FERMILAB-PUB-24-0784-CSAID, MCNET-24-18

Rejection Sampling with Autodifferentiation

Case study: Fitting a Hadronization Model

Nick Heller,[1,*] Phil Ilten,[2,†] Tony Menzo,[2,‡] Stephen Mrenna,[3,2,§] Benjamin Nachman,[1,¶] Andrzej Siodmok,[4,**] Manuel Szewc,[2,5,††] and Ahmed Youssef[2,‡‡]

[1] *Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA*
[2] *Department of Physics, University of Cincinnati, Cincinnati, Ohio 45221, USA*
[3] *Scientific Computing Division, Fermilab, Batavia, Illinois 60510, USA*
[4] *Jagiellonian University, Łojasiewicza 11, 30-348 Kraków, Poland*
[5] *International Center for Advanced Studies (ICAS), ICIFI and ECyT-UNSAM, 25 de Mayo y Francia, (1650) San Martín, Buenos Aires, Argentina*

We present an autodifferentiable rejection sampling algorithm termed *Rejection Sampling with Autodifferentiation* (RSA). In conjunction with reweighting, we show that RSA can be used for efficient parameter estimation and model exploration. Additionally, this approach facilitates the use of unbinned machine-learning-based observables, allowing for more precise, data-driven fits. To showcase these capabilities, we apply an RSA-based parameter fit to a simplified hadronization model.

I. INTRODUCTION

Simulations are essential tools for connecting fundamental theories with observations in many physical sciences. These simulations are built from physical and phenomenological models containing many parameters. In nearly every analysis of experimental data in particle and nuclear physics, there is a need to produce simulated datasets with one or more of the simulation parameters varied. Traditionally, varying the parameters requires generating new, statistically independent synthetic datasets. Fitting the parameters further requires interpolating between such datasets and this has only been possible by first summarizing the data into a small number of observables. Therefore, the current approaches are unable to use all of the available information and are computationally expensive.

One solution to these challenges is to build sophisticated surrogate models. Modern machine learning (ML), and generative modeling in particular, have introduced a number of tools that can accurately and rapidly emulate physics-based simulations. A key benefit of neural network-based surrogate models is that they are inherently *autodifferentiable* (AD). In the context of parameter fitting, derivatives are computed with respect to statistics comparing data and simulation (e.g. the χ^2 difference of two histograms). Gradient descent is an efficient approach to picking the model parameters that best describe the data. The *auto* in autodifferentiable refers to the ability to compute gradients accurately (machine-precision limited) and efficiently. A number of recent studies have shown how to fit a ML-based surrogate model to data using such a framework [1–6].

While highly flexible, ML surrogate models can also be unwieldy due to their large number of parameters. Alternatively, *differentiable simulations* benefit from both the physical structure of classical simulations and the AD of ML surrogate models. If the simulation itself is written in an AD-compatible programming language, then it could be possible to compute derivatives of statistics through the simulation itself. With far fewer parameters than neural networks, such an approach may be more effective than using a generic deep generative model. There have been a number of proposals for differentiable physics simulators in particle and nuclear physics, including for matrix element calculations [7–9], parton showers [10], and detector simulations [11–15].

An essential incompatibility between AD and Monte Carlo (MC) simulations is that the latter contains many operations that are not differentiable[1]. One widely-used technique in MC simulations is rejection sampling, which is a general algorithm for sampling from a probability density whose normalization is not known. By construction, a sample is accepted or rejected based on a (pseudo-) random number. Changes to the model parameters change the accept-reject threshold value. The derivative of the samples with respect to the underlying parameter is almost everywhere zero except at one value, where it is infinity.

We propose a new technique that allows for the autodifferentiation of rejection sampling without making any approximations. The key idea is to use event weights that are differentiable. Reweighting MC event samples is a powerful technology for the efficient exploration of model parameter-space [17]. An efficient algorithm, based on standard accept-reject sampling, for computing event weights in the absence of an explicit normalized probability density function was introduced in refs. [18, 19] and

* nheller@berkeley.edu
† philten@cern.ch
‡ menzoad@mail.uc.edu
§ mrenna@fnal.gov
¶ bpnachman@lbl.gov
** andrzej.siodmok@uj.edu.pl
†† szewcml@ucmail.uc.edu
‡‡ youssead@ucmail.uc.edu

[1] See ref. [16] for some cases where approximations are possible.

used to provide uncertainty estimation associated with different parton shower parameterizations. A similar procedure was recently developed in ref. [20] to provide hadronization uncertainty estimation. In this work, we employ this idea to build an automated black-box parameter tuning referred to from here on as *Rejection Sampling with Autodifferentiation* (RSA). To demonstrate this method, we show how to fit the parameters of a hadronization model, which has gained recent attention in the context of ML surrogate models [1–6]. An AD hadronization model is an alternative approach to generic neural networks that builds in physical priors in order to significantly reduce the number of parameters needed to describe the data. Along the way, we explore new ways of building test statistics for the parameter fitting using tools from optimal transport.

The paper is structured as follows: In Sec. II we introduce rejection sampling and the RSA methodology, in Sec. III we outline the RSA-based algorithm for parameter estimation and introduce the problem in the context of hadronization models, in Sec. IV we describe the data generation pipeline, in Sec. V we summarize our results and finally in Sec. VI we conclude with final remarks and highlight future work.

II. REJECTION SAMPLING WITH AUTODIFFERENTIATION

A. Accept-reject sampling

Consider a random variable x over the domain $\mathcal{X}$ with a probability density $P(x;\boldsymbol{\theta})$ known up to a normalizing constant with $\mathcal{N}$ tuneable parameters $\boldsymbol{\theta} \equiv \{\theta_1, \cdots, \theta_\mathcal{N}\}$. Assume the maximum $\hat{P}$ is known (or can be estimated numerically) such that $P(x) \leq \hat{P}$ for all $x \in \mathcal{X}$. Samples of x can be obtained by:

1. Uniformly sampling a trial value $x_1 \in \mathcal{X}$.
2. Computing the acceptance probability
$$P^1_{\text{accept}}(x_1;\boldsymbol{\theta}) = \frac{P(x_1;\boldsymbol{\theta})}{\hat{P}}. \tag{1}$$
3. Sampling an additional uniformly distributed value $s_1 \in [0, \hat{P}]$.
4. If $P^1_{\text{accept}}(x_1) \leq s_1$ accept the trial x_1, otherwise, reject and return to step 1. Repeat n-trials until $P^n(x_n) \leq s_n$ i.e. until a trial is accepted.

Note that $\hat{P}$ is unbounded from above, allowing the efficiency of the accept-reject algorithm to be tuned, with values larger than the minimum $\hat{P}_{\text{min}} \equiv \max[P(x)]$ decreasing the efficiency. A single element of an accept-reject sampling will have one accepted value x_{accept} and $n-1$ rejected trials $\boldsymbol{x}_{\text{reject}} \equiv \{x^1_{\text{reject}}, \cdots, x^{n-1}_{\text{reject}}\}$.

B. Alternative accept-reject sampling

Given a sample $\boldsymbol{x}$, a different sample using the same probability model but with different parameters $\boldsymbol{\theta'} \neq \boldsymbol{\theta}$ can be obtained by assigning each trial $x_i \in \boldsymbol{x}$ a statistical weight [18]. The total weight of each element is obtained by assigning the accepted value the relative acceptance probability

$$w_{\text{accept}} \equiv \frac{P_{\text{accept}}(x_{\text{accept}};\boldsymbol{\theta'})}{P_{\text{accept}}(x_{\text{accept}};\boldsymbol{\theta})} = \frac{P(x_{\text{accept}};\boldsymbol{\theta'})}{P(x_{\text{accept}};\boldsymbol{\theta})}, \tag{2}$$

and each rejected trial value the relative rejection probability

$$w^j_{\text{reject}} \equiv \frac{1 - P_{\text{accept}}(x^j_{\text{reject}},\boldsymbol{\theta'})}{1 - P_{\text{accept}}(x^j_{\text{reject}},\boldsymbol{\theta})} = \frac{\hat{P} - P(x^j_{\text{reject}},\boldsymbol{\theta'})}{\hat{P} - P(x^j_{\text{reject}},\boldsymbol{\theta})}. \tag{3}$$

The full weight, w, of an element in the sample is given by

$$w = w_{\text{accept}} \prod_{j=1}^{n-1} w^j_{\text{reject}}. \tag{4}$$

These weights encode the likelihood ratio between two alternative parameterizations of the same distribution with $w < 1$ and $w > 1$ implying the sample is less or more probable in the new parameterization $\boldsymbol{\theta'}$.

Assuming that $P(x)$ is smooth, the above prescription is inherently differentiable and facilitates the computation of *weight gradients* with respect to $\boldsymbol{\theta'}$. If $P(x)$ is sampled repeatedly within a larger simulation pipeline, we may also compute gradients of simulation outputs (which may be arbitrarily complicated functions of the sampled values of x) via weighted binned or *unbinned* distributions. In practice, this requires the ability to efficiently compute and store numerical gradients. Modern machine learning requirements meet this demand with well-established, GPU-accelerated, differential programming libraries, such as PyTorch [21] (used in this paper), providing powerful numerical autodifferentiation engines that allow for efficient, out-of-place gradient computations. The parameters of interest can be implemented as differentiable (learnable) objects within custom classes (such as a PyTorch `Module`) whose forward function facilitates the extraction of weights for a given batch of training data. We refer to this melding as *Rejection Sampling with Autodifferentiation* (RSA).

III. PARAMETER ESTIMATION WITH RSA

A practical use-case for our studies is to consider the fitting or tuning of parameters inside of Monte Carlo event generators [22–24]. Tuning consists of simulating a large number of events with a 'base' parameterization $\boldsymbol{\theta}$, comparing the output distributions to experimental data,

and updating the base parameterization to a new parameterization $\boldsymbol{\theta}'$ following a prescribed procedure [25–29]. The first and third steps present as the primary bottlenecks encountered when performing full event-generator tunes. The generation of simulated events at each parameter point can take $\mathcal{O}$(hours/million events) of CPU time. This makes even a modest tuning exercise (a few parameters scanned over a hypercube consisting of 5–10 parameter values per dimension) computationally prohibitive. The computational demand can be further compounded by the non-trivial task of choosing an updated parameterization, with inefficient choices significantly increasing computation time. Below, we describe a novel tuning paradigm that alleviates this computational burden through statistical reweighting[2] and utilizes modern autodifferention and optimization paradigms to choose updated parameterizations. This approach is similar but distinct to that of DCTR [30] which also performs event reweighting and parameter tuning, but utilizes a deep neural network classifier (instead of the analytic unnormalized density function) to extract event weights and perform parameter tuning. Other machine-learning-inspired tuning paradigms, such as those described in [31], do not utilize reweighting. Below we will illustrate the efficacy of parameter estimation using RSA in the context of fitting a hadronization model.

A. Hadronization Models

Hadronization models are used to connect particle-level measurements to perturbative quantum-chromodynamic (QCD) calculations. A model posits a mechanism for converting quarks and gluons (and possibly diquarks) into the observed hadrons, and is implemented through an algorithm and parametrized probability distributions. Precision tests of the standard model at colliders rely on these hadronization models, which are embedded inside event generators. It is of vital importance to test the validity of our hadronization models and to determine what range of parameters are consistent with data. This is often a time-consuming task, because the models depend on many parameters, each of which must be propagated forward to make a comparison with data.

In this work, we consider the Lund string fragmentation model [32, 33] as implemented in Pythia [22]. Uncertainties on the parameters of this model often lead to significant uncertainties on theory predictions compared to data at colliders. While the model can be more complex, we consider a simpler version and focus only on tuning kinematic parameters, which are assumed to be universal for all quark and diquark types. Flavor parameters are left unchanged from the current default values.

Hadron kinematics (p_x, p_y, p_z, E) at each string break are governed by the sampling of two correlated distributions – kinematics transverse to the string are sampled from a Gaussian

$$\mathcal{P}(p_x, p_y; \sigma_{p_T}) = \frac{1}{2\pi\sigma_{p_T}} \exp\left(-\frac{p_x^2 + p_y^2}{2\sigma_{p_T}^2}\right), \quad (5)$$

while kinematics longitudinal to the string motion are governed by the left-right Lund fragmentation function, defined up to a normalization constant as

$$f(z) \propto \frac{(1-z)^a}{z} \exp\left(-\frac{bm_T^2}{z}\right), \quad (6)$$

where σ_{p_T}, a, and b are tuneable parameters fit to data, z is the longitudinal momentum fraction defined as

$$z \equiv \frac{(E \pm p_z)_{\text{hadron}}}{(E \pm p_z)_{\text{string}}}, \quad (7)$$

with $m_T^2 \equiv m_h^2 + p_T^2$. The parameter b sets the importance of different string areas, the parameter a sets the fragmentation dependence as $z \to 1$, and σ_{p_T} sets the range of p_T values at each string break. Note that while the normalization factor cannot be calculated analytically, the maximum of $f(z)$ can be. For more details regarding the explicit implementation of the Lund model within Pythia, see ref. [22].

B. Fitting a hadronization model with RSA

To showcase the power of RSA, we focus on the two parameters which render rejection-based sampling necessary, a and b, and keep σ_{p_T} fixed to its default value, 0.335. To fit the a and b parameters of the Lund fragmentation function using RSA requires a data-structure comprised of three components: (i) the fragmentation-chain-level rejection sampling data e.g. the accepted and rejected longitudinal momentum fraction samples $\boldsymbol{z}$ as well as the transverse mass m_T of the fragmenting hadron, see eq. 6; (ii) the desired measurable observables from simulated hadronization events produced by the base parameterization $\boldsymbol{y}_{\text{sim}}$ and (iii) values of the same observables measured experimentally $\boldsymbol{y}_{\text{exp}}$.

Our simulation consists of the hadronization of a simple string system based on the probabilistic model described in Sec. III A. An example of the training data,

[2] For a detailed discussion and explicit timing comparisons between reweighted and re-simulated events we refer the reader to ref. [20].

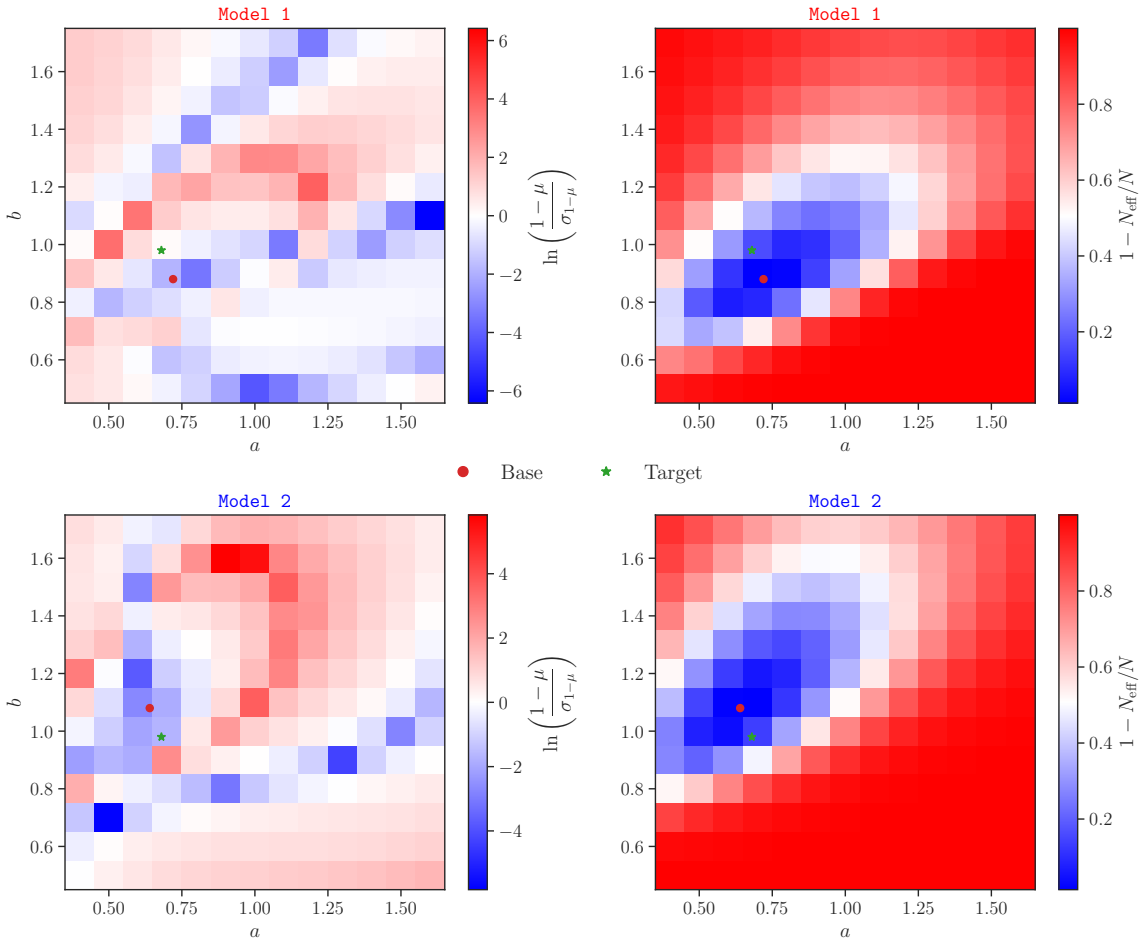

FIG. 1. The reweighting metrics defined in eq. 11 over the (a, b) parameter plane for the `Model 1` and `Model 2` base parameterizations denoted by the red dots. The green star denotes the 'experimental' or target parameterization. For both metrics, values closer to zero imply better coverage and effective statistics.

consisting of N hadronization events, is therefore[3]

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{z}_1 = \begin{pmatrix} \{m_T^{h_1}, z_{\text{accept}}^{h_1}, z_{\text{reject}}^{1,h_1}, \cdots, z_{\text{reject}}^{n_{h_1},h_1}\} \\ \{m_T^{h_2}, z_{\text{accept}}^{h_2}, z_{\text{reject}}^{1,h_2}, \cdots, z_{\text{reject}}^{n_{h_2},h_2}\} \\ \{m_T^{h_3}, z_{\text{accept}}^{h_3}, z_{\text{reject}}^{1,h_3}, \cdots, z_{\text{reject}}^{n_{h_3},h_3}\} \end{pmatrix}_1 \\ \boldsymbol{z}_2 = \begin{pmatrix} \{m_T^{h_1}, z_{\text{accept}}^{h_1}, z_{\text{reject}}^{1,h_1}, \cdots, z_{\text{reject}}^{n_{h_1},h_1}\} \\ \vdots \\ \{m_T^{h_4}, z_{\text{accept}}^{h_4}, z_{\text{reject}}^{1,h_4}, \cdots, z_{\text{reject}}^{n_{h_4},h_4}\} \end{pmatrix}_2 \\ \vdots \\ \boldsymbol{z}_N = \cdots \end{pmatrix}, \tag{8}$$

where n_{h_i} refers to the total number of rejections for the i^{th} fragmentation/hadron. Note that a string system will fragment hadrons until its invariant mass reaches a predetermined threshold (usually set at 1 GeV). When this threshold is reached, a different function is used to partition the remaining energy and momentum of the system between the final two hadrons such that energy and momentum is conserved and the left-right symmetry of the Lund model is preserved. This function, called `finalTwo`, can fail (roughly $10-15\%$ of the time for a $q\bar{q}$ string system with total energy of 50 GeV). Upon failure, the previous hadronization event is discarded and the hadronization routine is re-run on a reinitialized string. For this technical reason, the hadron multiplicity N_h of an event is not necessarily equal to the total number of accepted z-values.

[3] While not denoted explicitly in eq. 8, each array $\boldsymbol{z}_i$ is zero-padded to a fixed length. In principle, the series of rejections can be stored with the random number seed used in the numerical algorithm.

Because RSA utilizes reweighting to explore the model parameter space, at least one set of events, generated using a base parameterization $\boldsymbol{\theta}_B \equiv \{a, b\}_B$, is required

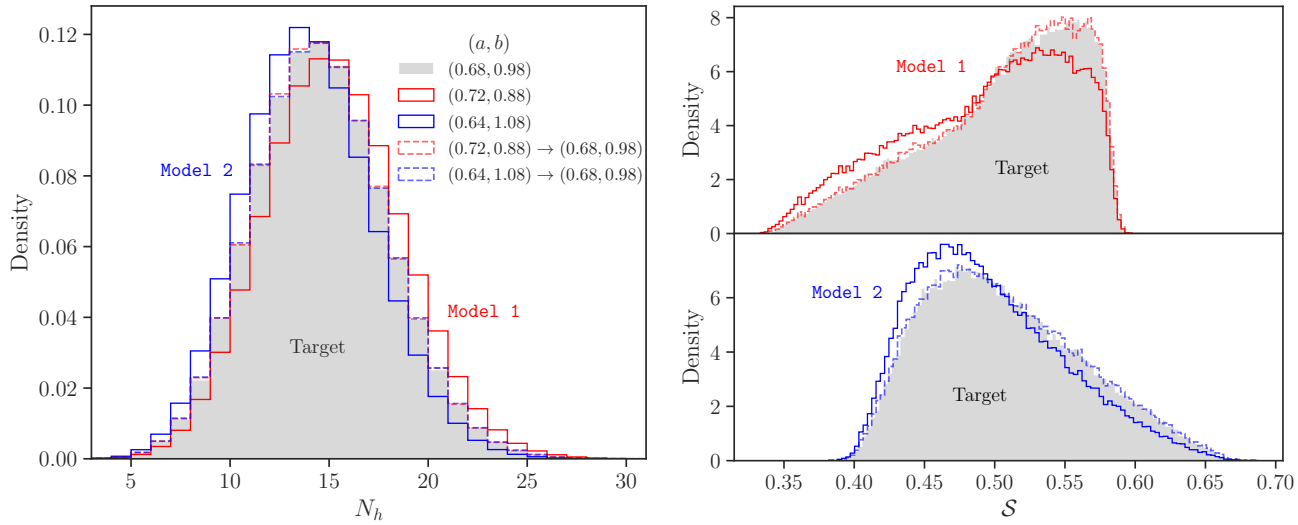

FIG. 2. Observable distributions for hadron multiplicity N_h (left) and classifier score $\mathcal{S}$ (right) at three distinct parameterizations of the Lund a, b parameters. Dotted histograms denote the reweighed distributions.

for tuning[4]. Given the unnormalized Lund fragmentation function, $f(z;\boldsymbol{\theta})$, the forward pass of training proceeds by first computing an event-weight array $\boldsymbol{w}$ for the proposed parameters, $\boldsymbol{\theta}_P \equiv \{a,b\}_P$, given by

$$\boldsymbol{w} = \begin{pmatrix} w_1 & w_2 & \cdots & w_N \end{pmatrix}^T, \text{ where} \tag{9}$$

$$w_n = \prod_{i=1}^{\tilde{N}_{h,n}} \left(\frac{f(z^{h_i}_{\text{accept}}; \{a,b\}_P)}{f(z^{h_i}_{\text{accept}}; \{a,b\}_B)} \right) \times \prod_{j=1}^{n_{h_i}} \left(\frac{\hat{f} - f(z^{j,h_i}_{\text{reject}}; \{a,b\}_P)}{\hat{f} - f(z^{j,h_i}_{\text{reject}}; \{a,b\}_B)} \right), \tag{10}$$

and $\hat{f}$ is the oversampling factor associated with the sampling of z and $\tilde{N}_{h,n}$ denotes the the total number of distinct accept-reject samplings in the n^{th} event (including those when `finalTwo` fails). Once the event-weight array has been computed, a 'cost' or 'loss' function must be defined and minimized to allow for the determination of the next set of parameters $\boldsymbol{\theta}_P$. The loss will parameterize the difference between $\boldsymbol{y}_{\text{exp}}$ and the weighted $\boldsymbol{y}_{\text{sim}}$ ensembles, see Sec. V. Once the loss has been computed, PyTorch's automatic numerical differentiation engine `autograd` paired alongside a chosen optimization algorithm (`Adam`, `SGD`, ... , etc.) can be used to choose the next model parameterization i.e. fragmentation function $f(z;\boldsymbol{\theta})$.

[4] When fitting to experimental data (or when a good guess for initial parameters is unknown) multiple base parameterizations would be desirable/required. For a detailed discussion see Sect. IV

IV. DATA

As a proof of principle for our method, we fit a simplified hadronizing system, namely a quark anti-quark pair hadronizing to pions ($q\bar{q} \to \pi$'s), using the Lund string model of hadronization. The events are generated from a 'particle gun'[5] where the initial configuration is a $q\bar{q}$ system outgoing back-to-back oriented along the z-axis with each quark having equal energies E. We simulate 10^6 of the specified hadronization events with $E = 50$ GeV using three parameterizations of the Lund fragmentation function: $\boldsymbol{\theta}_1 \equiv$ `Model 1` $= \{a = 0.72, b = 0.88\}$, $\boldsymbol{\theta}_2 \equiv$ `Model 2` $= \{a = 0.64, b = 1.08\}$, and the PYTHIA default $\boldsymbol{\theta}_{\text{exp}} \equiv$ Target $= \{a = 0.68, b = 0.98\}$. Both `Model 1` and `Model 2` will be used as base parameterizations during the fits. The rejection sampling uses an over-sampling factor of $\hat{f} = 10$, each accept-reject array is zero-padded to a fixed length of 200[6], and each event is zero-padded to a fixed length of 75.

The efficacy/performance of a reweighting between two parameterizations can be summarized by two metrics

$$\mu \equiv \sum_{i=1}^{N} \frac{w_i}{N}, \quad N_{\text{eff}} = \frac{\left(\sum_{i=1}^{N} w_i\right)^2}{\sum_{i=1}^{N} w_i^2}. \tag{11}$$

The two metrics provide complementary reweighinting diagnostics – the first gives a measure of possible lack

[5] See, for example, `main234` within the `Pythia/examples` subdirectory of release 8.312.

[6] We do not allow for > 199 rejections. This is a convention chosen to reduce the memory footprint of the accept-reject datasets required for reweighting. We have checked empirically that this choice does not effect fit performance.

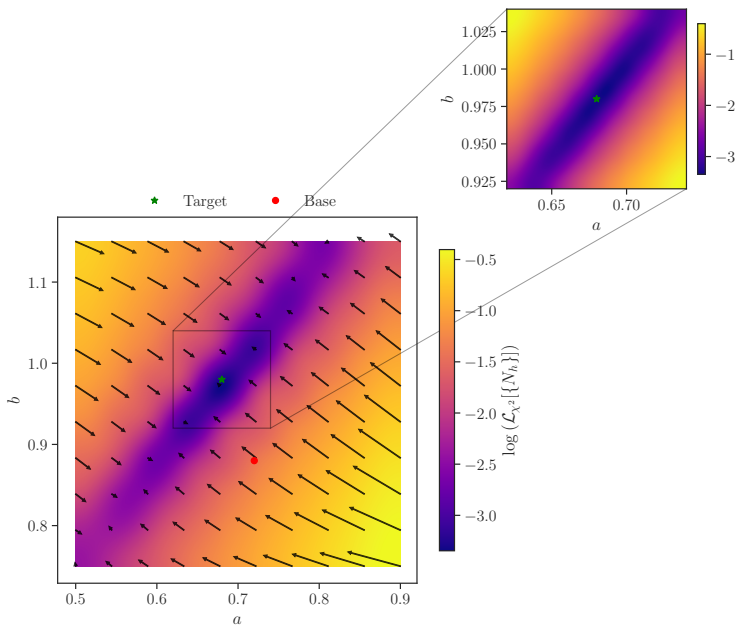

FIG. 3. An example of the two-dimensional Lund a and b parameter plane loss landscape for the pseudo-χ^2 loss defined in eq. 15. The red dot denotes the initial base parameterization from which we reweigh from and the green cross denotes the target parameterization from which the observable distribution was generated from. Both subplots are generated by computing the loss and gradients over a grid of 100 (a, b) pairs. For each parameterization the losses and gradients are averaged over four mini-batches of 5×10^4 events for a total 'batch size' of 2×10^5 events.

of coverage or 'unitarity violation' between the two parameterizations and the second reflects the possible loss in statistical power of the reweighed sample. For two parameterizations with sufficiently good coverage we expect $\mu \approx 1$. Likewise, $N_{\text{eff}}/N \approx 1$, with N being the total number of samples used during reweighting, would signal very little statistical power lost to reweighting. In figure 1 we show both of these reweighting metrics across the (a, b) parameter plane for both `Model 1` and `Model 2`. We express μ (left column) as $\ln((1-\mu)/\sigma_{1-\mu})$, where $\sigma_{1-\mu}$ is the uncertainty on $1-\mu$, to better assess consistency with $\mu = 1$. Written this way and for the sample sizes considered in this work, $\ln((1-\mu)/\sigma_{1-\mu})$ in the approximate range $(-1, 1)$ are considered as reflecting adequate reweighting. In general, values closer to 0 imply better coverage, values $\gg 0$ are inconsistent with 0 due to lack of coverage, and values $\ll 0$ indicate weights with large variance and signal poor reweighting performance. We see that the target parameterizations (green stars) sit in regions where the reweighting metrics indicate good coverage and effective statistics in both base models (red dots). We also see that the full parameter range of interest contains large regions with poor coverage and effective statistics. When performing a black-box fit (where a good estimate of model parameters is unknown) over the full parameter range, one would need to utilize multiple base parameterizations that, when combined, provide adequate coverage over the full parameter volume.

A. Observables

We categorize observables as either 'high-level' or 'low-level'. High-level data utilizes currently available event-level/jet-level observables such as (charged) hadron multiplicity, event-shape variables like thrust, etc. Low-level data, on the other hand, assumes access to all hadron-level data that can in principle be extracted from experimental data, such as the energy and momenta of all hadrons within an event.

For tuning, we consider two observables – one high-level and one low-level. The high-level observable is the hadron multiplicity N_h that counts the total number of hadrons in the hadronization chain accepted by `finalTwo`. The multiplicity distributions are shown in the left panel of figure 2 for the three model parameterizations as well as the reweighted distributions following eq. 10. The second observable is the low-level *truth score* $\mathcal{S}$. The truth score is the output of a trained DeepSets classifier [34, 35] that distinguishes between full low-level input from simulated (base) and experimental (truth/target) data. The classifier consists of a per-particle multi-layer-perceptron (MLP) encoder ϕ, an aggregation function chosen to be simple addition, and a MLP decoder ρ. Both the encoder and decoder networks consist of three layers and 64 hidden nodes. The input consists of low-level pion observables *e.g.* four-momenta that were preprocessed into transverse momentum, polar angle, and pseudorapidity $\mathbf{\Theta}_h^i \equiv (p_{T,i}, \phi_i, \eta_i)$ measured

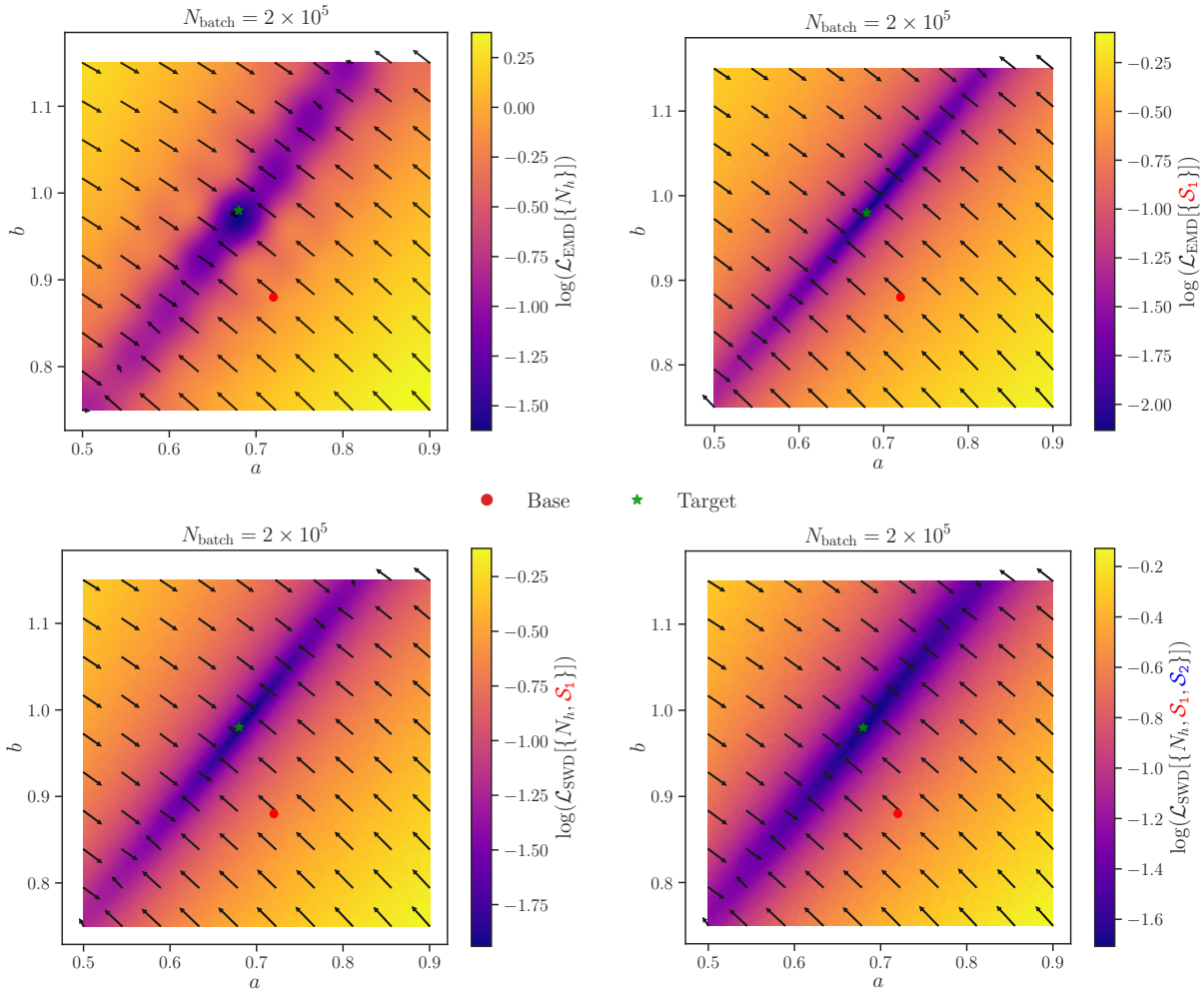

FIG. 4. Two-dimensional Lund a and b parameter plane loss landscape for one-dimensional and sliced Wasserstein distances over various dimensionalities of low and high-level observables, with the same grid and mini-batching as in figure 3.

relative to the initial z-axis where

$$\begin{aligned} p_{T,i} &= \sqrt{p_{x,i}^2 + p_{y,i}^2}, \\ \phi_i &= \tan^{-1}(p_{y,i}/p_{x,i}), \\ \eta_i &= -\log\left[\tan^{-1}(p_z/2p_{T,i})\right]. \end{aligned} \tag{12}$$

The classifier was first trained with labeled data minimizing the binary-cross-entropy loss (BCE) between simulated and experimental data. After training, the score is given by

$$\mathcal{S}(\boldsymbol{\Theta}_h^1, \cdots, \boldsymbol{\Theta}_h^{N_h}) = \rho\left(\sum_i^{N_h} \phi(\boldsymbol{\Theta}_h^i)\right). \tag{13}$$

As an observable, the score $\mathcal{S}$ effectively compresses the full-phase space kinematics $\left\{\boldsymbol{\Theta}_h^1, \cdots, \boldsymbol{\Theta}_h^{N_h}\right\}$ of all N_h hadrons in an event into a single scalar representation with hadron permutation-invariance that can be applied to any dataset irrespective of the actual parameter values considered for generation. The right panel of figure 2 shows the score distributions as well as the reweighted distributions for the two cases in which the classifier is initially trained to distinguish between `Model 1` (right top) or `Model 2` (right bottom) from Target which we will refer to as scores $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. Finally, when fitting, we consider multiplicity and scores individually as one-dimensional distributions as well as together to form two- and three-dimensional distributions, e.g.

$$\boldsymbol{y}_{\text{sim}}^d = \begin{pmatrix} \boldsymbol{y}_1 = C(\{N_{h,1}, \mathcal{S}_{1,1}, \mathcal{S}_{2,1}\}, d) \\ \vdots \\ \boldsymbol{y}_N = C\left(\{N_{h,N}, \mathcal{S}_{1,N}, \mathcal{S}_{2,N}\}, d\right) \end{pmatrix} \tag{14}$$

where $C(\boldsymbol{O}, d)$ implies $\boldsymbol{O}$ choose d, with $d = 1, 2$ or 3 the number of observables considered, and keeping the same choice for all $\{\boldsymbol{y}_{\text{sim}}^d\}_{n=1}^N$.

V. RESULTS

The algorithmic prescription for parameter estimation using RSA described in Sec. III offers an efficient method for analyzing and exploring model parameter space. In addition to the standard binned and unbinned observables used in modern tuning exercises, RSA facilitates the exploration of parameter space using *machine-learning-based observables* (such as the score $\mathcal{S}$). This facilitates a systematic study and comparison of metric spaces utilizing high- versus low-level observables. Here we first validate the algorithm through the construction of loss landscapes for a variety of metrics and then turn to the investigation of fit performance using high- versus low-level observables.

A. Loss landscapes

Given an accept-reject data-sample for an arbitrary parameterization $\boldsymbol{\theta}$, as described in Sec. III B, and a set of observables $\boldsymbol{y}_{\text{exp}}, \boldsymbol{y}_{\text{sim}}$, RSA with reweighting can be used as a powerful exploratory device over the parameter volume. Once a metric comparing $\boldsymbol{y}_{\text{exp}}$ and $\boldsymbol{y}_{\text{sim}}$ is defined, a 'loss landscape' that illustrates the magnitude and gradients of the metric over $\boldsymbol{\theta}$ can be created quickly using reweighting. Below we construct loss landscapes for various losses and observables.

a. Pseudo-χ^2 For binned observables, a commonly used metric to determine fit efficacy is the pseudo-χ^2 [27, 28]. Given a differentiable binning of $\boldsymbol{y}_{\text{sim}}, \boldsymbol{y}_{\text{exp}}$ and weights $\boldsymbol{w}$ the pseudo-χ^2 loss can be expressed as:

$$\mathcal{L}_{\chi^2}(\boldsymbol{y}_{\text{sim}}, \boldsymbol{y}_{\text{exp}}; \boldsymbol{w}) = \sum_{i=1}^{n_{\text{bins}}} \frac{(y_{\text{sim}}^{(i)} - y_{\text{exp}}^{(i)})^2}{\sigma_{\text{sim},i}^2 + \sigma_{\text{exp},i}^2} \tag{15}$$

where $y^{(i)}$ represents the normalized count density in the i^{th} bin and σ denotes the statistical uncertainty of the i^{th} simulated and experimental bin. Note that this loss is well-defined for arbitrary multi-dimensional observable distributions. The landscape of the pseudo-χ^2 for multiplicity N_h distributions over the parameter plane can be seen in figure 3.

b. 1D Wasserstein distance For unbinned observables, we utilize the optimal-transport-based Wasserstein, or Earth mover's distance (EMD). Unlike f-divergences (like the χ^2), the EMD does not require overlapping support. Tuning with the EMD is similar to a WGAN [36], only here, the EMD is computed explicitly and thus there is no trainable 'critic' (making the optimization more stable)[7]. A similar use of the EMD is found in ref. [38], which used this setup to learn surrogate models for likelihood ratios. The EMD loss is defined as

$$\mathcal{L}_{\text{EMD}}(\boldsymbol{y}_{\text{sim}}, \boldsymbol{y}_{\text{exp}}; \boldsymbol{w}) = \sum_{n=1}^{N}\sum_{m=1}^{M} f_{n,m}^* d_{n,m}\,, \tag{16}$$

where the elements of the flow matrix $f_{n,m}^*$ encode the fractional amount of weight to be transferred between event $\boldsymbol{y}_{\text{sim},n}$ and $\boldsymbol{y}_{\text{exp},m}$ and $d_{n,m} = ||\boldsymbol{y}_{\text{sim},n} - \boldsymbol{y}_{\text{exp},m}||_2$ is the distance between these two events. Here, M is the number of events observed in the experimental dataset. The top panel of figure 4 shows the EMD loss landscapes for both the multiplicity N_h (top left) and the score $\mathcal{S}_1$ (top right).

c. Multi-dimensional sliced Wasserstein distance To fit across multiple observables, such as multiplicity and classifier scores, we employ the sliced Wasserstein distance [39, 40]. The observable vector is randomly projected onto a large number N_P of random vectors θ_i over

[7] A GAN-based reweighting and fitting has been explored in ref. [37].

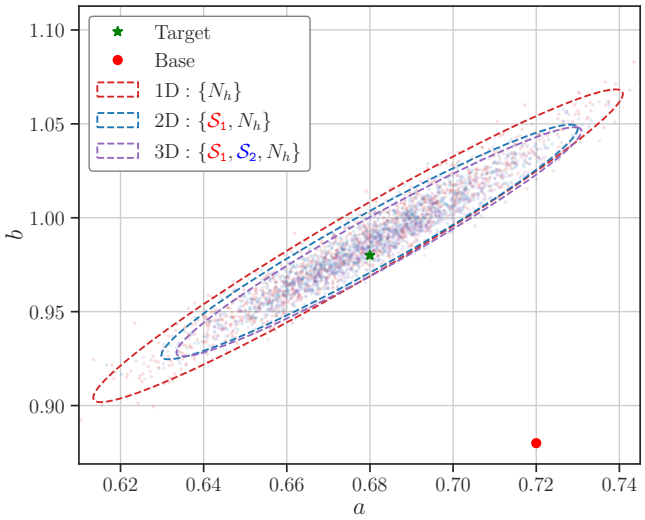

FIG. 5. The 95% confidence ellipses on a, b parameters estimated from 1,000 bootstrapped training runs over 300 epochs and identical tuning hyperparameters. Demonstrates the increasing accuracy of slicing over multiple low and high-level observables, from only N_h (red), $\{\mathcal{S}_1, N_h\}$ (blue), and $\{\mathcal{S}_1, \mathcal{S}_2, N_h\}$ (purple).

the full observable space:

$$\mathcal{L}_{\text{SWD}}(\boldsymbol{y}_{\text{sim}}, \boldsymbol{y}_{\text{exp}}; \boldsymbol{w}) = \frac{1}{N_P} \sum_{i=1}^{N_P} \mathcal{L}_{\text{EMD}}(\boldsymbol{y}_{\text{sim},\theta_i}, \boldsymbol{y}_{\text{exp},\theta_i}; \boldsymbol{w}). \tag{17}$$

We apply slicing for two and three dimensional combinations of both low- and and high-level observables, over one classifier score and multiplicity as well as over two classifier scores based of two (a, b) parameterizations and multiplicity. We chose $N_P = 128$ (with little sensitivity to small changes) and prescaled each observable to have zero mean and unit variance. The bottom panel of figure 4 shows the sliced Wasserstein loss landscapes for both the 2D observable combining multiplicity with the `Model 1` score $\{N_h, \mathcal{S}_1\}$ (bottom left) and the 3D observable combining multiplicity with the `Model 1` and `Model 2` scores $\{N_h, \mathcal{S}_1, \mathcal{S}_2\}$ (bottom right).

Note that in all cases (for all losses and observables), the loss landscapes in figures 3 and 4 reveal a valley in the (a, b) plane illuminating an unavoidable degeneracy in model parameter space and correlation between the a and b parameters. For more details see the discussions in Refs. [27, 28].

B. Confidence Intervals

To demonstrate that the classifier score observables $\mathcal{S}_1$ and $\mathcal{S}_2$ have learned to distinguish data based off of more than the high-level event multiplicity alone, we apply our Wasserstein tuning on the 1D, 2D, and 3D observables mentioned in Sec. IV A and compare their performance. For each observable we perform 1,000 independent tunes, each using a single bootstrapped sample [41] of 25,000 events (both from the target and `Model 1` parameterizations) randomly sampled from the full datasets. Each run is initialized at $\boldsymbol{\theta}_{\text{init}} =$ `Model 1` with the same training hyperparameters and trained over 300 epochs with the same bootstrapped dataset after each parameter update using the `AdamW` optimizer. In figure 5 we show the 95% confidence ellipses over all bootstrapped tunes in the (a, b) plane. The one-dimensional observable tunes utilize the 1D Wasserstein loss while the higher-dimensional observables utilize the sliced Wasserstein loss as described in Sec. V A. We observe that the higher-dimensional observables (1D versus 2D versus 3D) produce a steady narrowing of confidence ellipses, indicating better fit performance.

VI. CONCLUSIONS AND OUTLOOK

In this paper we investigated the implications and applications of pairing rejection sampling with modern autodifferentiation engines. In particular, we focused in the context of model parameter estimation via reweighting and validated the methodology using a hadronization model. We developed an algorithmic prescription for RSA-based parameter estimation and showed that the methodology provides an efficient tool for model parameter space exploration. An important implication of embedding rejection sampling into the computational framework of differential libraries is the possibility of utilizing machine-learning-based observables during tuning. By comparing observables utilizing high-level event information, such as multiplicity, with those utilizing all available event-level information, like the classifier score, we showed that ML-based observables can improve fit performance and convergence.

There are two caveats to RSA-based parameter fits. Firstly, as discussed in Sec. IV, due to the degraded performance of weighted distributions far from the base parameterization, full black-box tuning exercises will likely require multiple base parameterizations to ensure proper coverage over the full parameter space. In practice, however, the total number of base parameterization can be minimized through successive coarse graining of the loss landscapes. Secondly, while not a fundamental issue, the memory footprint of the accept-reject datasets, due to the necessity of zero-padding, can become computationally prohibitive for large oversampling factors or high multiplicity final states (large string energies). This can be mitigated through oversampling factor tuning, seed storage, or the direct computation and in-place storage of gradients during event generation which would completely eliminate the need for external datasets alltogether.

While we performed a two-parameter fit for simplicity, modern differential programming libraries and autodifferentiation engines are commonly used to store and compute numerical gradient information on hundreds to millions of parameters. Because of this, we do not expect that the addition of more tuning parameters will admit any fundamental issues to the numerical algorithm itself (although with more parameters the choice of non-degenerate observables and high-fidelity loss functions becomes extremely non-trivial). In the context of hadronization models, applying the methods developed here in the flavor sector, where there are $\mathcal{O}(10^2)$ of tuneable parameters, may prove to be a good testing ground for the capacity of these methods.

Tangentially, RSA methods may also have applications in numerical uncertainty quantification. As discussed in ref. [20], reweighting can be useful for estimating modeling uncertainties. With RSA, direct access to numerical gradient information can provide a more rigorous characterization of model uncertainties by facilitating the computation of covariance matrices on fit parameters or the construction of maximum likelihood test statistics. We leave these considerations for future work.

RSA presents a versatile and flexible approach to parameter estimation. The methods developed here serve as a promising foundation for the next generation of simulation-based parameter estimation, offering a scalable approach to accurate, data-driven model tuning.

VII. ACKNOWLEDGMENTS

NH and BN are supported by the U.S. Department of Energy (DOE), Office of Science under contract DE-AC02-05CH11231. PI is also supported by NSF grant NSF-PHY-2209769. PI, TM, SM, MS, and AY acknowledge support in part by NSF grants OAC-2103889, OAC-2411215 and OAC-2417682. SM is supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics. TM, MS, and AY are also in part funded by DOE grant DE-SC101977. AS is supported by grant 2019/34/E/ST2/00457 of the National Science Centre, Poland and by the Priority Research Area Digiworld under the program 'Excellence Initiative – Research University' at the Jagiellonian University in Krakow. AY acknowledges support in part by The University of Cincinnati URC Graduate Support Program.

[1] Phil Ilten, Tony Menzo, Ahmed Youssef, and Jure Zupan, "Modeling hadronization using machine learning," SciPost Phys. **14**, 027 (2023), arXiv:2203.04983 [hep-ph].

[2] Aishik Ghosh, Xiangyang Ju, Benjamin Nachman, and Andrzej Siodmok, "Towards a deep learning model for hadronization," Phys. Rev. D **106**, 096020 (2022), arXiv:2203.12660 [hep-ph].

[3] Jay Chan, Xiangyang Ju, Adam Kania, Benjamin Nachman, Vishnu Sangli, and Andrzej Siodmok, "Fitting a deep generative hadronization model," JHEP **09**, 084 (2023), arXiv:2305.17169 [hep-ph].

[4] Christian Bierlich, Phil Ilten, Tony Menzo, Stephen Mrenna, Manuel Szewc, Michael K. Wilkinson, Ahmed Youssef, and Jure Zupan, "Towards a data-driven model of hadronization using normalizing flows," SciPost Phys. **17**, 045 (2024), arXiv:2311.09296 [hep-ph].

[5] Jay Chan, Xiangyang Ju, Adam Kania, Benjamin Nachman, Vishnu Sangli, and Andrzej Siodmok, "Integrating Particle Flavor into Deep Learning Models for Hadronization," (2023), arXiv:2312.08453 [hep-ph].

[6] Christian Bierlich, Phil Ilten, Tony Menzo, Stephen Mrenna, Manuel Szewc, Michael K. Wilkinson, Ahmed Youssef, and Jure Zupan, "Describing Hadronization via Histories and Observables for Monte-Carlo Event Reweighting," (2024), arXiv:2410.06342 [hep-ph].

[7] Stefano Carrazza, Juan Cruz-Martinez, Marco Rossi, and Marco Zaro, "MadFlow: automating Monte Carlo simulation on GPU for particle physics processes," Eur. Phys. J. C **81**, 656 (2021), arXiv:2106.10279 [physics.comp-ph].

[8] Lukas Heinrich and Michael Kagan, "Differentiable Matrix Elements with MadJax," J. Phys. Conf. Ser. **2438**, 012137 (2023), arXiv:2203.00057 [hep-ph].

[9] Theo Heimel, Olivier Mattelaer, Tilman Plehn, and Ramon Winterhalder, "Differentiable MadNIS-Lite," (2024), arXiv:2408.01486 [hep-ph].

[10] Benjamin Nachman and Stefan Prestel, "Morphing parton showers with event derivatives," (2022), arXiv:2208.02274 [hep-ph].

[11] Tommaso Dorigo *et al.* (MODE), "Toward the end-to-end optimization of particle physics instruments with differentiable programming," Rev. Phys. **10**, 100085 (2023), arXiv:2203.13818 [physics.ins-det].

[12] Max Aehle, Mihály Novák, Vassil Vassilev, Nicolas R. Gauger, Lukas Heinrich, Michael Kagan, and David Lange, "Optimization using pathwise algorithmic derivatives of electromagnetic shower simulations," (2024), arXiv:2405.07944 [physics.comp-ph].

[13] Sanha Cheong, Josef C. Frisch, Sean Gasiorowski, Jason M. Hogan, Michael Kagan, Murtaza Safdari, Ariel Schwartzman, and Maxime Vandegar, "Novel light field imaging device with enhanced light collection for cold atom clouds," JINST **17**, P08021 (2022), arXiv:2205.11480 [physics.ins-det].

[14] Giles C. Strong *et al.*, "TomOpt: differential optimisation for task- and constraint-aware design of particle detectors in the context of muon tomography," Mach. Learn. Sci. Tech. **5**, 035002 (2024), arXiv:2309.14027 [physics.ins-det].

[15] Sean Gasiorowski, Yifan Chen, Youssef Nashed, Pierre Granger, Camelia Mironov, Ka Vang Tsang, Daniel Ratner, and Kazuhiro Terao, "Differentiable simulation of a liquid argon time projection chamber," Mach. Learn. Sci. Tech. **5**, 025012 (2024), arXiv:2309.04639 [physics.ins-det].

[16] Michael Kagan and Lukas Heinrich, "Branches of a Tree: Taking Derivatives of Programs with Discrete and Branching Randomness in High Energy Physics," (2023), arXiv:2308.16680 [stat.ML].

[17] James S. Gainer, Joseph Lykken, Konstantin T. Matchev, Stephen Mrenna, and Myeonghun Park, "Exploring Theory Space with Monte Carlo Reweighting," JHEP **10**, 078 (2014), arXiv:1404.7129 [hep-ph].

[18] S. Mrenna and P. Skands, "Automated Parton-Shower Variations in Pythia 8," Phys. Rev. D **94**, 074005 (2016), arXiv:1605.08352 [hep-ph].

[19] Johannes Bellm, Simon Plätzer, Peter Richardson, Andrzej Siódmok, and Stephen Webster, "Reweighting Parton Showers," Phys. Rev. D **94**, 034028 (2016), arXiv:1605.08256 [hep-ph].

[20] Christian Bierlich, Philip Ilten, Tony Menzo, Stephen Mrenna, Manuel Szewc, Michael K. Wilkinson, Ahmed Youssef, and Jure Zupan, "Reweighting Monte Carlo predictions and automated fragmentation variations in Pythia 8," SciPost Phys. **16**, 134 (2024), arXiv:2308.13459 [hep-ph].

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (Curran Associates, Inc., 2019) pp. 8024–8035.

[22] Christian Bierlich *et al.*, "A comprehensive guide to the physics and usage of PYTHIA 8.3," SciPost Phys. Codeb. **2022**, 8 (2022), arXiv:2203.11601 [hep-ph].

[23] Johannes Bellm *et al.*, "Herwig 7.0/Herwig++ 3.0 release note," Eur. Phys. J. C **76**, 196 (2016), arXiv:1512.01178 [hep-ph].

[24] Enrico Bothmann *et al.* (Sherpa), "Event Generation with Sherpa 2.2," SciPost Phys. **7**, 034 (2019), arXiv:1905.09127 [hep-ph].

[25] Peter Z. Skands, "The Perugia Tunes," in *1st International Workshop on Multiple Partonic Interactions at the LHC* (2009) pp. 284–297, arXiv:0905.3418 [hep-ph].

[26] Andy Buckley, Hendrik Hoeth, Heiko Lacker, Holger Schulz, and Jan Eike von Seggern, "Systematic event generator tuning for the LHC," Eur. Phys. J. C **65**, 331–357 (2010), arXiv:0907.2973 [hep-ph].

[27] Peter Skands, Stefano Carrazza, and Juan Rojo, "Tuning PYTHIA 8.1: the Monash 2013 Tune," Eur. Phys. J. C **74**, 3024 (2014), arXiv:1404.5630 [hep-ph].

[28] Philip Ilten, Mike Williams, and Yunjie Yang, "Event generator tuning using Bayesian optimization," JINST **12**, P04028 (2017), arXiv:1610.08328 [physics.data-an].

[29] Mohan Krishnamoorthy, Holger Schulz, Xiangyang Ju, Wenjing Wang, Sven Leyffer, Zachary Marshall, Stephen Mrenna, Juliane Müller, and James B. Kowalkowski, "Apprentice for Event Generator Tuning," EPJ Web

Conf. **251**, 03060 (2021), arXiv:2103.05748 [hep-ex].
[30] Anders Andreassen and Benjamin Nachman, "Neural Networks for Full Phase-space Reweighting and Parameter Tuning," Phys. Rev. D **101**, 091901 (2020), arXiv:1907.08209 [hep-ph].
[31] Marco Lazzarin, Simone Alioli, and Stefano Carrazza, "MCNNTUNES: Tuning Shower Monte Carlo generators with machine learning," Comput. Phys. Commun. **263**, 107908 (2021), arXiv:2010.02213 [physics.comp-ph].
[32] Bo Andersson, G. Gustafson, G. Ingelman, and T. Sjostrand, "Parton Fragmentation and String Dynamics," Phys. Rept. **97**, 31–145 (1983).
[33] Bo Andersson, *The Lund Model*, Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology, Vol. 7 (Cambridge University Press, 2023).
[34] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola, "Deep sets," (2018), arXiv:1703.06114 [cs.LG].
[35] Patrick T. Komiske, Eric M. Metodiev, and Jesse Thaler, "Energy Flow Networks: Deep Sets for Particle Jets," JHEP **01**, 121 (2019), arXiv:1810.05165 [hep-ph].
[36] Martin Arjovsky, Soumith Chintala, and Léon Bottou, "Wasserstein gan," (2017), arXiv:1701.07875 [stat.ML].
[37] Krish Desai, Benjamin Nachman, and Jesse Thaler, "Moment Unfolding," (2024), arXiv:2407.11284 [hep-ph].
[38] Chu-Cheng Pan, Xiang Dong, Yu-Chang Sun, Ao-Yan Cheng, Ao-Bo Wang, Yu-Xuan Hu, and Hao Cai, "Swd-Fold:A Reweighting and Unfolding method based on Optimal Transport Theory," (2024), arXiv:2406.01635 [physics.data-an].
[39] Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot, "Wasserstein barycenter and its application to texture mixing," in *Scale Space and Variational Methods in Computer Vision: Third International Conference, SSVM 2011, Ein-Gedi, Israel, May 29–June 2, 2011, Revised Selected Papers 3* (Springer, 2012) pp. 435–446.
[40] Soheil Kolouri, Kimia Nadjahi, Umut Simsekli, Roland Badeau, and Gustavo Rohde, "Generalized sliced wasserstein distances," Advances in neural information processing systems **32** (2019).
[41] B. Efron, "Bootstrap Methods: Another Look at the Jackknife," The Annals of Statistics **7**, 1 – 26 (1979).