Simulating Hadronization with Machine Learning

Michael K. Wilkinson[1,*] on behalf of the MLHAD collaboration.

[1]Department of Physics, University of Cincinnati, Cincinnati, Ohio 45221, USA

Abstract. Hadronization is an important part of physics modeling in Monte Carlo event generators, where quarks and gluons are bound into physically observable hadrons. Today's generators rely on finely-tuned phenomenological models, such as the Lund string model; while these models have been quite successful overall, there remain phenomenological areas where they do not match data well. A machine-learning-based alternative called MLHAD, intended ultimately to be data-trainable, can simulate hadronization by encoding latent-space vectors, trained to be distributed according to a user-defined distribution using the sliced-Wasserstein distance in the loss function, then decoding them. The multiplicities and cumulative kinematic distributions of pions generated with MLHAD in this way match those generated using PYTHIA 8.
While this architecture has been successful, an alternative using normalizing flows is convenient for generating non-pion hadrons and for taking advantage of reweighting techniques to reduce computing time. Combined with new methods for reweighting the output of phenomenological models, this updated architecture should prove convenient for comparing the output of MLHAD and of empirical models.

1 Introduction

The simulation of particle collisions can be considered in three blocks: (1) the hard process, (2) the parton shower, and (3) hadronization, as shown in figure 1. The hard process is the initial high-energy interaction between partons (quarks, gluons, electrons, etc.); hadronization is the combination of quarks and gluons into hadrons; and the parton shower, also known as "evolution", corrects the hard process with additional quark and gluon emissions. The hard process and parton shower are well-determined by Quantum Chromo-Dynamics (QCD) through perturbative calculations, but hadronization is in a non-perturbative regime and must be simulated through the use of phenomenological models [1].

There are two widely used models used to simulate hadronization in general-purpose event generators: (1) the Lund string model [2] and (2) the cluster model [3], both illustrated in figure 1. In the string model (used by PYTHIA 8 [4]), partons are connected via QCD color strings with linear potential; these strings are then iteratively split, emitting hadrons as long as there is enough energy to do so. In the cluster model (used by HERWIG 7 [5, 6]), partons are pre-confined into proto-clusters, which then split via two-body decay. These splittings are not determined by QCD (in either model), presenting an opportunity for contributions from Machine Learning (ML).

*e-mail: michael.wilkinson@uc.edu

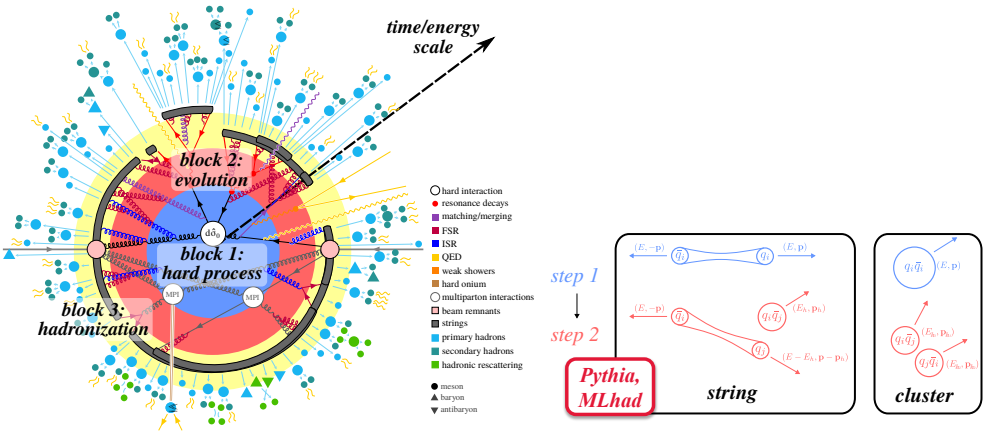

Figure 1. (Left) the three blocks of event generation; the abbreviations refer to Multi-Parton Interaction (MPI), Final-State Radiation (FSR), Initial-State Radiation (ISR), and Quantum Electro-Dynamics (QED); the cross-section of the hard process is $\mathrm{d}\hat{\sigma}_0$. (Right) the two primary hadronization models (used in block 3); in step 1, a quark q_i and an anti-quark $\bar{q}_i$, each with energy E and three-momentum $\mathbf{p}$, are connected either by a linear QCD potential (string model) or by pre-confinement (cluster model); in step 2, this connection is iteratively broken by the creation of a new quark q_j and anti-quark $\bar{q}_j$, which combine with q_i and $\bar{q}_i$ to form one or more hadrons with energies E_h and three-momenta $\mathbf{p}_h$.

MLhad is a novel ML-based approach to simulating hadronization using Neural Networks (NNs). It is currently trained on Pythia 8 simulations and relies on the Lund string model, but ultimately, it should be adaptable to train only on experimental data. Its original architecture, used to simulate pion emissions, is described in section 2. The benefits of reweighting simulated events and the calculation of such weights in hadronization are described in section 3. An updated MLhad architecture, which will be used to simulate non-pion emissions and take advantage of reweighting techniques, is described in section 4.

2 MLhad with cSWAE

2.1 cSWAE Architecture

The original version of MLhad[1] trained a conditional Sliced-Wasserstein Autoencoder (cSWAE) [7–9], as shown in figure 2. In the cSWAE architecture, the encoder NN ϕ takes a vector of observables $\boldsymbol{x}_i$ and a vector of condition labels $\boldsymbol{c}_i$ and returns a latent-space vector $\tilde{\boldsymbol{z}}_i = \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{c}_i)$. The decoder NN ψ takes $\tilde{\boldsymbol{z}}_i$ and $\boldsymbol{c}_i$ and returns a vector of observables $\tilde{\boldsymbol{x}}_i = \boldsymbol{\psi}(\tilde{\boldsymbol{z}}_i, \boldsymbol{c}_i)$. The loss function minimized in training is

$$\mathcal{L}(\psi, \phi) = \mathcal{L}_{\mathrm{SW}} + \mathcal{L}_{\mathrm{rec}} \tag{1}$$

where $\mathcal{L}_{\mathrm{SW}}$ is the sliced-Wasserstein distance and $\mathcal{L}_{\mathrm{rec}}$ is the reconstruction loss [7]. The sliced-Wasserstein distance becomes smaller as the distribution of $\tilde{\boldsymbol{z}}_i$ becomes more similar to a user-defined target latent-space distribution $I(\boldsymbol{z}_i, \boldsymbol{c}_i)$. The reconstruction loss is a measure

[1] A public repository with the code, which depends heavily on the Pythia 8, PyTorch, and scikit-learn libraries, as well as documentation and usage examples, can be found at https://gitlab.com/uchep/mlhad.

of the differences between $\boldsymbol{x}_i$ and $\tilde{\boldsymbol{x}}_i$, defined as

$$\mathcal{L}_{\text{rec}} = \frac{1}{N_{\text{tr}}} \sum_{i=1}^{N_{\text{tr}}} \left[\frac{1}{Q} \sum_{k=1}^{N_e} \left(x_k^{(i)} - \tilde{x}_k^{(i)} \right)^2 + \sum_{k=1}^{N_e} \left| x_k^{(i)} - \tilde{x}_k^{(i)} \right| \right], \tag{2}$$

where N_{tr} is the size of the training set, Q is a user-defined constant, and N_e is the size of the $\boldsymbol{x}_i$ and $\tilde{\boldsymbol{x}}_i$ vectors.

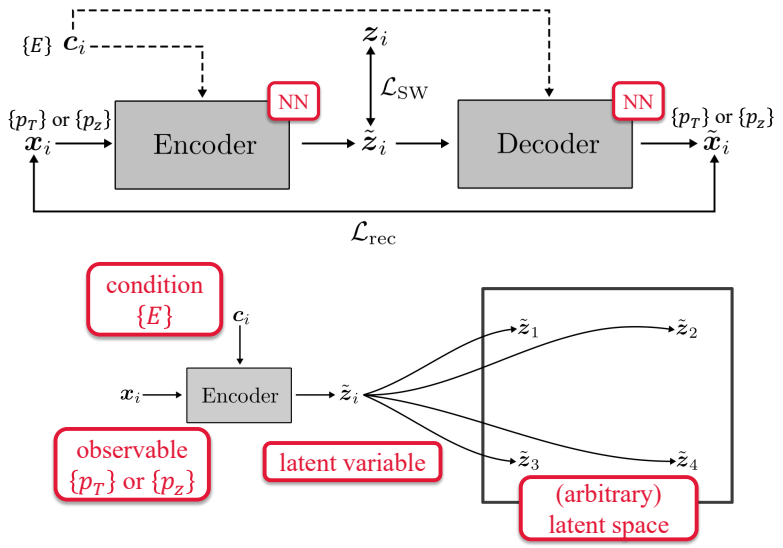

Figure 2. (Top) the cSWAE architecture used by MLHAD, adapted from reference [7]. Two NNs, an encoder and a decoder, are trained to minimize $\mathcal{L}_{\text{SW}} + \mathcal{L}_{\text{rec}}$, the sum of the sliced-Wasserstein distance and the reconstruction loss. The encoder maps an observable $\boldsymbol{x}_i$ (pion transverse momentum p_T or longitudinal momentum p_z) to a latent variable $\boldsymbol{z}_i$ that follows a user-defined distribution. The decoder maps back from the latent space to the observable space. The observables are given a label $\boldsymbol{c}_i$ indicating the energy E of the pion. (Bottom) the action of the encoder in the cSWAE architecture, adapted from reference [7]. The encoder maps the pion p_T or p_z to a location in latent space determined by the pion energy.

The use of the sliced-Wasserstein distance in the loss function allows the user to choose an arbitrary latent space [7]. The region of this space to which $\tilde{\boldsymbol{z}}_i$ is mapped is determined by the value of $\boldsymbol{c}_i$, as shown in figure 2. While $I(\boldsymbol{z}_i, \boldsymbol{c}_i)$ need only be sampleable for the cSWAE architecture to function, its form can impact the performance of the architecture signficantly; in particular, the more similar it is to the distribution of $\boldsymbol{x}_i$, the better [7].

2.2 cSWAE for hadronization

To make use of the cSWAE architecture in MLHAD, the training observables $\boldsymbol{x}_i$ are sets of either the longitudinal momentum p_z or transverse momentum p_T of PYTHIA 8-generated first-pion emissions [7]. While this version of MLHAD treated p_z and p_T as uncorrelated (that is, separately), the same architecture could be used to treat them together, thus taking any correlations into account. The generated emissions used in training are at a fixed initial string energy $E_i = 50$ GeV, used as the condition label $\boldsymbol{c}_i$. The decoder ψ can then be used to generate a range of potential values for p_z or p_T for a given E_i.

The cSWAE decoder is paired with the PYTHIA 8 flavor selector to generate hadrons, where the kinematics are determined by the decoder and the flavor by the flavor selector [7], as shown in figure 3. (Ultimately, the PYTHIA 8 flavor selector could be replaced with an ML-based, data-trained model.) A random value is sampled from $I(\boldsymbol{z}_i, \boldsymbol{c}_i)$ and passed to the decoder along with the energy label $\boldsymbol{c}_i$. A value of p_z or p_T is then randomly selected from the range produced by the decoder and, by imposing the conservation of energy and momentum, used to determine the four-momentum of the emitted hadron and of the remaining string. The flavor of the string is passed to the PYTHIA 8 flavor selector, which determines the flavor of the hadron and of the remaining string. This process is repeated iteratively until the energy of the remaining string falls below 5 GeV.

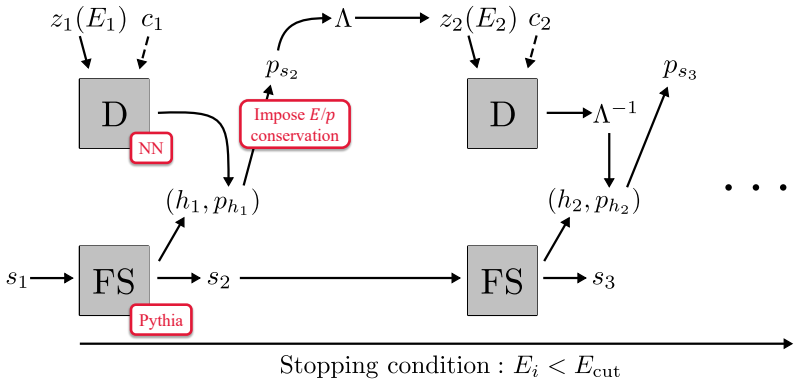

Figure 3. Generating hadrons using the cSWAE decoder and the PYTHIA 8 flavor selector, adapted from reference [7]. The flavor of a string fragment s_i is passed to the PYTHIA 8 flavor selector FS, which determines the flavor of the emitted hadron h_i and therefore also of the remaining string fragment s_{i+1}. Meanwhile, the energy of the string fragment E_i determines the condition label c_i and the distribution from which a random variable z_i is sampled, both of which are passed to the cSWAE decoder D. The decoder generates a range of possible hadron p_z or p_T values, from which one is randomly selected. The imposition of energy and momentum conservation determines the four-momentum of the hadron p_{h_i} and of the remaining string fragment $p_{s_{i+1}}$. The string fragment is boosted to its center-of-mass frame by a Lorentz transformation Λ prior to each emission. The emission process ends when the center-of-mass string energy falls below some pre-determined cutoff value $E_{\text{cut}} = 5$ GeV.

The properties of hadrons generated in this way are found to be distributed similarly to those generated by PYTHIA 8 using the analytic Lund string model; examples are shown in figure 4. The average number of hadron emissions from a single string is the same when using MLHAD as it is when using PYTHIA 8, and, remarkably, the kinematics of emitted pions produced in PYTHIA 8 are faithfully reproduced by MLHAD even at energies on which it was not trained. (The discrepancy at short chain lengths is a consequence of differences in the handling of the first emissions, as described in section 3 of reference [7].) The full validation of MLHAD using this architecture can be found in reference [7].[2]

[2]The computation time required for hadronization using MLHAD vs. using pure PYTHIA 8 is highly model-dependent, and its analysis is beyond the scope both of reference [7] and of these proceedings. It should be noted that, as is common for ML-based computations, the speed of MLHAD generally improves when running on GPUs rather than CPUs.

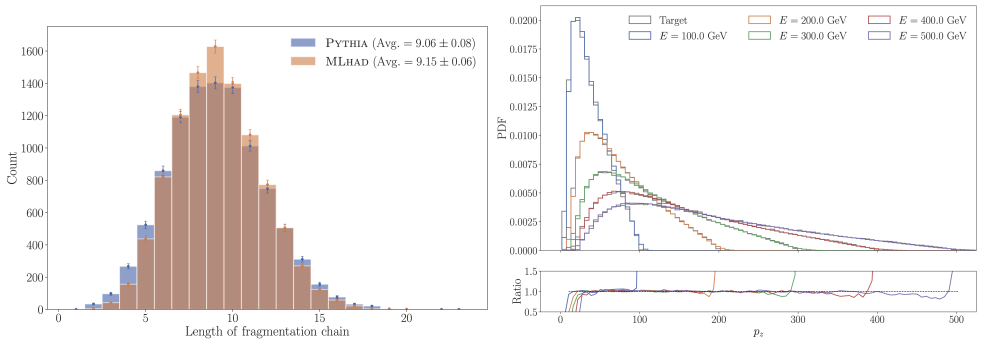

Figure 4. (Left) the number of hadrons produced from a single string (in a sample of 10^4 strings) before falling below the cutoff energy for Pythia 8 (blue) or MLHad (orange). (Right) the p_z distributions of pions generated using MLHad with string energies different from that on which it was trained (50 GeV), compared with those generated using Pythia 8 ("Target"). Taken from reference [7].

3 Reweighting Pythia 8 events

Event generation is time consuming, so it is beneficial to simply calculate per-event weights rather than to regenerate whole datasets, as shown in figure 5. This principle can be applied to the hadronization step in Pythia 8 using a modified accept-reject algorithm, allowing the consequences of varying the parameters of the Lund string model to be explored while generating significantly fewer datasets [10]. An example of the results of this procedure is shown in figure 6.

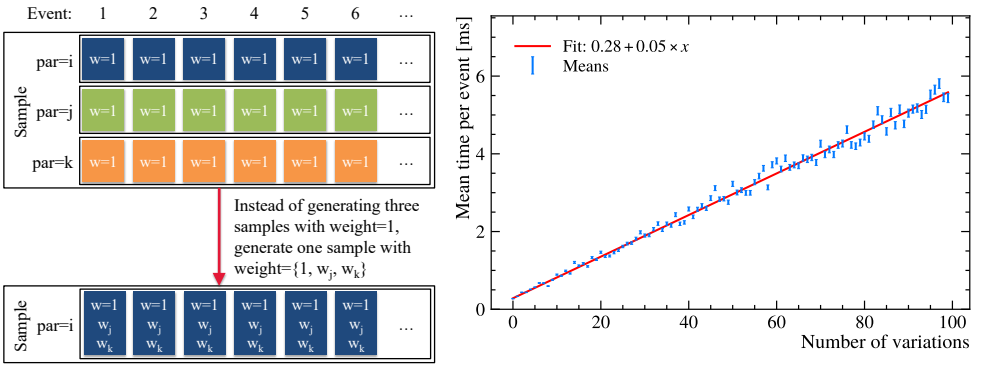

Figure 5. (Left) the difference between regenerating and reweighting. (Right) an example of the average time required to generate a single event as a function of the number of alternative hadronization parameter values calculated during the generation, taken from reference [10].

4 MLHad with NFs

An updated version of MLHad is being prepared using Normalizing Flows (NFs) instead of the cSWAE architecture in order to take advantage of reweighting. As shown in figure 7,

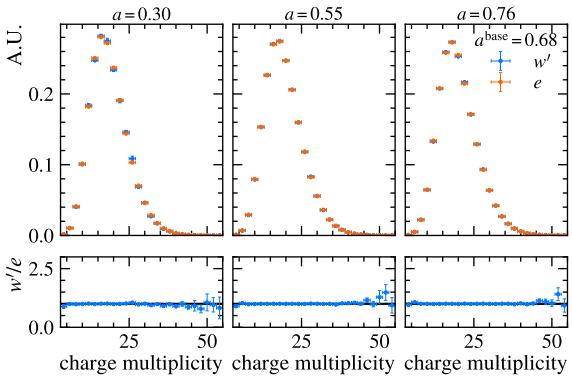

Figure 6. Event charge multiplicity when the Lund parameter parameter a is varied using different methods, taken from reference [10]. Distributions labeled e were generated with the value of the parameter a explicitly set to 0.30, 0.55, and 0.76. Those labeled w' were all taken from a reweighted sample generated with $a = a^{\text{base}} = 0.68$.

NFs work by transforming a probability distribution, typically a Gaussian, into some desired distribution by learning a sequence of differentiable, invertible functions f_j:

$$p_n(z_n, c_i) = p_0(z_0) \prod_{j=1}^{n} \left| \det \left(\frac{\partial f_j(z_{j-1}, c_i)}{\partial z_{j-1}} \right) \right|^{-1}, \tag{3}$$

where z_j is a latent variable, z_n is distrubted according to a target distribution p_n, z_0 is distributed according to a Gaussian distribution p_0, c_i is a condition label, and n is the number of functions f_j (learned by NNs). NFs can therefore learn a log likelihood directly:

$$\log\left(p_n(z_n, c_i)\right) = \log\left(p_0(z_0)\right) - \sum_{j=1}^{n} \log \left| \det \frac{\partial f_j(z_{j-1}, c_i)}{\partial z_{j-1}} \right|. \tag{4}$$

(See, e.g., reference [11] for a derivation of this in the context of ML.) The ratios of likelihoods can then be used to reweight events generated with different NFs.

An NF can be used for hadronization in much the same way as a cSWAE, as shown in figure 8, but since it learns the likelihood directly, its outputs can be easily reweighted, as shown in figure 9, saving computing resources.

5 Conclusions

Machine learning can be used to generate the kinematics of hadrons simulated using the Lund string model. The cSWAE architecture has been shown to be well-suited to this task [7], but there are various computational benefits associated with switching to an architecture based on normalizing flows. Among these are the ability to use reweighting techniques to avoid time-consuming regeneration of events. Coupled with the use of reweighting to compare values of phenomenological hadronization parameters [10], this could allow for the computationally inexpensive comparison of ML-based and phenomenological-based models of hadronization in particle collisions.

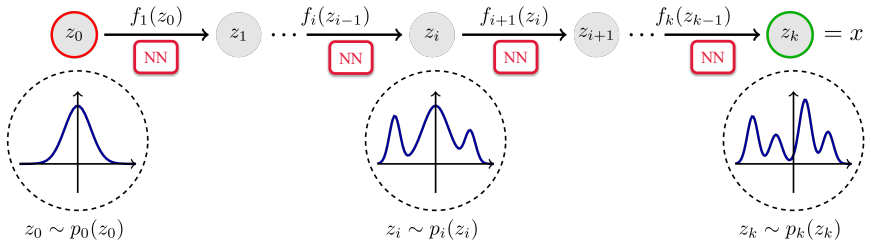

Figure 7. The architecture of NFs, adapted from reference [12]. The flow consists of k NNs that learn functions f_i that transform latent variables z_{i-1} into latent variables z_i, which follow a probability distribution $p_i(z_i)$. The NNs should be trained to maximize the similarity between the distribution of the output variable $p_k(z_k)$ and that of the target variable x.

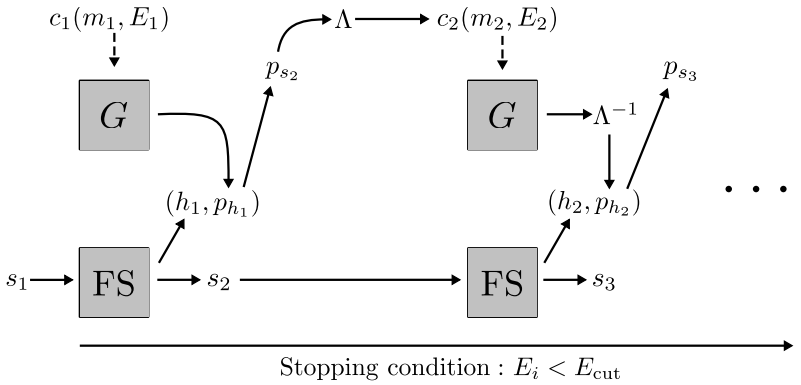

Figure 8. Using an NF as the generator G of hadron kinematics. Compare to figure 3. The condition c_i passed to G specifies the mass of the hadron m_i and the energy of the string E_i.

References

[1] A. Buckley et al., Phys. Rept. **504**, 145 (2011), 1101.2599

[2] B. Andersson, G. Gustafson, G. Ingelman, T. Sjöstrand, Phys. Rept. **97**, 31 (1983)

[3] B. Webber, Nuclear Physics B **238**, 492 (1984)

[4] C. Bierlich, S. Chakraborty, N. Desai, L. Gellersen, I. Helenius, P. Ilten, L. Lönnblad, S. Mrenna, S. Prestel, C.T. Preuss et al., SciPost Phys. Codebases p. 8 (2022), 2203.11601

[5] M. Bahr et al., Eur. Phys. J. C **58**, 639 (2008), 0803.0883

[6] J. Bellm et al., Eur. Phys. J. C **76**, 196 (2016), 1512.01178

[7] P. Ilten, T. Menzo, A. Youssef, J. Zupan, SciPost Phys. **14**, 027 (2023), 2203.04983

[8] S. Kolouri, P.E. Pope, C.E. Martin, G.K. Rohde, *Sliced-wasserstein autoencoder: An embarrassingly simple generative model* (2018), 1804.01947

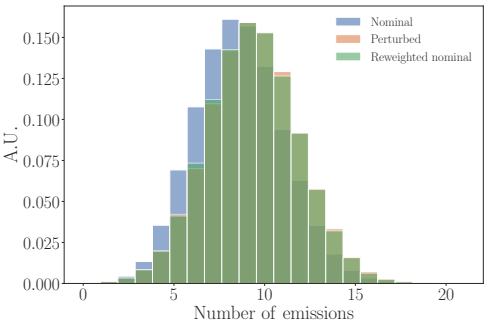

Figure 9. The number of hadrons produced from a single string before falling below the cutoff energy using a preliminary NF-based version of MLHAD. The nominal NF was trained on a PYTHIA 8 simulation with the Lund parameter b set to 0.98. The perturbed NF was trained with $b = 0.80$. The ratio of the likelihoods learned by these two NFs was used to reweight the nominal distribution, causing it to align with the perturbed one.

[9] I. Tolstikhin, O. Bousquet, S. Gelly, B. Schoelkopf, *Wasserstein auto-encoders* (2019), 1711.01558

[10] C. Bierlich, P. Ilten, T. Menzo, S. Mrenna, M. Szewc, M.K. Wilkinson, A. Youssef, J. Zupan (2023), 2308.13459

[11] G. Papamakarios, E. Nalisnick, D.J. Rezende, S. Mohamed, B. Lakshminarayanan, Journal of Machine Learning Research **22**, 1 (2021), 1912.02762

[12] J. Riebesell, K. Sachdeva, J.E. Johnson, F. Rozet, AifuHan, B. Ahn, H. Wang, H. Pourbozorg, H. Shon, J.F. Crenshaw et al., *janosh/awesome-normalizing-flows: v1.0.0* (2023), https://doi.org/10.5281/zenodo.8170087