration), Fast simulation of muons produced at the SHiP experiment using generative adversarial networks, J. Instrum. **14,** P11028 (2019).

[25] S. Carrazza and F. A. Dreyer, Lund jet images from generative and cycle-consistent adversarial networks, Eur. Phys. J. C **79,** 979 (2019).

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

[27] J. Lin, W. Bhimji, and B. Nachman, Machine learning templates for QCD factorization in the search for physics beyond the standard model, J. High Energy Phys. 05 (2019) 181.

[28] R. Di Sipio, M. F. 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 (2019) 110.

[29] B. Hashemi, N. Amin, K. Datta, D. Olivito, and M. Pierini, LHC analysis-specific datasets with generative adversarial networks, arXiv:1901.05282.

[30] K. Zhou, G. Endrodi, L.-G. Pang, and H. Stocker, Regressive and generative neural networks for scalar field theory, Phys. Rev. D **100,** 011501 (2019).

[31] K. Datta, D. Kar, and D. Roy, Unfolding with generative adversarial networks, arXiv:1806.00433.

[32] K. Deja, T. Trzcinski, and U. Graczykowski, Generative models for fast cluster simulations in the TPC for the ALICE experiment, EPJ Web Conf. **214,** 06003 (2019).

[33] D. Derkach, N. Kazeev, F. Ratnikov, A. Ustyuzhanin, and A. Volokhova, Cherenkov detectors fast simulation using neural networks, Nucl. Instrum. Methods Phys. Res., Sect. A **952,** 161804 (2020).

[34] H. Erbin and S. Krippendorf, GANs for generating EFT models, Phys. Lett. B **810,** 135798 (2020).

[35] J. M. Urban and J. M. Pawlowski, Reducing autocorrelation times in lattice simulations with generative adversarial networks, Mach. Learn. Sci. Technol. **1,** 045011 (2020).

[36] S. Farrell, W. Bhimji, T. Kurth, M. Mustafa, D. Bard, Z. Lukic, B. Nachman, and H. Patton, Next generation generative neural networks for HEP, EPJ Web Conf. **214,** 09005 (2019).

[37] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics (Springer, New York, 2001).

[38] M. Sugiyama, T. Suzuki, and T. Kanamori, *Density Ratio Estimation in Machine Learning* (Cambridge University Press, Cambridge, England, 2012).

[39] A. Andreassen and B. Nachman, Neural networks for full phase-space reweighting and parameter tuning, Phys. Rev. D **101,** 091901 (2020).

[40] C. Badiali, F. Di Bello, G. Frattari, E. Gross, V. Ippolito, M. Kado, and J. Shlomi, Efficiency parameterization with neural networks, arXiv:2004.02665.

[41] M. Stoye, J. Brehmer, G. Louppe, J. Pavez, and K. Cranmer, Likelihood-free inference with an improved cross-entropy estimator, arXiv:1808.00973.

[42] J. Hollingsworth and D. Whiteson, Resonance searches with machine learned likelihood ratios, arXiv:2002.04699.

[43] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, Constraining Effective Field Theories with Machine Learning, Phys. Rev. Lett. **121,** 111801 (2018).

[44] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, A guide to constraining effective field theories with machine learning, Phys. Rev. D **98,** 052004 (2018).

[45] J. Brehmer, F. Kling, I. Espejo, and K. Cranmer, MadMiner: Machine learning-based inference for particle physics, Comput. Software Big Sci. **4,** 3 (2020).

[46] J. Brehmer, G. Louppe, J. Pavez, and K. Cranmer, Mining gold from implicit models to improve likelihood-free inference, Proc. Natl. Acad. Sci. U.S.A. **117,** 5242 (2020).

[47] K. Cranmer, J. Pavez, and G. Louppe, Approximating likelihood ratios with calibrated discriminative classifiers, arXiv:1506.02169.

[48] A. Andreassen, B. Nachman, and D. Shih, Simulation assisted likelihood-free anomaly detection, Phys. Rev. D **101**, 095004 (2020).

[49] M. Erdmann, B. Fischer, D. Noll, Y. Rath, M. Rieger, and D. Schmidt, Adversarial neural network-based data-simulation corrections for jet-tagging at CMS, in *Proc. 19th Int. Workshop on Adv. Comp., Anal. Techn. in Phys. Research, ACAT2019* (2019), https://doi.org/10.1088/1742-6596/1525/1/011001.

[50] N. D. Gagunashvili, Parametric unfolding: method and restrictions, Eur. Phys. J. Plus **135**, 540 (2020).

[51] F. Chollet, Keras, https://github.com/fchollet/keras (2017).

[52] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, Tensorflow: A system for large-scale machine learning, in *OSDI* (2016), Vol. 16, pp. 265–283, https://dl.acm.org/doi/proceedings/10.5555/3026877.

[53] D. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980.

[54] P. T. Komiske, E. M. Metodiev, and J. Thaler, Energy flow networks: Deep sets for particle jets, J. High Energy Phys. 01 (2019) 121.

[55] M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, Deep sets, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17 (Curran Associates Inc., Red Hook, NY, 2017). p. 3394–3404.

[56] T. Sjöstrand, S. Mrenna, and P. Z. Skands, PYTHIA 6.4 physics and manual, J. High Energy Phys. 05 (2006) 026.

[57] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands, An introduction to PYTHIA 8.2, Comput. Phys. Commun. **191**, 159 (2015).

[58] M. Cacciari, G. P. Salam, and G. Soyez, The anti-k_t jet clustering algorithm, J. High Energy Phys. 04 (2008) 063.

[59] M. Cacciari, G. P. Salam, and G. Soyez, FastJet user manual, Eur. Phys. J. C **72**, 1896 (2012).

[60] M. Cacciari and G. P. Salam, Dispelling the N^3 myth for the k_t jet-finder, Phys. Lett. B **641**, 57 (2006).

[61] P. Skands, S. Carrazza, and J. Rojo, Tuning PYTHIA 8.1: The Monash 2013 Tune, Eur. Phys. J. C **74**, 3024 (2014).

[62] J. Thaler and K. Van Tilburg, Identifying boosted objects with N-subjettiness, J. High Energy Phys. 03 (2011) 015.

[63] J. Thaler and K. Van Tilburg, Maximizing boosted top identification by minimizing N-subjettiness, J. High Energy Phys. 02 (2012) 093.

[64] A. J. Larkoski, G. P. Salam, and J. Thaler, Energy correlation functions for let substructure, J. High Energy Phys. 06 (2013) 108.

[65] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lematre, A. Mertens, and M. Selvaggi (DELPHES 3 Collaboration), DELPHES 3, A modular framework for fast simulation of a generic collider experiment, J. High Energy Phys. 02 (2014) 057.

[66] A. Mertens, New features in DELPHES 3, J. Phys. Conf. Ser. **608**, 012045 (2015).

[67] M. Selvaggi, DELPHES 3: A modular framework for fast-simulation of generic collider experiments, J. Phys. Conf. Ser. **523**, 012033 (2014).

[68] https://github.com/hep-lbdl/SRGN.

[69] A. J. Andreassen and B. Nachman, DCTR: Pythia $e^+e^- \to Z \to dijets$ datasets, Zenodo, https://doi.org/10.5281/zenodo.3518708 (2019).

[70] A. Andreassen, S.-C. Hsu, B. Nachman, N. Suaysom, and A. Suresh, SRGN: $Pythia + Delphes \times \ pp \to tt - bar$, Zenodo, https://doi.org/10.5281/zenodo.4067673 (2020).

Correction: The penultimate sentence of the caption to Fig. 1, the first complete sentence after Eq. (5), and the first sentence in footnote 3 contained minor notation errors and have been fixed.

Second Correction: The originally requested change to the sentence after Eq. (5) was improperly implemented by the production staff and has been fixed (a prime was missing on "θ" after "from").