The Flow of LHC Events

Generative models for LHC simulations and inference

Dissertation

Theo Heimel

Dissertation

submitted to the

Combined Faculty of Natural Sciences and Mathematics
of Heidelberg University, Germany

for the degree of

Doctor of Natural Sciences

Put forward by

Theo Heimel

born in Frankfurt am Main, Germany

Oral examination: 17.04.2024

The Flow of LHC Events

Generative models for LHC simulations and inference

Referees: Prof. Dr. Tilman Plehn
Prof. Dr. Björn Malte Schäfer

Abstract

Generative neural networks have various applications in LHC physics, for both fast simulations and precise inference. We first show that normalizing flows can be used to generate reconstruction-level events with percent-level precision. To estimate their generation uncertainties, we apply Bayesian neural networks. Further, we study the weight distribution from a classifier network which can be used for reweighting, as a performance metric and as a diagnostic tool. Next, we introduce the MadNIS framework for neural importance sampling. It improves classical methods for phase-space integration and sampling using adaptive multi-channel weights and normalizing flows as learnable channel mappings. We show that it leads to significant performance gains for several realistic LHC processes implemented in the MadGraph event generator. Generative networks can also improve analyses by maximizing the amount of extracted information. The matrix element method uses the full kinematic information, making it the tool of choice for small event numbers. It relies on a transfer function to model the shower, detector and acceptance effects. We show how three networks can be used to encode these effects, and for efficient phase-space integration. We use normalizing flows for fast sampling, diffusion models for precise density estimation, and solve jet combinatorics with a transformer.

Zusammenfassung

Generative neuronale Netze haben zahlreiche Anwendungen in der LHC-Physik, sowohl für schnelle Simulationen als auch für präzise Messungen. Wir zeigen zunächst, wie rekonstruierte Events mithilfe von Normalizing Flows mit einer Präzision von einem Prozent generiert werden können. Desweiteren extrahieren wir mit einem Classifier Event-Gewichte, die zum Umgewichten der generierten Events, als Performancemetrik und als diagnostisches Hilfsmittel verwendet werden können. Danach führen wir MadNIS, ein Framework für neuronales Multi-Channel Importance Sampling, ein. Dieses verbessert klassische Phasenraumintegrations- und Samplingmethoden, indem es adaptive Channelgewichte mit Normalizing Flows als lernbare Transformationen kombiniert. Wir wenden unsere Methode mithilfe des MadGraph-Eventgenerators auf realistische LHC-Prozesse an, wo sie zu einer signifikanten Effizienzsteigerung führt. Generative Netzwerke können auch für verbesserte Analysemethoden verwendet werden, die die aus gemessenen Daten extrahierte Informationsmenge maximieren. Die Matrixelementmethode ist solch eine Inferenzmethode, die alle verfügbaren kinematischen Informationen nutzt. Daher ist sie gut für Prozesse mit sehr wenigen gemessenen Events geeignet. Sie benötigt jedoch eine Transferfunktion, die die Effekte von Partonshower, Detektor und Akzeptanz beschreibt. Wir verwenden drei neuronale Netze zum Modellieren dieser Effekte sowie zur effizienten Phasenraumintegration. Wir nutzen Normalizing Flows für schnelles Sampling, Diffusionsmodelle für präzise Extraktion der Phasenraumdichte und lösen die Jet-Kombinatorik mithilfe eines Transformers.

Contents

Preface

The research presented in this thesis was conducted at the Institute for Theoretical Physics at Heidelberg University from April 2021 to November 2023. The contents of Chapters 4 to 6 are based on work in collaboration with other researchers and have been previously published as

[1] Anja Butter, Theo Heimel, Sander Hummerich, Tobias Krebs, Tilman Plehn, Armand Rousselot, Sophia Vent,
Generative networks for precision enthusiasts,
SciPost Phys. 14 (2023) 078, arXiv:2110.13632 [hep-ph];

[2] Anja Butter, Theo Heimel, Till Martini, Sascha Peitzsch, Tilman Plehn,
Two Invertible Networks for the Matrix Element Method,
SciPost Phys. 15 (2023) 094, arXiv:2210.00019 [hep-ph];

[3] Theo Heimel, Ramon Winterhalder, Anja Butter, Joshua Isaacson, Claudius Krause, Fabio Maltoni, Olivier Mattelaer, Tilman Plehn,
MadNIS – Neural Multi-Channel Importance Sampling,
SciPost Phys. 15 (2023) 141, arXiv:2212.06172 [hep-ph];

[4] Ranit Das, Luigi Favaro, Theo Heimel, Claudius Krause, Tilman Plehn, David Shih,
How to Understand Limitations of Generative Networks,
SciPost Phys. 16 (2024) 031, arXiv:2305.16774 [hep-ph];

[5] Theo Heimel, Nathan Huetsch, Ramon Winterhalder, Tilman Plehn, Anja Butter,
Precision-Machine Learning for the Matrix Element Method,
submitted to SciPost, arXiv:2310.07752 [hep-ph];

[6] Theo Heimel, Nathan Huetsch, Fabio Maltoni, Olivier Mattelaer, Tilman Plehn, Ramon Winterhalder,
The MadNIS Reloaded,
submitted to SciPost, arXiv:2311.01548 [hep-ph].

Additionally, the author was involved in the following publications during this period,

[7] Anja Butter (ed.), Tilman Plehn (ed.), Steffen Schumann (ed.) et al.,
Machine learning and LHC event generation,
SciPost Phys. 14 (2023) 079, arXiv:2203.07460 [hep-ph];

[8] Jona Ackerschott, Rahool Kumar Barman, Dorival Gonçalves, Theo Heimel, Tilman Plehn,
Returning CP-Observables to The Frames They Belong,
submitted to SciPost, arXiv:2308.00027 [hep-ph].

Finally, the author is involved in ongoing projects that have not been ready for publication at the time of writing this thesis.

Chapter 1

Introduction

The Standard Model of particle physics (SM) describes the fundamental particles and interactions of nature, except for gravity. It was completed with the discovery of the Higgs boson at the Large Hadron Collider (LHC) in 2012 [9, 10]. Since then, the predictions of the Standard Model have been confirmed at the LHC with higher and higher precision. However, there is clear evidence for physics beyond the Standard Model. Results from astrophysics and cosmology indicate that there is dark matter which cannot be made up of Standard Model particles. Also, the Standard Model does not explain phenomena like the baryon asymmetry and neutrino oscillations. Yet, evidence for physics beyond the Standard Model from collider experiments is still missing. The upcoming high-luminosity LHC upgrade will increase its integrated luminosity by a factor of ten compared to the total from the first three runs [11]. This will provide an unprecedented amount of scattering data that will facilitate measurements with even higher precision, and might lead us to the discovery of physics beyond the Standard Model.

Meanwhile, artificial intelligence is transforming society. The continuous progress in the development of powerful hardware like graphics processing units (GPUs) has made it possible to build more and more complex neural networks, causing a rise in research interest in machine learning (ML) methods. Starting with the area of image recognition [12], machine learning methods have been applied to many different domains. The development of generative ML architectures has proven to be especially impactful. Powerful models for text [13] and image [14] generation have become easily available to end users and are starting to fundamentally change many aspects of our lives. The success of these ML models was facilitated by the availability of vast amounts of training data. This is a parallel to particle physics, where we not only have a large number of measured scattering events, but also have precise simulations based on first principles [15]. Hence, there are many possible applications of ML methods to various aspects of LHC physics.

Applications of ML in collider physics can be broadly split up into two categories. The first type of application uses ML to improve the way the measured data is analyzed. ML makes it possible to use high-dimensional data instead of a low number of hand-crafted summary statistics. Moreover, ML methods are often more computationally efficient. This way, they enhance established analysis techniques, or open up new ways to extract information from data. Examples for this type of ML application include jet tagging [16], anomaly detection [17–22], simulation-based inference [23, 24] and unfolding [25–32]. A better overview can be found in Ref. [33].

Most classical and ML-based data analysis techniques in particle physics rely on precise Monte-Carlo simulations. At the LHC, there is a chain of standard simulation tools [34–40]. Starting from the hard process, the effects of initial state radiation, parton

showering, hadronization, detector interactions and reconstruction are simulated to arrive at reconstruction-level events. The second type of ML application aims to enhance or replace parts of this simulation chain. Event generation needs to be accelerated to prevent it from becoming a computational bottleneck at the high-luminosity LHC [41]. Further, some parts of the simulations are based on phenomenological models instead of first-principle predictions, and ML might be able to improve these parts of the simulations by learning directly from collider data. Examples for ML applied to event generation include phase-space integration and sampling [42–50], amplitude surrogates [51–56], loop integration [57], end-to-end event generation [58–64], hadronization models [65,66], parton shower generation [67–74] and detector simulation [75–98].

There are multiple different generative neural network architectures that have been used in particle physics, including generative adversarial networks (GANs) [99], diffusion models [100–104], generative transformers [105] and normalizing flows [106]. Most of the work in this thesis is based on the latter, especially its invertible neural network (INN) variant [107–110]. INNs implement a bijective mapping between a simple latent distribution and a complex data distribution. This mapping has a tractable Jacobian and is equally fast in both the density estimation and sampling direction. Even though it has been shown that improved precision can be achieved with other generative architectures in event generation [64], INNs provide an excellent balance between fast and stable training, computationally cheap sampling and density estimation, as well as high precision. This makes them a good choice for many applications. In this thesis, we use generative networks both to accelerate event generation and as a tool for precision measurements. Independent of the choice of generative architecture, it is crucial to understand the uncertainties introduced by the use of neural networks. Bayesian neural networks (BNNs) [111–114] can be used to extract the uncertainties from limited training statistics. They can be applied to various network architectures, and are used in several of the ML applications presented in the following chapters.

This thesis is organized as follows: we start with an introduction to LHC event generation in Ch. 2. We discuss the generation of hard-scattering events using importance sampling in detail, and briefly review the other parts of the LHC event generation chain. After a short introduction to deep learning, we discuss the various network architectures used in this thesis in Ch. 3. Next, we present three applications of generative networks in LHC physics. In Ch. 4, we use invertible neural networks to generate reconstruction-level events for the example of leptonically decaying Z bosons associated with a variable number of jets. We show how Bayesian neural networks and classifier weights [115] can be used to control the uncertainties of the generated events. We then move to hard-scattering events in Ch. 5 where the exact likelihood is known from the matrix elements. We present MadNIS, a framework for neural multi-channel importance sampling. It combines channel mappings based on normalizing flows with adaptive channel weights to replace established importance sampling algorithms like Vegas [116–120]. After discussing several methods to improve the training stability and speed, as well as the integration and sampling performance, we benchmark our method for toy examples and realistic LHC processes. The knowledge of the exact likelihood is not only useful for event generation, but also for precision measurements of theory parameters using the matrix element method (MEM) [121, 122]. In Ch. 6, we show how the MEM can be combined with three networks to encode shower, detector and acceptance effects, and for efficient phase-space integration. As an example process, we use associated Higgs and single-top production to measure the CP-violating phase of the top Yukawa coupling. We summarize our results and discuss future research directions in Ch. 7.

Chapter 2

Collider physics and event generation

We describe the particles and interactions that make up our universe using quantum field theory (QFT). Such a theory is formulated in terms of symmetries, a set of particles and their corresponding quantum fields, a set of parameters like masses of particles and coupling strengths, and a Lagrangian that defines the dynamics of these quantum fields. The most successful model is the Standard Model of particle physics (SM) [123–127], which is symmetric under global Poincaré transformations, i.e. rotations, translations and boosts, and under local gauge transformations with the gauge group

$$\mathrm{SU}(3)_C \times \mathrm{SU}(2)_W \times \mathrm{U}(1)_Y \ . \tag{2.1}$$

For a more detailed introduction, see Ref. [128]. The Standard Model has 18 free parameters. For instance, it can be parameterized in terms of

- 10 masses of quarks, charged leptons, the weak gauge bosons and the Higgs boson,
- 4 parameters of the CKM matrix, defining the mixing between the three generations of fermions through the weak interaction and the amount of CP violation,
- 3 coupling constants for the strength of the gauge coupling,
- the vacuum expectation value of the Higgs field.

Based on the fundamental theory we can make predictions, often using perturbation theory. For instance, we can predict the rates and kinematics of scattering processes. The goal of particle physics at colliders is to test the predictions of the Standard Model, provide precise measurements of the parameters of the theory, and – most importantly – look for signs for physics beyond the Standard Model (BSM). All of these things are done by colliding particles at high energies, measuring the particles produced in the collisions, and comparing them to the theory prediction. This approach relies on fast and precise simulations that predict the outcome of an experiment given a fundamental theory.

This chapter covers the basics of LHC event generation and the underlying physics. It starts with a brief discussion of the way from a Lagrangian to a prediction of a differential cross section at a hadron collider like the LHC in Sec. 2.1. This is followed by a description of the Monte Carlo methods necessary to integrate and sample from these differential cross sections in Sec. 2.2. We then outline the subsequent steps needed to simulate events, like parton showers, hadronization, detector effects and reconstruction in Sec. 2.3.

2.1 Collider physics

There are several different types of colliders. Lepton colliders like LEP are based on electron-positron collisions. As electrons are fundamental particles, they provide a very clean environment that is ideal for precision measurements. The main disadvantage of electron colliders is the large energy loss through synchrotron radiation because of the low mass of the electron. This limits the available center of mass energy. Because the proton mass is roughly 2000 times larger than the electron mass, this problem is much smaller for hadron colliders that collide (anti-)protons like Tevatron or the LHC. However, protons are not fundamental particles, so precise measurements at hadron colliders require a detailed understanding of the proton substructure, described by parton distribution functions.

2.1.1 Parton distribution functions

Protons are composed of fundamental particles, most importantly quarks and gluons. These are referred to as partons. In a collision between two protons, interactions take place between partons carrying a part of the total proton momentum. This is described by the parton model. It assumes an infinite momentum frame where the masses and transverse momenta of the partons (quarks and gluons) can be neglected. The probability density to find a given type of parton a in a scattering process involving a proton,

$$f_a(x, Q^2) \,, \tag{2.2}$$

was originally assumed to only depend on the momentum fraction x carried by the parton [129]. The model was later refined to also consider the dependence on the energy scale Q^2 given by the momentum transfer during the scattering. This function is called a parton distribution function (PDF). PDFs are normalized such that the total momentum fraction for the sum of all types of partons is one,

$$\sum_a \int \mathrm{d}x \; x \; f_a(x, Q^2) = 1 \,. \tag{2.3}$$

PDFs cannot be computed using perturbative QCD. While precise first-principles predictions of PDFs using lattice QCD might become possible in the future [130], current methods extract the PDFs by performing fits to measured data. These fits combine measurements from deep inelastic scattering experiments and from hadron colliders to cover a wide range of momentum fractions and energy scales. The choice of functional shape of the PDFs depends on the methodology used in the fit. For instance, the NNPDF collaboration [131] uses neural network for precise PDF fits – one of the earliest successful applications of neural networks in particle physics.

While the functional form of the PDFs cannot be determined perturbatively, the evolution of the PDFs with the momentum scale Q^2 is described by the DGLAP equations [132–134]

$$\frac{\mathrm{d}f_b(x, Q^2)}{\mathrm{d}\log Q^2} = \sum_{a,c} \int_x^1 \frac{\mathrm{d}z}{z} \frac{\alpha_s(Q^2)}{2\pi} f_a(x/z, Q^2) P_{a\to bc}(z) \,. \tag{2.4}$$

The functions $P_{a\to bc}(z)$ are universal splitting kernels describing the splitting of a parton a into partons b and c in the soft and collinear limit. They only depend on the momentum fraction z and on the type of the partons a, b and c (quarks or gluons). They are not

only relevant to describe the proton substructure, but are also used to simulate parton showers, see Sec. 2.3.1.

2.1.2 Differential cross sections

Consider a $2 \to n$ scattering process at a proton-proton collider with a center-of-mass energy $\sqrt{s}$. Neglecting the proton mass, the momenta of the two protons in the lab frame are given by

$$p_A = \left(\frac{\sqrt{s}}{2}, 0, 0, \frac{\sqrt{s}}{2}\right) \quad \text{and} \quad p_B = \left(\frac{\sqrt{s}}{2}, 0, 0, -\frac{\sqrt{s}}{2}\right) . \tag{2.5}$$

The scattering process itself will happen between constituents of these protons, with momenta

$$p_a = \left(x_a \frac{\sqrt{s}}{2}, 0, 0, x_a \frac{\sqrt{s}}{2}\right) \quad \text{and} \quad p_b = \left(x_b \frac{\sqrt{s}}{2}, 0, 0, -x_b \frac{\sqrt{s}}{2}\right) , \tag{2.6}$$

where x_a and x_b are the momentum fractions carried by the partons. The squared center-of-mass energy of the partonic system is then given by

$$\hat{s} = s x_a x_b . \tag{2.7}$$

The first step to find the rate for a process with given initial and final state particles is to compute its matrix element $\mathcal{M}$. To this end, Feynman diagrams are constructed to the desired order in perturbation theory and the Feynman rules are used to compute the matrix element as a function of the initial state momenta (p_a, p_b) and the final state momenta $(p_1, \ldots, p_n)$. The next step is to compute the squared matrix element $\langle|\mathcal{M}|^2\rangle$. The brackets $\langle\,\rangle$ indicate that the squared matrix element is averaged over the the possible spins and colors of the incoming partons (corresponding to an unpolarized beam), and summed over the spins and colors of the outgoing particles. Furthermore, we have to introduce a symmetry factor to avoid double-counting from final states with two or more identical particles which we absorb into the averaging in the following. By combining the proton PDFs, the squared matrix element, the flux factor $1/(2\hat{s})$ (where we again assume massless initial state particles) and the Lorentz-invariant phase-space element $\mathrm{d}\Phi$, we get the full expression for the differential cross section,

$$\mathrm{d}\sigma(p_a, p_b; p_1, \ldots, p_n) = \frac{f_a(x_a) f_b(x_b)}{2 s x_a x_b} \langle|\mathcal{M}(p_a, p_b; p_1, \ldots, p_n)^2|\rangle \mathrm{d}\Phi(p_a, p_b; p_1, \ldots, p_n) . \tag{2.8}$$

The phase-space element enforces the momentum conservation

$$p_a + p_b = p_1 + \ldots + p_n , \tag{2.9}$$

as well as the on-shell conditions of the particles and the positivity of their energies. It is given by

$$\mathrm{d}\Phi(p_a, p_b; p_1, \ldots, p_n) = \prod_{i=1}^{n} \left(\frac{\mathrm{d}^3 \boldsymbol{p}_i}{2\pi^3 E_i}\right) (2\pi)^2 \delta^{(4)} \left(p_a + p_b - \sum_{i=1}^{n} p_i\right) . \tag{2.10}$$

Hence, there are two degrees of freedom of the initial state momenta corresponding to the momentum fractions x_a and x_b, and there are $3n - 4$ degrees of freedom of the final

state momenta. In total, the phase space has $3n - 2$ dimensions.

Integrating the differential cross section over the full phase space yields the total cross section

$$\sigma = \int \mathrm{d}\Phi \, \frac{\mathrm{d}\sigma}{\mathrm{d}\Phi} \, , \tag{2.11}$$

from which we can compute the expected rate at a collider with luminosity L,

$$\frac{\mathrm{d}N}{\mathrm{d}t} = L\sigma \, . \tag{2.12}$$

We can also understand the differential cross section as a probability distribution by normalizing it with the total cross section,

$$p(x) = \frac{1}{\sigma} \frac{\mathrm{d}\sigma}{\mathrm{d}x} \quad \text{with} \quad \int \mathrm{d}x \, p(x) = 1 \, . \tag{2.13}$$

Here we used x instead of Φ to denote the hard-scattering phase space, which will be used as a convention throughout the rest of this thesis. To generate hard-scattering events, we need to sample from this probability distribution. In Sec. 2.2, we will discuss how the integration in Eq. (2.11) and the sampling from Eq. (2.13) is done in practice.

2.1.3 Kinematic observables

The momenta of particles can be written as four-vectors

$$p^\mu = (E, \boldsymbol{p}) = (E, p_x, p_y, p_z) \quad \text{with} \quad g^{\mu\nu} = \mathrm{diag}(1, -1, -1, -1) \, . \tag{2.14}$$

Often it is useful to express the momenta in a way that takes the symmetries of particle collisions and the geometry of typical detectors into account. Differential cross sections are invariant under a global rotation around the beam axis. Therefore, momenta are often parameterized in terms of the azimuthal angle ϕ around the beam axis and the transverse momentum

$$p_T = \sqrt{p_x^2 + p_y^2} \tag{2.15}$$

perpendicular to the beam axis. A second important symmetry is that the matrix element is invariant under Lorentz boosts along the beam axis. This is no longer true for the differential cross section as it also includes parton densities. Both the azimuthal angle ϕ and the transverse momentum p_T are invariant under such boosts. The invariant mass

$$m = \sqrt{E^2 - |\boldsymbol{p}|^2} \tag{2.16}$$

is another example for a Lorentz-invariant quantity. The rapidity

$$y = \frac{1}{2} \log \left(\frac{E + p_z}{E - p_z} \right) \tag{2.17}$$

is Lorentz-boosted by applying a simple additive shift. As the rapidity depends both on the spatial components of the momentum and the energy, it is often replaced with the pseudo-rapidity

$$\eta = \frac{1}{2} \log \left(\frac{|\boldsymbol{p}| + p_z}{|\boldsymbol{p}| - p_z} \right) = \mathrm{arctanh} \left(\frac{p_z}{|\boldsymbol{p}|} \right) = -\log \tan \left(\frac{\theta}{2} \right) \, , \tag{2.18}$$

which does not depend on the energy. The pseudo-rapidity can be expressed as a function of the polar angle

$$\theta = \arccos\left(\frac{p_z}{|\boldsymbol{p}|}\right) . \tag{2.19}$$

For massless particles or in the limit of high energies $E \gg m$, the pseudo-rapidity and rapidity are equal. These properties make (m, p_T, η, ϕ) a convenient parameterization for momenta. The transformation back into four-momenta is given by

$$p_x = p_T \sin\phi , \quad p_y = p_T \cos\phi , \quad p_z = p_T \sinh\eta , \quad E = \sqrt{m^2 + |\boldsymbol{p}|^2} . \tag{2.20}$$

2.2 Generating hard-scattering events

In this section, we will first review the basics of Monte Carlo integration and then show how they can be applied to Monte Carlo event generation, as implemented in event generators like MG5aMC [135], Sherpa [35] or Pythia8 [34]. This short introduction uses the notation conventions and some of the equations from Refs. [3, 6].

Starting from a Lagrangian and the parameters of a model, we can use QFT to predict the matrix elements for processes described by the model. To find the expected rate of such a process at a collider and to determine the kinematics of the events, we have to compute the total cross section defined in Eq. (2.11) and sample from the probability distribution defined by the differential cross section, Eq. (2.13). For most processes, it is infeasible or impossible to solve this integral analytically, and no closed-form expression for sampling from the distribution is known. Instead, both of these tasks can be solved numerically using importance sampling [136].

2.2.1 Importance sampling

Let f be a function of D-dimensional phase space and let I be the integral of that function over phase space,

$$I = \int \mathrm{d}^D x \, f(x) . \tag{2.21}$$

Deterministic numerical integration methods typically compute such integrals by discretizing space and then applying quadrature rules like the trapezoidal rule. While this works well for low-dimensional integrands, it quickly becomes prohibitive for higher-dimensional functions because these methods scale with $\mathcal{O}(n^D)$ with the number of grid nodes n.

This problem can be solved with Monte Carlo integration. We define a normalized proposal distribution $q(x)$ that has to be non-zero over the entire phase space and that we can sample from. We can then rewrite Eq. (2.21) as

$$I = \int \mathrm{d}^D x \, q(x) \frac{f(x)}{q(x)} = \left\langle \frac{f(x)}{q(x)} \right\rangle_{x \sim q(x)} . \tag{2.22}$$

This expectation value can then be approximated as the mean of the reweighted integrand computed for N samples $\{x_i\}$ from the proposal distribution,

$$I \approx I_N \equiv \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{q(x_i)} \quad \text{with} \quad x_i \sim q(x_i) . \tag{2.23}$$

We also refer to the ratio between the integrand and the sampling probability, normalized by the integral, as weights

$$w(x) \equiv \frac{1}{I}\frac{f(x)}{q(x)} \qquad \text{and} \qquad w_i = \frac{1}{I_N}\frac{f(x_i)}{q(x_i)} \,. \tag{2.24}$$

The variance of the integral is given by

$$\Delta_N^2 = \frac{\sigma^2}{N} \,, \tag{2.25}$$

where σ^2 is the variance of the unnormalized weights,

$$\begin{aligned}
\sigma^2 &\equiv \operatorname{Var}\left(\frac{f(x)}{q(x)}\right)_{x\sim q(x)} \\
&= \left\langle \left(\frac{f(x)}{q(x)} - I\right)^2 \right\rangle_{x\sim q(x)} \\
&= \left\langle \frac{f(x)^2}{q(x)^2} \right\rangle_{x\sim q(x)} - \left\langle \frac{f(x)}{q(x)} \right\rangle_{x\sim q(x)}^2 \\
&= \int \mathrm{d}^D x \, \frac{f(x)^2}{q(x)} - \left(\int \mathrm{d}^D x \, f(x)\right)^2 \,.
\end{aligned} \tag{2.26}$$

In practice, we estimate σ^2 as the sample variance from the samples $\{x_i\}$. To make the sample variance an unbiased estimator, we have to introduce the correction factor $N/(N-1)$ (often referred to as Bessel's correction), resulting in

$$\sigma^2 \approx \sigma_N^2 \equiv \frac{N}{N-1}\sum_{i=1}^{N}\left(\frac{f(x_i)}{q(x_i)} - I_N\right)^2 \,. \tag{2.27}$$

Note that the form of the variance in the last two lines of Eq. (2.26) should not be used in practical computations because the two terms are often of very similar size and their cancellation can lead to numerical problems. Instead, the form in Eq. (2.27) should be used.

As seen in Eq. (2.25), we can reduce the error of our integral estimate either by increasing the sampling statistics, resulting in an $\mathcal{O}(\sqrt{N})$ scaling of the error with the number of samples, or by reducing the variance of the integrand, σ^2. We can see from Eq. (2.26) that the variance becomes zero if

$$q_{\text{opt}}(x) = \frac{f(x)}{I(x)} \equiv p(x) \qquad \Longleftrightarrow \qquad w(x) = 1 \,, \tag{2.28}$$

where $p(x)$ is the normalized probability distribution defined by the integrand. This implies that we want to choose our sampling distribution in a way that is as close to p as possible. However, it is not always possible to efficiently draw samples distributed according to some arbitrary function. Instead, we typically can only generate uniformly distributed random numbers and have to define a mapping between the unit hypercube and phase space,

$$x \in \mathbb{R}^D \quad \underset{\leftarrow \overline{G}(y)}{\overset{G(x)\rightarrow}{\longleftrightarrow}} \quad y \in [0,1]^D \,. \tag{2.29}$$

The distribution of samples obtained by mapping uniform random numbers into phase

space is then given by the Jacobian determinant of the mapping,

$$q(x) = g(x) \equiv \left| \frac{\partial G(x)}{\partial x} \right| \quad \text{with} \quad \int \mathrm{d}^D x \, g(x) = 1 \, . \tag{2.30}$$

The expression for the integral from Eq. (2.22) then becomes

$$\begin{aligned} I &= \int\limits_{[0,1]^D} \mathrm{d}^D y \; \left. \frac{f(x)}{g(x)} \right|_{x=\overline{G}(y)} \\ &= \left\langle \left. \frac{f(x)}{g(x)} \right|_{x=\overline{G}(y)} \right\rangle_{y\sim u(y)} = \left\langle \frac{f(x)}{g(x)} \right\rangle_{x\sim g(x)} \, , \end{aligned} \tag{2.31}$$

where $u(y)$ denotes a uniform distribution over the unit hypercube. In the following sections, we not explicitly write down the mapping $\overline{G}$ for the sake of brevity.

2.2.2 Vegas algorithm

In many cases we have some knowledge about the shape of the integrand that can be used to construct mappings. We will discuss some examples in LHC physics in Sec. 2.2.3. However, the exact mapping to reproduce the shape of f is typically not known. This problem can be tackled with algorithms for adaptive importance sampling, the most common one in particle physics being Vegas [116–120].

The challenge of coming up with an adaptive integrator for high-dimensional integrands is to beat the curse of dimensionality, i.e. the method has to be built in a way that does not scale as a power law of the phase-space dimension. The central assumption in Vegas is that the high-dimensional phase-space distribution $p(x)$ can be approximated as a product of one-dimensional distributions,

$$p(x) \approx g(x) = g(x_1, \ldots, x_D) \approx g_1(x_1) \cdot \ldots \cdot g_D(x_D) \, . \tag{2.32}$$

An adaptive mapping is then defined to approximate these one-dimension distributions, resulting in a linear scaling with the number of dimensions. The integration domain of the Vegas algorithm is the unit hypercube, so in this section we will assume $x \in [0,1]^D$. To use the algorithm for other integration domains like phase space, it has to be combined with an analytic mapping. This mapping can also be used to encode prior knowledge about the integrand.

For each individual dimension, the integration interval $[0,1]$ is divided into K bins with boundaries $W_0, \ldots, W_K$ that satisfy

$$W_0 = 0 \, , \quad W_K = 1 \, , \quad \text{and} \quad W_k < W_{k+1} \, . \tag{2.33}$$

The widths of these bins are

$$w_k = W_k - W_{k-1} \quad \text{such that} \quad \sum_{k=1}^{K} w_k = 1 \, . \tag{2.34}$$

The probability density is then defined as a piecewise-constant function with

$$g(x) = \frac{1}{K w_k} \quad \text{for} \quad x \in [W_{k-1}, W_k] \, . \tag{2.35}$$

Consequently, all bins have the same total probability

$$P_k = \int_{W_{k-1}}^{W_k} \mathrm{d}x\, g(x) = \frac{1}{K}\,, \tag{2.36}$$

and the integral over the full integration domain is normalized,

$$\int_0^1 \mathrm{d}x\, g(x) = \sum_{k=1}^{K} P_k = 1\,. \tag{2.37}$$

The corresponding mapping G is a piecewise-linear function, or linear spline,

$$G(x) = \frac{1}{K}\left(k - 1 + \frac{x - W_{k-1}}{w_k}\right) \quad \text{for} \quad x \in [W_{k-1}, W_k]\,, \tag{2.38}$$

with $g(x) = \partial G(x)/\partial x$. Inserting Eq. (2.35) into the formula for the integral variance, Eq. (2.26), yields

$$\sigma^2 = \sum_{k=1}^{K} \left(K w_k \int_{W_{k-1}}^{W_k} \mathrm{d}x\, f(x)^2 \right) - \left(\int_0^1 \mathrm{d}x\, f(x) \right)^2 . \tag{2.39}$$

This is minimized if [120]

$$\frac{(K w_k)^2}{w_k} \int_{W_{k-1}}^{W_k} \mathrm{d}x\, f(x)^2 = C \quad \text{for} \quad k = 1, \ldots, K \tag{2.40}$$

for some constant C that is the same for all bins. The optimization of the VEGAS grid is done iteratively by sampling points $x_k^{(i)}$ for $i = 1, \ldots, N_k$ and for each bin k to estimate the training objective from Eq. (2.40),

$$d_k \equiv \frac{(K w_k)^2}{w_k} \frac{w_k}{N_k} \sum_{i=1}^{N_k} f\left(x_k^{(i)}\right)^2 . \tag{2.41}$$

The bin boundaries are then adapted such that all d_k are as close to the same value as possible. In practice, a smoothing kernel and a damping function with the damping parameter α are applied to d_k first for a more stable convergence [120]. Then, new samples are drawn and the steps are repeated.

While VEGAS is extremely efficient in finding mappings for distributions that factorize, it is not able to model correlations between the integral dimensions. A simple example for such a case is a two-dimensional Gaussian mixture model (GMM) with two modes that are located along the diagonal. In this case, each marginal distribution is a GMM with two modes and their product is a GMM with four modes. Only two of those modes are found in the distribution while the other two modes will be detrimental to the convergence of the integral. Furthermore, it is hard for VEGAS to map out sharp peaks in the integrand due to its limited resolution. Consequently, simple adaptive MC integration algorithms like VEGAS are typically combined with analytic mappings that align the structures of the integrand with the integration axes and make use about prior knowledge about narrow structures in the integrand.

2.2.3 Phase-space mappings

Phase-space mappings fulfill two important tasks in phase-space integration. Firstly, they provide a mapping from the unit hypercube, where it is simple to define adaptive integration algorithms, to phase space, where they enforce momentum conservation and phase-space boundaries like the largest available center-of-mass energy. Secondly, they are useful to encode physics knowledge into the sampling and therefore make it more efficient. For instance, for processes with an s-channel resonance, the position and shape of the resonance is known. A phase-space mapping can then be constructed to map out this resonance.

For a single Feynman diagram of a given process, phase-space mappings can be constructed automatically. There are several ways how this construction can be done in practice. The common idea behind most of these approaches is to factorize the full phase space into multiple mappings according to the structure of the Feynman diagram. Tree-level diagrams are decomposed into $1 \to 2$-decays, time-like invariants s, and $2 \to 2$ or $2 \to n$ scattering processes with t-channel propagators [36, 50, 137].

A propagator with momentum transfer q appears in the squared matrix element as

$$|\mathcal{M}|^2 \propto \frac{1}{(q^2 - M^2)^2 + M^2\Gamma^2} \tag{2.42}$$

where M is the mass and Γ is the decay width of the propagating particle. A mapping from uniform random numbers z to q^2 following this Breit-Wigner distribution is given by [137]

$$\begin{aligned} q^2 = \overline{G}(z) &= M\Gamma \tan(y_1 + (y_2 - y_1)z) + M^2 \quad \text{with} \\ y_{1,2} &= \arctan\left(\frac{q^2_{\text{min,max}} - M^2}{M\Gamma}\right) \end{aligned} \tag{2.43}$$

and the corresponding Jacobian is

$$g(q^2) = \frac{M\Gamma}{(y_2 - y_1)((q^2 - M^2)^2 + M^2\Gamma^2)} \,. \tag{2.44}$$

The above mapping is only defined for massive particles. The momentum transfer for a massless propagator can be mapped out with

$$\begin{aligned} q^2 = \overline{G}(z) &= \left[z(q^2_{\text{max}})^{1-\nu} + (1-r)(q^2_{\text{min}})^{1-\nu}\right]^{\frac{1}{1-\nu}} \\ g(q^2) &= \frac{1-\nu}{(q^2)^\nu \left[(q^2_{\text{max}})(1-\nu) - (q^2_{\text{min}})^{1-\nu}\right]} \,, \end{aligned} \tag{2.45}$$

with a parameter $\nu \neq 1$, where $\nu = 2$ recovers the $1/q^2$ scaling of the propagator. Still, $\nu \lesssim 1$ is often a good choice in practice due to partial cancellation of the propagator poles in the collinear limit [137].

A $1 \to 2$-decay is best described in the rest frame of the decaying particle, where the only two degrees of freedom parameterize the angular direction of the decay. It can be parameterized in terms of the azimuthal angle ϕ and polar angle θ. A uniform distribution of these angles on a sphere can be achieved by sampling $\phi \in [-\pi, \pi]$ and $\cos\theta \in [-1, 1]$ uniformly. Similar to the decay, a $2 \to 2$-scattering with given incoming momenta also has 2 degrees of freedom in its center of mass frame. In the case of a t-channel propagator,

they can be parameterized in terms of the uniformly sampled azimuthal angle ϕ and the momentum transfer $q^2 = -t$. A phase-space mapping for the latter can be constructed using Eq. (2.43) or Eq. (2.45). Finally, it is also possible to perform flat sampling of the n-particle phase space. RAMBO on diet [138] is an algorithm to construct such a mapping that is flat for massless particles in the final state, and approximately flat for massive particles.

2.2.4 Multi-channel integration

For simple integrands, the combination of analytic mappings encoding knowledge about the structure of the integrand and an adaptive importance sampling algorithm like VEGAS is sufficient to get a good convergence of the integral. However, it is not always possible to find a mapping that captures the full structure of the integrand. It is often easier to find mappings that encode a part of the structure. As described in Sec. 2.2.3, we can systematically build phase-space mappings that reflect the squared matrix element of a single Feynman diagram. Since the full process is typically described by more than one diagram and there can be interference effects between the diagrams, a single mapping is not sufficient for efficient importance sampling.

This problem can be addressed with multi-channel integration, where the integrand is split up into multiple channels [139–141]. Following the multi-channel integration setup used in MG5AMC [142], we define channel weights $\alpha_i(x)$ as functions over phase space with the channel index $i = 1, \ldots, n_c$. They satisfy the normalization condition

$$\sum_{i=1}^{n_c} \alpha_i(x) = 1 \,. \tag{2.46}$$

We can use this partition of unity to rewrite the integral as

$$I = \sum_{i=1}^{n_c} \alpha_i(x)\, I = \sum_{i=1}^{n_c} \int \mathrm{d}^D x\, \alpha_i(x) f(x) \equiv \sum_{i=1}^{n_c} I_i \,. \tag{2.47}$$

In analogy to Eqs. (2.29) to (2.31), we introduce mappings

$$x \in \mathbb{R}^D \quad \underset{\leftarrow \overline{G}_i(y)}{\overset{G_i(x) \rightarrow}{\longleftrightarrow}} \quad y \in [0,1]^D \tag{2.48}$$

with Jacobians

$$g_i(x) = \left| \frac{\partial G_i(x)}{\partial x} \right| \qquad \text{with} \qquad \int \mathrm{d}^D x\, g_i(x) = 1 \tag{2.49}$$

to write the full integral as

$$\begin{aligned} I &= \sum_{i=1}^{n_c} \int_{[0,1]^D} \mathrm{d}^D y \; \frac{\alpha_i(x) f(x)}{g_i(x)} \bigg|_{x = \overline{G}_i(y)} \\ &= \sum_{i=1}^{n_c} \left\langle \frac{\alpha_i(x) f(x)}{g_i(x)} \right\rangle_{x \sim g_i(x)} \,. \end{aligned} \tag{2.50}$$

The variance for a single channel with N_i samples can be computed using Eq. (2.26),

$$\Delta^2_{i,N_i} = \frac{1}{N_i}\sigma_i^2 = \frac{1}{N_i}\left[\left\langle \frac{\alpha_i(x)^2 f(x)^2}{g_i(x)^2}\right\rangle_{x\sim g_i(x)} - \left\langle \frac{\alpha_i(x) f(x)}{g_i(x)}\right\rangle^2_{x\sim g_i(x)}\right] . \tag{2.51}$$

For a single channel i with channel weights $\alpha_i(x)$, the variance becomes zero if

$$g_{i,\text{opt}}(x) = \frac{\alpha_i(x) f(x)}{I_i} . \tag{2.52}$$

The variance of the combined integral is given by

$$\Delta^2_N = \sum_{i=1}^{n_c} \Delta^2_{i,N_i} = \sum_{i=1}^{n_c} \frac{\sigma_i^2}{N_i} \qquad \text{with} \qquad N = \sum_{i=1}^{n_c} N_i . \tag{2.53}$$

The combined variance Δ^2_N for a given total number of samples N can be minimized through the choice of the N_i. We can find the optimal N_i using the method of Lagrangian multipliers with the Lagrange function

$$\mathcal{L} = \sum_{i=1}^{n_c} \frac{\sigma_i^2}{N_i} - \lambda\left(N - \sum_{i=1}^{n_c} N_i\right) \tag{2.54}$$

We then minimize $\mathcal{L}$ by setting its derivative to zero and recover the result known from stratified sampling [136],

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial N_i} &= -\frac{\sigma_i^2}{N_i^2} + \lambda \overset{!}{=} 0 \\ \Longrightarrow \quad N_i &= \lambda^{-1/2}\,\sigma_i \\ \Longrightarrow \quad N_i &= N\frac{\sigma_i}{\sum_{i=1}^{n_c}\sigma_i} . \end{aligned} \tag{2.55}$$

Multi-channel integration in MG5aMC

The construction of channels in MG5aMC is based on the single diagram enhancement method [142], where a channel is constructed for each Feynman diagram of the process. The channel weight is then defined as the matrix element squared for this Feynman diagram, normalized by the sum of all matrix elements squared,

$$\alpha_i^{\text{MG}}(x) = \frac{|\mathcal{M}_i(x)|^2}{\sum_j |\mathcal{M}_j(x)|^2} . \tag{2.56}$$

This channel decomposition works best if the interference between diagrams is small,

$$|\mathcal{M}(x)|^2 = \sum_i |\mathcal{M}_i(x)|^2 + \sum_{i\neq j} \mathcal{M}_i(x)\overline{\mathcal{M}_j(x)} \approx \sum_i |\mathcal{M}_i(x)|^2 . \tag{2.57}$$

At leading order, differential cross sections are given by a function $f_{\text{PS}}(x)$ of phase space containing the flux factor and parton distribution functions, and the squared matrix element,

$$\frac{\mathrm{d}\sigma}{\mathrm{d}x} = f_{\text{PS}}(x)|\mathcal{M}(x)|^2 . \tag{2.58}$$

Under the assumption of vanishing interference terms, the integrands for the individual channels are given by

$$\alpha_i(x)\frac{\mathrm{d}\sigma}{\mathrm{d}x} = f_{\mathrm{PS}}(x)\frac{|\mathcal{M}_i(x)|^2}{\sum_j |\mathcal{M}_j(x)|^2}|\mathcal{M}(x)|^2 \approx f_{\mathrm{PS}}(x)|\mathcal{M}_i(x)|^2 \,. \tag{2.59}$$

That means that the phase-space mappings only have to model the dynamics of one Feynman diagram, which makes it possible to construct phase-space mappings from the propagator structure of the diagram as described above.

Neglecting the interference terms is not always a good assumption and can be detrimental to the convergence of the phase-space integration. There is a second method to construct the channel weights in MG5aMC which is based on the propagators of the Feynman diagrams instead of their squared matrix elements [143]. There, the channels weights are defined as

$$\alpha_i^{\mathrm{MG}}(x) = \frac{P_i(x)}{\sum_j P_j(x)} \quad \text{with} \quad P_i(x) = \prod_{k\in\mathrm{prop}} \frac{1}{|q_k(x)^2 - m_k^2 - \mathrm{i}m_k\Gamma_k|^2} \,, \tag{2.60}$$

where the product encompasses all the propagators in a given Feynman diagram, with momenta $q_k(x)$, masses m_k and widths Γ_k. MG5aMC chooses between the two methods in a process-dependent way.

Standard multi-channel integration

The standard method to define multi-channel importance sampling [136, 144], as implemented in Sherpa, is not based on a physics-informed partition of unity like the single-diagram enhancement method. Instead, it defines a total probability from a weighted combination from the channel mappings,

$$g(x) = \sum_{i=1}^{n_c} \beta_i g_i(x) \quad \text{with} \quad \sum_{i=1}^{n_c} \beta_i = 1 \quad \text{and} \quad 0 \leq \beta_i \leq 1 \,, \tag{2.61}$$

with phase-space independent channel weights β_i. The integral is then written as

$$I = \sum_{i=1}^{n_c} \beta_i \left\langle \frac{f(x)}{g(x)} \right\rangle_{x\sim g_i(x)} \,. \tag{2.62}$$

The weights β_i are optimized during the integration to minimize the total variance. Alternatively, we can express this channel decomposition in terms of phase-space dependent channel weights

$$\alpha_i(x) = \beta_i \frac{g_i(x)}{g(x)} \,. \tag{2.63}$$

The disadvantage of this method is that to recover $g(x)$ it is necessary to evaluate all the channel mappings $g_i(x)$ for every sample x. This makes it more expensive compared to the channel decomposition from Eqs. (2.56) and (2.60) where it is sufficient to evaluate $g_i(x)$ for the single channel used to generate the sample. Further, it is less flexible compared to a method with arbitrary $\alpha_i(x)$ as the functional form of the channel partitioning is fixed by the choice of mappings.

2.2.5 Generating unweighted events

So far, we have only discussed Monte Carlo integration as a tool to compute the total cross section for a process of interest. However, we are often interested in event generation, i.e. randomly drawing momenta x from the probability distribution $p(x)$ defined by the differential cross section, see Eq. (2.13). As Monte Carlo integration aims to find mappings that approximate this probability distribution, integration and event generation are closely related problems.

The mapping found during integration is usually not perfect. Hence the generated events follow some sample distribution $x \sim q(x)$ that is related to the truth distribution through known weights $w(x)$, as defined in Eq. (2.24). For samples x_i, we have

$$p(x_i) = w_i \, q(x_i) \; . \tag{2.64}$$

In some cases, it is sufficient to work with weighted events. In that case, the weights w_i have to be taken into account in downstream tasks like making histograms of kinematic observables. However, typically we are interested in events that follow the distribution $p(x)$ exactly in the limit of infinite statistics. This is because there are subsequent, computationally expensive simulation steps like the simulation of detector effects. In cases with a wide distribution of the weights w_i we would have to perform these simulations for a large number of events with low weights, which would make event generation very costly.

Events that exactly follow the distribution $p(x)$ are referred to as unweighted events. They are obtained from a larger set of weighted events through unweighting, which is done by rejection sampling (also known as the hit-or-miss method). We start with the assumption that there is a $w_{\max}$ such that

$$p(x) \leq w_{\max} \, q(x) \tag{2.65}$$

Then samples can be drawn from $p(x)$ with the following algorithm:

1. Draw a sample $x_i \sim q(x_i)$ and a uniform random number $r \in [0, 1]$.
2. Compute the weight of the sample, $w_i = p(x_i)/q(x_i)$.
3. Accept the sample if $r < w_i/w_{\max}$.
4. Repeat until the required number of samples have been accepted.

In practice, $w_{\max}$ is not known analytically in MC event generation and has to be estimated from samples. It is often dominated by a few events with very high weights, resulting in low acceptance rates. Also, determining $w_{\max}$ from a large number of samples requires storing all of them until events are rejected, leading to a high memory usage. These problems can be addressed by replacing the rejection sampling algorithm with partial unweighting, where it is allowed to have some events with $w > w_{\max}$. The algorithm above is modified by assigning accepted events the weight

$$\tilde{w}_i = \max\left(\frac{w_i}{w_{\max}}, 1\right) \; . \tag{2.66}$$

Then there will be some events with overweight $\tilde{w}_i > 1$ that are kept as weighted events.

The acceptance rate of events during unweighting, also called unweighting efficiency ϵ, is given by

$$\epsilon = \frac{\langle w_i \rangle}{w_{\max}} \,. \tag{2.67}$$

It is useful as a quality metric for the mapping constructed by the importance sampling algorithm as it is related to the average time to generate an unweighted event. Improving the mapping will lead to a narrower weight distribution, and therefore a larger unweighting efficiency, resulting in less time to generate a fixed number of unweighted events. Note that this is not an exactly proportional relation. For example, there may be events rejected because of phase-space cuts. Then the matrix element does not have to be evaluated, making it cheap to reject these events.

For practical use as a performance metric, the unweighting efficiency has to be defined in a way that is robust to outliers in the weight distribution. In Ref. [47], the authors propose to use a bootstrapping procedure, where from a set of $n \cdot N$ events with weights w_i, m replicas with N events each are resampled. For each replica r, the maximum weight $w^r_{\max}$ is computed. Next, the value of $w_{\max}$ is defined as the median over the $w^r_{\max}$ of the replicas. The unweighting efficiency is then computed using Eq. (2.67), averaging over all replicas and events. N is chosen based of the number of samples during the last optimization step. m and n have to be chosen sufficiently large to ensure a stable estimate of the unweighting efficiency.

2.3 Event generation toolchain

So far, we have discussed the sampling of hard-scattering events in detail. These are however not directly observed in experiments. First, we have to consider QCD effects where quarks and gluons will undergo soft and collinear splittings, as well as photon radiation. This applies both to partons in the initial and final state, leading to initial state radiation (ISR) and final state radiation (FSR). As a consequence, we see collimated showers of particles, called jets, instead of observing the partons produced in the hard process. The soft and collinear splittings are simulated using parton shower algorithms. Furthermore, we do not observe quarks and gluons as free particles because of confinement [145]. Instead, they form hadrons which are potentially unstable and decay further. This process is described by hadronization models. Moreover, there can be interactions between more than one parton of the incoming protons in a hadron collider, leading to multi-parton interaction (MPI). There can also be collisions between more than one proton, as bunches of multiple protons are collided at the same time. This effect is referred to as pileup. In the end, events observed in colliders typically consist of a large number of hadrons, leptons and photons (and unobserved neutrinos). These particles interact with the material of the detector, and their type and momenta have to be reconstructed from the raw detector data. All of these effects have to be considered for accurate simulations of LHC data. This means a whole chain of simulation tools is needed, with dedicated simulators for the various steps outlined above. In the following, we will discuss parts of this chain in more detail.

2.3.1 Parton shower

Computing matrix elements and sampling events for very large numbers of final state particles quickly becomes prohibitive because the number of Feynman diagrams explodes.

Most of these particles are the result of soft or collinear QCD radiation, which can be treated separately with parton shower algorithms. They are based on the fact that in the soft and collinear limit, the $(n+1)$-particle cross-section factorizes into the n-particle cross section and a universal splitting kernel [132]. Consider a parton a with momentum p_a, splitting up into b and c with energy fractions z and $1-z$, respectively. The cross-section can then be factorized as

$$\sigma_{n+1} \approx \sigma_n \int \mathrm{d}z \frac{\mathrm{d}p^2}{p^2} \frac{\alpha_s}{2\pi} P_{a\to bc}(z) \; . \tag{2.68}$$

The form of the splitting kernels is only dependent on whether the particles a, b and c are quarks or gluons, so there are three kernels $P_{q\to qg}(z)$, $P_{g\to gg}(z)$ and $P_{g\to q\bar{q}}(z)$. These splitting kernels are the same as in the DGLAP equations, see Eq. (2.4). For final state radiation, the differential probability for a splitting with momentum fraction z to occur can be derived from Eq. (2.68),

$$\mathrm{d}\mathcal{P}_a(z, Q^2) = \frac{\mathrm{d}Q^2}{Q^2} \frac{\alpha_s(Q^2)}{2\pi} \sum_{b,c} P_{a\to bc}(z)\mathrm{d}z \; . \tag{2.69}$$

There are multiple choices for the evolution scale Q^2, such as mass or transverse momentum [146]. Eq. (2.69) is used to generate an ordered sequence of splittings that starts from the scale of the hard process and ends at the scale where QCD becomes non-perturbative. A similar equation can be derived for initial state radiation, where the parton distribution functions have to be taken into account,

$$\mathrm{d}\mathcal{P}_b(x, Q^2) = \frac{\mathrm{d}Q^2}{Q^2} \frac{\alpha_s(Q^2)}{2\pi} \sum_a \int \frac{\mathrm{d}z}{z} \frac{f_a(x/z, Q^2)}{f_b(x, Q^2)} P_{a\to bc}(z) \; . \tag{2.70}$$

The differential probabilities from Eqs. (2.69) and (2.70) can be integrated to get the probability that no splitting occurs between two scales Q_1^2 and Q_2^2, known as the Sudakov form factor. This can be turned into a Monte Carlo simulation through the Sudakov veto algorithm, which iteratively samples the scale Q for the next splitting until a cutoff scale is reached [146]. As one-to-two splittings between massless particles are not possible without violating four-momentum conservation, the change in momentum has to be compensated by recoiling against other constituents of the shower [147]. Parton shower algorithms differ in their choice of evolution scale and recoil scheme as well as their potential treatment of higher-order effects [146, 148]. Different implementations of parton shower algorithms are included in Pythia8 [146], Sherpa [35] and Herwig [37].

2.3.2 Hadronization

Simulations of the hard process and parton showers are derived from first principles using perturbation theory. However, we do not have a good theoretical understanding of non-perturbative QCD. Therefore the hadronization of the partons below the cutoff scale of the parton shower can be only simulated using phenomenological models that are fitted to measured data.

The Lund string model [149, 150] is based on the assumption that the field of the strong force between two colored partons is a narrow flux tube, or string, resulting in a linear increase of the potential energy of the system as a function of the distance with approximately 1 GeV/fm. As particles move apart, more and more of their kinetic energy

is transferred to the string. At some point, the string snaps and a new $q\bar{q}$ pair is created. Once a cutoff scale is reached, each pair of quarks connected by a string forms a meson. In this basic form, no baryons are formed during hadronization. This problem can be solved by allowing strings to also break into pairs of diquarks [151, 152]. As a result, baryons can then be formed by strings that connect three quarks. PYTHIA8 [146] as well as recent versions of HERWIG [37, 153] are examples for event generators that implement the Lund string model.

Alternatively, SHERPA [35] and HERWIG implement the cluster model. It works by clustering color-connected quarks after letting the remaining gluons from the parton shower decay into quark pairs. These clusters then decay further through the creation of quark pairs [154].

Finally, the decays of unstable hadrons have to be simulated, for example neutral pions decaying into two photons. The output of this part of the event generation pipeline is called "particle level" or "truth level"

2.3.3 Detector simulation and reconstruction

Detectors are needed to identify the particles produced in a collision, and measure their momenta. Detectors typically consist of multiple components responsible for different types of particles. The main components of general purpose detectors like ATLAS or CMS are the tracking system, electromagnetic and hadronic calorimeters and a muon spectrometer [155, 156]. The tracking system detects charged particles with high angular resolution and measures their momentum from the curvature of the tracks in a magnetic field. Calorimeters measure the energy that both charged and neutral particles deposit in the detector material. The electromagnetic calorimeter focuses on the measurement of the energies of electrons and photons, whereas the typically larger hadronic calorimeter measures the energy of hadrons like protons, neutrons and pions. Note that these hadrons also deposit energy in the electromagnetic calorimeter, but more stopping power is required to absorb all of their energy. Finally, muons do not deposit a significant part of their energy in the calorimeters. Hence, a dedicated tracking system for muons is placed outside of the calorimeters to allow for a more precise measurement of their momenta. The measured data from the various detector components is then used to reconstruct the type and momenta of the particles entering the detector. One important method that combines the information from the tracker and the calorimeter is the Particle Flow algorithm, described in detail in Refs. [157, 158].

A detailed description of detector effects is an important part of precise simulations of LHC events. This involves, among other things, modeling the geometry of the detector, the interactions of particles with the detector material and the response of the detector electronic. The software used to simulate the detector effects in most particle physics experiments is GEANT4 [40]. The disadvantage of such highly detailed detector simulations is that they are very computationally expensive. This is especially the case for particles with a high energy that cause a shower with a large multiplicity of daughter particles in the calorimeter. Consequently, detector simulation is one of the most expensive steps in the LHC simulation chain. In practice, not every LHC analysis requires such expensive simulations. Fast detector simulations [84, 159] are an alternative that make simplifying assumptions about the detector geometry and the interactions with the material to achieve a large speedup. Fast simulations of calorimeter showers are also a natural application of generative machine learning methods, and they have

recently become a part of the ATLAS fast simulation software [84]. For phenomenological studies, even more simplified simulations of the detector effect are often sufficient. In DELPHES, the detector geometry, calorimeter effects and reconstruction are modeled using simple smearing functions [39]. This results in a further speedup compared to the fast simulation packages used by ATLAS and CMS.

2.3.4 Jet algorithms

In many cases we are not interested in the large number of hadrons produced due to QCD effects, but in the final state of the hard process from which the shower originated. Based on the observation that the QCD effects result in collimated sprays of particles, jet algorithms find clusters of particles to reconstruct the outgoing momenta of the hard process. There are different ways to construct a jet algorithm. Most of them rely on a distance measure between particles. The distance measure for an important family of jet algorithms is given by [160]

$$\begin{aligned} d_{ij} &= \min\left(k_{ti}^{2p}, k_{tj}^{2p}\right) \frac{(\Delta R_{ij})^2}{R^2} \\ d_{iB} &= k_{ti}^{2p} \,, \end{aligned} \tag{2.71}$$

with the angular separation ΔR and the transverse momentum k_t. d_{ij} denotes the distance between two particles and d_{iB} the distance between a particle and the beam. R is the jet radius which is a parameter of the jet algorithm. This distance measure is used by the k_t algorithm [161] ($p = 1$), the Cambridge/Aachen algorithm [162] ($p = 0$) and the anti-k_t algorithm [160] ($p = -1$). These algorithms start with a list of the momenta of all detected particles and compute their pairwise distances and distances to the beam. If the smallest distance is between two momenta i and j, they are combined, reducing the length of the list by one. If the smallest distance is between momentum i and the beam, it is classified as a jet and removed from the list. This procedure is repeated until the list is empty.

The shape of the jet depends on the choice of distance measure. The k_t algorithm can be understood as reconstructing the history of collinear splittings from the parton shower, starting from soft momenta. This leads to irregular shapes of the jets. In contrast, the anti-k_t algorithm starts by combining hard momenta with close-by soft momenta. This leads to a more regular, conical shape of the jets [160]. After reconstructing jets, cuts are often imposed that only select events with at least a certain number of jets that fulfill some kinematic constraints. Often, there is a threshold for the transverse momentum of the jets to reject very soft jets, and a maximum pseudo-rapidity to reject jets that are very close to the beam.

Chapter 3

Machine learning

This chapter starts with a short introduction to deep learning in Sec. 3.1. We then describe the network architectures used in this thesis in detail. We introduce normalizing flows with a focus on invertible neural networks (INNs) in Sec. 3.2. We then show how the uncertainties from limited training statistics can be quantified with Bayesian neural networks in Sec. 3.3. Lastly, we describe transformers and how they can be used to build generative networks in Sec. 3.4.

3.1 Introduction to deep learning

In this section, we briefly review the basics of deep learning. We then discuss classifier networks with a focus on their probabilistic interpretation of extracting a weight function between two probability distributions.

3.1.1 Fully connected networks

The core of deep learning is to approximate complex high-dimensional functions as a series of linear functions combined with simple non-linearities, called activation functions. As every input feature is linked to every output feature through these linear functions, the resulting function is called a fully connected network. A single layer of such a network can be written as

$$y = \Phi(Wx + b) \; . \tag{3.1}$$

where x is a d_x-dimensional vector, y is a d_y-dimensional vector, W is a $(d_y \times d_x)$-dimensional matrix, b is a d_y-dimensional vector and Φ is the activation function. The components of the matrix W and bias vector b are called the trainable parameters of the network, as they have to be adapted to fit the network to a target function. Popular choices for the activation function [163] include component-wise functions such as

$$\text{ReLU}(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \tag{3.2}$$

$$\text{LeakyReLU}(x) = \begin{cases} \alpha x & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \tag{3.3}$$

$$\text{Softplus}(x) = \log(1 + \exp x) \tag{3.4}$$

$$\text{Sigmoid}(x) = \frac{1}{1 + \exp(-x)} \tag{3.5}$$

and functions operating on vectors,

$$\text{Softmax}(x_i) = \frac{\exp x_i}{\sum_j \exp x_j} \quad \text{such that} \quad \sum_i \text{Softmax}(x_i) = 1 \,. \tag{3.6}$$

Often, no activation function is applied in the last layer of a network, for example in regression tasks where the output needs to be unconstrained.

3.1.2 Loss functions and optimization

To optimize a neural network, the training objective has to be expressed as a function of its trainable parameters and the training data. The network parameters are then adapted to minimize this loss function. A simple example for a such a function is the mean squared error (MSE) loss used in regression, where a network f_θ with parameters θ is applied to input vectors x_i and the output is then compared to a truth label y_i. It can be written as

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (y_i - f_\theta(x_i))^2 \,. \tag{3.7}$$

The minimization of the loss function can then be performed using the gradient descent algorithm, where the network parameters are updated in small steps, in the direction of the gradient of the loss function with respect to the network parameters,

$$\theta \leftarrow \theta - \lambda \nabla_\theta \mathcal{L} \,, \tag{3.8}$$

where λ is called the learning rate. Gradient descent is prone to get stuck in local minima of the loss function. It can be improved by randomly splitting up the training dataset into smaller batches (also called mini-batches), and performing the optimizations steps batch-wise. This method is referred to as stochastic gradient descent (SGD). More sophisticated optimization algorithms implement further improvements to SGD. For instance, the Adam optimizer [164] is used for all network trainings discussed in this thesis. It tracks running means and variances of the gradients and computes the update step based on these instead of the raw gradients, leading to a faster and more stable convergence. Moreover, the learning rate is often adapted over the course of the training through a learning rate schedule. Large learning rates at the beginning of the training allow for a faster convergence and make it less likely to converge to a local minimum of the loss function. Small learning rates make the training more stable and increase the precision in the final stages of the training.

The MSE loss function given in Eq. (3.7) belongs to the category of likelihood losses. They are based on assuming some form of the likelihood $p_\theta(s)$ of a training point s given the network parameters θ. The training objective is to minimize the Kullback-Leibler (KL) divergence [165] between the truth distribution $p(s)$ and the learned distribution $p_\theta(s)$,

$$\begin{aligned} \mathcal{L} &= \text{KL}(p(s), p_\theta(s)) \\ &= \int \mathrm{d}s \, p(s) \log \frac{p(s)}{p_\theta(s)} \\ &= -\int \mathrm{d}s \, p(s) \log p_\theta(s) + \text{const.} \,. \end{aligned} \tag{3.9}$$

As shown in the last line, this is equivalent to maximizing the log-likelihood. Neglecting the constant offset, the integral can also be rewritten as an expectation value

$$\mathcal{L} = -\left\langle \log p_\theta(s) \right\rangle_{s\sim p(s)} \ . \tag{3.10}$$

It can then be approximated as a mean over the training samples $\{s_i\}$,

$$\mathcal{L} \approx -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(s_i) \ , \tag{3.11}$$

where N is the size of the training dataset. Consider labeled training data $s = (x, y)$ with inputs x and truth labels y. Under the assumption of a Gaussian shape of the conditional probability $p_\theta(y|x)$, the joint probability is

$$p_\theta(x, y) = p_\theta(y|x)p(x) \propto \exp\left(\frac{(y - f_\theta(x))^2}{2}\right) p(x) \ . \tag{3.12}$$

By inserting this into Eq. (3.10) and neglecting the constant offset, the MSE loss from Eq. (3.7) is recovered. Further examples for likelihood losses include the binary cross entropy loss for classification discussed in Sec. 3.1.3 and the normalizing flow loss discussed in Sec. 3.2.1.

3.1.3 Classification

A common application of deep learning is to categorize input samples into two or more classes in a probabilistic way, i.e. the outputs of the network are the probabilities for each class. In the following, we will discuss the special case of binary classifiers and its applications beyond classification.

The training data of a binary classifier consists of tuples (x, y), where x is the input vector and $y \in \{0, 1\}$ is the truth label. In general, both classes may contain samples in a certain region of input space, so it is not always possible to a assign an input x to a single class. Let $p_0(x) = p(x|y = 0)$ be the probability distribution of class 0, $p_1(x) = p(x|y = 1)$ the distribution of class 1, and $P(y)$ the probabilities for samples to be in each category, i.e. $P(y = 0) + P(y = 1) = 1$. Then the joint distribution is given by

$$p(x, y) = p(x|y)P(y) = p_y(x)P(y) \ . \tag{3.13}$$

For classification, we model this as a conditional Bernoulli distribution of the two categories where the probability of each category is a learnable function of x,

$$P_\theta(y = 1|x) = f_\theta(x) \quad \text{and} \quad P_\theta(y = 0|x) = 1 - f_\theta(x) \quad \text{with} \quad 0 \leq f_\theta(x) \leq 1. \ , \tag{3.14}$$

with trainable parameters θ. We can then write the joint distribution modeled by the network as

$$p_\theta(x, y) = P_\theta(y|x)p(x) \quad \text{with} \quad p(x) = p_0(x)P(y = 0) + p_1(x)P(y = 1) \ . \tag{3.15}$$

Because y can only be 0 or 1, we can rewrite Eq. (3.14) as

$$\log P_\theta(y|x) = y \log f_\theta(x) + (1 - y) \log(1 - f_\theta(x)) \ . \tag{3.16}$$

The function $f_\theta(x)$ can be implemented as a simple fully-connected network with a sigmoid activation function in the final layer to guarantee outputs between 0 and 1. Depending on the application, more complex architectures can be beneficial for the performance. The network objective now is to minimize the KL divergence between $p(x, y)$ and $p_\theta(x, y)$,

$$\begin{aligned}
\mathcal{L} &= \mathrm{KL}(p(x,y), p_\theta(x,y)) \\
&= \sum_{y=0}^{1} \int \mathrm{d}x\, p(x,y) \log \frac{p(x,y)}{p_\theta(x,y)} \\
&= \sum_{y=0}^{1} \int \mathrm{d}x\, p(x,y) \log \frac{p(x,y)}{P_\theta(y|x)p(x)} \\
&= -\sum_{y=0}^{1} \int \mathrm{d}x\, p(x,y) \log P_\theta(y|x) + \text{const.} \\
&= -\sum_{y=0}^{1} \int \mathrm{d}x\, p(x,y) \Big[y \log f_\theta(x) + (1-y) \log(1 - f_\theta(x)) \Big] + \text{const.} \\
&= -\Big\langle y \log f_\theta(x) + (1-y) \log(1 - f_\theta(x)) \Big\rangle_{(x,y)\sim p(x,y)} + \text{const.} \,.
\end{aligned} \tag{3.17}$$

The resulting loss function is known as binary cross-entropy.

In addition to the interpretation of $f_\theta(x)$ as the probability for a point x to be in class 1, we can also use the classifier to obtain a weight function $w(x)$ that relates the probability distributions of the two classes as

$$p_1(x) = w(x) p_0(x) \,. \tag{3.18}$$

First, we assume a training dataset in which both classes are present in equal proportions, i.e. $P(y = 0) = P(y = 1) = 0.5$. For a trained classifier network f_θ, we can use Eqs. (3.13) to (3.15) to show that

$$f_\theta(x) \approx P(y = 1|x) = \frac{p_1(x)}{p_0(x) + p_1(x)} \,. \tag{3.19}$$

Inserting Eq. (3.18) results in

$$f_\theta(x) \approx \frac{w(x) p_0(x)}{p_0(x) + w(x) p_0(x)} = \frac{w(x)}{1 + w(x)} \,. \tag{3.20}$$

Solving this for w leaves us with

$$w_\theta(x) \equiv \frac{f_\theta(x)}{1 - f_\theta(x)} \approx w(x) \,. \tag{3.21}$$

Note that, if the network is implemented with a sigmoid activation in the final layer, the weight can be directly obtained by taking the exponential of the output of the final layer without an activation function.

This result means that classifier networks are not only useful for classification. They are also a powerful tool that allows us to learn reweighting factors between probability distributions. In Ch. 4, we will discuss how they can be used to reweight samples obtained from generative networks and to evaluate their performance.

3.2 Normalizing flows

Normalizing flows are a type of deep neural networks that defines a bijective mapping between two probability distributions and has a tractable Jacobian [106, 166]. Typically, a normalizing flow is built by chaining multiple simple transformations. Let $z_1, \ldots, z_{n+1}$ be vectors in the spaces that these transformations map between. Let $G_1, \ldots, G_n$ be bijective mappings such that

$$z_{i+1} = G_i(z_i) \ . \tag{3.22}$$

Then the full transformation is

$$z_{n+1} = G(z_1) \equiv (G_n \circ \cdots \circ G_1)(z_1) \ . \tag{3.23}$$

We can use the change of variables formula to obtain the Jacobian of the full transformation,

$$g(z_1) = \left| \frac{\partial G(z_1)}{\partial z_1} \right| = \prod_{i=1}^{n} \left| \frac{\partial G_i(z_i)}{\partial z_i} \right| = \prod_{i=1}^{n} g_i(z_i) \ . \tag{3.24}$$

Assuming that z_{n+1} is distributed according to some probability distribution $p(z_{n+1})$, we can use this result to write

$$\log p(z_1) = \log p(z_{n+1}) + \sum_{i=1}^{n} \log \left| \frac{\partial G_i(z_i)}{\partial z_i} \right| \ . \tag{3.25}$$

Normalizing flows are often used to learn a mapping between a complicated distribution of interest and a simple latent distribution, like a normal or uniform distribution. Once trained, they can be used for density estimation by transforming from the complex data distribution into the latent space, and for sampling, by drawing a sample from the latent distribution and transforming it to the data space. This is also the motivation for the name "normalizing flows". In this thesis, the following conventions for naming these different spaces and transformations will be used:

- The *latent space* ($z_{n+1} \sim p(z_{n+1})$ in the equations above) follows a distribution for which we can compute the probability and that we can easily sample from.
- The *data space* ($z_1 \sim p(z_1)$ in the equations above) follows the distribution of interest that we want to capture with the normalizing flow to perform density estimation, or to sample from it.
- The *forward transformation* ($G = G_n \circ \cdots \circ G_1$ in the equations above) maps from the data space into the latent space. It is used to train the normalizing flows on samples. Once the network is trained, the forward direction can be used to estimate the density of data points.
- The *inverse transformation* ($\bar{G} = \bar{G}_1 \circ \cdots \circ \bar{G}_n$ in the equations above) maps from the latent space into the data space. It is used for sampling from the learned data distribution.

While mapping between a data distribution and a simple latent distribution is the most common use case for normalizing flows, they can also be used to learn mappings between different data distributions [19, 109]. If neither of these data distributions have a tractable likelihood and both distributions can only be inferred from data, training these networks

is more challenging. In this thesis, we will only use normalizing flows with simple latent distributions. The two most important examples are Gaussian latent spaces with

$$\log p(z) = -\frac{\|z\|^2}{2} - \frac{D}{2}\log(2\pi) = -\frac{\|z\|^2}{2} + \text{const.}\,, \tag{3.26}$$

and uniform latent spaces over an interval $[a, b]$ with

$$\log p(z) = -D\log(b-a) = \text{const.}\,, \tag{3.27}$$

where D is the number of dimensions.

The definitions above can be easily generalized to allow for conditional probability distributions. In this case, the data space distribution is a conditional probability $p(z_1|c)$. By defining the forward transformations $z_{i+1} = G_i(z_i; c)$ and inverse transformations $z_i = \bar{G}_i(z_{i+1}; c)$ as functions of the condition c, we obtain a conditional version of Eq. (3.25),

$$\log p(z_1|c) = \log p(z_{n+1}) + \sum_{i=1}^{n} \log \left| \frac{\partial G_i(z_i; c)}{\partial z_i} \right| . \tag{3.28}$$

3.2.1 Loss functions

The training objective for normalizing flows is to approximate the truth distribution as close as possible. The truth distribution can be either given in the form of weighted or unweighted samples, or it can have a known analytic form. In both cases, we have to define some measure of similarity between the truth and learned distribution to express the objective as a minimization problem. In the following, we discuss several options for such divergences. We denote the data space as x, the truth distribution as $p(x)$, and the flow distribution as $g_\theta(x)$. The latter includes both the Jacobian and the latent space distribution. The subscript θ indicates the dependence on the trainable network parameters. In the most general case, the training samples x do not have to follow the truth distribution $p(x)$. We denote their distribution as $q(x)$.

Written in the most general way, the training objective is to minimize some divergence D between $p(x)$ and $g_\theta(x)$,

$$\mathcal{L} = D(p(x), g_\theta(x)) . \tag{3.29}$$

Two important options for the divergence D are the KL divergence and the variance. These will be discussed in greater detail below. However, in general there are many other options, for example arbitrary f-divergences, including the reverse KL divergence and the Jensen-Shannon divergence [106].

KL divergence

The KL divergence between g_θ and p is defined as

$$\begin{aligned} \mathrm{KL}(p(x), g_\theta(x)) &= \int \mathrm{d}x\, p(x) \log \frac{p(x)}{g_\theta(x)} \\ &= \left\langle \frac{p(x)}{q(x)} \log \frac{p(x)}{g_\theta(x)} \right\rangle_{x\sim q(x)} , \end{aligned} \tag{3.30}$$

where we expressed the integral as an expectation value over the sample distribution $q(x)$ in the second line. By absorbing terms that do not depend on the trainable parameters θ into an additive constant, we can further simplify this equation to get

$$\mathrm{KL}(p(x), g_\theta(x)) = -\left\langle \frac{p(x)}{q(x)} \log g_\theta(x) \right\rangle_{x\sim q(x)} + \text{const.} \,. \tag{3.31}$$

There is one important special case of the KL divergence loss. If the network is trained directly on samples from the truth distribution, we have

$$q(x) = p(x) \,. \tag{3.32}$$

Then we recover the usual form of a likelihood loss from Eq. (3.10),

$$\mathcal{L} = -\left\langle \log g_\theta(x) \right\rangle_{x\sim p(x)} \,. \tag{3.33}$$

The main advantage of this loss function is that the truth distribution does not have to be known as long as samples from it are available. The log-likelihood loss for cINNs with a Gaussian latent space is the most-used loss function in this thesis. Given a forward transformation $G_\theta(x|c)$ with trainable parameters θ and a training distribution $p(x, c)$, we can combine Eqs. (3.26), (3.28) and (3.33) to get the loss function

$$\mathcal{L} = \left\langle \frac{\|G_\theta(x|c)\|^2}{2} - \log \left| \frac{\partial G_\theta(x|c)}{\partial x} \right| \right\rangle_{(x,c)\sim p(x,c)} \,. \tag{3.34}$$

If the distributions p and q are not the same, we can define weights in analogy to Eq. (2.24),

$$w(x) \equiv \frac{p(x)}{q(x)} \,. \tag{3.35}$$

The KL divergence loss then has the form

$$\mathcal{L} = -\left\langle w(x) \log g_\theta(x) \right\rangle_{x\sim q(x)} + \text{const.} \,. \tag{3.36}$$

This can be understood as a weighted log-likelihood loss. The network can again be trained without knowing the truth or sample distributions as long as their relative weight $w(x)$ is known and samples $x \sim q(x)$ are available.

Variance

One advantage of normalizing flows over other generative architectures is that the exact likelihood for every generated sample is available without additional computational cost. This makes them perfect candidates for importance sampling proposal distributions as introduced in Sec. 2.2.1. The training objective then is to minimize the integral variance [45, 167] defined in Eq. (2.26),

$$\begin{aligned} \mathcal{L} &= \mathrm{Var}\left(\frac{f(x)}{g_\theta(x)} \right)_{x\sim g_\theta(x)} \\ &= \left\langle \frac{f(x)^2}{g_\theta(x)^2} \right\rangle_{x\sim g_\theta(x)} - \left\langle \frac{f(x)}{g_\theta(x)} \right\rangle^2_{x\sim g_\theta(x)} \,. \end{aligned} \tag{3.37}$$

Note that the training samples are drawn from the learned distribution $g_\theta(x)$ itself, so the training does not rely on a pre-generated dataset. Furthermore, while the target function $f(x)$ needs to be known, it does not have to be a normalized probability distribution. The variance loss and its application to neural multi-channel importance sampling will be discussed in detail in Ch. 5.

3.2.2 Architectures

So far we have only discussed how to define normalizing flows and their training objective. Next, we need a neural network architecture that provides expressive bijective transformations with tractable Jacobians. This is achieved by breaking up the transformation into a chain of smaller, invertible blocks. There are two important options how these blocks can be constructed.

Coupling blocks

The first option are normalizing flows based on coupling blocks [107, 108]. Let $C(x_i|c)$ be a transformation that is invertible with respect to a scalar input x_i under a condition c. $\bar{C}(y_i|c)$ is the inverse of that transformation. Let x be a d-dimensional input vector. Coupling blocks first split the input vector into two halves $x = (x^A, x^B)$. The first half is then used as the input of a fully-connected neural network $\psi(x^A)$. This network is called the sub-network of the coupling block. Crucially, this network does not have to be invertible itself. The second half of the input vector is then transformed component-wise using the output of the sub-network as the condition to the transformation C. The output of the full coupling block is then given by

$$\begin{aligned} y_i^A &= x_i^A \\ y_j^B &= C(x_j^B|\psi(x^A)) \ , \end{aligned} \tag{3.38}$$

with the inverse

$$\begin{aligned} x_i^A &= y_i^A \\ x_j^B &= \bar{C}(y_j^B|\psi(x^A)) \ . \end{aligned} \tag{3.39}$$

The Jacobian of the transformation is

$$\frac{\partial y}{\partial x} = \begin{pmatrix} \mathbb{1} & 0 \\ \frac{\partial y^B}{\partial x^A} & \frac{\partial y^B}{\partial x^B} \end{pmatrix} \quad \text{with} \tag{3.40}$$

$$\frac{\partial y^B}{\partial x^A} = \frac{\partial C(x_j^B|\psi(x^A))}{\partial x^A} \quad \text{and} \tag{3.41}$$

$$\frac{\partial y^B}{\partial x^B} = \mathrm{diag}_j \left(\frac{\partial C(x_j^B|\psi(x^A))}{\partial x_j^B} \right) \ . \tag{3.42}$$

Because of the diagonal form of Eq. (3.42), the Jacobian in Eq. (3.40) is a lower triangular matrix and its determinant is given by

$$\left| \frac{\partial y}{\partial x} \right| = \prod_j \frac{\partial C(x_j^B|\psi(x^A))}{\partial x_j^B} \ . \tag{3.43}$$

The complexity of computing the determinant is therefore linear with the number of dimensions, making it computationally tractable even for high-dimensional input vectors. The transformation can be made conditional by concatenating a condition vector with the sub-network input,

$$y_j^B = C(x_j^B|\psi(x^A, c)) \quad \text{and} \quad x_j^B = \bar{C}(y_j^B|\psi(x^A, c)) \; . \tag{3.44}$$

The main advantage of coupling blocks is that they are equally fast in both directions. In the following, we will refer to this type of normalizing flow as an Invertible Neural Network (INN) or conditional Invertible Neural Network (cINN).

Autoregressive flows

The other important way to implement normalizing flows is based on factorizing the probability autoregressively,

$$g(x) = g(x_1)\, g(x_2|x_1) \, \cdots \, g(x_d|x_1, \ldots, x_{d-1}) \; . \tag{3.45}$$

This allows the network to model correlation between all dimensions while only performing one-dimensional transformations C. Similar to coupling blocks, autoregressive flows can be implemented using conditional transformations, where the condition is computed by a neural network. The autoregressive structure, like coupling blocks, guarantees that the Jacobian of the transformation is a triangular matrix. Therefore, its determinant is fast to compute. One example how such an autoregressive structure can be implemented is to use sub-networks with an autoregressive mask of the network weights [168]. $g(x)$ can be computed efficiently in the forward direction, because all $g(x_i|\ldots)$ in Eq. (3.45) are independent, allowing for parallelization. However, in the inverse direction the terms can only be evaluated sequentially because every term depends on the results of the previous terms. This is in contrast to coupling block-based architectures, where both directions have the same computational cost. To solve this problem, it has been proposed to use the fast density estimation network to train a fast sampling network in a second step in a teacher-student or distillation setup [85, 169]. While this accelerates the sampling, it also makes the training more complex.

Permutations and rotations

A single coupling block only transforms one half of the input features. Also, it does not allow for correlations between the features being transformed. This problem can be solved by chaining multiple coupling blocks and permuting the features between the blocks [107, 108]. While a single autoregressive block would theoretically be sufficient to model arbitrary correlations, practical implementations of such blocks are typically not expressive enough. Therefore, permuting the features is also necessary for autoregressive flows.

There are several ways to construct these permutations. One option is to sample the permutations randomly when the network is constructed. This approach can lead to problems for low numbers of coupling blocks since it is not guaranteed that every feature is conditioned on every other feature at least once. A more systematic approach to construct the permutations is proposed in Ref. [45]. It guarantees that correlations

between all features can be modeled, leading to a number of coupling blocks n_b that scales logarithmically with the input dimension D,

$$n_b = 2 \lceil \log_2 D \rceil \ . \tag{3.46}$$

An even simpler approach is to just keep exchanging the first and the second half of the features. This approach can work for highly correlated data like images, where the locations of features in the first and second half can be arranged in a checkerboard pattern [108]. However, correlations between some features can then only be learned indirectly. For normalizing flows operating on a Gaussian latent space, the random permutations can be generalized and replaced with random $O(d)$ rotations matrices [170]. To increase the flexibility of this approach, the fixed rotations can be replaced with trainable rotations that are constructed using trainable Euler angles [3].

3.2.3 Coupling transformations

Both INNs and autoregressive flows are based on one-dimensional, invertible mappings C that can be conditioned on the output of a neural network.

Affine transformations

A simple way to define an invertible mapping is to multiply the input with a positive number $\exp(r)$ and then add an offset s, where both r and s are the output of a neural network [107, 108]. The transformation and its Jacobian then have the form

$$\begin{aligned} y = C(x|r,s) &= \exp(r)\, x + s \\ x = \bar{C}(y|r,s) &= \exp(-r)\, (y - s) \\ c(x|r,s) = \frac{\partial C(x|r,s)}{\partial x} &= \exp(r) \ . \end{aligned} \tag{3.47}$$

This transformation is often combined with a soft, invertible clamping function, for instance

$$y = \frac{2\alpha}{\pi} \arctan\left(\frac{s}{\alpha}\right) \ , \tag{3.48}$$

with a hyperparameter α, to improve the training stability and prevent exploding values especially in the early stages of the training [170]. Affine coupling blocks are easy to implement and fast to evaluate. They are however not very expressive, so it is necessary to chain many coupling blocks to model complex distributions.

Neural spline flows

The expressivity can be improved by replacing the affine transformation with a spline-based transformation. Spline flows are defined by splitting up the input interval into bins. Within each bin, the transformation is modelled by a simple function, for example linear [167], quadratic [167], cubic [171] or rational-quadratic [110] functions. We will discuss the latter option in more detail as it combines high expressivity with good numerical stability.

For simplicity, we will restrict our discussion to splines where both input and output are defined on the unit interval $[0, 1]$. The interval is split up into K bins with $K + 1$ boundaries. These bins are defined by their widths w, heights h, and the derivatives d at the boundaries. Ref. [110] proposes to set the derivative at the outer boundaries of the unit interval to 1. Then the transformation can be smoothly continued as the identity function outside of the unit interval. In cases where the feature being transformed is restricted to be within the unit interval, this limits the expressivity close to the boundaries. This can be solved by also making the outer derivatives free parameters of the spline [6]. Depending on that choice, there are $3n - 1$ or $3n + 1$ parameters of the spline which have to be predicted by a trainable sub-network. Starting from unnormalized network outputs

$$(\Theta^w, \Theta^h, \Theta^d) \tag{3.49}$$

the normalized widths, heights and derivatives are given by

$$\begin{aligned} w &= \operatorname{Softmax} \Theta^w \\ h &= \operatorname{Softmax} \Theta^h \\ d &= \frac{\operatorname{Softplus} \Theta^d}{\log 2} \equiv \frac{\log\left(1 + \exp \Theta^d\right)}{\log 2} \,. \end{aligned} \tag{3.50}$$

These fulfill the normalization conditions

$$\sum_{k=1}^{K} w_k = 1 \,, \quad \sum_{k=1}^{K} h_k = 1 \quad \text{and} \quad d_k > 0 \,. \tag{3.51}$$

The factor $\log 2$ in Eq. (3.50) is introduced to make the spline equal to the identity transformation if $\Theta^w = \Theta^h = \Theta^d = 0$. This makes it easy to initialize the sub-networks such that the normalizing flow training starts from the identity mapping. The transformation within each bin is a monotonic rational-quadratic function, i.e. the ratio between two quadratic polynomials. The inverse and derivative of this function can be computed analytically. The details on how to construct this function can be found in Ref. [110].

In many cases where generative models are used, there are periodic features like azimuthal angles. Spline coupling transformations can be easily extended such that they ensure periodicity by setting $d_0 = d_K$ [172]. While this makes the transformation periodic, its flexibility is limited because there is still a fixed boundary at 0 and 1. This problem can be solved by introducing an additive shift [8].

3.2.4 Continuous flows

INNs and autoregressive flows are based on transformations that are restricted such that the Jacobian is a triangular matrix and therefore computationally tractable. A different approach is to transform all dimensions simultaneously by making the transformation a continuous function of time [173]. Its dynamics are written as an ordinary differential equation (ODE) with a function v_θ encoded by a neural network and the vector of features $x(t)$ at time t,

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = v_\theta(x(t), t) \,. \tag{3.52}$$

In analogy to fluid dynamics, the function v_θ is called velocity field. The ODE can be turned into a normalizing flow by imposing the boundary conditions for the probability density of x at two points in time,

$$p(x, t = 0) = p_{\text{latent}}(x) \quad \text{and} \quad p(x, t = 1) = p_{\text{model}}(x) \; . \tag{3.53}$$

Again in analogy to fluid dynamics, the probability density $p(x, t)$ is described by the continuity equation,

$$\frac{\partial p(x,t)}{\partial t} + \nabla_x \cdot \Big[p(x,t) v_\theta(x,t) \Big] = 0 \; . \tag{3.54}$$

Given $x(t_0)$ at one point in time, t_0, the ODE can be used to determine $x(t)$ at all other times. Hence, it defines an invertible mapping between the latent space and the data space. Solving the continuity equation for a given trajectory yields the Jacobian of this transformation,

$$p(x, t = 1) = p(x, t = 0) \exp \left[\int_0^1 \mathrm{d}t \, \mathrm{tr} \Big(\nabla_x v_\theta(x(t), t) \Big) \right] \; . \tag{3.55}$$

The mapping and the Jacobian are computed by integrating out t with an ODE solver. Numerically, this is done by splitting up the time into smaller steps. The network v_θ and its gradients with respect to x (if the Jacobian is needed) have to be evaluated for every time step. The main advantage of continuous flows is that there are no restrictions for the architecture of the network v_θ. This can increase the expressivity of continuous flows compared to other types of normalizing flows.

The first proposed training method for continuous flows was to directly use the probability from Eq. (3.55) in a log-likelihood loss [173]. The ODE over the gradients of the velocity field then has to be evaluated for every weight update during the training, making it slow and unstable. Several improvements have been proposed to increase the training speed and stability [174, 175].

Conditional Flow Matching (CFM) is a different way to train continuous flows where the ODE does not have to be solved during training [102–104]. It combines the basic structure of continuous flows with a pre-defined time evolution between the latent and data distribution. This approach combines the invertible setup of continuous flows with the training method used for diffusion models [100, 101]. A simple choice for such a target trajectory is linear interpolation between the target distribution and a normal distribution [104],

$$p(x,t) = \int \mathrm{d}x_0 \, p(x,t|x_0) \, p_{\text{data}}(x_0) \quad \text{with} \quad p(x,t|x_0) = \mathcal{N}(x; (1-t)x_0, t) \; . \tag{3.56}$$

By construction, $p(x, t)$ fulfills the boundary conditions. The corresponding velocity field can be found by inserting it into the continuity equation. The training objective is now to minimize the difference between $v(x, t)$ and $v_\theta(x, t)$ with a MSE loss function,

$$\mathcal{L} = \Big\langle \Big[v_\theta(x,t) - v(x,t) \Big]^2 \Big\rangle_{(x,t) \sim p(x,t)} \; . \tag{3.57}$$

While both sampling from $p(x, t)$ and the target velocity field $v(x, t)$ are intractable, it can be shown that the loss function can be rephrased in terms of the conditional

probability $p(x, t|x_0)$. The loss function then becomes [102]

$$\mathcal{L}_{\text{CFM}} = \left\langle \left[v_\theta((1-t)x_0 + t\epsilon, t) - (\epsilon - x_0) \right]^2 \right\rangle_{t\sim u, x_0\sim p_{\text{data}}, \epsilon\sim\mathcal{N}} , \tag{3.58}$$

where x_0 is sampled from training dataset, t is sampled uniformly from the interval $[0, 1]$ and ϵ is sampled from a normal distribution. This way, the complex continuous flow training has been replaced with a simple regression task. For a detailed derivation of this loss function, we refer to Ref. [102]. CFMs can be easily adapted to learn conditional distributions by giving the condition as an additional input to the network encoding the velocity field.

3.3 Bayesian neural networks

One of the first lessons that a physicist learns is that a measurement is only complete when it comes with an uncertainty estimate. When neural networks are applied in physics, we have to extend this rigorous treatment of uncertainties to deep learning. In this section, we will review the different types of uncertainties that arise when neural networks are used and then discuss Bayesian neural networks (BNNs), a type of network that allows us to estimate a part of these uncertainties.

3.3.1 Types of uncertainties

When we train a neural network, we have to distinguish between three types of uncertainties. Firstly, there are intrinsic uncertainties in the data. For example, when a classifier is trained, multiple labels can have non-zero probabilities. Because it is intrinsic, this uncertainty cannot be overcome by constructing a better network or improving the training.

Secondly, networks are usually trained on a limited amount of training samples. Even for a network that is expressive enough to exactly model the target, this will lead to an uncertainty on the weights of the network. This uncertainty vanishes in the limit of infinite training data.

Lastly, real-world neural networks might not be expressive enough to perfectly model the truth. But even if they are, the training might get stuck in a local minimum or might not be fully converged. This will lead to uncertainties that are neither intrinsic to the data nor go away in the limit of infinite training data.

The first uncertainty is often referred to as epistemic or systematic uncertainty, and the second one is called aleatoric or statistical uncertainty [113]. The names statistical and systematic uncertainty are problematic in a physics context because they usually refer to uncertainties that do or do not scale with the number of observations, whereas here, these names are used to describe the scaling with respect to the number of training samples. In this thesis, we will only use the terms statistical and systematic in the way that they are understood in physics, and refer to the uncertainty arising from the limited training statistics as predictive uncertainty.

Note that our discussion of the different types of uncertainties was based on the assumption that we can trust the distribution of the training data. If this is not the case, there are

additional uncertainties from the simulations used to generate the training data. These systematic uncertainties are often addressed with nuisance parameters in particle physics.

3.3.2 Building a BNN

Bayesian neural networks [111–114] can be constructed for any model with a likelihood loss that is trained on a fixed training dataset. Examples for likelihood losses include regression (see Sec. 3.1.2), classification (see Sec. 3.1.3) and (conditional) normalizing flows (see Sec. 3.2). Consider a training dataset $\mathcal{D}$ of N paired points (x_i, c_i). Likelihood losses assume some functional form of the likelihood $p(x|c,\theta)$ of these points, where we make the conditional dependence on the network parameters explicit. The training objective then is to minimize the loss function

$$\mathcal{L} = -\log p(\mathcal{D}|c,\theta) = -\sum_{i=1}^{N} \log p(x|c,\theta) \ . \tag{3.59}$$

The following derivation can be easily applied to non-conditional models by omitting the condition c.

Even if the exact form of the likelihood of the training data was known, we could only get an estimate of the parameters θ due to the limited number of points in the training dataset. To capture the resulting uncertainty, we assume some prior distribution of the network weights $p(\theta)$, and ask for the posterior distribution of the network weights given the training data, $p(\theta|\mathcal{D})$. While Markov Chain Monte Carlo methods can be used to sample from the posterior for small models [176], this is intractable for models with millions of parameters. An alternative approach is variational inference (VI) [177] where the true posterior is approximated by a simpler, tractable distribution $q_\phi(\theta)$ with trainable parameters ϕ. The training objective then becomes to minimize the KL divergence between this approximation and the true posterior,

$$\phi_{\text{opt}} = \arg\min_{\phi} \mathrm{KL}(q_\phi(\theta), p(\theta|\mathcal{D})) \ . \tag{3.60}$$

We can express this in terms of the prior and likelihood using Bayes' theorem,

$$\begin{aligned}
\mathrm{KL}(q_\phi(\theta), p(\theta|\mathcal{D})) &= \int \mathrm{d}\theta \, q_\phi(\theta) \log \frac{q_\phi(\theta)}{p(\theta|\mathcal{D})} \\
&= \int \mathrm{d}\theta \, q_\phi(\theta) \log \frac{q_\phi(\theta) p(\mathcal{D})}{p(\mathcal{D}|\theta) p(\theta)} \\
&= -\int \mathrm{d}\theta \, q_\phi(\theta) \log p(\mathcal{D}|\theta) + \int \mathrm{d}\theta \, q_\phi(\theta) \log \frac{q_\phi(\theta)}{p(\theta)} + \int \mathrm{d}\theta \, q_\phi(\theta) \log p(\mathcal{D}) \\
&= -\left\langle \log p(\mathcal{D}|\theta) \right\rangle_{\theta \sim q_\phi(\theta)} + \mathrm{KL}(q_\phi(\theta), p(\theta)) + \log p(\mathcal{D}) \ .
\end{aligned} \tag{3.61}$$

The log-likelihood of the full dataset can written as a sum over the log-likelihoods for the single points, giving us

$$\log p(\mathcal{D}|\theta) = \sum_{i=0}^{N} \left(\log p(x_i|c_i,\theta) + \log p(c_i) \right) \ . \tag{3.62}$$

By splitting up the dataset into mini-batches of size M, we can approximate this likelihood as

$$\log p(\mathcal{D}|\theta) \approx \frac{N}{M} \sum_{i=0}^{M} \left(\log p(x_i|c_i, \theta) + \log p(c_i)\right) \ . \tag{3.63}$$

The evidence $p(\mathcal{D})$ does not depend on the trainable weights ϕ, so we do not have to include it in the loss function. The same is true for the distribution of the condition, $p(c)$. The expectation value of the likelihood with respect to the parameters θ can be approximated by drawing a single sample from $p(\theta)$ per training batch. Then the BNN loss function is

$$\mathcal{L} = -\frac{1}{M} \sum_{i=0}^{M} \log p(y_i|x_i, \theta) + \frac{1}{N} \mathrm{KL}(q_\phi(\theta), p(\theta)) \quad \text{with} \quad \theta \sim q_\phi(\theta) \ , \tag{3.64}$$

where we divided Eq. (3.61) by N to recover the usual log-likelihood loss in the first term. The second term effectively acts like a regularization of the network weights.

A common choice for the form of the prior and posterior is to assume uncorrelated Gaussians. While this is a very restrictive assumption, it is still able to model complex uncertainties. Because the network is deep, varying parameters in early layers of the network can lead to complex, non-linear effects on the network output [113]. In some cases, it is even sufficient to only use VI for the first layer of the network and use deterministic layers for the rest of the network [178]. The prior is often chosen to have a mean $\mu_p = 0$ and a standard deviation σ_p. The approximated posterior for a network parameter θ_j has means μ_j and standard deviations σ_j. Going from a deterministic network with trainable parameters θ_j to a Bayesian network with parameters $\phi_j = (\mu_j, \sigma_j)$ therefore doubles the number of parameters. For this choice of prior, the KL divergence can be computed analytically,

$$\mathrm{KL}(q_\phi(\theta), p(\theta)) = \sum_j \left(\log \frac{\sigma_p}{\sigma_j} + \frac{\sigma_j^2 + \mu_j^2}{2\sigma_p^2} - \frac{1}{2} \right) \ . \tag{3.65}$$

Sampling from $q_\phi(\theta)$ while still allowing to compute gradients is possible using the local reparameterization trick, where the sampling is expressed as

$$\theta = \mu + \sigma\epsilon \quad \text{with} \quad \epsilon \sim \mathcal{N}(0, 1) \ . \tag{3.66}$$

Then the gradients of the loss with respect to μ and σ can be computed through backpropagation as the gradients $\partial\theta/\partial\mu$ and $\partial\theta/\partial\sigma$ are available.

We give the loss function for Bayesian cINNs [179] explicitly, because they are used several times throughout this thesis. By combining Eqs. (3.34) and (3.64), we get

$$\mathcal{L} = \frac{1}{M} \sum_{i=0}^{M} \left(\frac{\|G_\theta(x_i|c_i)\|^2}{2} - \log \left| \frac{\partial G_\theta(x_i|c_i)}{\partial x} \right| \right) + \frac{1}{N} \mathrm{KL}(q_\phi(\theta), p(\theta)) \ , \tag{3.67}$$

with $\theta \sim q_\phi(\theta)$.

3.4 Transformers

Transformer networks were originally proposed as an architecture for translation tasks where a sequence of words in one language is generated, conditioned on a sequence

of words in another language [105]. Since then, they have been applied to various applications in language processing and beyond [180, 181]. In particle physics, examples for the use of transformers include jet tagging [182], representation learning [183], and generative tasks [64, 184].

Transformers consist of two parts, the transformer encoder and decoder. The encoder transforms an input sequence of length N_E into an output sequence of length N_E. The decoder takes a sequence of length N_D and the output sequence of the encoder as inputs. Its output is a sequence of length N_D. Each sequence consists of d-dimensional vectors. Both encoder and decoder are permutation-equivariant with respect to their input sequences. In cases where the order of the sequence is relevant, positional encoding can be used to embed this information into the vectors encoding the sequence elements.

3.4.1 Attention mechanism

The core of transformers is the scaled dot-product attention mechanism [105]. Let x_i^K, x_i^V and x_i^Q be d-dimensional vectors, where the index i indicates that they are elements of a sequence. First, trainable matrices $W^K, W^V, W^Q \in \mathbb{R}^{m\times d}$ are applied to those vectors, resulting in m-dimensional vectors

$$k_i = W^K x_i^K \ , \quad v_i = W^V x_i^V \ , \quad q_i = W^Q x_i^Q \ . \tag{3.68}$$

These vectors are often called key, value and query. Next, the attention matrix A_{ij} is constructed from the keys and queries,

$$A_{ij} = \mathrm{Softmax}_j \left(\frac{q_i \cdot k_j}{\sqrt{m}} \right) \ . \tag{3.69}$$

Finally, the attention matrix is multiplied with the value vectors, yielding a sequence of output vectors

$$y_i = A_{ij} v_j \ . \tag{3.70}$$

The attention matrix can be interpreted as learning the relationships between sequence elements. Since there might be multiple different meaningful ways to construct these relationships, the simple attention can be extended to a multi-head attention mechanism. Let h be the number of heads. For each head, an independent attention matrix is computed and the resulting y_i are concatenated. For the choice $m = d/h$, this results in a d-dimensional output vector

$$y_i = \mathrm{Concat}(y_i^1, \ldots, y_i^h) \ , \tag{3.71}$$

where the y_i^k denote the outputs of the individual attention heads from Eqs. (3.69) and (3.70). The attention transformation is then followed by one or more fully-connected layers applied to each sequence element. These layers are the same for all elements, therefore preserving the permutation-equivariance. The transformer encoder and decoder are built from a chain of multiple attention blocks. In the case of the encoder, all inputs are chosen to be the same,

$$x_i^K = x_i^V = x_i^Q = x_i \ . \tag{3.72}$$

This is referred to as self-attention. The decoder contains both self-attention and cross-attention blocks. The latter construct their attention matrix from the output of the

previous attention block in the decoder x_i and the output of the encoder e_i,

$$x_i^Q = x_i \quad \text{and} \quad x_i^K = x_i^V = e_i \; . \tag{3.73}$$

3.4.2 Generative transformer

Transformers were introduced as a generative language model for translation [105]. The inputs to transformer encoder and decoder are words embedded into a vector space combined with a positional encoding. The original sequence is passed as the input of the transformer encoder while the decoder generates the translated sequence word by word, using the previously generated words as its input. Hence, the generative transformer is an example for an autoregressive generative architecture. While the original transformer was built to predict discretized tokens (words), it can be easily generalized to support continuous distributions. To build a non-conditional generative model, it is sufficient to use a series of self-attention blocks similar to the transformer encoder defined above. It can be easily replaced with a full transformer, including an encoder and decoder, to make the generation conditional on another sequence.

Generative transformers are built from three components. Firstly, we need to assume a form of the likelihood of a single sequence item x_i given some condition c_i, $p_\theta(x_i|c_i)$. Secondly, we need an embedding of the sequence items into vectors of size d where d is the internal number of features of the transformer, $e_\phi(x_i)$. Lastly, we have the transformer t_ξ itself with the sequence of embeddings shifted by one as its input and the conditions c_i as its output. In the most general case, all three components are neural networks with trainable parameters θ, ϕ and ξ. The full likelihood for a sequence $(x_1, \ldots, x_n)$ is then given by

$$\begin{aligned} p(x_1, \ldots, x_n) &= \prod_{i=1}^{n} p_\theta(x_i \mid c_i) \\ &= \prod_{i=1}^{n} p_\theta(x_i \mid t_\xi(e_\phi(x_0), \ldots, e_\phi(x_{i-1}))) \; . \end{aligned} \tag{3.74}$$

Here, x_0 acts as a starting token. Its specific definition depends on the type of sequence.

The generic structure of Eq. (3.74) allows for many different types of likelihoods to be used for the items in the sequence. In the case of a finite set of tokens, p_θ can be implemented as a fully-connected network with a categorical cross-entropy loss [105]. If the sequence consists of one-dimensional continuous features, a fully-connected network that predicts the parameters of a Gaussian Mixture Model can be used [64, 185]. To model more complex, multidimensional likelihoods, transformers can be combined with normalizing flows. We will discuss the latter variant of generative transformers in Sec. 6.5.2.

Chapter 4

Precision event generation

The research presented in this chapter is based on work in collaboration with Anja Butter, Ranit Das, Luigi Favaro, Sander Hummerich, Claudius Krause, Tobias Krebs, Tilman Plehn, Armand Rousselot, David Shih and Sophia Vent, and has been previously published in Ref. [1] and Ref. [4]. All tables and figures as well as parts of the text are similar or identical to the content of these articles. In particular, Secs. 4.1 and 4.3 are based on Ref. [1], Sec. 4.4 is based on Ref. [4].

Having introduced the basics of machine learning and LHC event generation, we can now combine the two. Often, ML-applications focus on one specific part of the event generation chain. We will discuss one such example in Ch. 5, where we accelerate the generation of hard scattering events with neural importance sampling. A different approach is to build a generative network surrogate for the entire LHC event generation chain that directly generates reconstructed events. This is also referred to as an end-to-end model. Various network architectures have been applied to this task, including generative adversarial networks (GANs) [61], variational autoencoders [63], diffusion models and generative transformers [64]. This work focuses on invertible neural networks, as they provide a good balance between high precision, fast and stable trainings, and fast generation of new samples. We first describe our benchmark dataset, leptonically decaying Z-bosons with a variable number of jets, in Sec. 4.1. We then present the architecture of our precision INN generator and discuss its performance as a surrogate for the full simulation chain in Sec. 4.2.

Compared to a more fine-grained approach where generative models replace smaller parts of the event generation chain, an end-to-end model has the potential to provide the largest speed-up. It skips various high-dimensional intermediate representations like the particle level or hits in calorimeter cells. It however also reduces the amount of control that we have, as the entire simulation chain is now distilled into a single neural network. A thorough understanding of the uncertainties involved with the training of such a network is therefore crucial. There are two key sources of uncertainties that we investigate in this work. The first one is caused by the limited amount of training statistics of the network. The second one is caused by the lack of expressiveness or the training of the ML model itself.

Generative models have been shown to amplify their trainings statistics [186, 187], i.e. they can be used to generate more events than they were trained on until the effective sample size becomes limited. This is because neural networks behave similar to parametric fits [179,188] and are powerful tools to smoothly approximate high-dimensional probability distributions. In practice, this means that we need to understand the uncertainties arising

from the limited training statistics. We discuss how these uncertainties can be extracted using Bayesian Neural Networks [111–114, 179, 189, 190] in Sec. 4.3.

Second, we show in Sec. 4.4 how classifiers discriminating between generated and truth data can be used both as a performance metric and diagnostic tool to find the failure modes of generative networks. When classifiers are used as a performance metric, the area under the ROC curve is often used to reduce the generator performance to a single number. We instead show how the local structures picked up by the classifier can help us to better understand the shortcomings of the generative model. Further, we demonstrate that the precision of the generated samples can be improved by reweighting them using the weights from the classifier [115]. In addition to reweighting and its use as a diagnostic tool, there are also ways to use the discriminator feedback to improve the generator directly, which are not the focus of this thesis. Examples include the DiscFlow method, where the discriminator feedback is used during the training of an INN [1], and latent space refinement [191, 192]. Lastly, we show how the error estimates from the classifier and Bayesian network are related.

4.1 Dataset

We choose the production of leptonically decaying Z-bosons with a variable number of jets as our benchmark process,

$$pp \to Z_{\mu\mu} + \{1, 2, 3\} \text{ jets} . \tag{4.1}$$

The events are simulated with Sherpa2.2.10 [35] at 13 TeV. We use CKKW merging [193] to get a merged sample with one to three hard jets. We include initial state radiation, parton shower and hadronization, but exclude detector effects. This is because detector effects lead to a smearing of sharp features in phase space. Not including them makes the generation task more challenging, and we expect that the performance of our method would be equal or better for reconstructed objects at the detector level. Moreover, there are detector-specific artifacts, for instance from the calorimeter geometry, that we neglect to focus on the underlying physics.

We use Fastjet3.3.4 [194] to reconstruct anti-k_T jets [195] with

$$p_{T,j} > p_{T,\min} = 20 \text{ GeV} \qquad \text{and} \qquad \Delta R_{jj} > R_{\min} = 0.4 . \tag{4.2}$$

The input of the jet algorithm comprises all particles at the hadronization-level, excluding neutrinos, the two muons from the Z-decay as well as photons radiated by these muons. Because of the variable number of hard jets and QCD effects like initial and final state radiation, there will be a variable number of jets. We sort the jets by p_T and include up to three jets in our dataset. The resulting dataset contains 5.4M events in total, with 4.0M one-jet events, 1.1M two-jet events, and 300k three-jet events. It is further split up into a training and a test dataset of equal sizes. The momenta of the two muons have three degrees of freedom, while the invariant mass is an additional degree of freedom for the jets. Consequently, the phase-space dimensions for one-, two- and three-jet events are 10, 14 and 18.

This process has two features that are difficult to learn for generative networks. The Z-boson decaying into two muons causes a sharp resonance in the combined invariant mass of the two muons, $M_{\mu\mu}$. For a network trained on the momenta of the two muons,

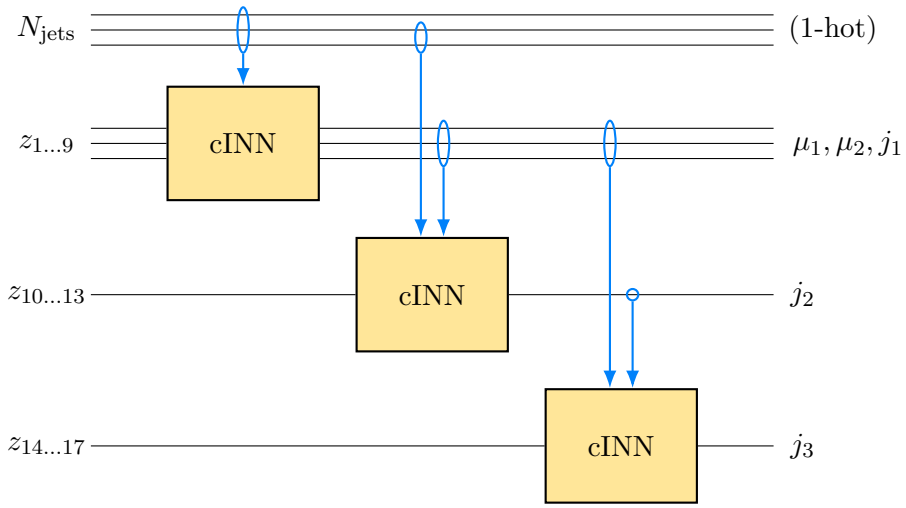


Figure 4.1: Generative flow architecture for events with two muons and one to three jets. The INNs relate the latent space (left) to the physical phase space (right), conditioned on the one-hot encoded jet multiplicity.

this mass peak is a complex correlation between multiple input features, making it challenging to learn. Secondly, for events with more than one jet, QCD effects cause an enhancement of the rate when the two jets are close to collinear. At the same time, there is a cutoff in the angular separation ΔR_{jj} imposed in the jet algorithm. This again leads to a very sharp feature that is hidden in the correlation of the momenta of two jets. Furthermore, the cutoff leads to a hole in phase space which can cause problems with generative architectures like normalizing flows that preserve the topology of their latent space distribution [191].

4.2 INN generator

4.2.1 Network architecture

We use an INN with the architecture and loss function described in Sec. 3.2.2. It is implemented in PyTorch [196] with the Adam optimizer [164], and a one-cycle learning-rate scheduler [197]. The coupling blocks are based on cubic spline transformations [171]. The network uses a Gaussian latent space and we ensure that almost all points stay within the bounds of the spline transformations by setting the limits of the transformations to $[-10, 10]$. Values outside of the bounds are mapped onto themselves. After each coupling block, a random, but fixed rotation matrix is applied to the features, so that correlations between all input variables can be modeled.

Invertible neural networks without any modifications are restricted to datasets with a fixed input dimension. The simplest way to account for a variable jet multiplicity would be to train a separate generative network for every multiplicity. This approach has several disadvantages. There are structures that are similar for all different multiplicities like the decay of the Z into two muons, or basic QCD patterns. These structures would have to be relearned by every generator. As the number of jets increases, the number of training events gets smaller while the size of the generator networks gets larger, making the training more unstable. A better way to account for the variable multiplicity is to train

a common network that generates the momenta of the two muons and the first jet, as these are present in all events in our training data. Then a smaller network is trained for each additional jet to generate the jet momentum conditioned on all previously generated momenta. This makes the architecture autoregressive at the level of particles. This chain of conditional generators is illustrated in Fig. 4.1. As the first two network are trained on mixed multiplicities, the information about the multiplicity is passed to the network as an additional condition in one-hot encoded form. Moreover, the multiplicities are balanced during training, such that every training batch contains events with the three different multiplicities in equal proportions. All three networks are trained separately, and they are combined to generate samples. We list the hyperparameters for our networks in Tab. A.1.

Preprocessing

The choice of an appropriate preprocessing is crucial for a successful network training. This is especially true for generative models because sharp features are difficult to model from a smooth latent distribution. For INNs with a Gaussian latent space, the input data should be preprocessed in a way that brings it as close to a Gaussian as possible. Each muon or jet is represented by

$$\{ p_T, \eta, \phi, m \} , \tag{4.3}$$

where the mass is only present for jets as the muons are on the mass-shell. The p_T distribution has a sharp cutoff at $p_{T,\text{min}}$ from the cuts imposed during reconstruction. It can be turned into an approximately Gaussian distribution with

$$\tilde{p}_T = \log(p_T - p_{T,\text{min}}) . \tag{4.4}$$

The phase space distribution is invariant under rotations around the beam axis, so we can remove a degree of freedom by training the networks on azimuthal angles relative to the muon with the largest transverse momentum. We take into account the periodicity of the azimuthal angle, resulting in the range $\Delta\phi \in [-\pi, \pi]$. We transform these angle differences into a distribution close to a Gaussian using

$$\widetilde{\Delta\phi} = \operatorname{arctanh}\left(\frac{\Delta\phi}{\pi}\right) . \tag{4.5}$$

We then centralize and normalize the phase-space variables q_i as

$$\tilde{q}_i = \frac{q_i - \langle q_i \rangle}{\sigma(q_i)} . \tag{4.6}$$

In Ref. [1], we then used a multiplicity-wise PCA/whitening transformation. We later found that this preprocessing led to hard to learn correlations between different multiplicities and removing it improves the network performance. We implemented these changes in Ref. [4]. In the following, we will refer to the model using the original preprocessing as "standard generator" and to the improved version as "state-of-the-art generator".

Magic transformation

While preprocessing is effective in removing some challenging features like the hard cut in the p_T distribution, it is not always possible to find a simple mapping that addresses such challenges. One of the major challenges of the $Z+$ jets final state is the the correlation between $\Delta\phi$ and $\Delta\eta$ of the jets. We illustrate this for the exclusive 2-jet final state in Fig. 4.2. In most events, the two jets have a back-to-back topology, but there are events where the two jets recoil against the Z. Collinear enhancement leads to a local maximum close to the cutoff from the jet algorithm at $\Delta R_{jj} > 0.4$. This sharp feature is difficult for the network to learn for several reasons. There is a low number of training events in this phase-space region, compared to events with a back-to-back topology. Further, to correctly model this feature, the network has to learn the correct correlation between several input variables. As a consequence, the network smoothly interpolates through this ring-hole structure. While the main difficulty is the lack of precision of the network, the non-trivial topology of the phase-space distribution might give the network further incentive to smoothly interpolate through the hole in phase space [191].

We can exploit the tendency of neural networks to interpolate through sharp structures by defining a weighting function that smooths the sharply peaked distribution into a distribution that is easier to learn for the network. In the case of the ring-hole structure, the complicated shape of the distribution is the most apparent in the observable ΔR_{jj}. Therefore, we reweight the training data with a simple linear smoothing function,

$$f(\Delta R) = \begin{cases} 0 & \text{for } \Delta R < R_- \\ \dfrac{\Delta R - R_-}{R_+ - R_-} & \text{for } \Delta R \in [R_-, R_+] \\ 1 & \text{for } \Delta R > R_+ \,, \end{cases} \tag{4.7}$$

where the lower cutoff $R_- < R_{\text{min}} = 0.4$ ensures non vanishing weights for all training events, and the upper boundary R_+ is in a phase-space region without complicated features. We use the values $R_- = 0.2$ and $R_+ = 1.5$. The same procedure can be easily applied to events with more than two jets and multiple critical ΔR_{jj}-distributions by

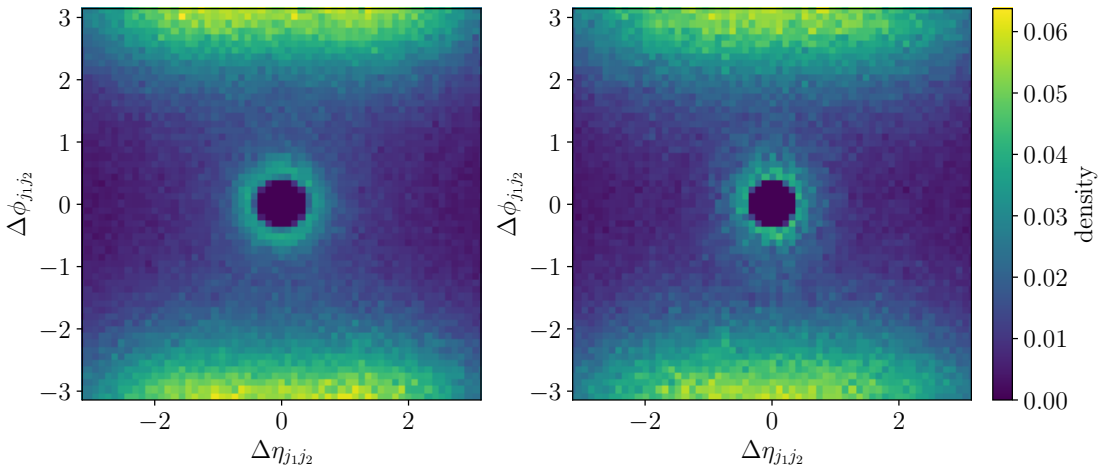


Figure 4.2: Correlations of the difference in pseudorapidity and azimuthal angle between jets for events with two jets. We show truth (left) and INN-generated events (right).

defining the weights

$$\begin{aligned} w^{(\text{1-jet})} &= 1 \\ w^{(\text{2-jet})} &= f(\Delta R_{j_1,j_2}) \\ w^{(\text{3-jet})} &= f(\Delta R_{j_1,j_2}) f(\Delta R_{j_2,j_3}) f(\Delta R_{j_1,j_3}) \ . \end{aligned}$$

The ΔR_{jj}-distributions for $Z + 2$-jet events before and after reweighting are shown in the left panel of Fig. 4.3. The reweighted distribution no longer has a local maximum at $\Delta R_{jj} \approx 0.6$ and is less steep close to $R_{\text{min}} = 0.4$. The INN is trained on weighted events using the weighted version of the standard INN loss from Eq. (3.36) with a batch-wise normalization of the weights from Eq. (4.8),

$$\mathcal{L}_G = \frac{1}{B} \sum_{i=1}^{B} \frac{w(x_i)}{\sum_{j=1}^{B} w(x_j)} \log g_\phi(x_i|c_i) \ , \tag{4.8}$$

where B is the batch size. The magic transformation has to be inverted for events generated by an INN trained on reweighted training data. The inverse weights are

$$\bar{f}(\Delta R) = \begin{cases} 0 & \text{for } \Delta R < R_{\text{min}} \\ \dfrac{R_+ - R_-}{\Delta R - R_-} & \text{for } \Delta R \in [R_{\text{min}}, R_+] \\ 1 & \text{for } \Delta R > R_+ \end{cases} \ , \tag{4.9}$$

where we set the weights of events with $\Delta R < R_{\text{min}}$ to zero to enforce the jet separation. We show the $\Delta\phi$ and $\Delta\eta$ correlation of events generated using the magic transformation in the right panel of Fig. 4.2. It can be seen that the network has captured the ring-hole structure. The main disadvantage of the magic transformation is that the resulting events are now weighted. We show the weight distribution of the generated events in the right panel of Fig. 4.3. Most events have weights between one and seven, so an unweighted sample could be obtained efficiently through rejection sampling as described in Sec. 2.2.5. The advantages of the method are its versatility, as it could be easily applied to other localized structures that are difficult to learn for a generative network, and that it is complementary to other approaches like improving phase-space mappings or latent space

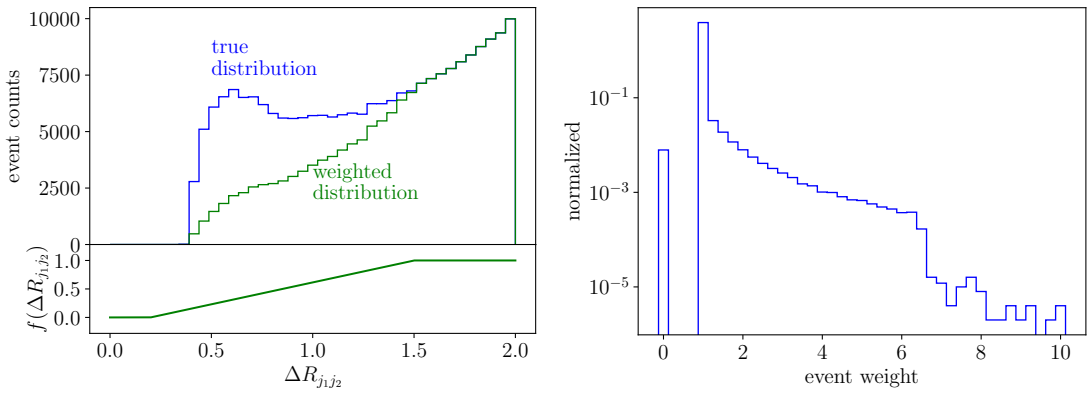


Figure 4.3: Left: $\Delta R_{j_1 j_2}$-distribution for $Z + 2$ jets events before and after the transformation of Eq. (4.8). Right: histogram of the weights of the generated events.

refinement. Note that the magic transformation is different from the standard approach used in Monte Carlo integration, where phase-space mappings are used to remove the leading features of a distribution and an adaptive model is used to learn a small but non-trivial difference. Instead, our approach relies on the tendency of INNs to smoothly interpolate, as their training behaves similarly to a fit [179].

4.2.2 Results

We train an INN generator with the setup, preprocessing and magic transformation described above. We then use it to generate samples. In Fig. 4.4, we show histograms of

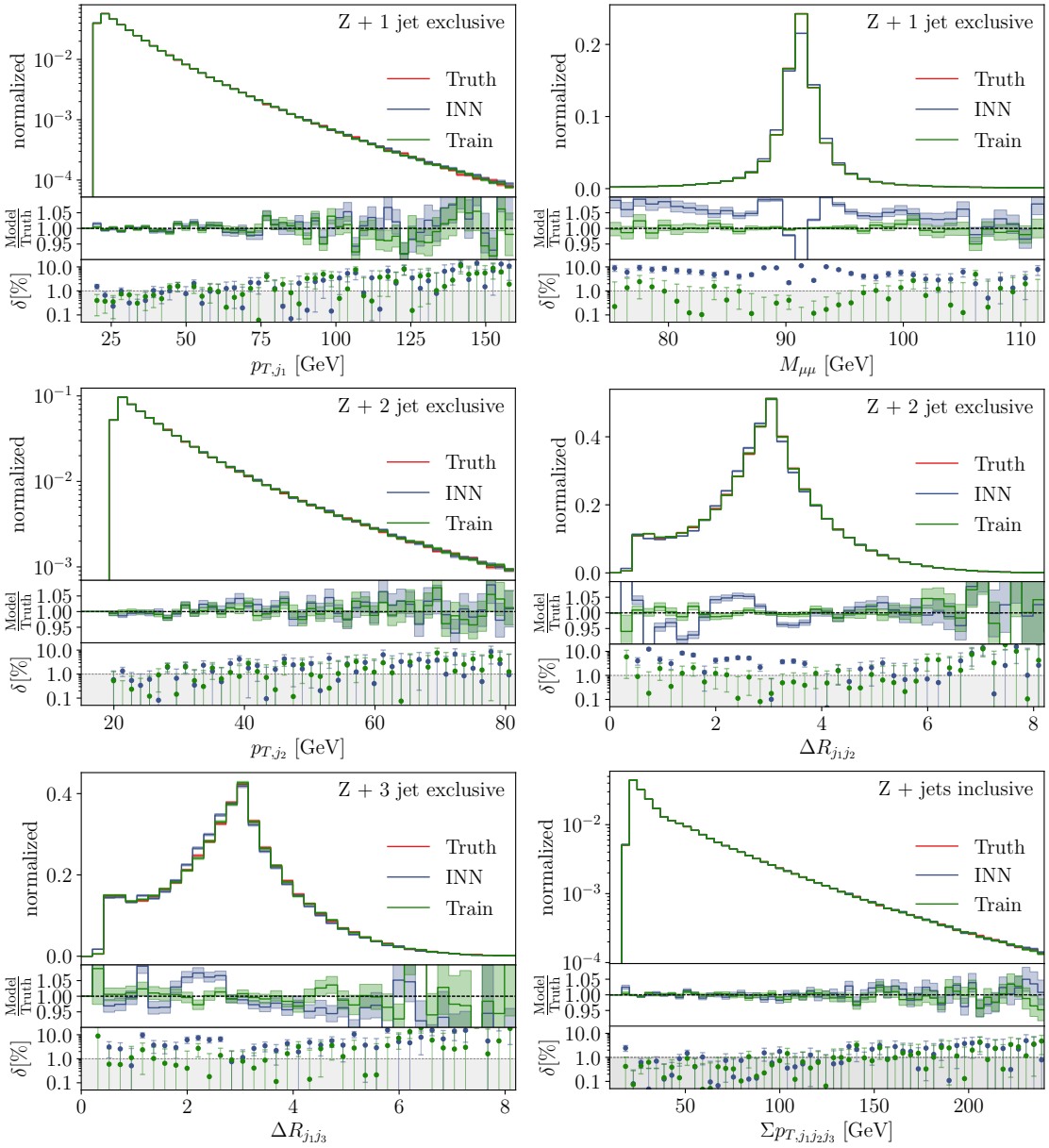


Figure 4.4: INN distributions for $Z+1$ jet (upper), $Z+2$ jets (middle), $Z+3$ jets (lower left) and an inclusive distribution (lower right) from a combined $Z+$jets generation. We show weighted events using the magic transformation of Eq. (4.8) to improve the ΔR distributions.

kinematic distributions for our training data ("Train"), generated data ("INN") and test data ("Truth"). We show exclusive distributions for $Z + \{1, 2, 3\}$ jets samples as well as one inclusive distribution. Each histogram also contains the relative deviation between the generated and truth data, expressed as a ratio and as the percentage of deviation,

$$\delta[\%] = 100 \, \frac{|\text{Model} - \text{Truth}|}{\text{Truth}} \, . \tag{4.10}$$

As seen in the top row, the p_T of the jets is learned with high precision, and the network is able to extract the challenging peak in the intermediate Z mass with only a small amount of smearing. In the second row, we can see distributions for events with two jets. Again, the p_T distribution of the second jet is learned with high precision, comparable to that of the first jet. For two-jet events, the network now also has to learn the structure of $\Delta R_{j_1 j_2}$. With the help of the magic transformation, the network is able to map out this structure well. This is still true after adding a third jet in the final state. Lastly, we also show an inclusive distribution, the scalar sum of all jet p_T. Like the exclusive p_T distributions, it is in excellent agreement with the truth distribution.

We find that the network agrees with the truth data at the percent level or better in the bulk of the distributions of many kinematic observables, like in the p_T distribution shown in Fig. 4.4. It is close or only slightly worse than the agreement between the truth data and the statistically independent training data. In the tails of the distributions, there are larger fluctuations. These are however also present in the training data due to the limited size of the training dataset. For more complicated observables like $M_{\mu\mu}$ and ΔR_{jj}, we find a lower precision between one and ten percent in the bulk.

4.3 Uncertainties from Bayesian networks

We can extend the INN from Sec. 4.2 to a Bayesian INN as introduced in Sec. 3.3.2. To this end, we train a network with the same architecture as in Sec. 4.2. We adjust the hyperparameters as given in Tab. A.1. The result of the training can be understood as the uncertainties of the phase-space density arising from the limiting amount of training data. These uncertainties can be evaluated in the forward direction where for a given point in phase space, multiple samples from the distribution of possible densities can be drawn, or in the inverse direction, resulting in an uncertainty of the generated phase-space distributions. In this section, we focus on the latter interpretation. We extract the uncertainties by repeatedly sampling a set of events for different network replicas drawn from the the Bayesian network posterior distribution. For a given observable and each set of events, a histogram is created. The bin-wise means and standard deviations of these histograms then define the error bands for the observable.

We show histograms for different phase-space observables including the BINN error bars in Fig. 4.5. In the p_T distribution of the first jet in the upper left panel, we can see that the deviation between the generated and true distribution is covered by the error bands from the Bayesian network. These are especially large in the tail of the distribution, indicating that estimating the density in this region of phase space is harder due to the lower amount of training statistics. In contrast, we can see that the deviation from the truth distribution is not covered by the uncertainty bands for the smeared peak from the intermediate Z resonance in the upper right panel. This highlights that BNNs only find the uncertainties from a lack of training statistics, but not from a sub-optimal training procedure or a lack of network expressivity. In the central left panel, we show

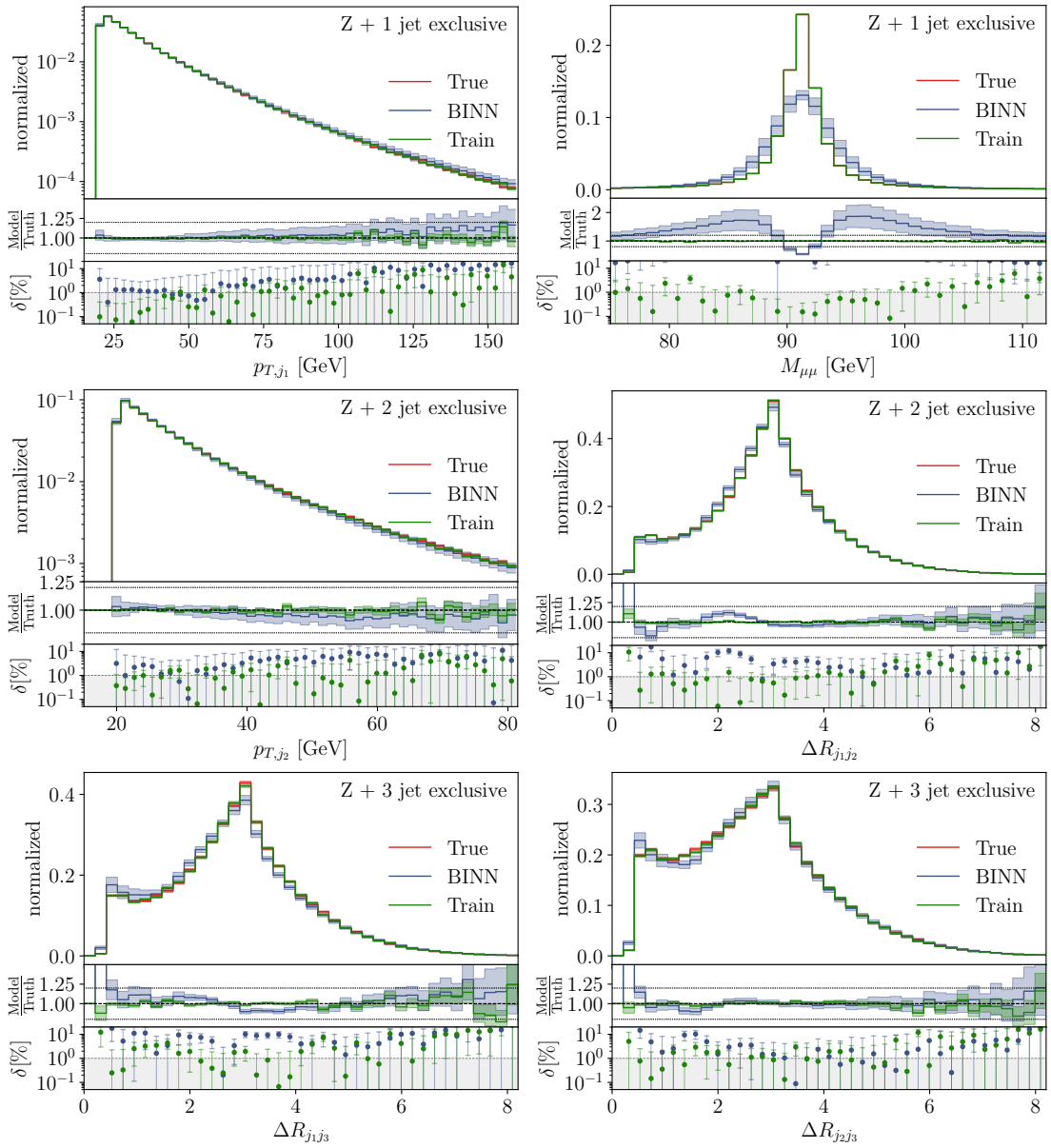


Figure 4.5: BINN densities and uncertainties for $Z + 1$ jet (upper), $Z + 2$ jets (middle), and $Z + 3$ jets (lower) from a combined $Z +$ jets generation. The architecture and training data correspond to the deterministic network results shown in Fig. 4.4, including the magic transformation of Eq. (4.8).

the p_T distribution of the second jet for $Z + 2$-jet events. The slightly larger error bars in the bulk are caused by the lower number of training events with two or more jets. Finally, the ΔR_{jj} distributions for events with two or three jets are still learned well thanks to the magic transformation and deviations from the truth distribution are mostly covered by the Bayesian network uncertainties. Overall, there is some degradation of the performance compared to the deterministic network. This is caused by the more difficult training of a BNN compared to a deterministic network because of a larger number of trainable parameters and noise from the stochastic network weights. An improved version of the Bayesian generator, similar to the improved version of the deterministic generator will be discussed in Sec. 4.4.3.

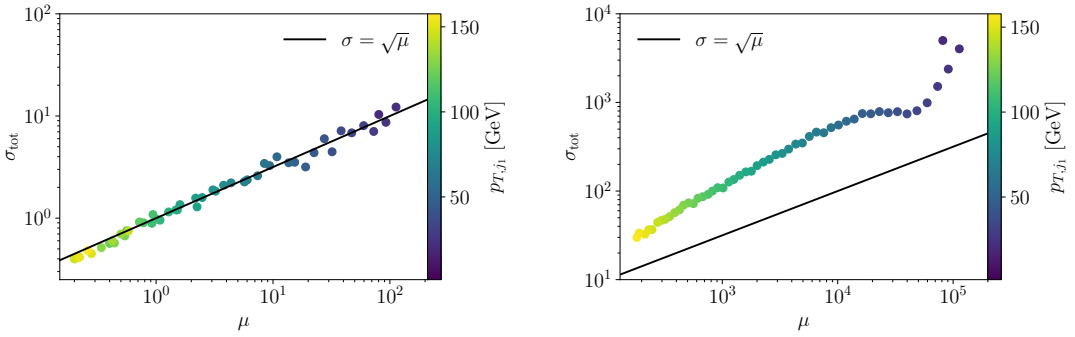


Figure 4.6: Correlation between event count and BINN uncertainty for 1000 (left) and 1M (right) generated events. The diagonal line shows the Poissonian scaling for a statistically limited sample.

Decomposition of bin-wise uncertainties

Bin-wise means and standard deviations are a simple way to visualize the uncertainties found by a BINN. The resulting error bars do however not only contain the uncertainty of the distribution itself, but also the Poissonian noise from sampling. We can decompose the error bars into these two components. In the following, we will look at the mean μ and standard deviation σ of the number of events in a single histogram bin. The mean is given by

$$\mu \equiv \langle n \rangle = \sum_n n P_N(n) \,, \tag{4.11}$$

with the Poissonian probability $P_N(n)$ to observe n events in that bin. In the case of a generative neural network, $P_N(n)$ depends on the the probability to generate events in that bin, which in turn depends on the network's trainable parameters θ. Consequently, it is a conditional distribution $P_N(n|\theta)$. As we are also averaging over the distribution of network parameters by making several histograms with different sampled θ, the mean from Eq. (4.11) becomes

$$\langle n \rangle = \int d\theta \; q(\theta) \sum_n n P_N(n|\theta) \equiv \int d\theta \; q(\theta) \; \langle n \rangle_\theta \,. \tag{4.12}$$

We can now compute the corresponding standard deviation, and decompose it into two terms,

$$\begin{aligned} \sigma_{\text{tot}}^2 &= \langle (n - \langle n \rangle)^2 \rangle \\ &= \int d\theta \; q(\theta) \left[\langle n^2 \rangle_\theta - 2 \langle n \rangle_\theta \langle n \rangle + \langle n \rangle^2 \right] \\ &= \int d\theta \; q(\theta) \left[\langle n^2 \rangle_\theta - \langle n \rangle_\theta^2 + (\langle n \rangle_\theta - \langle n \rangle)^2 \right] \equiv \sigma_{\text{stoch}}^2 + \sigma_{\text{pred}}^2 \,. \end{aligned} \tag{4.13}$$

following the argument given in Ref. [190]. The first term is the variance of the Poissonian noise,

$$\sigma_{\text{stoch}}^2 = \int d\theta \; q(\theta) \left[\langle n^2 \rangle_\theta - \langle n \rangle_\theta^2 \right] = \langle n \rangle \,. \tag{4.14}$$

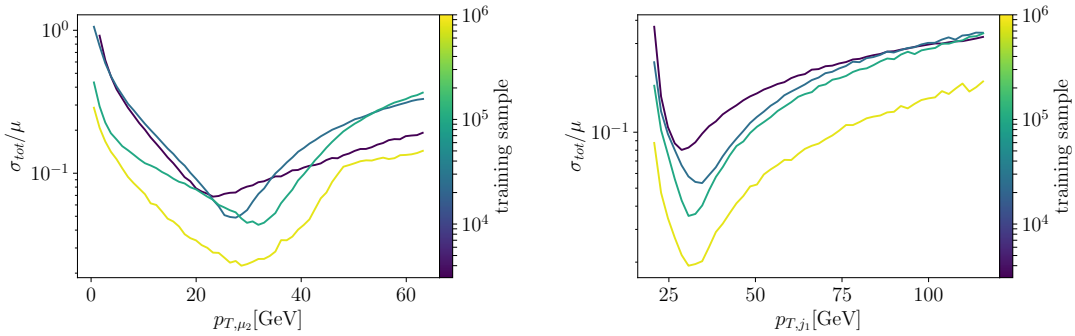


Figure 4.7: Relative uncertainty from the BINN for the $Z + 1$ jet sample, as a function of the size of the training sample.

The second term is the variance of the network prediction due to the limited number of training points, extracted by the Bayesian network,

$$\sigma^2_{\text{pred}} = \int d\theta \, q(\theta) \left(\langle n \rangle_\theta - \langle n \rangle \right)^2 \, . \tag{4.15}$$

Note that this decomposition of the uncertainties is specific to the histogram-based approach. A strategy to define uncertainties as a continuous function of phase space by using the network for density estimation will be discussed in Sec. 4.4.3.

We demonstrate this decomposition of the uncertainties for a histogram of the $p_{T,j}$-distribution for $Z + 1$ jet events in Fig. 4.6. Each dot in the figure denotes the mean and standard deviation of the events count for one of the 60 histogram bins. The black line shows the Poisson scaling $\sigma \propto \sqrt{\mu}$, which is the minimum of the total uncertainty. We show this plot for a histogram of 1000 and 1M events. Neglecting statistical fluctuations, the point follow the Poisson line in the low statistics case. That means that σ_{stoch} is the dominant contribution to the total uncertainty. In the high statistics case, the predictive uncertainty of the BINN, σ_{pred}, becomes more important.

Effect of training statistics

As the purpose of a Bayesian network is to capture the uncertainties caused by the limited amount of training data, we can test the BINN by varying the size of the training dataset. We train the network with dataset sizes between 3072 and 2.7M and show the relative uncertainty of two p_T distributions for one-jet events in Fig. 4.7. It can be seen that the uncertainty is reduced for trainings on larger dataset compared to smaller sizes. The uncertainty of p_{T,μ_1} is no longer estimated correctly in the training with the smallest dataset. This shows that uncertainty estimate becomes unreliable in situations when it becomes hard for the network to learn the phase-space density due to a very low amount of training data.

4.4 Classifier metric

So far, we have discussed how specific challenges and failure modes of a generative network, like washed out sharp features, wrongly learned phase-space boundaries and

underpopulated kinematic tails, can be addressed through problem-specific measures such as preprocessing, reweighting of the training data or changes to the architecture. These problems are typically identified by constructing high-level observables and looking at their distributions. However, problematic features can sometimes be hidden in complex correlations. In this section, we will show how classifier networks can be used to identify such failure modes in a systematic way, making them a powerful quality metric for generative models. Moreover, the classifier feedback can be used to refine the generated events through reweighting.

According to the Neyman-Pearson lemma, the likelihood ratio is the optimal discriminant between two different hypotheses. As explained in Sec. 3.1.3, the training objective of a binary classifier trained with a binary cross-entropy loss is to extract the likelihood ratio from the training data. An optimal classifier trained on samples from a generative network (label $C = 0$) and samples from the truth distribution (label $C = 1$) can therefore extract the reweighting factor between these two distributions. Given the classifier output $C(x)$ as a function of phase space, these weights can be written as

$$w(x) = \frac{p_{\text{data}}(x)}{p_{\text{model}}(x)} = \frac{C(x)}{1 - C(x)} \,. \tag{4.16}$$

For a good generative model the distribution of these weights will be peaked close to one. Phase-space regions where the generator is overestimating the density lead to tails on the left side of the weight distribution ($w \ll 1$) and regions where it is overestimating will lead to tails on the right side of the distribution ($w \gg 1$). For instance, a missed tail will result in large weights, whereas in a case where the network interpolates through a phase-space boundary there will be weights close to zero. A smoothed out peak leads to low weights $w < 1$ in phase-space regions close to the peak, and to weights $w > 1$ further away from the peak.

To be able to use a classifier as a tool for reweighting, it has to reach a higher precision than the generative network to which it is applied. As demonstrated above, INNs applied to LHC event generation reach percent-level accuracy. GANs are a different example for generative networks that have been applied to similar data. They reach accuracies around 10% [61]. Regression networks, which are conceptually more similar to classifiers, have been shown to reach per-mille level accuracies when they were applied to amplitude regression [55]. The reason for this difference in performance is that generative networks typically are more constrained or require a special mode of training. For example, INNs

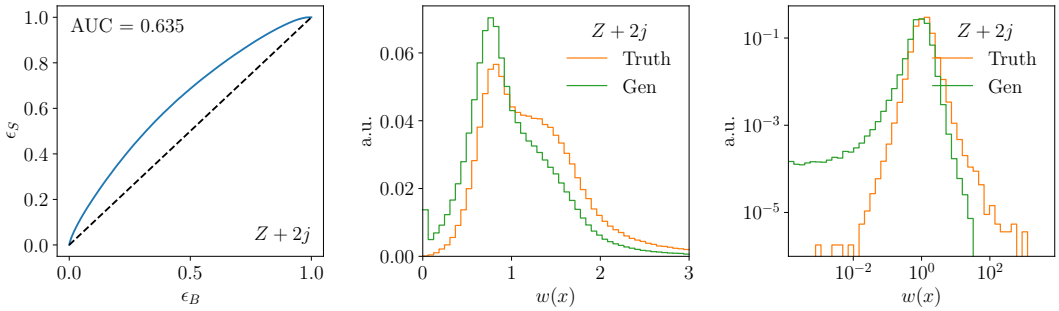


Figure 4.8: Left to right: ROC curve, weight distribution on a linear scale, and weight distribution on a logarithmic scale for $Z + 2$ jets events, using the standard generator. The weights are evaluated separately on the truth dataset and the generated dataset.

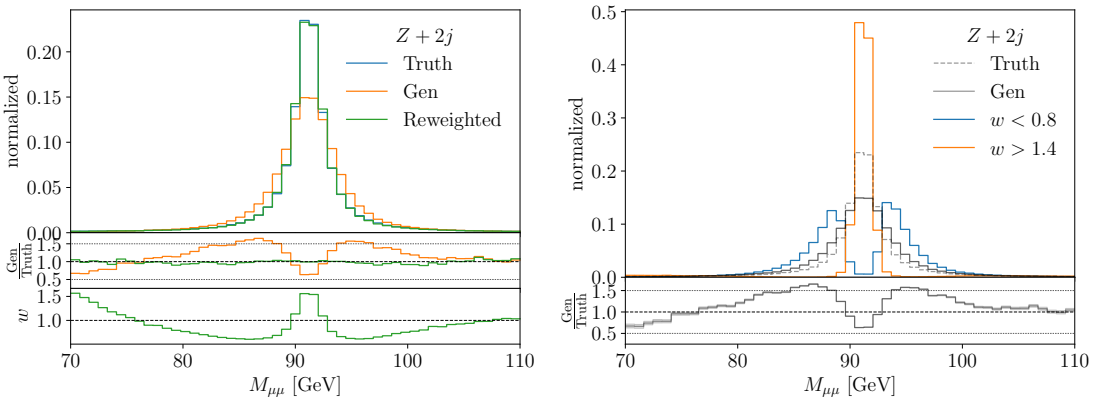


Figure 4.9: Z-peak distributions for $Z+2$ jets events from the standard generator. We show the agreement between the generated events with the truth data (left) and events in different weight ranges (right). The events with small weights are taken from the generated distribution, the events with large weights are taken from the truth distribution.

have a constrained architecture to make them invertible with a tractable Jacobian. GANs need to find an equilibrium between two networks during the training, and models based on flow matching learn a velocity field instead of directly estimating the density.

We train a classifier on INN-generated events with the same parameterization as for the INN training. Because classifiers do not have an invertibility constraint, we can add further training observables that correspond to known failure modes. This enables the classifier to extract the weights with higher accuracy. In this case, we know that the peak in the $M_{\mu\mu}$ correlation and the ΔR_{jj} distributions are challenging to learn, so we append these features to the classifier inputs. The input variables are then given by

$$\Big\{p_{T,i}, \eta_i, \Delta\phi_{1,i}, M_i\Big\} \cup \Big\{M_{\mu\mu}\Big\} \cup \Big\{\Delta R_{j_1,j_2}\Big\} \cup \Big\{\Delta R_{j_2,j_3}, \Delta R_{j_1,j_3}\Big\} \,, \tag{4.17}$$

with no $\Delta\phi$ input for the first particle because of the symmetry in the azimuthal angle and no mass inputs for the muons. We take the inverse of ΔR and apply a cutoff as preprocessing steps to help the classifier focus on small values of ΔR. The hyperparameters of our classifier networks are given in Tab. A.2. We train classifiers for both the standard generator discussed above as well as the state-of-the-art generator with improved preprocessing to highlight the impact of the changes on the classifier weight distribution. In both cases, we do not use the magic transformation to investigate the performance of the classifier near the ΔR cutoff.

4.4.1 Standard generator and mass peak

We first train a classifier on events generated with the old network architecture and show the resulting ROC curves and weight distributions in Fig. 4.8. As discussed in the previous sections, one of the main difficulties of this generator is to reproduce the Z mass peak in the $M_{\mu\mu}$ distribution, especially for events with two or three jets. We demonstrate this for the example of $Z + 2$ jets in the left panel of Fig. 4.9. It can be seen that the network correctly extracted the position of the mass peak, but the width is larger than in the truth dataset with relative deviations up to 50% between the learned and truth distributions. This failure mode can be interpreted as a smearing of the mass

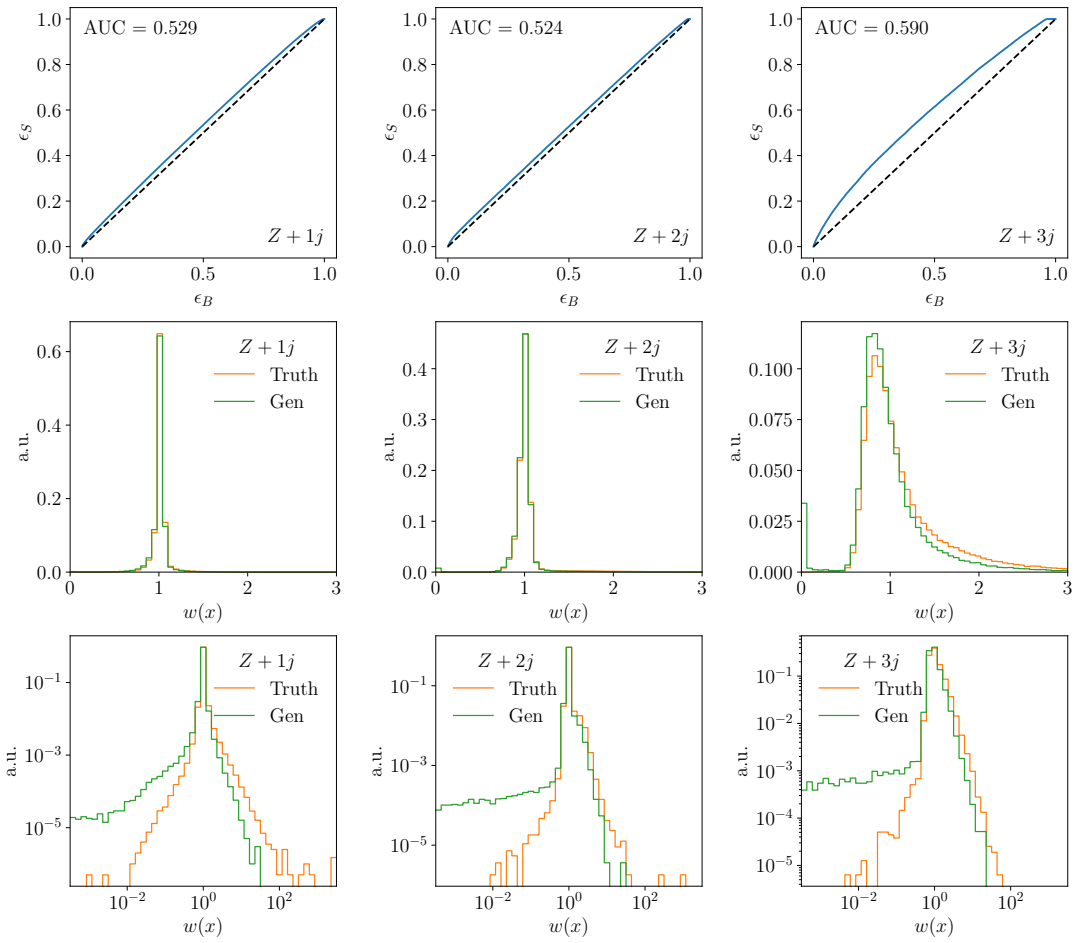


Figure 4.10: Left to right: $Z + \{1, 2, 3\}$ jets using the state-of-the-art generator. Top to bottom: ROC curve, weight distribution on a linear scale, and weight distribution on a logarithmic scale. The weights are evaluated separately on the truth dataset and the generated dataset.

distribution due to a lack of precision of the generative network. This smearing means that the phase space close to the peak is underpopulated, and it is overpopulated on the sides of the peak. The green line is a histogram of the distribution after reweighting the generated samples with the weights learned by the classifier. The marginalized weights are shown in the lower sub-panel. The smeared mass peak leads to weights up to 1.5 at M_Z and weights in the range $w = 0.6 \ldots 0.8$ on the shoulders of the peak. After the reweighting, the distributions match at the percent level, indicating that the classifier has successfully extracted the difference between the generated and truth samples. This leads to a characteristic structure in the weight distribution, best seen in the middle panel of Fig. 4.8. Instead of peaking close to $w = 1$, the weight distribution is shifted to $w < 1$ because of the overpopulated shoulders of the $M_{\mu\mu}$ distribution. Due to the underpopulated peak, a second shoulder appears in the weight distribution around $w \approx 1.5$. To confirm this, we show the distribution of $M_{\mu\mu}$ for events with large and small weights in the right panel of Fig. 4.9.

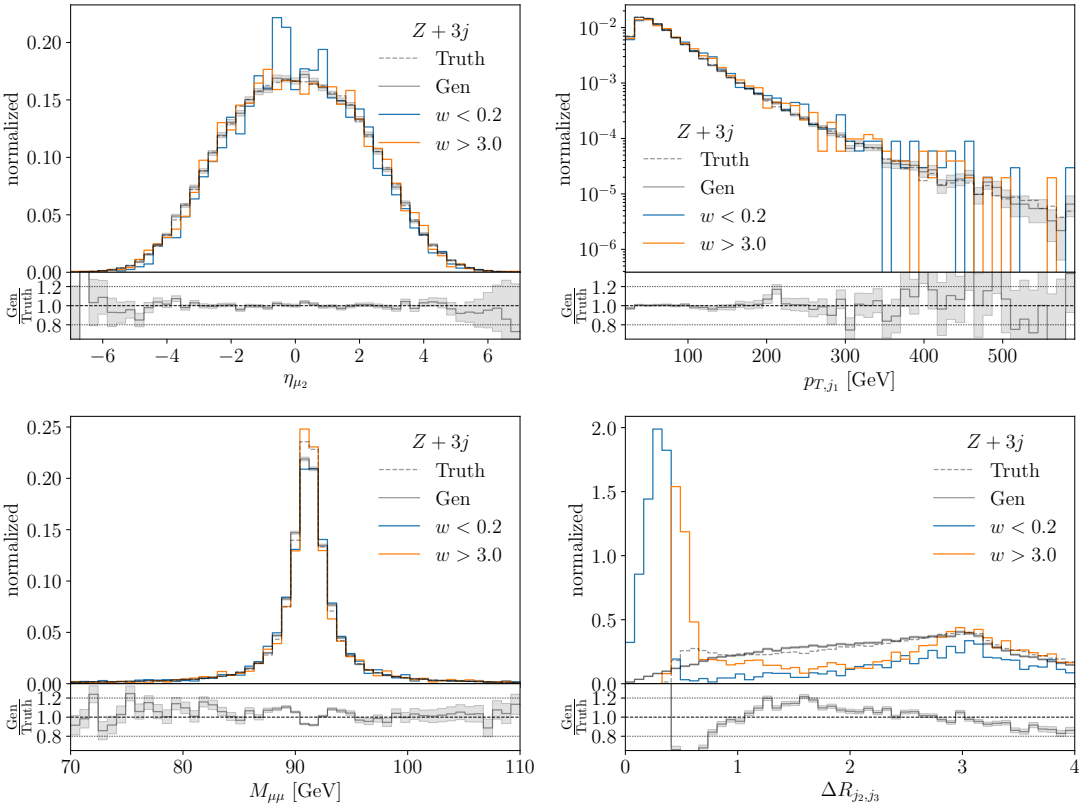


Figure 4.11: Kinematic distributions for $Z+3$ jets events from the state-of-the-art generator in different weight ranges, to see if events with large corrections cluster in phase space. The events with small weights are taken from the generated distribution, the events with large weights are taken from the truth distribution.

4.4.2 State-of-the-art generator and feature scan

Next, we look at the state-of-the-art version of the Z+jets event generator, for which we show the weight distributions in Fig. 4.10. The modeling of the mass peak is significantly improved in the updated version, so this failure mode is no longer the dominant contribution to the classifier weight distribution. Overall, the central peaks of the weight distribution are much narrower compared to the distributions shown in Fig. 4.8. The width of these peaks is similar for one- and two-jet events. There are however still tails of the weight distributions that hint at further failure modes of the network. The tail towards small weights is best evaluated for generated events, as the corresponding phase-space regions will be more populated compared to the training events. Similarly, the tail towards large weights is best studied using training events.

We search for clustering in phase space by looking at the phase-space distributions of events with anomalous weights. Because the weight distribution for $Z + 3$ jets has the most sizeable tails, both towards small and large weights, we show the clustering plots for these events in Fig. 4.11. η_{μ_2} and p_{T,j_1} are examples for distributions where almost no clustering in phase space is visible. Only in the tail of the p_T-distribution, there are some fluctuations caused by low statistics. This is also true for the distribution of $M_{\mu\mu}$, as there is only very little smearing compared to the truth distribution. In contrast, there is strong clustering visible for the observable ΔR_{jj} due to the mismodeling of the collinear enhancement and cutoff. Events with $\Delta R_{jj} < 0.4$ only appear in the generated

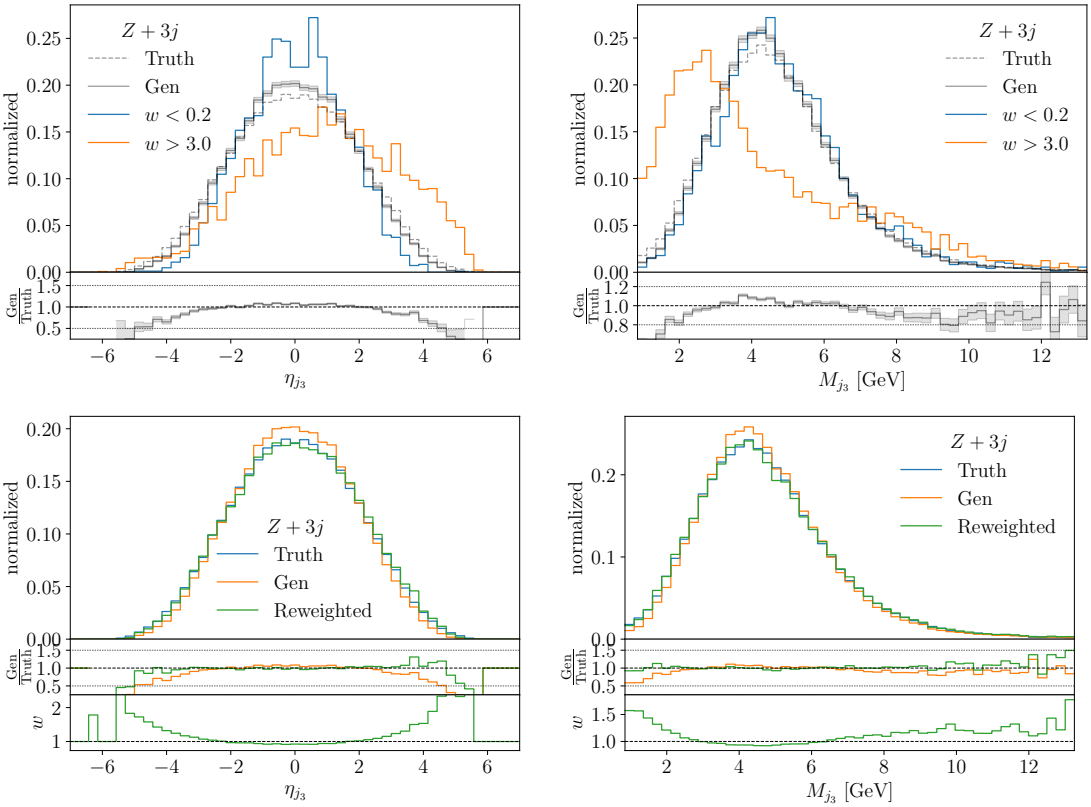


Figure 4.12: Critical kinematic distributions for $Z + 3$ jets events from the state-of-the-art generator in different weight ranges (upper panels) and comparing generated data with truth (lower panels). The events with small weights are taken from the generated distribution, the events with large weights are taken from the truth distribution.

dataset and have weights close to 0. This can be confirmed by looking at the events in the leftmost bin in the weight histograms for three-jet events in Fig. 4.10, corresponding to weights $0 < w < 0.06$. Indeed, 95% of the events in this bin have at $\Delta R_{jj} < 0.4$ for one pair of jets.

The classifier weights can also be used to identify unexpected failure modes that are not clearly visible by just plotting histograms of the generated events. Two examples for observables where this is the case are shown in Fig. 4.12. The upper panels show a strong clustering of events with large weights in one tail of the distribution. The reweighted distributions in the lower panels confirms that the classifier extracts the correct weights.

4.4.3 Classifier metric for Bayesian generators

As discussed in Sec. 3.3, INNs can be trained as a Bayesian network to extract the uncertainty of the network parameters arising from limited availability of training statistics. This uncertainty can be propagated to find the uncertainty of the predicted phase-space density. Similarly, we can understand classifiers as a tool to find the deviation of the predicted and true phase-space density. We can compare these two types of uncertainty

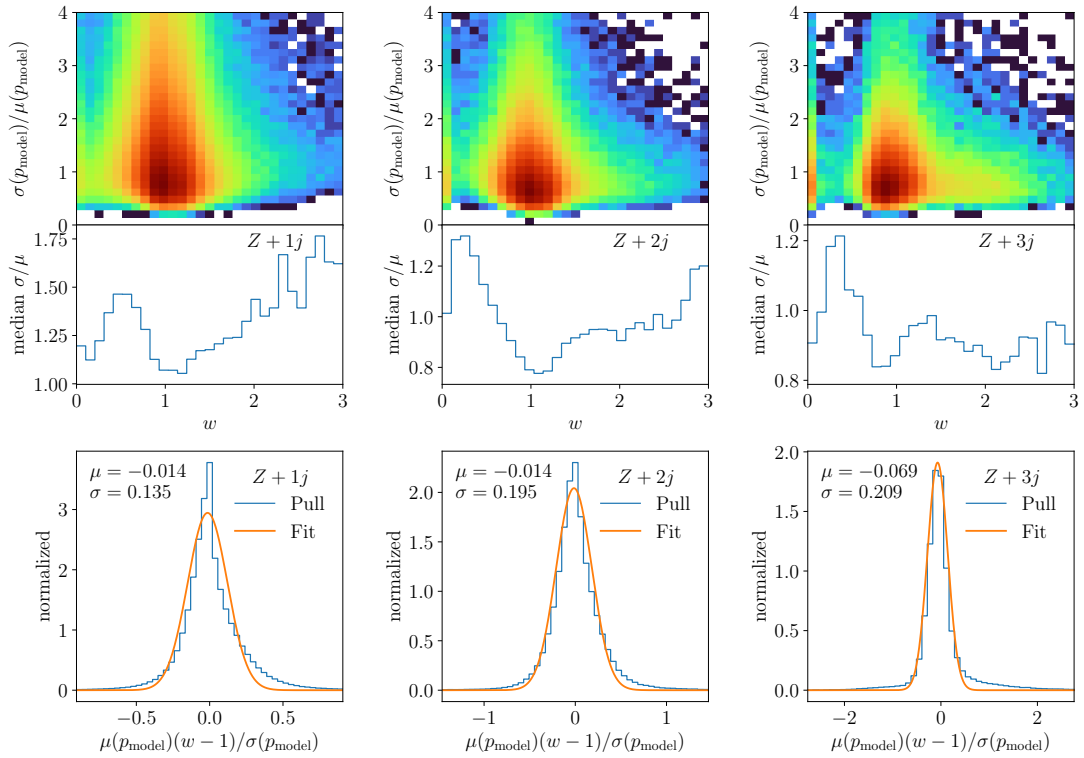


Figure 4.13: Top: correlation between the classifier weights and the relative standard deviation of the event weights from the Bayesian generator. Center: medians over the w-bins. Bottom: pulls combining the standard deviation of the event weight distribution with the error estimate from the Bayesian generator.

estimate by defining the mixed ratio

$$t(x_i) = \frac{\mu(x_i)[1 - w(x_i)]}{\sigma(x_i)} \,, \tag{4.18}$$

where $\mu(x_i)$ and $\sigma(x_i)$ are the mean and standard deviation of the estimated density of a sample x_i, and $w(x_i)$ is the classifier weight.

We train a Bayesian version of the state-of-the-art generator, again without the use of the magic transformation. As a further improvement to the Bayesian training used in Sec. 4.3, we choose a smaller initial standard deviation of the network parameters. This results in a more stable training and an improvement in the precision of the learned densities. To get the uncertainty estimate of the density for a set of generated samples, we first generate the samples by setting the network parameters to the maximum of the posterior distribution learned by the Bayesian generator. As the next step, we use the network as a density estimator and compute the density for different networks sampled from the parameter posterior distribution. The width of this distribution then gives us $\sigma(x_i)$.

The quantity defined in Eq. (4.18) superficially looks like a pull between the learned density and true density,

$$t(x_i) = \frac{\mu(x_i) - \mu(x_i)w(x_i)}{\sigma(x_i)} \approx \frac{p_{\text{model}}(x_i) - p_{\text{data}}(x_i)}{\sigma(x_i)} \,, \tag{4.19}$$

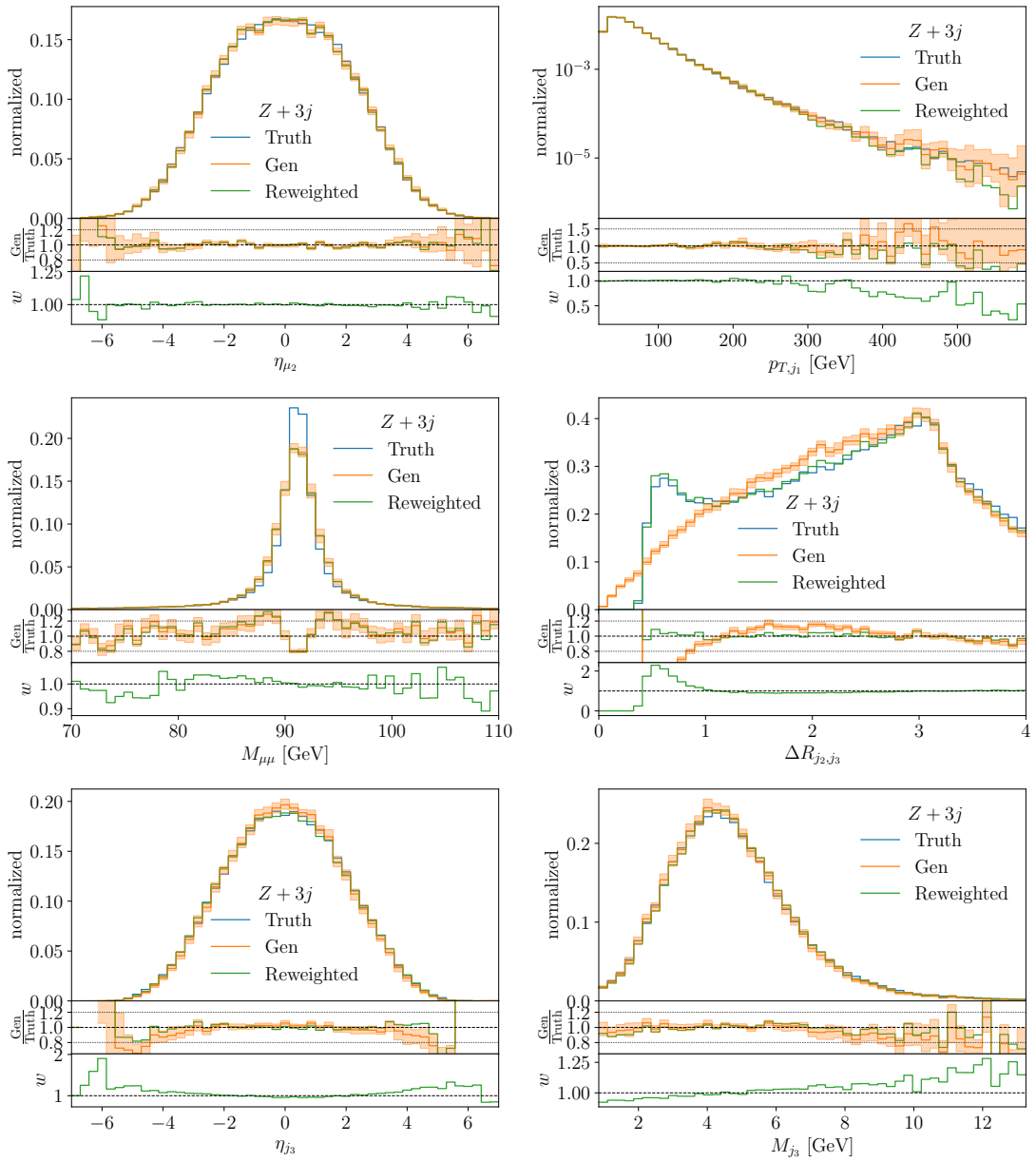


Figure 4.14: Kinematic distributions for $Z + 3$ jets events from the Bayesian state-of-the-art generator, with bin-wise error bars on the generated events. We include the distributions from Fig. 4.11 as well as the challenging distributions from Fig. 4.12.

and one would therefore expect $t(x_i)$ to follow a standard normal distribution. However, this interpretation assumes that the densities are independently distributed in every point. This is not true because of correlations between points, especially if they are in close proximity. The lower panels of Fig. 4.13 show the distributions of $t(x_i)$ for the three different jet multiplicities. While the distributions roughly have a Gaussian shape, the width is much smaller than one, confirming that the uncertainties are overestimated when their correlations are neglected.

The upper panels of Fig. 4.13 show the correlation between the classifier weight $w(x)$ and the relative error $\sigma(p_{\text{model}})/\mu(_{\text{model}})$. These two quantities are correlated for events with

one and two jets, with the lowest median relative errors for events with weights close to one. This indicates that the generator performs worst in phase-space regions with low training statistics in these cases. The situation changes for $Z + 3$ jet events, where the correlation is less clear. For these events missing features are the dominant failure mode of the generator, which is not detected by Bayesian networks. We can confirm this observation by looking at observable histograms again. Fig. 4.14 shows the same observables as in Fig. 4.11 and Fig. 4.12. Like in Sec. 4.3, we obtain uncertainty bands by taking the bin-wise means and standard deviations of the histograms for different samples from the network parameter distribution. In the upper two panels, there is a good agreement between the generated and truth distribution, so the classifier weights are close to one. The uncertainties from the BNN are over-conservative in the tails of these distributions and both the deviations from the truth as well as the reweighting are covered by them.

In the two middle panels, we show the histograms of $M_{\mu\mu}$ and ΔR_{jj}, as these are examples for distributions with missing or smoothed out features. As expected, we can see that these missing features are not found by the BNN, but the classifier is able to correctly reweight the missing collinear enhancement. This is an example where the predictions of the BNN and the classifier are not correlated. They can be understood as orthogonal because the classifier focuses on missing features and the BNN focuses on regions of low training statistics. Lastly, we look at the distributions from Fig. 4.12. We can see that the reweighted distributions are covered by the Bayesian uncertainty bands, indicating that these issues might be caused by a lack of training statistics in the 3-jet dataset. These observables are an example where the combination of the information from the classifier and from the BINN can be used to better understand the behavior of the generative network.

4.5 Conclusion

There are two crucial steps to establish generative models as tools for LHC event generation. First, we need architectures that estimate the phase-space density with the required precision. Second, we need tools to understand the uncertainties introduced by these generative networks. We showed that invertible neural networks are able to reach percent-level precision in the bulk of important phase-space distributions for LHC events with a leptonically decaying Z boson and a variable number of jets.

We first described how a chain of conditional INNs can be used to model events with variable numbers of jets without having to train a separate model for every multiplicity. Then we showed how preprocessing and weighting of the training data can be used to help the network extract challenging features from the training data for the example of the collinear enhancement and cutoff in ΔR_{jj} between two jets. As shown in Fig. 4.4, this allowed us to learn the phase-space density with a precision close to that of the training dataset in the bulk of many phase-space distributions. To better understand the uncertainty arising from the limited size of the training dataset, we then moved from a deterministic to a Bayesian INN setup with the results shown in Fig. 4.5. We checked the extracted uncertainties by varying the size of the training dataset and the number of samples generated by the network.

Finally, we showed that classifier networks are able to identify typical failure modes of generative models, for example missing features, underpopulated tails or sharp features

learned with a reduced resolution. These failure modes and their clustering in phase space can be understood from the classifier weight distribution. Furthermore, we showed that reweighting the generated data with the classifier weights successfully improved the challenging ΔR_{jj} and $M_{\gamma\gamma}$ phase-space distributions. We ended our discussion by showing that the uncertainties found by classifiers and Bayesian networks are complementary, especially in cases where the performance of the generator is limited by its lack of expressivity and not primarily by the size of the training dataset.

To sum up, our study shows that INNs are a promising architecture for fast surrogate networks for LHC event generation, and that Bayesian neural networks and classifiers are a key ingredients to understand and reduce the uncertainties associated with generative networks.

Chapter 5

Neural importance sampling

The research presented in this chapter is based on work in collaboration with Anja Butter, Nathan Huetsch, Joshua Isaacson, Claudius Krause, Fabio Maltoni, Olivier Mattelaer, Tilman Plehn and Ramon Winterhalder, and has been previously published in Ref. [3] and Ref. [6]. All tables and figures as well as parts of the text are similar or identical to the content of these articles. In particular, Sec. 5.2 is based on Ref. [3], Sec. 5.3 is based on Ref. [6], and Sec. 5.1 is based on both articles.

In the last chapter, we discussed how generative networks can be used as a fast and precise surrogate for LHC simulations. However, these generative networks still rely on Monte Carlo training data from the established LHC event generation chain. We should therefore also ask the question: How can ML help to speed up these simulation tools without relying on pre-generated training data or making compromises in terms of their precision? In this chapter, we will show how this can be done for the sampling of hard-scattering events, where the exact likelihood is known from first principles.

The generation of hard-scattering events in tools like MadGraph5_aMC@NLO [135], Sherpa [35], Pythia8 [34] or Whizard [119] is based on adaptive multi-channel importance sampling, described in detail in Sec. 2.2.1. The basic principle of this method is to find a mapping from random numbers to phase space that approximates the probability distribution defined by the differential cross section as closely as possible. Finding such a mapping is also the training objective of normalizing flows. Because the Jacobian of this mapping is tractable, the most desirable property of importance sampling, exact sampling from the target distribution, is preserved. These properties make normalizing flows the perfect ML architecture for hard-scattering event generation. Neural importance sampling [167] was first applied to LHC physics in the Sherpa event generator [45, 47, 48]. It was shown that normalizing flows can be used to replace the standard Vegas algorithm [116–120] for phase-space integration and importance sampling, and lead to improvements in the unweighting efficiency.

These early applications of neural importance sampling have two key limitations. First, they only focus on training phase-space mappings for individual channels of the multi-channel integral without improving the channel decomposition itself. Second, they rely on an online training mode, where the training data is generated by the network during the training, used once, and then discarded. While this makes the training less susceptible to overfitting, it is also more costly compared to a regular neural network training where each sample is used multiple times. This is especially problematic for processes with high multiplicities where the matrix element computation is expensive.

In this chapter, we present MadNIS – MadGraph Neural Importance Sampling, a tool that addresses these limitations. Based on the local phase-space weights implemented in MG5aMC, we show how a simple feed-forward network can be used to improve the multi-channel decomposition. Furthermore, we introduce buffered training which allows to reuse samples generated during the online training multiple times, resulting in faster trainings.

This chapter is organized as follows. In Sec. 5.1 we present the basic MadNIS setup, including trainable channel mappings and weights. Next, we introduce buffered training and additional improvements to the training. This includes using the Vegas algorithm for the initialization of the networks, and using stratified sampling and channel dropping to make the training more stable and efficient. We then study the MadNIS approach for one- and two-dimensional toy examples and a simple physics process in Sec. 5.2, before we interface MadNIS with MG5aMC to apply it to more complicated LHC examples in Sec. 5.3.

5.1 MadNIS

Repeating the results from Sec. 2.2, multi-channel Monte Carlo phase-space integrals are computed by decomposing the integrand f using local, normalized channel weights $\alpha_i(x)$ as

$$f(x) = \sum_{i=1}^{n_c} \alpha_i(x)\, f(x) \qquad \text{with} \quad \sum_{i=1}^{n_c} \alpha_i(x) = 1 \quad \text{and} \quad \alpha_i(x) \geq 0 \,, \tag{5.1}$$

and defining phase-space mappings for every channel,

$$x \in \mathbb{R}^D \quad \underset{\leftarrow \overline{G}_i(z)}{\overset{G_i(x)\rightarrow}{\longleftrightarrow}} \quad z \in [0,1]^D \,, \tag{5.2}$$

with corresponding densities

$$g_i(x) = \left| \frac{\partial G_i(x)}{\partial x} \right| \quad \text{with} \quad \int \mathrm{d}^D x\, g_i(x) = 1 \,. \tag{5.3}$$

The full multi-channel phase-space integral then becomes

$$\begin{aligned} I &= \sum_{i=1}^{n_c} \int_{[0,1]^D} \mathrm{d}^D y \;\; \frac{\alpha_i(x) f(x)}{g_i(x)} \bigg|_{x=\overline{G}_i(y)} \\ &= \sum_{i=1}^{n_c} \left\langle \frac{\alpha_i(x) f(x)}{g_i(x)} \right\rangle_{x\sim g_i(x)} \,, \end{aligned} \tag{5.4}$$

with a channel variance

$$\sigma_i^2 \equiv \sigma_i^2 \left[\frac{\alpha_i f}{g_i} \right] = \left\langle \frac{\alpha_i(x)^2 f(x)^2}{g_i(x)^2} \right\rangle_{x\sim g_i(x)} - I_i[f]^2 \,. \tag{5.5}$$

Given a fixed set of channel weights $\alpha_i(x)$, this variance is minimized by choosing the optimal mapping with

$$g_i(x)\big|_{\text{opt}} = \frac{\alpha_i(x) f(x)}{I_i[f]} \,. \tag{5.6}$$

In the following, we will describe how both channel weights and mappings can be encoded in neural networks, and how these networks can be trained to minimize the variance of the integral.

5.1.1 Neural multi-channel importance sampling

Trainable channel weights

Trainable channel weights are implemented in MadNIS by expressing the channel weights from Eq. (5.1) as a neural network (CWnet) with trainable parameters ξ,

$$\alpha_i(x) \equiv \alpha_{i\xi}(x) \; . \tag{5.7}$$

The normalization condition from Eq. (5.1) can be implemented using a Softmax activation function, as defined in Eq. (3.6). To improve the convergence of the training, it is often useful to use a prior assumption about the form of the channel weights, encoding our knowledge about the integrand and channel decomposition. An example for such a prior is the single-diagram enhanced multi-channel method implemented in MG5aMC [142, 143] and described in Sec. 2.2.4. We denote the channel weight prior as α^{MG}. The channel weight is then computed using the network prediction as a multiplicative correction to the prior weight,

$$\alpha_{i\xi}(x) = \text{Softmax}\left[\log \alpha_i^{\text{MG}}(x) + \Delta_{i\xi}(x)\right] = \frac{\alpha_i^{\text{MG}}(x)\ \exp \Delta_{i\xi}(x)}{\sum_j [\alpha_j^{\text{MG}}(x)\ \exp \Delta_{j\xi}(x)]} \; . \tag{5.8}$$

The network predicting $\Delta_{j\xi}(x)$ is implemented as a simple feed-forward network. By initializing the weights and biases of the last layer to zeros, we make the prior weights the starting point of the network training .

Trainable channel mappings

The next step is to combine the analytic channel mappings from the unit hypercube into phase space with an invertible neural network mapping between two unit hypercubes,

$$x \in \mathbb{R}^D \xleftarrow{\text{analytic}} y \in [0,1]^D \xleftarrow{\text{INN}} z \in [0,1]^D \; . \tag{5.9}$$

This is similar to the way the Vegas algorithm would normally be used to refine the analytic channel mappings. We denote the full mapping including the analytic part and the part parameterized by an INN as

$$z = G_{i\theta}(x) \qquad \text{or} \qquad x = \overline{G}_{i\theta}(z) \; , \tag{5.10}$$

with trainable parameters θ. Like for the channel weights, this allows as to combine our physics knowledge with the flexibility of a neural network.

Like in the previous chapter, we use an INN architecture based on rational quadratic spline coupling blocks [110], as it combines a high expressivity with fast sampling and density estimation. We use a modified version of the RQ spline transformations with learnable derivatives at the boundaries to increase the flexibility. Further, we normalize the derivatives such that the spline is the identity transformation if all conditional inputs

are zeros. This modified version of RQ splines is described in Sec. 3.2.3, and a detailed description of coupling-block based normalizing flows is given in Sec. 3.2.2. To build an INN with a minimal number of coupling blocks while ensuring that every feature is conditioned on every other feature at least once, we construct the permutations of the features based on a logarithmic decomposition of the integral dimension [45]. The number of coupling blocks then scales with $\log D$.

5.1.2 Training and loss function

Combining the neural multi-channel weights and mappings, the integral from Eq. (5.4) is given by

$$I[f] = \sum_{i=1}^{n_c} \left\langle \frac{\alpha_{i\xi}(x) f(x)}{g_{i\theta}(x)} \right\rangle_{x \sim g_{i\theta}(x)} . \tag{5.11}$$

The goal of the training is to find channel weights and mappings that minimize the variance of this integral. The optimality criterion from Eq. (5.6) alone is not sufficient to perform this optimization as it yields a solution for the channel mappings given a specific choice of channel weights, but cannot be used to optimize the channel weights itself. This means we cannot use an f-divergence like the KL divergence to optimize the channel mappings. Instead, we run a joint training of the weights and mappings with the variance of the integral as our loss function. Given a total number of samples N, and N_i samples in channel i, we can use the variance of multi-channel integration from Eqs. (2.51) and (2.53) to define the variance loss

$$\begin{aligned} \mathcal{L}_{\text{variance}} &= \sum_{i=1}^{n_c} \frac{N}{N_i} \sigma_i^2 \\ &= \sum_{i=1}^{n_c} \frac{N}{N_i} \left(\left\langle \frac{\alpha_{i\xi}(x)^2 f(x)^2}{g_{i\theta}(x)^2} \right\rangle_{x \sim g_{i\theta}(x)} - \left\langle \frac{\alpha_{i\xi}(x) f(x)}{g_{i\theta}(x)} \right\rangle^2_{x \sim g_{i\theta}(x)} \right) . \end{aligned} \tag{5.12}$$

Buffered and online training

This loss was derived under the assumption that both the generation of the samples and the optimization are done with the same network and identical trainable parameters (θ, ξ). We refer to this way of training as *online training*. The online training algorithm is as follows:

1. Draw samples z from the uniform latent space distribution;
2. Evaluate the inverse mapping $\overline{G}_{i\theta}(z)$ to get phase-space samples x, without keeping track of gradients;
3. Evaluate the integrand $f_i(x)$ for these samples;
4. Compute the density $g_{i\theta}(x)$ by evaluating the forward mapping;
5. Compute the loss function and its gradients $\nabla_{\theta,\xi} \mathcal{L}_{\text{variance}}$. Update the network parameters.

The algorithm is illustrated in 5.1. Note that in principle it is possible to evaluate the density $g_{i\theta}(x)$ during the sampling pass, and naively, one would expect that it is possible to already compute the gradients for the network optimization during the sampling pass

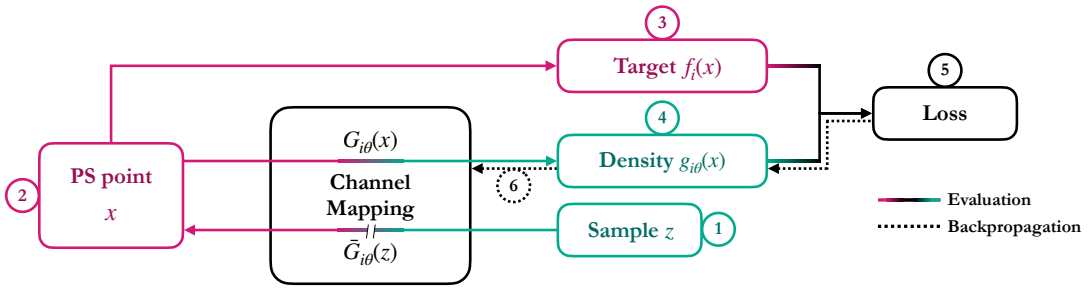


Figure 5.1: Workflow of the online training of the INN. The discontinuous line from (1) to (2) indicates that it only allows forward sampling but no gradient backpropagation.

to reduce the number of network evaluations. However, the samples x then have non-zero gradients with respect to the network parameters. To get the correct gradients of the loss functions, it then becomes necessary to also differentiate the target function $f(x)$. This alternative training strategy is explained in more detail in Ref. [3]. As the gradients of f are not always available, this thesis focuses on the standard online training outlined above.

The above algorithm requires two evaluations of the neural channel mappings, and one evaluation of the neural channel weights and target density in every optimization step. This can become costly if f is computationally expensive, for instance if it is the differential cross-section for a process with high multiplicity. It is possible to accelerate the training by sampling x and evaluating $f(x)$ once, and then updating the network multiple times using this sample. To this end, we generalize the loss function from Eq. (5.12),

$$\mathcal{L}_{\text{variance}} = \sum_{i=1}^{n_c} \frac{N}{N_i} \left(\left\langle \frac{\alpha_{i\xi}(x)^2 f(x)^2}{g_{i\theta}(x)\, q_i(x)} \right\rangle_{x\sim q_i(x)} - \left\langle \frac{\alpha_{i\xi}(x) f(x)}{q_i(x)} \right\rangle^2_{x\sim q_i(x)} \right) . \tag{5.13}$$

In this generalized version, we distinguish between the network density $g_{i\theta}(x)$ and the sampling density $q_i(x)$. It can be understood as a reweighted version of the online training loss with weights

$$\frac{g_{i\theta}(x)}{q_i(x)} . \tag{5.14}$$

For online training, we have $q_i(x) = g_{i\theta}(x)$. If the sample was generated by an earlier version of the network with parameters $\hat{\theta}$, we have $q_i(x) = g_{i\hat{\theta}}(x)$. The training is most stable for $q_i(x) \approx g_{i\theta}(x)$. Therefore, every sample should only be reused a limited amount of times before it is replaced with a new sample. We refer to this training mode as *buffered training*. The training workflow, illustrated in Fig. 5.2, is as follows:

1. Start with a buffered phase-space point x with integrand $f(x)$ and sampling density $q_i(x)$;
2. Compute the density $g_{i\theta}(x)$ by evaluating the forward mapping;
3. Compute the loss function and its gradients $\nabla_{\theta,\xi}\mathcal{L}_{\text{variance}}$. Update the network parameters.

Because new samples need to be generated to replace older buffered samples, there is no pure buffered training, and we need to switch between the two training modes. The

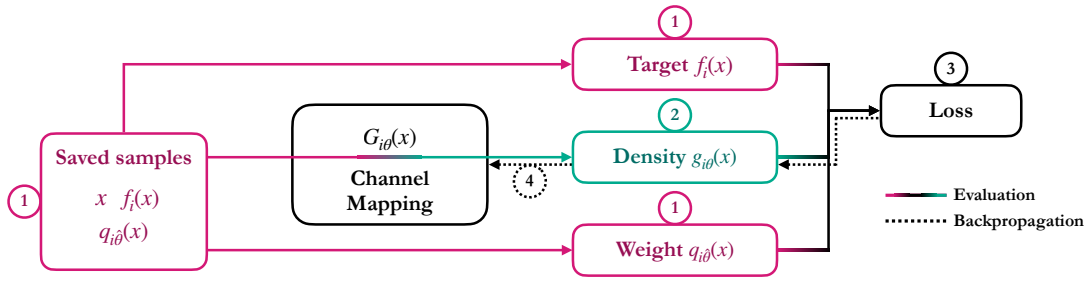


Figure 5.2: Workflow of the buffered training of the INN.

schedule that determines when to switch between the training modes is a hyperparameter of the training.

There is a trade-off between the reduction in training statistics through buffered training and the number of network weight updates within a given time. Let $r_@$ be the fraction of the total training time T used for online training. The times T_{buff} for buffered training and $T_@$ for online training are

$$T_{\text{buff}} = T \cdot (1 - r_@) \quad \text{and} \quad T_@ = T \cdot r_@ \,. \tag{5.15}$$

Let t_{buff} and $t_@$ be the times for a single weight update in buffered and online mode, excluding the integrand evaluation, and let t_f be the time for the integrand evaluation. t_{buff} is larger than $t_@$ because of the additional network evaluation for sampling, with $t_@/t_{\text{buff}} \approx 4/3$. The total time for a weight update in online training including the integrand evaluation is $t_@ + t_f$. For a fixed training time T, the total number of weight updates is

$$n = n_{\text{buff}} + n_@ = \frac{T_{\text{buff}}}{t_{\text{buff}}} + \frac{T_@}{t_@ + t_f} \,. \tag{5.16}$$

We compare this with the number of weight updates in a pure online training,

$$n_{\text{base}} = \frac{T}{t_@ + t_f} \,. \tag{5.17}$$

We can use Eqs. (5.15) to (5.17) to compute the increase in the number of weight updates for a fixed training time,

$$\frac{n}{n_{\text{base}}} = \left(1 - \frac{1}{R_@}\right) \frac{t_@ + t_f}{t_{\text{buff}}} + \frac{1}{R_@} \qquad \text{with} \qquad R_@ = \frac{n_{\text{base}}}{n_@} = \frac{1}{r_@} \,, \tag{5.18}$$

where we introduced the reduction in training statistics $R_@$. In the left panel of Fig. 5.3, we show the increase in weight update as a function of $R_@$ for values of t_{buff} and $t_@$ extracted from a test run on a CPU and different values of t_f. In the right panel we show the change in training time for a fixed number of weight updates. It can be seen that buffered training leads to a considerable speedup for costly integrands.

There are different ways to implement a buffered training schedule with a given $R_@$. For the toy physics example shown in Sec. 5.2.3, we choose a simple schedule where the buffer is filled up with 500k samples, corresponding to one online training epoch. These samples are then reused k_{buff} times for a reduction in training statistics of $R_@ = k_{\text{buff}} + 1$. While this schedule works well for simple integrands, it has relatively long intervals between updates of the buffered samples. For the results presented in Sec. 5.3, we instead use a buffer size of 1000 batches and first fill the buffer using online training. Then we alternate

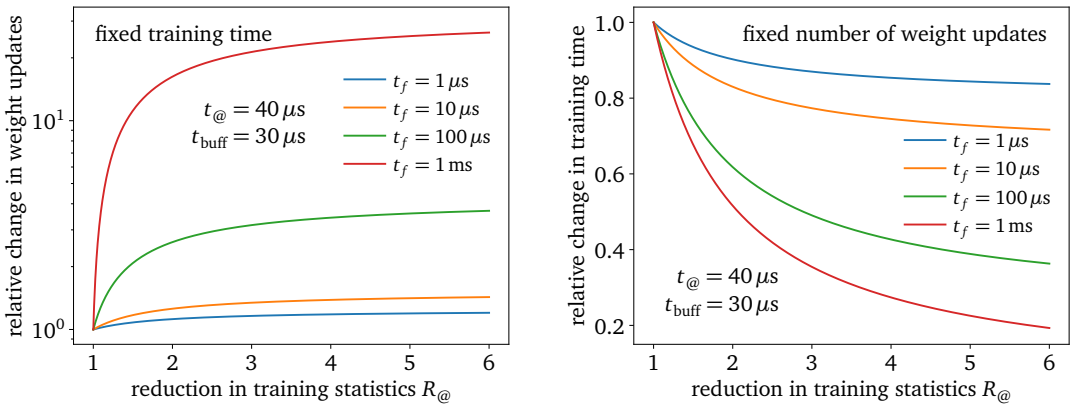


Figure 5.3: Hypothetical change in weight updates (left panel) and training time (right panel) as a function of the reduction in training statistics $R_@$ for integrands with different computational costs.

between online and buffered training and replace quarter or half of the buffer in each online training epoch, resulting in gain factors of $R_@ = 5$ and $R_@ = 3$ respectively.

A major challenge of buffered training is that the memory consumption of the buffer can quickly become prohibitive. This is mainly caused by the prior channel weights which have to be stored for all channels to compute the normalization of the channel weights during the training. Saving the complete set of weights quickly becomes prohibitive, as $\mathcal{O}(1M)$ phase-space points with $\mathcal{O}(1k)$ channels require several gigabytes of memory. Based on the observation that for any given sample most channels weights are close to zero, we can choose a more efficient way to store them. We store the indices and weights $(i, \alpha_i^{\text{MG}})$ of the $m < n_c$ channels with the largest weights, and make sure to always include the channel used to generate the sample in that list. We then set the other weights to zero during buffered training and adjust the normalization accordingly. This approximation does not introduce a bias as it only affects the buffered training but not online training, integration or sampling.

Stratified loss and training

So far we have not specified how the number of samples per channel N_i has to be chosen. This choice affects the optimal channel weights and mappings, so the N_i used for integral evaluation and sampling have to be known at training time. As derived in Sec. 2.2.4, the optimal choice of N_i known from stratified sampling [136, 198] is given by

$$N_i = N\frac{\sigma_i}{\sum_j \sigma_j} \qquad \Leftrightarrow \qquad \frac{N}{N_i} = \frac{\sum_j \sigma_j}{\sigma_i} \,. \tag{5.19}$$

Inserting this into the variance loss from Eq. (5.13) results in the MadNIS loss

$$\begin{aligned} \mathcal{L}_{\text{MadNIS}} &= \sum_{i=1}^{n_c} \left(\sum_{j=1}^{n_c} \sigma_j \right) \sigma_i = \left[\sum_{i=1}^{n_c} \sigma_i \right]^2 \\ &= \left[\sum_{i=1}^{n_c} \left(\left\langle \frac{\alpha_{i\xi}(x)^2 f(x)^2}{g_{i\theta}(x)\, q_i(x)} \right\rangle_{x\sim q_i(x)} - \left\langle \frac{\alpha_{i\xi}(x) f(x)}{q_i(x)} \right\rangle^2_{x\sim q_i(x)} \right)^{1/2} \right]^2 . \end{aligned} \tag{5.20}$$

Note that even though we derived the loss function based on the choice of N_i for integration and sampling, the number of samples per channel during training can be chosen independently. In the simplest case, we choose a uniform distribution of the N_i during training. However, when MadNIS is applied for physical processes, the variance of the channels scales with their cross sections and can therefore be very imbalanced. This will lead to noisy gradients from channels with a large variance and a small number of training samples. This problem can be solved by adjusting the number of samples per channel, and, optionally, dropping channels with a very low contribution to the total integral.

The variance of the channels depends on the channel mappings and weights, so we compute them using running means during the training. Channels with a very small number of points can also cause unstable trainings, so we choose a fraction r of points that is distributed evenly, and distribute the rest of the points using stratified sampling as given in Eq. (5.19). The fraction r can then be used to interpolate between uniform sampling ($r = 0$) and stratified sampling ($r = 1$). While it is not guaranteed that every channel will contain samples with non-zero weights because of phase-space cuts, this method is sufficient to ensure stable trainings. Also note that the estimates of the channel variances from running means will be unstable in the beginning of the training, so we start the training with uniform N_i for 1000 batches for the results shown in Sec. 5.3. We refer to this training mode as *stratified training*.

Some channels only have a tiny contribution to the total cross section. Such channels can be dropped altogether such that no training time is invested for channels with no significant contribution. We identify these channels by tracking a running mean over the channel-wise integrals I_i. We then specify a small fraction of the total cross section as a threshold, for example $10^{-3} \cdot I$. Starting with the lowest I_i, we drop channels until the sum of the dropped I_i reaches the threshold. Dropped channels are also removed from the sample buffer, and the normalization of the channel weights $\alpha_{i\xi}$ is adjusted to ensure an unbiased integral.

Symmetries between channels

For multi-channel integration of differential cross sections, the channels are constructed from the Feynman diagrams of the process. Some of these diagrams have identical matrix elements up to permutations of the final-state particles. While this is also true for initial-state particles, there could be PDF effects that cause the differential cross section to be different. Similar to the approach implemented in MG5aMC, we combine the channels mappings for such symmetry-related channels. Further, we also combine them in the loss function such that there is only one σ_i for each group of channels in the MadNIS loss in Eq. (5.20). This effectively reduces the number of channels, and leads to less noisy gradients and an improved training stability.

5.1.3 Vegas initialization

The Vegas algorithm [116, 117, 120] is used to refine the phase-space mappings in most LHC event generators. It is very efficient and converges much faster than a neural network trained with gradient descent. However, it assumes that the integrand factorizes and is therefore not able to model any correlations. To combine the advantages of Vegas with the much better expressivity of normalizing flows, we use Vegas to initialize our

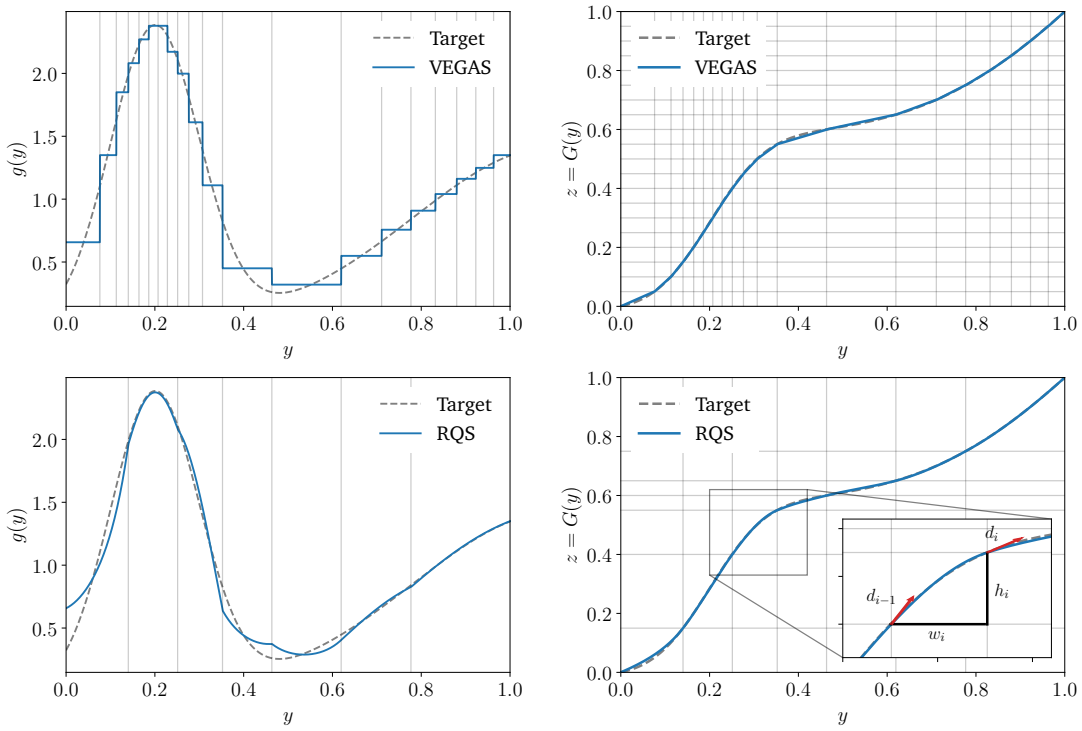


Figure 5.4: Top: learned VEGAS density $g(x)$ with 20 bins (left) and its transformation $G(x)$ (right). Bottom: the RQS density $g(x)$ after embedding the VEGAS grid and performing bin reduction to 7 bins (left). The right plot shows the corresponding mapping $G(x)$ including a zoom-in box illustrating the definition of the widths w_i, heights h_i of the bins and the derivatives d_i on the bin edges.

flow networks. In contrast to simply inserting a fixed VEGAS grid as an additional analytical mapping, this procedure allows the network to freely move away from its initialization while still using the advantages of the fast VEGAS training. As introduced in Sec. 2.2.2, the VEGAS algorithm divides the unit interval for each dimension into K bins with widths w_k. The density of the target distribution is then modeled as a piecewise-constant function $g(y)$ and the corresponding CDF is a piecewise-linear function $G(y)$. We illustrate g and G with $K = 20$ bins for a Gaussian mixture model in the upper panels of Fig. 5.4.

To initialize an INN with a pre-trained VEGAS mapping, we need to turn the VEGAS grid defined in terms of the widths w_k into a rational quadratic spline as introduced in Sec. 3.2.3. RQ splines are defined in terms of bin widths w_k, bin heights h_k and the derivatives d_k at the bin edges. Because of their higher expressivity, fewer bins are needed for the RQS grid than for the VEGAS grid. While the most accurate results could be achieved by fitting the RQ spline transformation to the transformation given by VEGAS, we find that results with sufficient precision can also be achieved with the following simple algorithm. First, we initialize the bin widths to be the same as in VEGAS, the heights to

be equal for all bins, and estimate the derivatives as the ratio of bin heights and widths,

$$\begin{aligned}
\text{widths:} \quad & w_k & & \text{for} \quad k = 1, \dots, K \\
\text{heights:} \quad & h_k = \frac{1}{K} & & \text{for} \quad k = 1, \dots, K \\
\text{endpoints:} \quad & d_0 = \frac{h_1}{w_1} = \frac{1}{K w_1} \qquad d_K = \frac{h_K}{w_K} = \frac{1}{K w_K} & & \\
\text{internal points:} \quad & d_k = \frac{h_{k+1} + h_k}{w_{k+1} + w_k} = \frac{2}{K(w_{k+1} + w_k)} & & \text{for} \quad k = 1, \dots, K-1 \,.
\end{aligned} \tag{5.21}$$

We then iteratively reduce the number of bins by merging adjacent bins until the desired number is reached:

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

$$\Delta_k = \left| \frac{w_k}{h_k} - \frac{w_{k+1}}{h_{k+1}} \right| \,. \tag{5.22}$$

2. Choose k with the lowest Δ_k, corresponding to the two bins k and $k+1$ with the most similar average slopes. This can lead to inaccuracies for very large bins as there can be considerable derivations of the derivative from the average slope. We prevent this by introducing a cutoff that stops very large bins from forming until all smaller bins are merged.

3. Reduce the number of bins by one,

$$\begin{aligned}
w &\leftarrow (w_1, \dots, w_{k-1}, w_k + w_{k+1}, w_{k+2}, \dots, w_K) \,, \\
h &\leftarrow (h_1, \dots, h_{k-1}, h_k + h_{k+1}, h_{k+2}, \dots, h_K) \,, \\
d &\leftarrow (d_0, \dots, d_{k-1}, d_{k+1}, \dots, d_K) \,, \\
K &\leftarrow K - 1 \,.
\end{aligned} \tag{5.23}$$

We illustrate the result of this algorithm in the lower panels of Fig. 5.4. The number of bins of the example VEGAS grid from above is reduced from $K = 20$ to $K = 7$. We show w, h and d for one bin. As the last step, we inject the RQS grid into the INN as its initialization. The bin widths, heights and derivatives are encoded by a fully-connected trainable sub-network. We initialize the transformations of all coupling blocks except for the last two to the identity transformation by setting the weights and biases of the final sub-network layer to zero. Each of the last two coupling blocks transforms half of the input features. We invert the normalization from Eq. (3.50) to get Θ^w, Θ^h and Θ^d from the w, h and d yielded by the bin reduction algorithm. These values are then assigned to the bias vectors of the final sub-network layers, while the the weight matrices are set to zero. The resulting INN then encodes a transformation very similar to the pre-trained VEGAS grid that was used as a starting point.

5.2 Toy examples

We start by testing the basic functionality of the MADNIS integrator for simple toy distributions. The first example is a one-dimensional camel back for which we examine the learned channel weights without trainable channel mappings. The second example is a two-dimensional crossed ring, where we combine trainable channel weights and mappings.

We end the discussion with a simple LHC example where we benchmark the buffered training.

In all these tests, we use a simplified version of the MadNIS integrator. We assume a flat distribution of the counts per channel instead of the stratified loss function from Eq. (5.20). We do not use Vegas initialization, stratified training and channel dropping. These features are only necessary for more complicated examples, and will be benchmarked in Sec. 5.3.

5.2.1 One-dimensional camel back

We first illustrate the effect of the trainable channel weights combined with fixed, analytical mappings for a one-dimensional bi-modal Gaussian mixture model, or camel back. The target function is given by

$$\begin{aligned} f_{\text{GM}}(x) = \frac{a_1}{\sqrt{2\pi}\sigma_1} \exp\left[-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right] + \frac{1-a_1}{\sqrt{2\pi}\sigma_2} \exp\left[-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right] \\ \text{with} \qquad \mu_1 = 2 \qquad \sigma_1 = 0.5 \qquad \mu_2 = 5 \qquad \sigma_2 = 0.1 \qquad a_1 = 0.35\,. \end{aligned} \tag{5.24}$$

In this simple example, we could use two channels with a Gaussian distribution to exactly recover the truth distribution. To bring the example closer to actual applications where the integrand is not modeled exactly by the channel mappings, we choose Cauchy or Breit-Wigner mappings instead,

$$\begin{aligned} x = \overline{G}_i(y) &= \mu_i + \sqrt{2}\sigma_i \tan\left[\pi\left(y - \frac{1}{2}\right)\right] \\ g_i(x) &= \frac{1}{\pi}\frac{\sqrt{2}\sigma_i}{(x-\mu_i)^2 + 2\sigma_i^2}\,. \end{aligned} \tag{5.25}$$

While these distributions look similar to the ideal Gaussian distributions close to the peak, they decay much slower in the tails. The full multi-channel integral from Eq. (5.4) is then given by

$$\begin{aligned} I[f_{\text{GM}}] &= \int_{-\infty}^{\infty} \mathrm{d}x\, f_{\text{GM}}(x) \\ &= \sum_{i=1}^{2} \int_{-\infty}^{\infty} \mathrm{d}x\, \alpha_{i\xi}(x)\, f_{\text{GM}}(x) \\ &= \sum_{i=1}^{2} \int_0^1 \mathrm{d}y\ \alpha_{i\xi}(x)\, \frac{f_{\text{GM}}(x)}{g_i(x)}\bigg|_{x=\overline{G}_i(y)}\,. \end{aligned} \tag{5.26}$$

We train MadNIS with the fixed channel mappings from Eq. (5.25) and with a small fully-connected network to learn the channel weights $\alpha_{i\xi}$. The hyperparameters are given in Tab. A.3. We repeat the training ten times to test the training stability. In Tab. 5.1 we compare the relative error of the integral as a metric of convergence for different choices of $\alpha_i(x)$. For constant $\alpha_i(x) = 0.5$, the error is relatively large. It is much smaller for the nearly optimal choice

$$\alpha_i^{\text{opt}}(x) = \frac{g_i(x)}{\sum_i g_i(x)}\,. \tag{5.27}$$

Function	$\alpha_i(x)$	Rel. Error [%]
Camel back	Uniform	2.553 ± 0.017
	Optimal	0.769 ± 0.006
	NN (flat prior)	0.770 ± 0.005
	NN (opt. prior)	0.767 ± 0.006
Cut camel back	Uniform	3.412 ± 0.048
	Optimal	1.031 ± 0.006
	NN (flat prior)	1.032 ± 0.017
	NN (opt. prior)	1.030 ± 0.009

Based on 10^4 events

Table 5.1: Relative errors of the camel back integrals using the trained channel weights (means and standard deviations from ten runs).

The result with trained channel weights is similar to the results for the optimal weights. We do not see a significant difference between training runs that start from flat prior weights compared to runs starting from optimal prior weights. We show the target function, the distributions from the two analytical channel mappings, the prior channel weights and the learned channel weights in Fig. 5.5. In the left panel, α_i^{opt} was used as a prior. It can be seen that the learned $\alpha_{i\xi}$ remain close to α_i^{opt}. This is consistent for multiple trainings of the network. There are only deviations from α_i^{opt} in the exponentially suppressed tails of the Gaussians where they have a low contribution to the variance loss. Consequently, these deviations also have no impact on the relative error shown in Tab. A.3. The right panel shows the result of the training starting from flat prior weights. Compared to the results starting from the optimal prior, there is more variation between runs in the suppressed regions. Near the peaks, the weights are similar to the ones shown in the left panel. Again, there is no significant difference in the relative error of the integral.

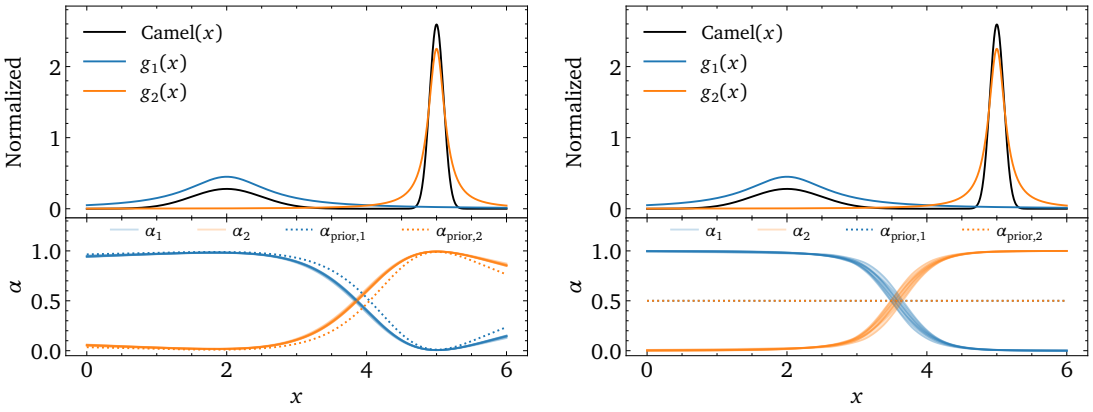


Figure 5.5: Learned channel weights for the camel back function for ten different trainings. We train NN-weights starting from a near-optimal (left) or flat (right) prior. The prior weights are illustrated as dotted lines.

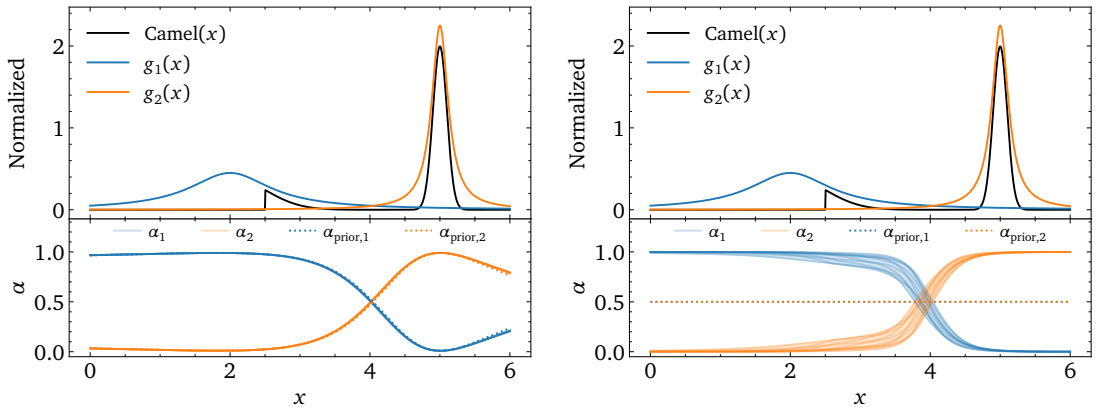


Figure 5.6: Learned channel weights for the cut camel back function for ten different trainings. We train NN-weights starting from a near-optimal (left) or flat (right) prior. The prior weights are illustrated as dotted lines.

Camel back with cut

Our channel mappings $g_i(x)$ for the camel-back function were relatively close to the target distribution. We can look at a variation of the function from Eq. (5.24) with a cut in the left peak,

$$f_{\text{GM}}(x) \;\to\; \begin{cases} f_{\text{GM}}(x) & x \geq \mu_1 + \sigma_1 \\ 0 & x < \mu_1 + \sigma_1 \end{cases} , \tag{5.28}$$

with $\mu_1 + \sigma_1 = 2.5$. We use the same mappings as before to study the effect of a larger deviation between the mappings and the target function. Again, we show the relative error in Tab. 5.1. All relative errors are now slightly larger than for the perfect camel back, but the errors for the integration with α_i^{opt} and learned weight $\alpha_{i\xi}$ are still similar. There is however a larger variation between runs when the network is trained from scratch. This is also confirmed by Fig. 5.6 where we show the learned weights starting from α_i^{opt} or a flat prior.

5.2.2 Two-dimensional crossed ring

For our next toy example, we combine the trained channel weights with trained channel mappings. The integrand is a two-dimensional Gaussian ring crossed by a diagonal Gaussian line,

$$\begin{aligned} f_{\text{no-parking}}(x) &= \frac{1}{2}\left[f_{\text{ring}}(x) + f_{\text{line}}(x)\right] \\ f_{\text{line}}(x) &= N_1 \exp\left[-\frac{(\tilde{x}_1 - \mu_1)^2}{2\sigma_1^2}\right] \exp\left[-\frac{(\tilde{x}_2 - \mu_2)^2}{2\sigma_2^2}\right] \\ f_{\text{ring}}(x) &= N_2 \exp\left[-\frac{\left(\sqrt{x_1^2 + x_2^2} - r_0\right)^2}{2\sigma_0^2}\right] \\ \text{with} \quad & r_0 = 1 \qquad \sigma_0 = 0.05 \qquad \mu_1 = 0 \qquad \sigma_1 = 3 \qquad \mu_2 = 0 \qquad \sigma_2 = 0.05 \\ \text{and} \quad & \tilde{x}_{1,2} = (x_1 \mp x_2)/\sqrt{2}\,, \end{aligned} \tag{5.29}$$

and with normalization constants N_0 and N_1 chosen such that the integrals of f_{line}, f_{ring} and $f_{\text{no-parking}}$ are one.

Channel mappings

Even though we know that the target function is constructed using Gaussian distributions, we build analytic channel mappings based on Breit-Wigner distributions to give the network a non-trivial learning task. We construct two different mappings, one corresponding to the diagonal line and one to the ring. We build the mappings in two steps. For the line, we start with a transformation $x \to y = G_1(x)$ that aligns the first axis with the diagonal line,

$$x_{1,2} = \frac{y_2 \pm y_1}{\sqrt{2}} \qquad \text{with} \qquad g_1(x) = \left| \frac{\partial G_1(x)}{\partial x} \right| = 1 \; . \tag{5.30}$$

Note that here and for the following transformations, we give the expression for the inverse direction, $y \to x = \overline{G}_1(y)$. As the second step, we use a similar mapping as for the camel back, Eq. (5.25), to get a Breit-Wigner distribution. The second transformation $y \to z = G_2(y)$ is then given by

$$\begin{aligned} y_{1,2} &= \mu_{1,2} + \gamma_{1,2} \tan\left[\pi\left(z_{1,2} - \frac{1}{2}\right)\right] \\ \text{with} \qquad g_2(y) &= \frac{1}{\pi^2} \prod_{j=1}^{2} \frac{\gamma_j}{\gamma_j^2 + (y_j - \mu_j)^2} \; , \end{aligned} \tag{5.31}$$

resulting in the combined density

$$g_{\text{line}}(x) = 1 \cdot g_2(G_1(x)) \; . \tag{5.32}$$

The first step of the ring mapping is a transformation $x \to (r, \theta) = G_3(x)$ into polar coordinates,

$$x_1 = r \cos\theta \qquad \text{and} \qquad x_2 = r \sin\theta \qquad \text{with} \qquad g_3(x) = r \; . \tag{5.33}$$

Again, the second step $(r, \theta) \to z = G_4(r, \theta)$ is a transformation to a Breit-Wigner distribution,

$$\begin{aligned} r &= r_0 + \gamma_0 \tan\left[\pi\left(\omega_0 z_1 - C_0\right)\right] \\ \theta &= 2\pi z_2 \qquad \text{with} \qquad g_4(r) = \frac{1}{2\pi} \, \frac{1}{\omega_0 \pi} \frac{\gamma_0}{\gamma_0^2 + (r - r_0)^2} \; . \end{aligned} \tag{5.34}$$

We have to ensure the positivity of the radius r, so we set $\pi C_0 = \arctan(r_0/\gamma_0)$ and $\omega_0 = (1 + 2C_0)/2$. The combined Jacobian is

$$g_{\text{ring}}(x) = r \, g_4(G_3(x)) \; . \tag{5.35}$$

These analytic mappings simplify the training by aligning the structures of the distributions with the integration variables, and they resolve the topologically challenging ring structure. When they are combined with learnable channel mappings, we still want to make the training challenging. Therefore, we choose large widths of the Breit-Wigner mappings,

$$\gamma_{0,1,2} = \sqrt{40} \, \sigma_{0,1,2} \; . \tag{5.36}$$

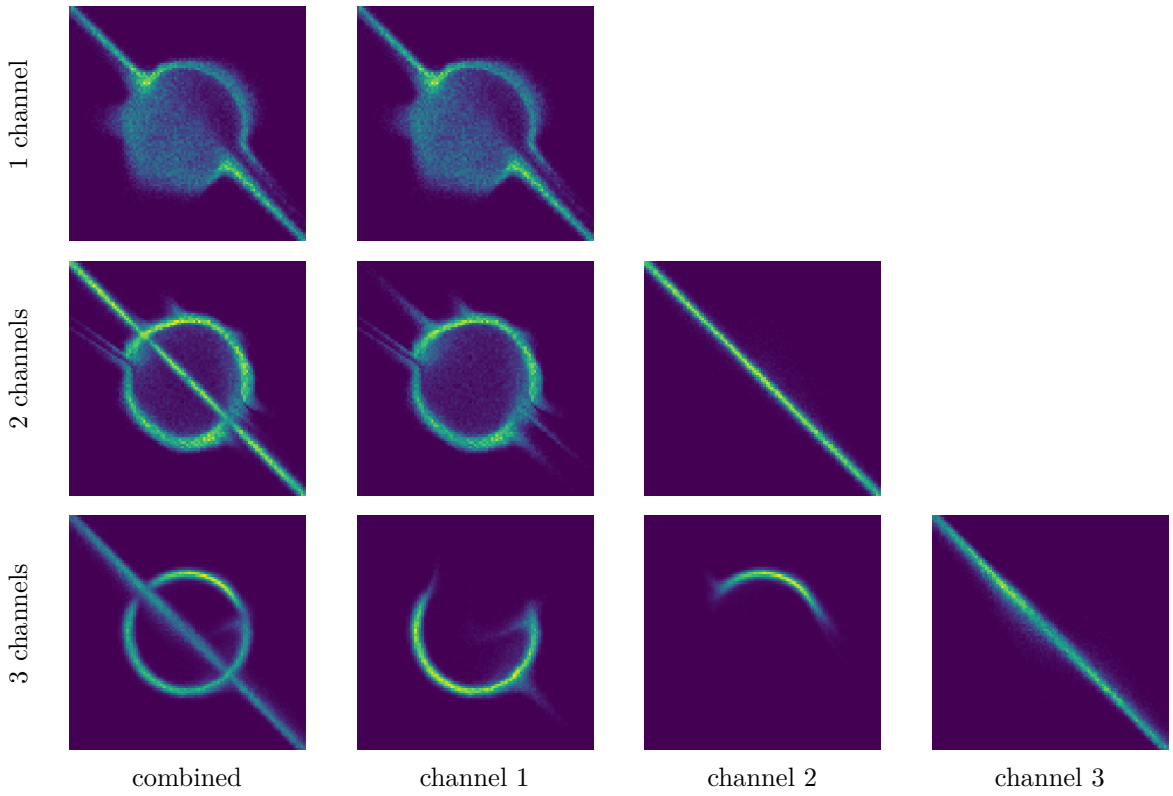


Figure 5.7: Combined and channel-wise (the latter not weighted by channel weights) distributions learned by a one-, two- and three-channel integrator with flat mappings and a mode-specific prior. Note that the splitting in the three-channel case is not unique and learned differently by the network for each run.

Results

As a first experiment, we investigate whether the MadNIS approach is able to find a sensible channel partitioning in an unsupervised way. As INNs based on spline coupling blocks are easily able to learn the crossed ring distribution without the need for multiple channels, we replace them with the less expressive affine coupling blocks, see Sec. 3.2.3.

Fig.	Analytic Mappings	Rel. Error [%]
5.7	flat	1.17 ± 0.13
5.7	flat, flat	0.71 ± 0.15
5.7	flat, flat, flat	0.50 ± 0.15
5.8	ring, flat	0.30 ± 0.11
	ring, line	0.14 ± 0.06
	ring, line, flat	0.29 ± 0.14

Based on 10^4 events

Table 5.2: Relative integration errors for different numbers of channels and choices of analytic mappings. We show the means and standard deviations for ten independent trainings.

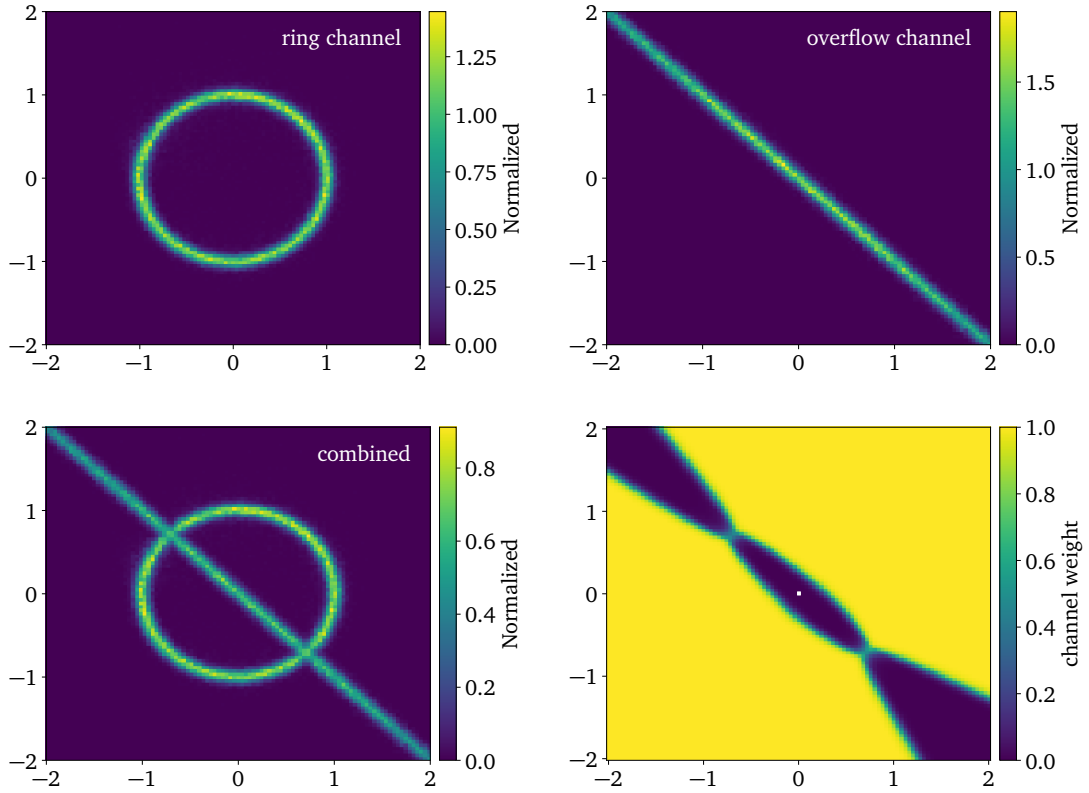


Figure 5.8: Distribution learned by a 2-channel integrator with a ring mapping and a flat mapping and flat prior. Top: individual channels, not weighted by channel weights. Bottom: combined distribution and channel weight of the ring channel.

The hyperparameters of our setup are given in Tab. A.4. We use one, two or three channels with flat analytic mappings combined with learnable INN mappings. We find that trainings with a flat prior of the channel weights tend to only make use of a single channel, while the channel weights of the other channels are close to zero. We can improve this by choosing the channel weight prior such that ring-shaped distributions are encouraged in one or two channels, and a line-shaped distribution is encouraged in the last channel. We show examples for the distributions learned by one, two or three channels in Fig. 5.7. There are large variations between multiple runs but we find that using two or three channels improves the performance. This is also confirmed by the relative integration errors shown in Tab. 5.2. As seen in the lower panels, the network is able to partition the distribution in a way that makes the topology in each channel easier to learn by removing holes. However, the setup is very sensitive to the choice of hyperparameters and channel weight priors, so an unsupervised approach to channel partitioning is not sufficiently reliable for practical use in phase-space integrators.

A better approach is to use analytic channel mappings specifically constructed to map out the structure of the integrand. In the case of the crossed ring, a mapping based on polar coordinates is especially helpful because it removes the challenging topology. When a ring mapping is used, the training task is much easier and an INN with fewer trainable parameters is sufficient. We show the learned distributions and channel weights for a training with a ring mapping and a flat mapping in Fig. 5.8. In this setup the channel with the flat mapping acts as an overflow channel that captures all features that the ring

mapping missed. The combined distribution is a closer match to the truth distribution than in a training with only flat mappings. We find that the distribution learned by the overflow channel only captures the diagonal line as expected. The channel weights cleanly separate between the two channels with weights close to 0.5 in the two points where the ring and line cross. The relative integral error for this setup is lower than for trainings with only flat mappings, as shown in Tab. 5.2. Replacing the flat mapping in the overflow channel with a line mapping leads to another improvement of the relative error. Adding a third channel as an overflow channel does not lead to further improvements as it makes the training more complex, and two channels are already sufficient to capture the target distribution.

5.2.3 Drell-Yan plus Z′ at the LHC

Next, we look at a Drell-Yan process with an additional Z′-resonance as a simple example for an LHC process,

$$\mathrm{pp} \to \gamma, \mathrm{Z}^*, \mathrm{Z}'^* \to \mathrm{e}^+\mathrm{e}^- \,, \tag{5.37}$$

with 13 TeV center-of-mass energy and

$$\begin{aligned} M_\mathrm{Z} &= 91.19 \text{ GeV} & \Gamma_\mathrm{Z} &= 2.44 \text{ GeV} \,, \\ M_{\mathrm{Z}'} &= 400.0 \text{ GeV} & \Gamma_{\mathrm{Z}'} &= 0.5 \text{ GeV} \,. \end{aligned} \tag{5.38}$$

The Feynman diagrams for this process are shown in Fig. 5.9. We use the leading-order NNPDF4.0 PDF set [131] via LHAPDF6 [199] with a fixed factorization scale $\mu_F = M_\mathrm{Z}$ and $\alpha_s(M_\mathrm{Z}) = 0.118$, and neglect b quarks in the initial state. We require all events to have

$$m_{\mathrm{e}^+\mathrm{e}^-} > 15 \text{ GeV} \,. \tag{5.39}$$

We use a hand-written implementation of the differential cross section to have full control over the integrand. The squared and spin-color averaged leading order matrix element is

$$\langle |\mathcal{M}|^2 \rangle = \frac{1}{4N_\mathrm{c}} \sum_\text{spins} |\mathcal{M}_\gamma + \mathcal{M}_\mathrm{Z} + \mathcal{M}_{\mathrm{Z}'}|^2 \,, \tag{5.40}$$

with $N_\mathrm{c} = 3$. The hadronic differential cross section can then be computed using Eq. (2.8) as a function of $(x_1, x_2, \cos\theta, \phi)$. Lastly, we define phase-space mappings from the unit hypercube $U = [0,1]^4$ to the two-particle phase space in two steps

$$\begin{aligned} G_1 :&\quad \{y_1, y_2, y_3, y_4\} \to \{s, y_2, y_3, y_4\} \\ G_2 :&\quad \{s, y_2, y_3, y_4\} \to \{x_1, x_2, \cos\theta, \phi\} \,. \end{aligned} \tag{5.41}$$

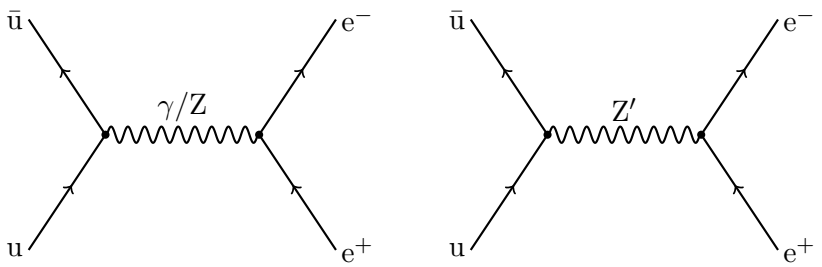


Figure 5.9: Leading-order Feynman diagrams contributing to the Z′-extended Drell-Yan process $\mathrm{pp} \to \mathrm{e}^+\mathrm{e}^-$ for one partonic channel.

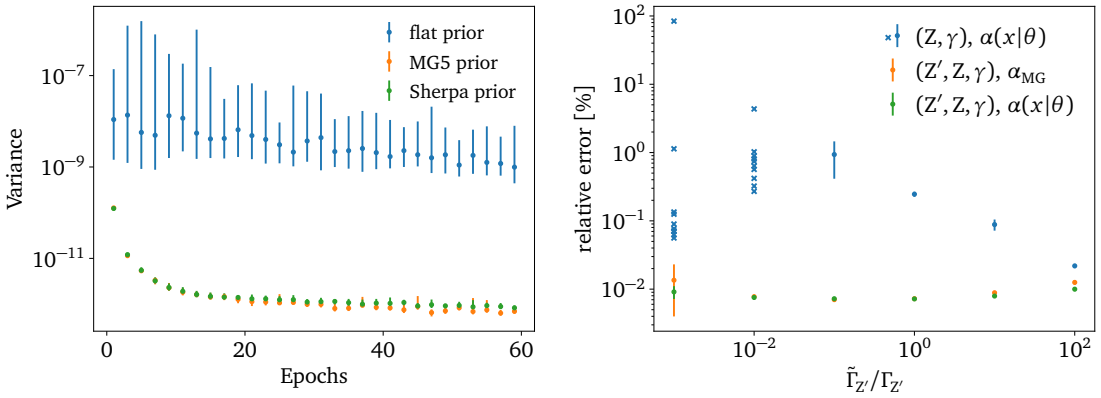


Figure 5.10: Left: mean and spread (5% to 95% percentile) of 25 evaluations of the variance for three priors of the network weights α. Right: integration error as a function of $\Gamma_{Z'}$ for two and three channels, with and without trained channel weights. We give means and standard deviations for ten runs, or the individual results in case of large variation. For very narrow peaks, the two-channel integrator misses the Z′ peak entirely.

We construct three channels, one for each possible propagator, that differ only in the mapping $y_1 \to s$. We use the Breit-Wigner mapping defined in Eq. (2.43) for the Z and Z′ resonance. For the photon, we use the mapping for massless propagators defined in Eq. (2.45) with $\nu = 2$. The mapping G_2 is the same for all channels and is given by

$$\begin{aligned} x_1 &= \left(\frac{s}{s_{\max}}\right)^{y_2} \qquad x_2 = \left(\frac{s}{s_{\max}}\right)^{1-y_2} \\ \cos\theta &= 2y_3 - 1 \qquad \phi = 2\pi y_4 - \pi \qquad \text{with} \quad g_2 = -\frac{s_{\max}}{4\pi \log(x_1 x_2)} \,. \end{aligned} \tag{5.42}$$

We integrate the differential cross section and confirm that it is in agreement to the standard MG5aMC prediction of $\sigma = (4349.7 \pm 0.32)$ pb within the integration uncertainties. Our MadNIS setup uses the hyperparameters given in Tab. A.5.

Results

As seen for the crossed ring, the choice of mappings and channel weight priors has a large impact on the performance of the MadNIS integrator. The most challenging features of our Drell-Yan process are the two narrow resonances in $M_{e^+e^-}$. These are best addressed by constructing appropriate channels and phase-space mappings. As a first check we use a three-channel setup as described above. We test three different choices for the channel weight prior: a flat prior, a Sherpa-like prior as defined in Eq. (2.63), and a MG5aMC-like prior as defined in Eq. (2.56). Figure 5.10 shows the variance of the integrand extracted from 25 batches of generated samples over the course of a 60-epoch training in intervals of 2 epochs. It can be seen that the variance is stable and quickly converges for both the MG5aMC-like and the Sherpa-like prior. However, there is a huge spread in the variance for the training with a flat prior, and the training only converges slowly. This again demonstrates that the choice of prior is crucial for successful training. In the following tests, we use the MG5aMC-like prior of Eq. (2.56).

Second, we study the impact of the physics-informed Breit-Wigner mapping for different

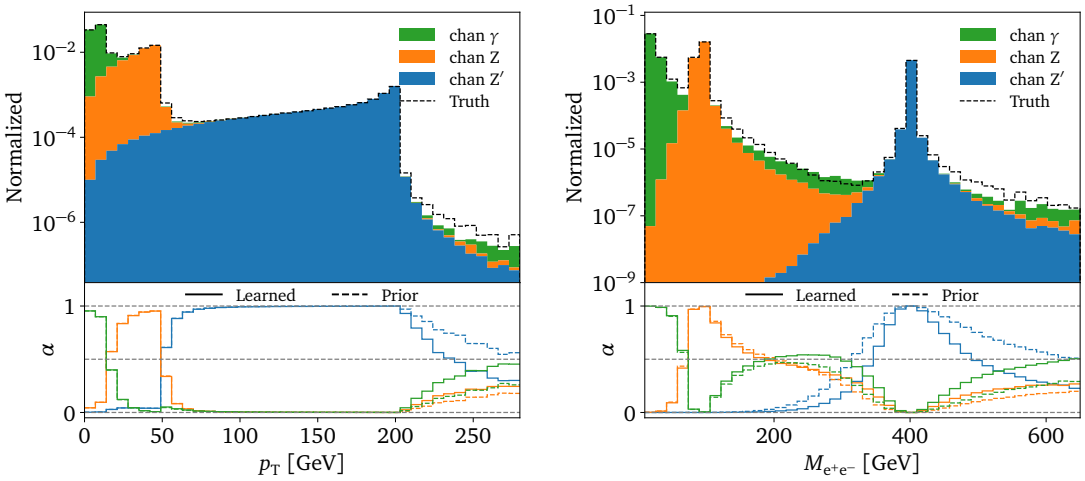


Figure 5.11: Learned p_T and $M_{e^+e^-}$ distributions for the Z′-extended Drell-Yan process. In the lower panels we show the learned and prior channel weights.

widths of the Z′ peak,

$$\tilde{\Gamma}_{Z'} = \Gamma_{Z'} \cdot \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\} \,, \tag{5.43}$$

while keeping the Z width constant. We train a MadNIS integrator with two channels for the Z and photon propagators and trainable channel mappings, and with three channels including a Z′ mapping with the appropriate width. We run the latter training with and without trainable channel weights. The relative errors of the phase-space integral are shown in the right panel of Fig. 5.10. It can be seen that the integration including a Z′ mapping consistently outperforms the two-channel integration, and that the relative error for the three-channel integrators is roughly constant for the different Z′ widths spanning five orders of magnitude. There is only a very small improvement from the trainable channel weights. This is because interferences are negligible for this process and the MG5aMC channel weights are already close to optimal. The relative error of the two channel integrator gets larger for smaller widths of the Z′ peak, until the peak is so narrow that it is missed during sampling. This leads to an overconfident estimate of the integration error and a large spread of the relative error between multiple MadNIS trainings.

Next, we show the learned distributions and local channel weights for a three-channel integrator starting from the MG5aMC prior as a function of the two phase-space observables p_T and $M_{e^+e^-}$ in Fig. 5.11. The stacked histograms show the contributions of each channel without reweighting. The dashed line for the truth distribution is a weighted histogram of the combined samples from all three channels. The learned distribution is close to the truth distribution even when the weights are neglected. The channel weight network only learns small corrections to the MG5aMC prior. Note that instead of spreading across the full phase space, every channel is active in one region of $M_{e^+e^-}$.

Lastly, we add buffered training to our MadNIS setup. Note that our modified Drell-Yan process is computationally cheap. Hence, buffered training will only lead to a small reduction of the training time of about 20% because of the smaller number of network evaluations during buffered training. Still, we can study whether the buffered training slows down the convergence of the training, or if it is detrimental to the integration

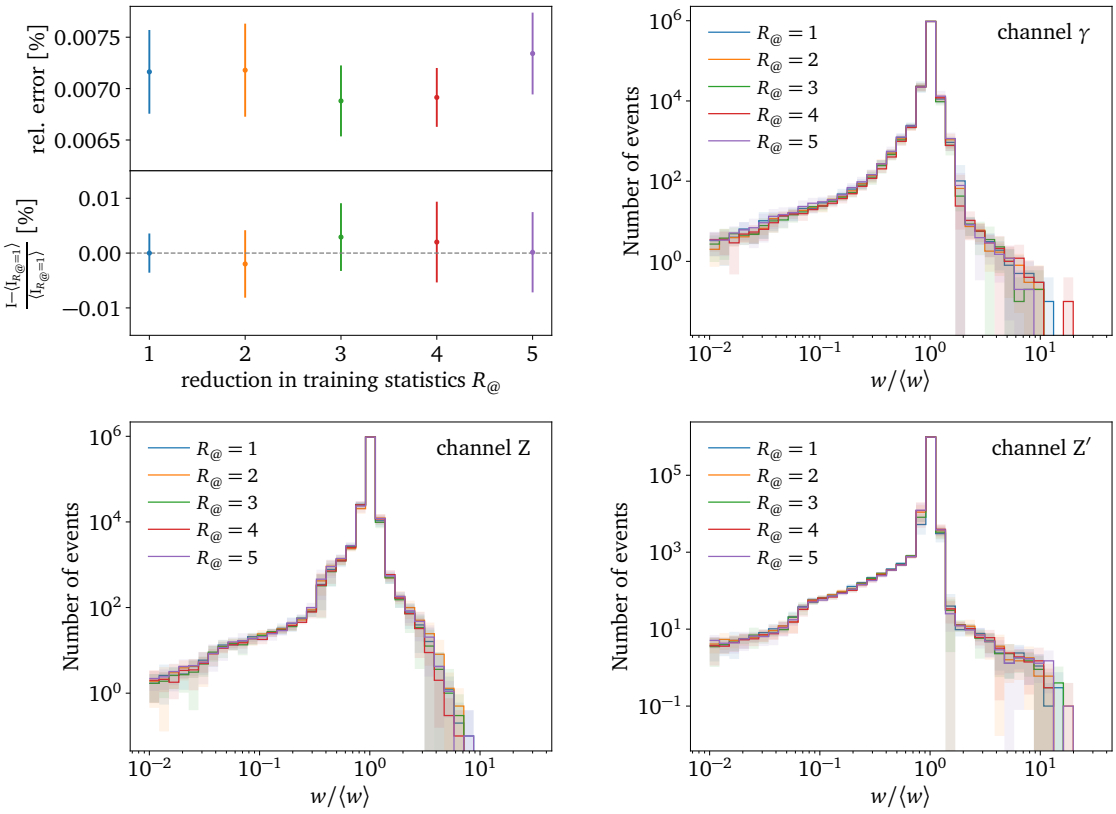


Figure 5.12: Relative integration error (from 10^6 events), relative deviation from the mean $R_@ = 1$ result, and weight distributions for different reduction factors $R_@$ in training statistics for the Z′-extended Drell-Yan process. The points/lines and error bars/bands show means and standard deviations over ten runs.

performance. We use a simple schedule for the buffered training where we first train the network online for one epoch, and save all samples generated during that epoch. Next, we run k_buff epochs of buffered training using the saved samples, shuffling them at the beginning of each epoch. Then, we discard the samples and start with the next online training epoch. We repeat this schedule $60/(k_\text{buff} + 1)$ times for a total of 60 epochs and run trainings with five different values of k_buff. The corresponding reductions in training statistics are

$$R_@ = k_\text{buff} + 1 \qquad \text{with} \qquad k_\text{buff} = 0, 1, 2, 3, 4\,. \tag{5.44}$$

The relative integration error, the relative deviation from the mean result for $R_@ = 1$ and the weight distributions for the three channels are shown in Fig. 5.12. The error bands are extracted by running ten independent MadNIS trainings for each value of $R_@$. Even for $R_@ = 5$, we find no significant deviations in the relative error or weight distributions compared to the pure online training. This shows that buffered training is a simple and effective measure to accelerate the training without compromising its performance.

5.3 Boosting MG5aMC

So far, we have only studied the MadNIS performance for simple and low-dimensional toy examples. As the next step, we move to the matrix elements and multi-channel setup implemented in MG5aMC. To this end, we implement an API that allows us to call MG5aMC from MadNIS. The inputs to this API are vectors of random numbers r_s with $s = 1 \dots D$ and the index $i = 1 \dots n_c$ of the channel used to sample the event. For each event, MG5aMC returns its four-momenta p, the event weight w, and a vector of channel weights α_j^{MG} with $j = 1 \dots n_c$. At the time of writing this thesis, this API is not yet fully optimized as MG5aMC was not originally intended to allow for fast switching between channels. Hence, we do not show runtime comparisons, but compare unweighting efficiencies instead. Furthermore, we only look at processes with fixed partonic initial states for simplicity. We move from the simplified MadNIS setup used in the previous section to the stratified loss function from Eq. (5.20). The hyperparameters used for our benchmark study are given in Tab. A.6.

5.3.1 Reference processes

We benchmark the MadNIS performance for a set of LHC processes,

$$\begin{aligned}
&\text{Triple-W} && \mathrm{u\bar{d}} \to \mathrm{W^+W^+W^-} && && \\
&\text{VBS} && \mathrm{uc} \to \mathrm{W^+W^+\,ds} && && \\
&\text{W+jets} && \mathrm{gg} \to \mathrm{W^+d\bar{u}} && \mathrm{gg} \to \mathrm{W^+d\bar{u}g} && \mathrm{gg} \to \mathrm{W^+d\bar{u}gg} \\
&\text{t}\bar{\text{t}}\text{+jets} && \mathrm{gg} \to \mathrm{t\bar{t}g} && \mathrm{gg} \to \mathrm{t\bar{t}gg} && \mathrm{gg} \to \mathrm{t\bar{t}ggg}\,,
\end{aligned} \tag{5.45}$$

where we do not consider further decays of the heavy particles. Triple-W production and vector-boson scattering (VBS) are included because they have a large number of gauge-related Feynman diagrams with potentially large interferences. These are especially challenging when a multi-channel decomposition based on the assumption of small interference terms is used, see Sec. 2.2.4. In such cases, there could be a large potential for improvements from trainable channel weights. Furthermore, we include W+jets and t$\bar{\text{t}}$+jets production to study the scaling with additional jets. We show the number of Feynman diagrams and the number of channels constructed by MG5aMC in Tab. 5.3. The number of channels is lower than the number of diagrams because MG5aMC does not build a separate channel for every diagram. Examples for diagrams where no channel is constructed include t-channel Z propagators (as there always is a similar diagram with a photon), and diagrams with four-point vertices. The number of independent channels is further reduced by grouping channels that only differ by a permutation of the final state momenta. The last column shows the number of channels that remain active after a MadNIS training with channel dropping, as discussed in Sec. 5.3.3.

5.3.2 Benchmarking MadNIS features

We benchmark the various MadNIS features introduced in Sec. 5.1 by applying them to our reference processes. Our two metrics for this comparison are the relative standard deviation σ/I minimized by stratified sampling and the unweighting efficiency ϵ as defined in Ref. [47], see Sec. 2.2.5. Note that the relative standard deviation differs from the relative integration error used in a previous section as it is independent of the number of

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

Table 5.3: Number of Feynman diagrams, channels and channel groups after accounting for symmetries. The last column shows the number of channels that remain active after MadNIS channel dropping. Its range reflects ten independent trainings.

samples used to compute the integral. We capture the variations between trainings by running ten independent trainings for each setup and computing the means and standard deviations of the two performance metrics. Further, we confirm that the cross section computed by the integrator, as well as the phase-space distribution of the generated samples, are compatible with the MG5aMC output.

We show the results for W+2jets, W+3jets, VBS and Triple-W in Fig. 5.13. The leftmost point shows the result of a Vegas training. To ensure a fair comparison, we use a longer Vegas training with more training statistics than used in MG5aMC. We use this point as a baseline for the following results. Because Vegas assumes a factorized integrand and does not rely on gradient descent, its training is much faster than a MadNIS training. Choosing between the two methods is a tradeoff between a fast training and efficient sampling. For cheap integrands and low numbers of samples, Vegas-based importance sampling will be faster overall. Also note that the cost of training networks is not amortized for the integration alone, as the training requires more samples than needed for a precise estimate of the integral. In situations where the upfront cost of the training and integration becomes negligible, the unweighting efficiency is proportional to the total run time. MadNIS is best applied in such situations, so we focus on the unweighting efficiency as a performance metric.

The following points in Fig. 5.13 successively include more MadNIS features. In analogy to the results shown for the Drell-Yan process in Fig. 5.10, we start with a training of the channel mappings where the channel weights are kept fixed to the MG5aMC output. We observe a large gain both in unweighting efficiency and relative standard deviation for all processes compared to the Vegas baseline. The gain for the unweighting efficiency is much larger than for the relative standard deviation. We observe this for most processes except for Triple-W production. The reason for this effect is the higher sensitivity of the unweighting efficiency to the tails of the weight distribution. As a next step, we run joint trainings of channel mappings and weights, and observe a further large improvement. For instance, the unweighting efficiency for VBS is around 8 times larger than for the standard method without trained channel weights and more than 10 times larger if the weight training is enabled.

Next, we study the impact of the Vegas-initialization from Sec. 5.1.3. For all processes, there are some improvements compared to the trainings starting from a uniform initialization, both with and without adaptive channel weights. We see a much larger performance

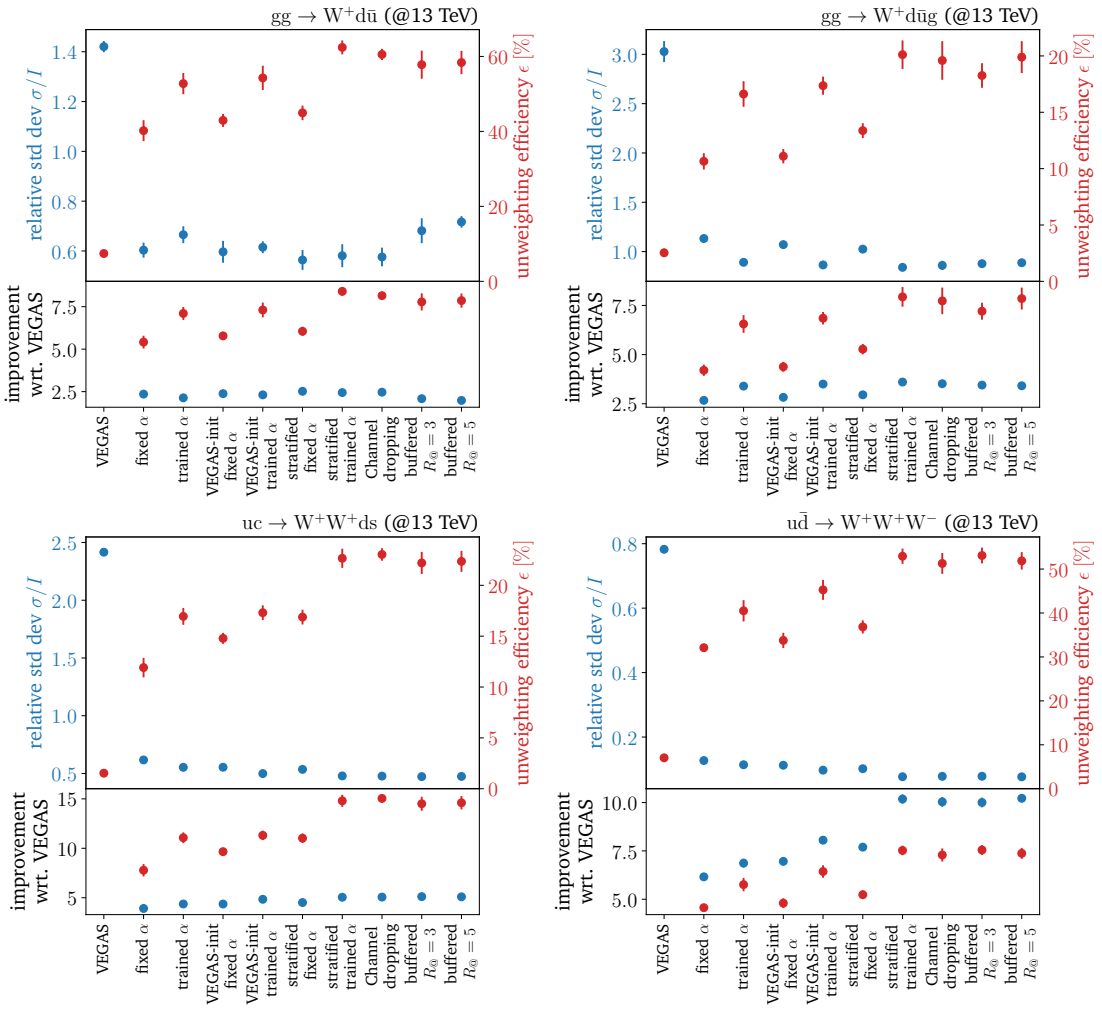


Figure 5.13: Relative standard deviation and unweighting efficiency for W+2jets, W+3jets, VBS and Triple-W for various combinations of MadNIS features.

gain when we also use stratified training (see Sec. 5.1.2) instead of evenly distributing the training samples among channels. Stratified training has the strongest effect when it is combined with adaptive channel mappings. In this setup, the unweighting efficiency for VBS increases by a factor 15 compared to the Vegas baseline.

In Fig. 5.12, we saw that buffered training had no negative impact on the performance for our Drell-Yan toy process. To confirm that this is still true for more challenging processes, we test our setup with two different reductions in training statistics, $R_{@} = 3$ and $R_{@} = 5$. We see that the training remains stable, even for large $R_{@}$. Furthermore, we also show the effect of channel dropping, as introduced in Sec. 5.1.2, on the performance. For all processes, there is no significant performance difference from channel dropping. For processes with an even larger number of channels, we find that channel dropping leads to a significant improvement in training stability. Hence, we use the full MadNIS setup including adaptive channel weights, Vegas initialization, stratified training, channel dropping and buffered training with $R_{@} = 5$ for the following results.

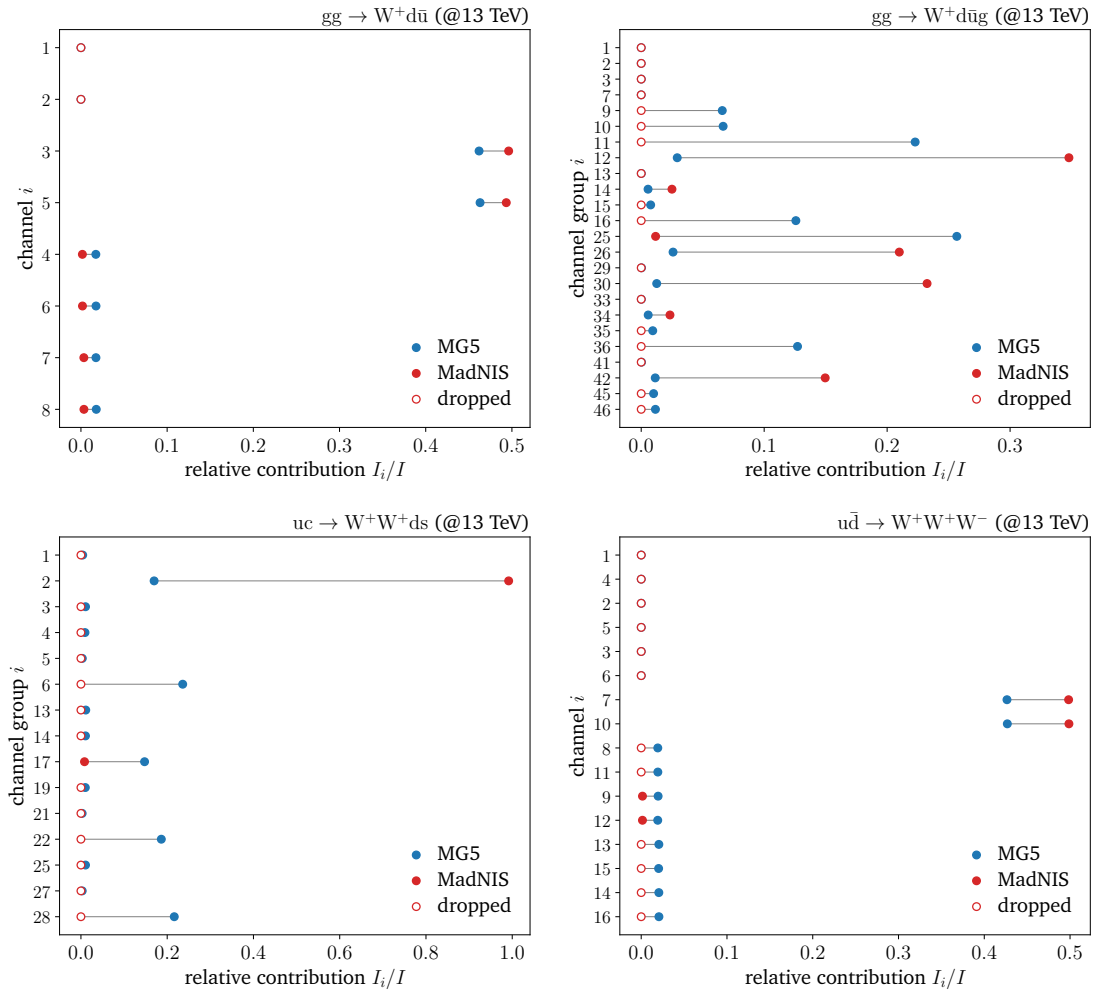


Figure 5.14: Relative contributions of the channels for W+2jets and Triple-W production, and for the channel groups for W+3jets and VBS. We show the channel weights defined by MG5aMC, and the learned channel weights from MadNIS. Empty circles indicate dropped channels.

5.3.3 Trained channel weights

In the previous section, we saw that using trainable channel weights led to a large performance gain. To take a closer look at these channel weights, we compute the contribution of each channel to the total cross section,

$$\frac{I_i}{I} = \frac{\int \mathrm{d}^D x \; \alpha_i(x) \, f(x)}{\int \mathrm{d}^D x \; f(x)} \,, \tag{5.46}$$

While the weights of channels within a group connected by permutations of the final state have different behaviors locally, we find that their contributions to the total cross section change coherently. For W+3jets and VBS, where the number of channels is larger, we therefore compute the group-wise sums of the channel contributions. We show the contributions for four processes in Fig. 5.14. The blue circles show the results using the initial channel weights from MG5aMC. The red circles show the contributions after the channel weights were adapted by MadNIS, with empty circles indicating dropped

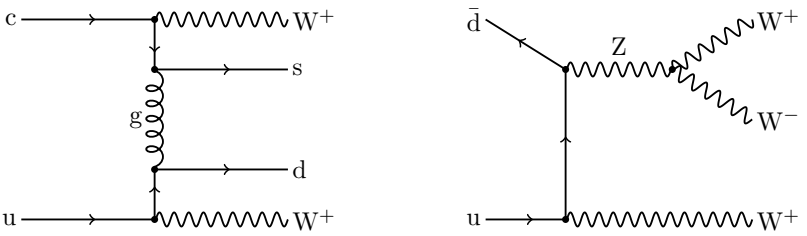


Figure 5.15: Feynman diagrams corresponding to the dominant channels after training MadNIS for VBS (left) and Triple-W production (right).

channels. We find that MadNIS prefers fewer channels, illustrating the benefit of our channel dropping feature.

For VBS and Triple-W production, MadNIS even chooses to adapt the channel weights such that the cross section is almost completely made up from a single group of symmetry-related channels. The choice of channels is consistent between repetitions of the training, and the corresponding Feynman diagrams are shown in Fig. 5.15. The diagram shown in the left panel corresponds to one of the five groups of channels that significantly contribute to the cross section for VBS. Four of those channels have a t-channel gluon propagator and one has a t-channel photon propagator. The diagram enhanced by MadNIS does not have an s-channel quark propagator. For Triple-W production, the diagram shown in the right panel corresponds to the channel group with the largest contribution in MG5aMC, and it is further enhanced by MadNIS.

5.3.4 Scaling with jet multiplicity

Neural importance sampling is most beneficial for very costly integrands. The computational cost of matrix elements scales with the number of Feynman diagrams, which in turn increases when additional jets are included in the final state. We study the performance scaling of MadNIS for W+jets and $t\bar{t}$+jets production by adding gluons to the final state. We show the relative standard deviation σ/I and the unweighting efficiency ϵ for trainings with the full MadNIS setup and compare it to the Vegas baseline in Fig. 5.16. For both Vegas and MadNIS, the unweighting efficiency decreases and the integration

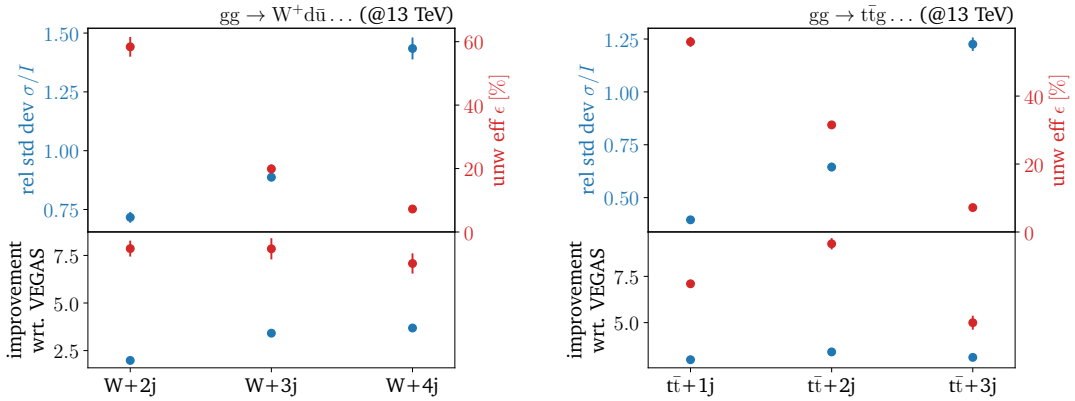


Figure 5.16: Relative standard deviation and unweighting efficiency for W+jets and $t\bar{t}$+jets with different numbers of gluons in the final state. The final MadNIS performance gain is illustrated in the lower panels, just as in Fig. 5.13.

error increases towards higher multiplicities. However, we can see that the gain from MadNIS over Vegas stays roughly constant between 7 and 8 for W+jets. While we see a decrease in gain going from $t\bar{t}$+2 jets to $t\bar{t}$+3 jets, a gain of 5 is still a very promising result as it directly translates to a speed-up for expensive processes like $t\bar{t}$+3 jets.

5.4 Conclusion

In this chapter we introduced MadNIS, a tool for neural importance sampling and multi-channel integration. It is based on invertible neural networks that replace the simpler adaptive transformations used in conventional phase-space generators. This is combined with a multi-channel strategy where the channel decomposition is refined using trainable channel weights encoded by a fully-connected network. The training samples for the networks are generated during the training process, but we proposed buffered training as an approach to reuse these samples and reduce the number of costly integrand evaluations. In addition, we introduced several features to improve the training process, like Vegas initialization, stratified training and channel dropping.

We first studied the performance of the MadNIS method for simple toy examples and found that the network was able to extract the optimal channel mappings and split a complicated topological structure into easy-to-learn substructures. However, the best results were achieved in combination with problem-specific analytic mappings. We confirmed this for a simple physics example, a Drell-Yan process with an additional Z' resonance. Further, we showed that buffered training can be used to reduce the number of matrix element evaluations without degrading the integration performance.

Next, we implemented an interface between MadNIS and MG5aMC that allows us to use the matrix elements, channel mappings and channel weights constructed by MG5aMC. We benchmarked the performance gain from MadNIS compared to the standard Vegas algorithm for W+2,3,4 jets, VBS, Triple-W and $t\bar{t}$+1,2,3 jets. They combine large numbers of gauge-related Feynman diagrams with a large number of particles in the final state. We showed that the trained channel weights, Vegas initialization and stratified trainings each led to a improvement in the integration error and unweighting efficiency. Buffered training and channel dropping made the training efficient and stable for complicated processes with many channels. We found improvements in unweighting efficiency between 5 and 15 compared to the standard approach.

Chapter 6

ML for the matrix element method

The research presented in this chapter is based on work in collaboration with Anja Butter, Nathan Huetsch, Till Martini, Sascha Peitzsch, Tilman Plehn and Ramon Winterhalder, and has been previously published in Ref. [2] and Ref. [5]. Most of the tables and figures as well as parts of the text are similar or identical to the content of these articles. In particular, Secs. 6.1 and 6.3 are based on Ref. [2], and Secs. 6.2, 6.4 and 6.5 are based on Ref. [5].

In this chapter, we turn from event generation to ML-assisted measurements of theory parameters. One of the disadvantages of classical analysis strategies at the LHC is that they typically rely on binned data from a small number of hand-crafted high-level observables. This can lead to a loss of information. Machine learning can help to make better use of the available data, as it is able to work with high-dimensional inputs and extract complex correlations. These techniques are summarized under the term simulation-based inference [200] (SBI). Some SBI-methods are based on Bayesian inference, and use neural networks for posterior estimation. Example applications include measuring theory parameters of parton showers [24] or kinematic reconstruction [30]. A different class of SBI-methods learns the likelihood ratio using classifier or regression networks [23]. According to the Neyman-Pearson lemma, the likelihood ratio is the optimal test statistic to decide between two hypotheses. Hence, these methods make optimal use of the available data in the limit of a perfectly trained and sufficiently expressive network.

One common disadvantage of most SBI methods is that they do not make use of any theory knowledge at inference time. For a theory parameter that enters the Lagrangian, the likelihood at the hard-scattering level is given by the differential cross-section and therefore, the matrix element. Hence, the likelihood ratio for the theory parameter can be computed from first principles. This is the idea behind the matrix element method (MEM) [121, 122]. It was used in the top mass measurement [201–204] and the discovery of single-top production [205] at the Tevatron. There are also several studies [206–212] and analysis applications [211, 213–216] at the LHC. As the likelihood is only known analytically at the hard-scattering level, the matrix element method uses transfer functions that model the effects of shower, hadronization, detector and reconstruction. This makes it necessary to integrate out the space of possible parton-level configurations for every measured event to obtain the likelihood at the reconstruction level. This integral has to be computed for all possible combinatorics from the hard and reconstructed objects, making the MEM computationally costly. Furthermore, the form of the transfer function is not known from first principles, so simple smearing functions are often assumed.

It the following we show how machine learning can be used to address these problems. Previous work on ML-applications for the matrix element method was focused on neural network surrogates to replace the slow phase-space integration [217] while still relying on a simple transfer function. We demonstrate that generative models can be used to learn the transfer function from data, and keep the phase-space integration efficient using neural importance sampling. Furthermore, we show how acceptance effects can be learned with classifier networks. As our benchmark LHC process, we use associated Higgs and single-top production.

We start with the description of our reference process in Sec. 6.1. Next, we describe how ML can be applied in the matrix element method in Sec. 6.2. We then first look at a simplified ML-MEM setup based on two invertible neural networks to model the transfer function and for importance sampling in Sec. 6.3. We start with the simple case of a leptonically decaying top and then move on to hadronic decays with and without initial state radiation. There are three main issues with this simple approach. The first issue is the missing treatment of acceptance effects, which we solve with classifier networks in Sec. 6.4.2. Second, the simple INN does not provide a sufficient level of precision to learn the transfer function, so we replace it with a diffusion model in Sec. 6.5.1. Finally, we show how transformer networks can be used to tackle jet combinatorics in Sec. 6.5.2.

6.1 LHC process

As a reference process to benchmark our machine learned matrix element method, we choose associated single-top and Higgs production

$$pp \to tHj \ , \tag{6.1}$$

as it will allow us to measure the CP phase of the top Yukawa coupling in future LHC runs [218–226]. The three leading-order Feynman diagrams for this process are shown in Fig. 6.1, where we neglect the second diagram in the limit of a massless bottom quark in the initial state. We choose the decay $H \to \gamma\gamma$. This allows us to focus on the signal process and we do not have to include continuum backgrounds. We express the Lagrangian of the top-Higgs interaction as a mixture of CP-even and CP-odd interactions [227],

$$\mathscr{L}_{t\bar{t}H} = -\frac{y_t}{\sqrt{2}}\Big[a\cos\alpha\ \bar{t}t + ib\sin\alpha\ \bar{t}\gamma_5 t\Big]H \ , \tag{6.2}$$

with $a = 1$, $b = 2/3$ and the CP-phase α. $\alpha = 0°$ corresponds to a CP-even and $\alpha = 180°$ to CP-odd Yukawa coupling. The parameters a and b are chosen such that the total

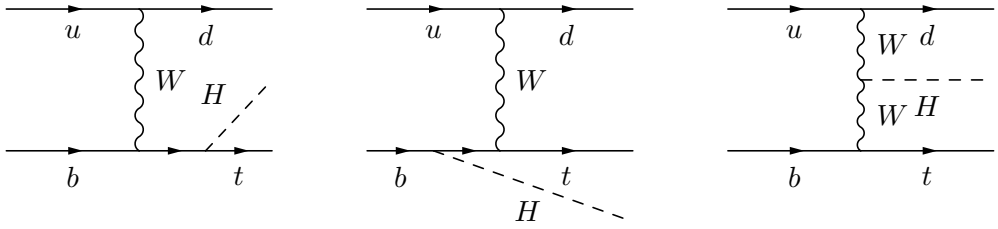


Figure 6.1: Leading-order Feynman diagrams for the hard process $pp \to tHj$. We neglect the second diagram in the limit of a massless bottom quark. The diagrams also appear with an inverted light-quark line.

Dataset	cut	rate [ab]	fraction
leptonic	σ	$43.6 \cdot 10^3$	
	$\sigma \times$ BR	7.38	
	≥ 2 photons with $p_T > 20$ GeV and $\eta < 2.4$	3.58	0.485
	≥ 1 muon with $p_T > 20$ GeV and $\eta < 2.4$	2.29	0.310
	≥ 2 jets	1.69	0.230
	1 b-jet with $p_T > 25$ GeV and $\eta < 2.4$	1.00	0.136
	≥ 1 jets with $p_T > 25$ GeV and $\eta < 2.4$	0.41	0.055
hadronic, no ISR	σ	$43.6 \cdot 10^3$	
	$\sigma \times$ BR	44.28	
	≥ 2 photons with $p_T > 20$ GeV and $\eta < 2.4$	19.56	0.442
	≥ 4 jets	7.09	0.160
	1 b-jet with $p_T > 25$ GeV and $\eta < 2.4$	3.93	0.089
	≥ 3 jets with $p_T > 25$ GeV and $\eta < 2.4$	1.23	0.028
hadronic, with ISR	σ	$43.6 \cdot 10^3$	
	$\sigma \times$ BR	44.26	
	≥ 2 photons with $p_T > 20$ GeV and $\eta < 2.4$	18.37	0.415
	≥ 4 jets	12.67	0.286
	1 b-jet with $p_T > 25$ GeV and $\eta < 2.4$	6.44	0.146
	≥ 3 jets with $p_T > 25$ GeV and $\eta < 2.4$	3.06	0.069

Table 6.1: Cut flow for $pp \to tHj$ with $H \to \gamma\gamma$ and for SM events ($\alpha = 0°$). We assume $m_b = 0$ and intermediate on-shell particles.

$gg \to H$ cross section remains constant when α is varied. From Eq. (6.2) we can see that any observable O obtained by integrating over hard-scattering phase space has the functional form

$$O(\alpha) = A + B(1 - \cos\alpha) + \sin\alpha\,(C\sin\alpha + D + E\cos\alpha) \ , \tag{6.3}$$

which allows us to express likelihoods or the fiducial cross-section using only five parameters.

Data samples

To train neural networks, we need paired events consisting of the CP-phase α, the hard-scattering momenta x_{hard}, and the reconstruction-level momenta x_{reco}. For a given α we run MG5aMC, v3.1.0, with LO-NNPDF and $\alpha_s = 0.119$ [228]. We assume the incoming b-quark to be massless. We define x_{hard} as the momenta of the top, Higgs and light quark on their respective mass shells and decay them in a second step. We then simulate the parton shower with Pythia8 [34] and detector effects with Delphes [39]. Next, we use FastJet [194] to reconstruct anti-k_T jets with a jet radius of 0.4.

We generate three different datasets. After the top decays into a b and a W, the W either decays leptonically into a muon and a neutrino,

$$pp \to tHj \to (b\mu^+\nu_\mu)\,(\gamma\gamma)\,j \ , \tag{6.4}$$

Dataset	A [fb]	B [fb]	C [fb]	D [fb]	E [fb]
leptonic	$4.07 \cdot 10^{-4}$	$2.37 \cdot 10^{-3}$	$-1.22 \cdot 10^{-3}$	$1.86 \cdot 10^{-6}$	$-2.90 \cdot 10^{-7}$
hadronic, no ISR	$1.23 \cdot 10^{-3}$	$7.59 \cdot 10^{-3}$	$-3.78 \cdot 10^{-3}$	$1.24 \cdot 10^{-5}$	$-7.96 \cdot 10^{-6}$
hadronic, with ISR	$3.06 \cdot 10^{-3}$	$2.05 \cdot 10^{-2}$	$-9.50 \cdot 10^{-3}$	$1.90 \cdot 10^{-5}$	$-5.77 \cdot 10^{-6}$

Table 6.2: Fit parameters for the fiducial cross sections, for the formula given in Eq. (6.3).

or hadronically into two jets,

$$pp \to tHj \to (bjj)\ (\gamma\gamma)\ j\ . \tag{6.5}$$

In both cases, we allow for additional reconstructed jets from final state radiation, but do not include initial state radiation or multi-parton interactions. In the third dataset, we again consider hadronic decays of the top, and include the effects of initial state radiation, resulting in additional jets,

$$pp \to tHj \to (bjj)\ (\gamma\gamma)\ j + \text{QCD jets}\ . \tag{6.6}$$

In all cases, we allow for up to four additional jets from final state radiation, or from initial state radiation in the case of the third dataset. We always combine top and anti-top production, resulting in a total cross section of 43.6 fb. We do not apply cuts in p_T or η at the hard-scattering level, but apply a series of cuts during reconstruction. The cut flow for the three datasets is shown in Tab. 6.1.

Every data point then consists of the CP-phase α, the four-momenta at the hard-scattering level,

$$x_{\text{hard}} = (p_t, p_H, p_q)\ , \tag{6.7}$$

and the reco-level four-momenta,

$$\begin{aligned} x_{\text{reco}} &= (p_{\gamma,1}, p_{\gamma,2}, p_b, p_\mu, p_{j,1}, \dots) && \text{leptonic decay} \\ x_{\text{reco}} &= (p_{\gamma,1}, p_{\gamma,2}, p_b, p_{j,1}, p_{j,2}, p_{j,3}, \dots) && \text{hadronic decay}\ . \end{aligned} \tag{6.8}$$

The photon momenta and light jet momenta are separately ordered by p_T. The dots indicate additional light jet momenta from FSR or ISR. As we define the hard-scattering level before any initial state radiation is added, we do not need to perform a boost into the

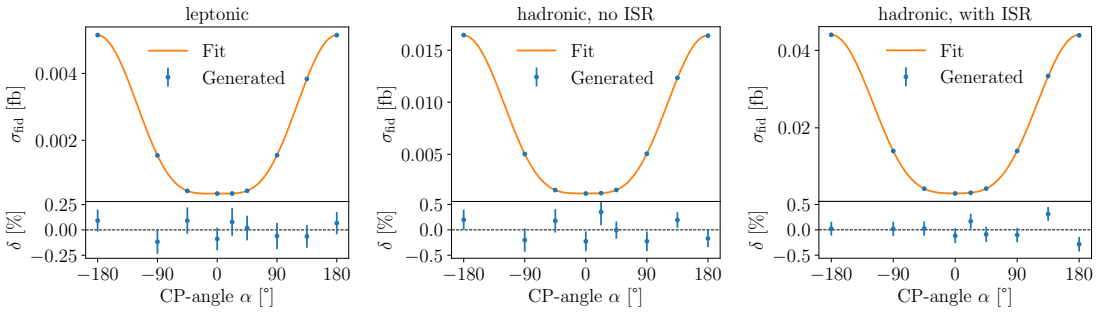


Figure 6.2: Fiducial cross section including decays and after cuts as a function of the CP-angle α. The lower panels illustrate the agreement between the generated data and the fitted continuous function defined in Eq. (6.3) and Tab. 6.2.

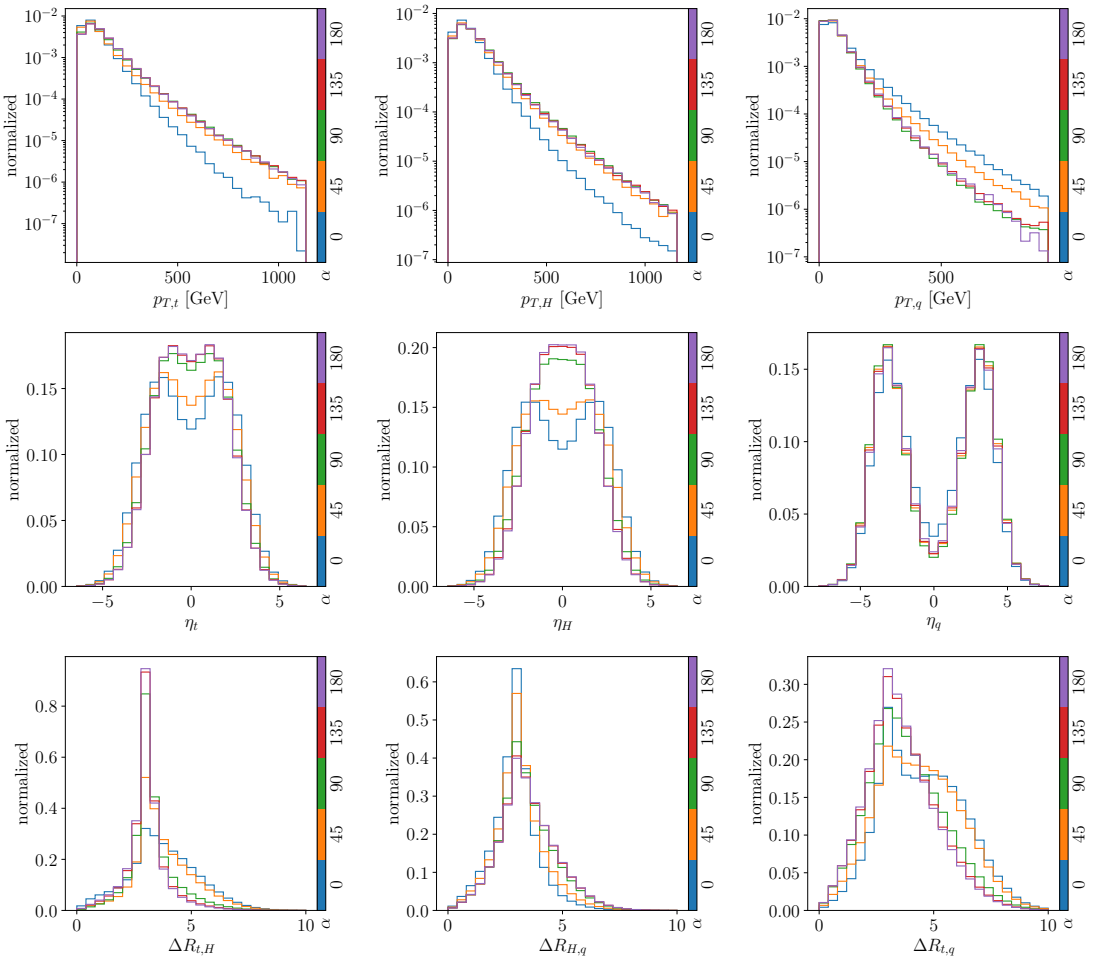


Figure 6.3: Kinematic distributions for the hard-scattering tHj final state for different values of the CP-angle.

hard-scattering rest frame. The two training datasets without ISR contain 1.3M paired events with α drawn from a uniform distribution. The third dataset contains 3.4M events to address the more challenging training task due to initial state radiation. In all cases, the test datasets contain 260k events for each of the angles $\alpha \in \{0°, 45°, 90°, 135°, 180°\}$.

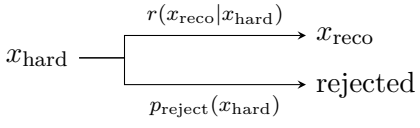
We simulate events for the CP-angles $\alpha = -180°, -90°, -45°, 0°, 22.5°, 45°, 90°, 135°, 180°$ and then perform a fit of the fiducial cross section including decays after cuts using Eq. (6.3). The fit parameters for the three different datasets are shown in Tab. 6.2. We find that the fiducial cross section is almost symmetric under a sign change in α, as $D, E \ll A, B, C$. We show the fiducial cross section for the set of CP-angles listed above as well as the fitted function in Fig. 6.2. It can be seen that the cross section around the Standard Model value ($\alpha = 0°$) is below 0.01 fb and almost flat for small $\alpha \lesssim 40°$, caused by destructive interference between the left and right diagram in Fig. 6.1. This turns into constructive interference as α gets further away from the Standard Model value. We show the distributions for the hard-scattering tHj kinematics in Fig. 6.3. Unlike for the rate, we can see that there is significant change in the shape of the kinematic distributions between $\alpha = 0°$ and $\alpha = 45°$. Because of the low rates, we have to include the kinematic information into our analysis to make optimal use of the available data, making this process a perfect example application for the matrix element method.

6.2 ML-matrix element method

As we already saw in Sec. 2.1.2, the differential cross section of a hard scattering process can be interpreted as a probability distribution when it is normalized by the total cross section. Similarly, the differential cross section for a given parameter of interest α can be interpreted as the likelihood of an event given the parameter,

$$\frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_\text{hard}} = \sigma(\alpha)\, p(x_\text{hard}|\alpha) \qquad \Leftrightarrow \qquad p(x_\text{hard}|\alpha) = \frac{1}{\sigma(\alpha)} \frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_\text{hard}} \,. \tag{6.9}$$

The hard-scattering momenta are not observed directly. Instead, we have to take into account the effects of parton shower, hadronization, detector and reconstruction. Because of the detector geometry and acceptance cuts performed during reconstruction, not every x_hard will result in a valid reco-level event x_reco. As the reconstruction level data before cuts is a very complex object that would be hard to learn for a neural network, we instead model the forward process in terms of a rejection probability $p_\text{reject}(x_\text{hard})$ and a forward-transfer or response function r [229]. Any hard scattering event is then either rejected, or it will be associated with a valid set of reco-level momenta x_reco,



(6.10)

We assume that the transfer function and the rejection probability are independent of the theory parameter α. Note that our method does not require us to make that assumption, but it is a very good approximation in many cases and allows for several numerical optimizations, see Sec. 6.4.1. The transfer function r is not normalized, and its normalization defines the efficiency or acceptance function,

$$\epsilon(x_\text{hard}) := \int dx_\text{reco}\; r(x_\text{reco}|x_\text{hard}) = 1 - p_\text{reject}(x_\text{hard}) \,. \tag{6.11}$$

We can then obtain the differential cross section at the reconstruction level using the transfer function,

$$\frac{\mathrm{d}\sigma_\text{fid}(\alpha)}{\mathrm{d}x_\text{reco}} = \int \mathrm{d}x_\text{hard}\; r(x_\text{reco}|x_\text{hard}) \frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_\text{hard}} \,, \tag{6.12}$$

where the subscript 'fid' indicates that the reco-level phase space is different from the hard-scattering level. We can then replace the transfer function r with the normalized transfer probability $p(x_\text{reco}|x_\text{hard})$ using Eq. (6.11),

$$r(x_\text{reco}|x_\text{hard}) = \epsilon(x_\text{hard})\, p(x_\text{reco}|x_\text{hard}) \qquad \text{with} \qquad \int \mathrm{d}x_\text{reco}\; p(x_\text{reco}|x_\text{hard}) = 1 \,. \tag{6.13}$$

Next, we insert Eq. (6.13) in Eq. (6.12) to obtain the final expression for the differential cross section,

$$\frac{\mathrm{d}\sigma_\text{fid}(\alpha)}{\mathrm{d}x_\text{reco}} = \int \mathrm{d}x_\text{hard}\; \epsilon(x_\text{hard})\, p(x_\text{reco}|x_\text{hard}) \frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_\text{hard}} \,. \tag{6.14}$$

Since we are interested in the likelihood of the reco-level events given a theory parameter, we need a normalized version of this differential cross section in analogy to Eq. (6.9),

$$\frac{\mathrm{d}\sigma_\text{fid}(\alpha)}{\mathrm{d}x_\text{reco}} = \sigma_\text{fid}(\alpha)\, p(x_\text{reco}|\alpha) \qquad \Leftrightarrow \qquad p(x_\text{reco}|\alpha) = \frac{1}{\sigma_\text{fid}(\alpha)} \frac{\mathrm{d}\sigma_\text{fid}(\alpha)}{\mathrm{d}x_\text{reco}} \,. \tag{6.15}$$

We can compute the fiducial cross section $\sigma_{\text{fid}}(\alpha)$ by integrating Eq. (6.14) over the reco-level phase space,

$$\begin{aligned}
\sigma_{\text{fid}}(\alpha) &= \int \mathrm{d}x_{\text{reco}} \int \mathrm{d}x_{\text{hard}}\; \epsilon(x_{\text{hard}})\; p(x_{\text{reco}}|x_{\text{hard}})\; \frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_{\text{hard}}} \\
&= \int \mathrm{d}x_{\text{hard}}\; \epsilon(x_{\text{hard}})\; \frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_{\text{hard}}} \\
&= \sigma(\alpha) \int \mathrm{d}x_{\text{hard}}\; \epsilon(x_{\text{hard}})\; p(x_{\text{hard}}|\alpha) \\
&= \sigma(\alpha)\; \big\langle \epsilon(x_{\text{hard}}) \big\rangle_{x\sim p(x_{\text{hard}}|\alpha)} \,,
\end{aligned} \tag{6.16}$$

where we used the normalization condition from Eq. (6.13) to integrate out the reco-level phase space, and then replaced the differential cross section using Eq. (6.9). This allows us to express the integral in terms of the average acceptance $\langle\epsilon\rangle_\alpha$ which is used to compute the fiducial cross section numerically, for example to obtain the results shown in Tab. 6.1. Inserting Eq. (6.14) into Eq. (6.15) yields the final expression for the reco-level likelihood,

$$p(x_{\text{reco}}|\alpha) = \frac{1}{\sigma_{\text{fid}}(\alpha)} \int \mathrm{d}x_{\text{hard}}\; \frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_{\text{hard}}}\; \epsilon(x_{\text{hard}})\; p(x_{\text{reco}}|x_{\text{hard}}) \,. \tag{6.17}$$

The training data for our neural networks consists of paired events $(x_{\text{reco}}, x_{\text{hard}}, \alpha)$, so it only contains events that have passed the phase-space cuts. Hence, the marginal distribution of the x_{hard} is different from the distribution defined by the differential cross section of the hard process, Eq. (6.9). The modified distribution of the accepted x_{hard} is given by

$$p_{\text{fid}}(x_{\text{hard}}|\alpha) = \frac{1}{\sigma_{\text{fid}}(\alpha)}\; \frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_{\text{hard}}}\; \epsilon(x_{\text{hard}}) \,. \tag{6.18}$$

Consequently, the reco-level likelihood and the modified parton-level likelihood are related by a completeness relation,

$$p(x_{\text{reco}}|\alpha) = \int \mathrm{d}x_{\text{hard}}\; p(x_{\text{reco}}|x_{\text{hard}})\; p_{\text{fid}}(x_{\text{hard}}|\alpha) \,. \tag{6.19}$$

Acceptance classifier and transfer network

The transfer probability $p(x_{\text{reco}}|x_{\text{hard}})$ and acceptance $\epsilon(x_{\text{hard}})$ needed to compute the reco-level likelihood using Eq. (6.17) are not analytically tractable and only defined through a forward simulation. We encode both functions in neural networks trained on these simulations.

We extract the acceptance from the data using a classifier network

$$x_{\text{hard}} \xrightarrow{\text{Acceptance network}} \epsilon_\psi(x_{\text{hard}}) \,, \tag{6.20}$$

with trainable parameters ψ. The classifier is trained on a dataset containing the momenta x_{hard} of both accepted (label 1) and rejected (label 0) events. The network is trained with a binary cross entropy loss. As explained in Sec. 3.1.3, this means that the output of the network will be the acceptance probability for a given input event.

The transfer probability introduced in Eq. (6.13) is encoded in a generative network. To compute the integral in Eq. (6.17), the transfer probability has to be available as a numerical value and not only through samples. The generative architecture therefore

needs to be able to perform density estimation. Examples for such networks include cINNs and CFMs. The network is trained on a data set with event pairs $(x_\text{reco}, x_\text{hard})$, where we only include accepted events. The trained network then defines a bijective mapping between random numbers and reco-level phase space conditioned on parton-level events,

$$x_\text{reco} \sim p_\theta(x_\text{reco}|x_\text{hard}) \quad \overset{\text{Transfer network}}{\longleftrightarrow} \quad r \sim p_\text{latent}(r) \; , \tag{6.21}$$

with trainable parameters θ. This mapping is evaluated in the forward direction for density estimation and in the inverse direction to sample from the transfer probability.

Sampling-cINN

We compute the integral in Eq. (6.19) using Monte Carlo integration, as introduced in Sec. 2.2.1. It is challenging because the differential cross section spans several orders of magnitude, and the transfer probability typically forms a narrow peak. Therefore, we have to find a proposal distribution to sample $x_\text{hard} \sim q(x_\text{hard}|x_\text{reco}, \alpha) \equiv q(x_\text{hard})$ such that the variance of the integral is minimized. Classically, this integral is computed by constructing analytic mappings that take the specific structure of the matrix element and transfer probability into account. As the transfer probability is only available through a generative network, we instead use neural importance sampling. The proposal distribution then also has to be extracted from the training data.

We start by writing the phase-space integral as an expectation value,

$$\begin{aligned} p(x_\text{reco}|\alpha) &= \int \mathrm{d}x_\text{hard}\; p_\text{fid}(x_\text{hard}|\alpha)\; p_\theta(x_\text{reco}|x_\text{hard}) \\ &= \left\langle \frac{1}{q(x_\text{hard})}\; p_\text{fid}(x_\text{hard}|\alpha)\; p_\theta(x_\text{reco}|x_\text{hard}) \right\rangle_{x_\text{hard}\sim q(x_\text{hard})} \; , \end{aligned} \tag{6.22}$$

Ideally, our network surrogate for the transfer probability has learned the phase-space density perfectly,

$$p_\theta(x_\text{reco}|x_\text{hard}) = p(x_\text{reco}|x_\text{hard}) \; . \tag{6.23}$$

We can then use Bayes theorem to rewrite the the integral as

$$\begin{aligned} p(x_\text{reco}|\alpha) &= \left\langle \frac{1}{q(x_\text{hard})}\; p_\text{fid}(x_\text{hard}|\alpha)\; p(x_\text{reco}|x_\text{hard}) \right\rangle_{x_\text{hard}\sim q(x_\text{hard})} \\ &= \left\langle \frac{1}{q(x_\text{hard})}\; p(x_\text{hard}|x_\text{reco}, \alpha) p(x_\text{reco}|\alpha) \right\rangle_{x_\text{hard}\sim q(x_\text{hard})} \; . \end{aligned} \tag{6.24}$$

We can see that the integrand becomes independent of x_hard and the variance vanishes when

$$q(x_\text{hard}) \equiv q(x_\text{hard}|x_\text{reco}, \alpha) \propto p(x_\text{hard}|x_\text{reco}, \alpha) \; , \tag{6.25}$$

where $p(x_\text{hard}|x_\text{reco}, \alpha)$ corresponds to the generative unfolding probability from reconstruction level to parton level [28].

In practice, we cannot expected the learned transfer probability to match the truth distribution perfectly. To compensate for the imperfect training, we define the sampling distribution in terms of the learned transfer probability. Then the condition in Eq. (6.25)

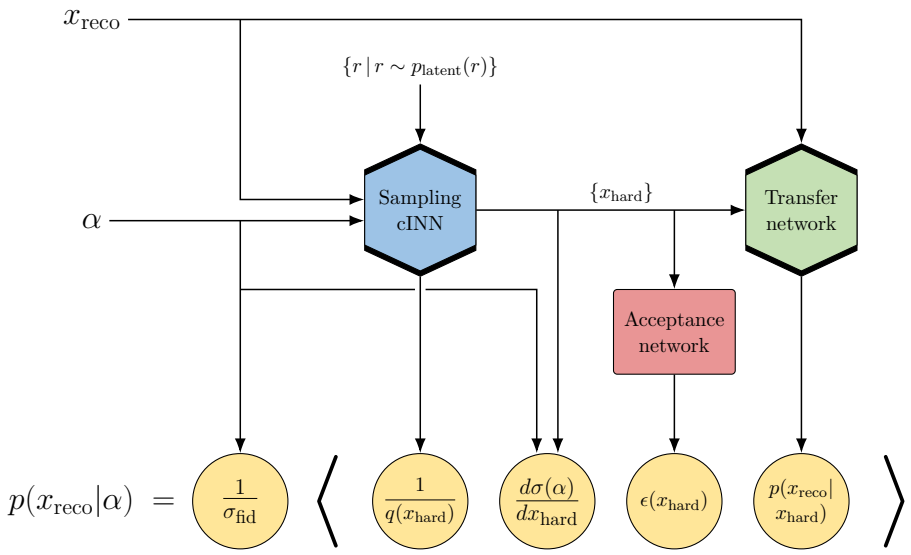


Figure 6.4: Three-network MEM integrator evaluating Eq. (6.29) through sampling r. The Sampling-cINN is conditioned on the CP-angle α and the reco-level event x_{reco}. The Transfer network is conditioned on the hard-scattering event x_{hard}. For the three-network setup the acceptance $\epsilon(x_{\text{hard}})$ is encoded in a network.

becomes

$$q(x_{\text{hard}}|x_{\text{reco}},\alpha) \propto p_{\text{fid}}(x_{\text{hard}}|\alpha)\; p_\theta(x_{\text{reco}}|x_{\text{hard}}) \; . \tag{6.26}$$

We train a conditional normalizing flow with trainable parameters φ to learn either the proposal distribution from Eq. (6.25) or Eq. (6.26),

$$r \sim p_{\text{latent}}(r) \quad \xleftrightarrow{\text{Sampling-cINN}} \quad x_{\text{hard}}(r) \sim q_\varphi(x_{\text{hard}}|x_{\text{reco}},\alpha) \; . \tag{6.27}$$

In the first case the network can be directly trained on paired samples $(\alpha, x_{\text{hard}}, x_{\text{reco}})$, whereas in the second case a synthetic dataset has to be created that follows the learned transfer probability. In both cases, we can then parameterize the conditional sampling density as

$$\begin{aligned} q_\varphi(x_{\text{hard}}|x_{\text{reco}},\alpha) \equiv q_\varphi(x_{\text{hard}}(r)|x_{\text{reco}},\alpha) &= \frac{p_{\text{latent}}(r)}{J_\varphi(r)} \\ \text{with} \qquad J_\varphi(r) &= \left| \frac{\partial x_{\text{hard}}(r;x_{\text{reco}},\alpha;\varphi)}{\partial r} \right| \; . \end{aligned} \tag{6.28}$$

Inserting this into the MEM integral from Eq. (6.17) yields

$$\begin{aligned} p(x_{\text{reco}}|\alpha) &= \frac{1}{\sigma_{\text{fid}}(\alpha)} \int \mathrm{d}r \; J_\varphi(r) \; \left[\frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_{\text{hard}}} \epsilon_\psi(x_{\text{hard}})\, p_\theta(x_{\text{reco}}|x_{\text{hard}}) \right]_{x_{\text{hard}}(r;x_{\text{reco}},\alpha;\varphi)} \\ &= \frac{1}{\sigma_{\text{fid}}(\alpha)} \left\langle \frac{J_\varphi(r)}{p_{\text{latent}}(r)} \; \left[\frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_{\text{hard}}} \epsilon_\psi(x_{\text{hard}})\, p_\theta(x_{\text{reco}}|x_{\text{hard}}) \right]_{x_{\text{hard}}(r;x_{\text{reco}},\alpha;\varphi)} \right\rangle_{r\sim p(r)} \; . \end{aligned} \tag{6.29}$$

The architecture of our three-network MEM integrator is illustrated in Fig. 6.4.

6.3 Two-network setup

A common assumption in the MEM literature [206] is that the phase-space dependence of the acceptance can be neglected. We will therefore start our discussion with a simplified setup where the acceptance only enters the MEM integration through the fiducial cross section. The reco-level likelihood can then be written as

$$p(x_{\text{reco}}|\alpha) \approx \frac{1}{\sigma_{\text{fid}}(\alpha)} \int \mathrm{d}x_{\text{hard}} \, \frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_{\text{hard}}} \, p_\theta(x_{\text{reco}}|x_{\text{hard}}) \; . \tag{6.30}$$

We use cINNs with rational-quadratic spline coupling blocks [110] for the transfer network and the sampling network. For each network, we train a deterministic and a Bayesian version, with the loss functions from Eq. (3.34) and Eq. (3.67), respectively. The input features of both networks are the components of the hard-scattering and reco-level four momenta. For the hard-scattering momenta, we remove redundant degrees of freedom from the on-shell conditions and transverse momentum conservation, resulting in a seven-dimensional input

$$\{p_t^x, p_t^y, p_t^z, p_H^x, p_H^y, p_H^z, p_j^z\} \; . \tag{6.31}$$

For the reco-level momenta, we remove the energies of photons and muons because they are redundant degrees of freedom, resulting in $5 \cdot 4 - 3 = 17$ input dimensions for the leptonic dataset, and $6 \cdot 4 - 2 = 22$ for the hadronic dataset. We do not include potential additional jets in the input of the Transfer-cINN as it requires a fixed dimension, but

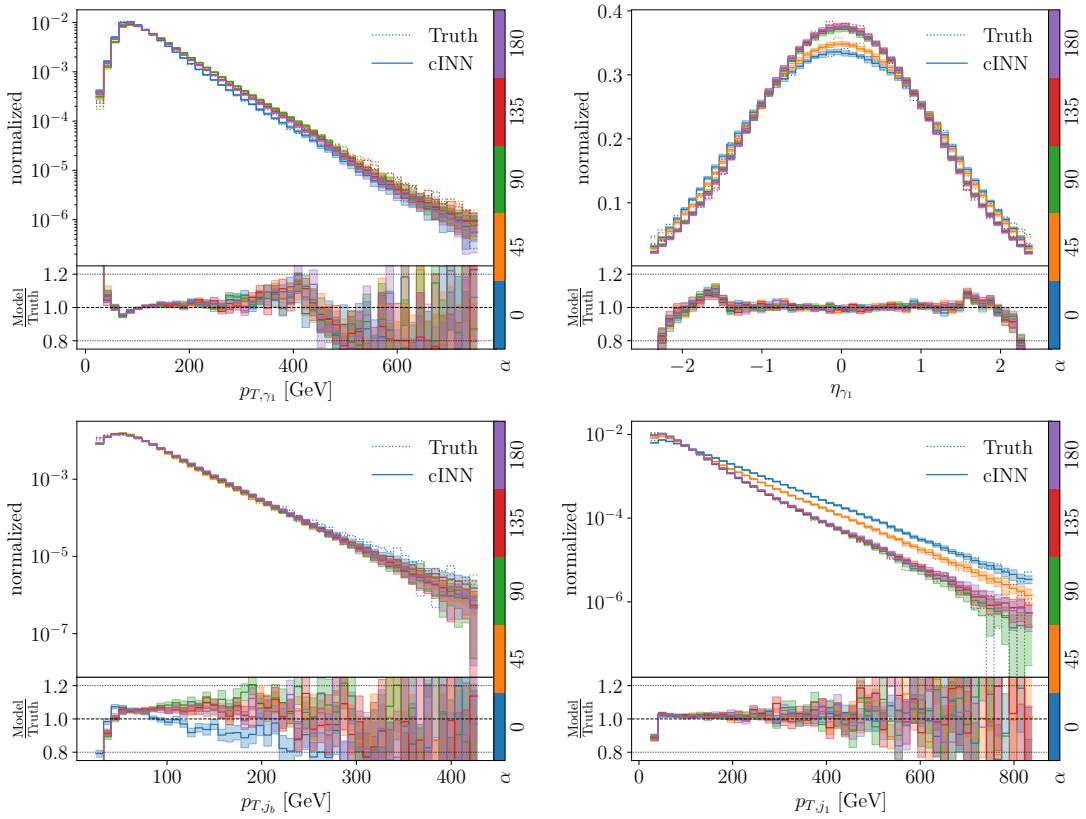


Figure 6.5: Forward-simulated kinematic distributions for the leptonic top decay, assuming five different CP-angles and including uncertainties from the Bayesian cINN. These distributions test the Transfer-cINN.

append the zero-padded four momenta of up to four extra jets to the conditional input of the Sampling-cINN. We expect the distribution learned by the Sampling-cINN to depend on the CP-phase α, so we pass it to the network as another conditional input. In contrast, we expect the forward-simulation to be α-independent. For all network inputs, we subtract the feature-wise mean and divide by the standard deviation as a preprocessing step. We use the hyperparameters shown in Tab. A.7 for both cINNs.

6.3.1 Leptonic top decay

We start by training a Bayesian Transfer-cINN and Sampling-cINN on our leptonic dataset. The kinematic reco-level distributions obtained from the Transfer-cINN compared to the truth distribution from the test dataset are shown in Fig. 6.5. We show histograms for each of the five values of the CP-phase in our test dataset. The error bands are defined by the bin-wise Bayesian uncertainties, similar to the results for precision event generation shown in Sec. 4.3. While there are deviations from the truth distribution, they are mostly covered by the uncertainties in the case of the three p_T distributions, with some deviations in the η distribution. These results also allow us to test our assumption that the forward-simulation is independent of the CP-phase α. While there are differences in the reco-level distributions, these are caused by the variation in the hard-scattering distributions. Small performance differences for different α are mostly covered by the training uncertainties, so the Transfer-cINN works equally well for all choices of α. This confirms that we do not need to assume an α-dependent transfer probability.

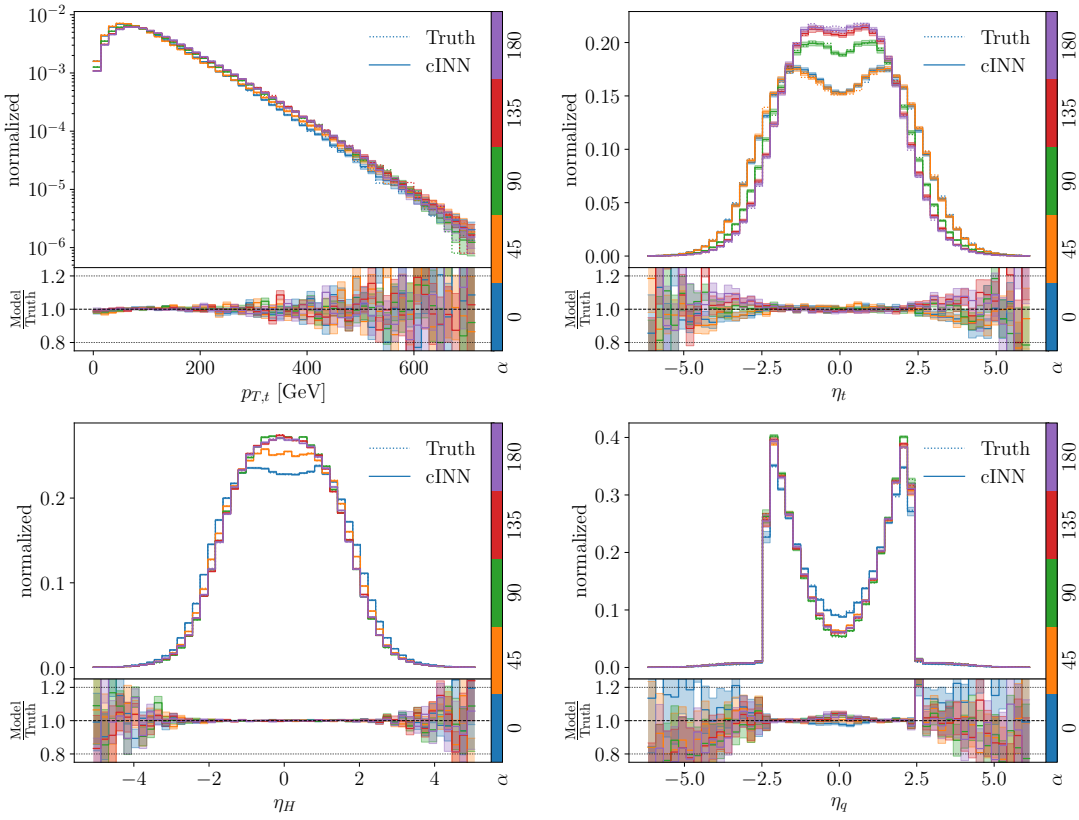


Figure 6.6: Hard-scattering level kinematic distributions for the leptonic top decay, assuming five different CP-angles and including uncertainties from the Bayesian cINN. These distributions test the Sampling-cINN.

Next, we show the kinematic distributions from the Sampling-cINN in Fig. 6.6, again with uncertainty bands estimated using a Bayesian network. We find that there is an excellent agreement with the truth distribution within the uncertainties. Further, the α dependence is correctly reproduced. The level of precision achieved by the Sampling-cINN is higher than for the Transfer-cINN. This is because the forward simulation is the more challenging task as it samples into a higher-dimensional space. In the following, we will replace the Bayesian Sampling-cINN with a deterministic version. Because it is used for importance sampling, deviations from the truth distribution only have an effect on the convergence of the integral but will not bias the result.

After testing both networks individually, we combine them to compute the integral from Eq. (6.30). We evaluate the integral for multiple sets of reco-level events, each one with a different value of the CP-phase α. For each event and α, we sample 100k points in the hard-scattering phase space using the Sampling-cINN conditioned on x_reco and α. We improve the numerical stability using trimmed means and standard deviations where we discard 1% of the largest and smallest values of the integrand. To guarantee physical events, we remove hard-scattering events with momentum fractions $x > 1$, and events that have a negative differential cross section because of the parton distribution functions. We then compute the negative log-likelihood $-2 \log L_i(\alpha)$ for a given event i, and combine the log-likelihoods of multiple events by adding them.

To test the results of our method, we combine the likelihoods of 400 events for three assumed truth values of the CP-angle, $\alpha = 0°, 45°, 90°$. In Fig. 6.7 we show the resulting log-likelihoods with uncertainties computed using Gaussian error propagation of the Monte Carlo integration error. We perform a polynomial fit to get a smooth reco-level log-likelihood, and normalize them such that the minimum of the fit curve is at 0. As a comparison, we show the log-likelihood obtained by directly evaluating the differential

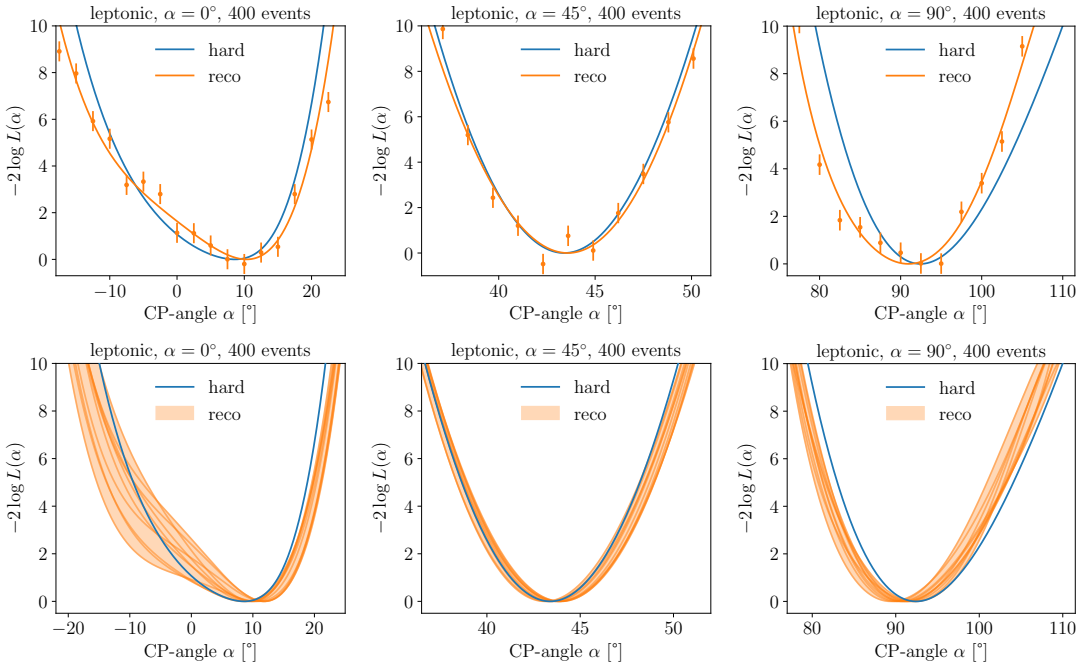


Figure 6.7: Likelihoods for the leptonic top decay as a function of the CP-angle α, extracted from 400 events for three assumed truth angles. For the Bayesian uncertainties we show the integrated likelihoods from 10 sampled networks.

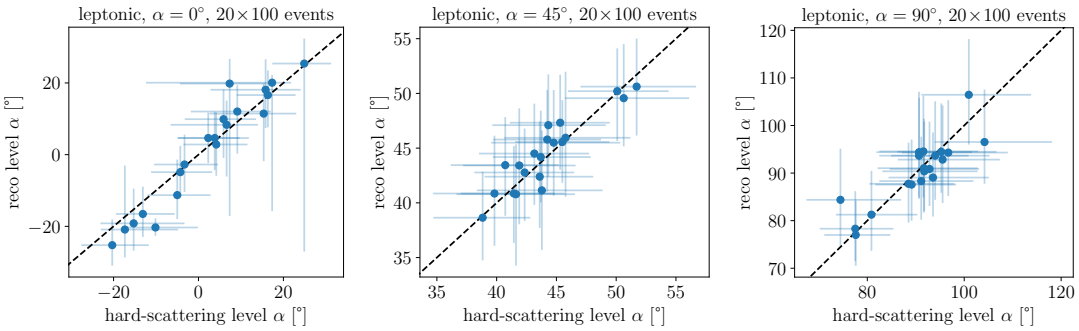


Figure 6.8: Calibration of the α-measurement from leptonic top decays, in terms of mean values and 68% confidence intervals extracted from 20 sets of 100 events at hard-scattering level and reco level.

cross sections for the hard-scattering momenta from the simulation. Note that even in the limit of a perfect Transfer-cINN, we cannot expect the reco-level likelihood and the hard-scattering level likelihood to be the same, as there could be a loss of information in the forward simulation. It can be seen that there is a very good agreement for $\alpha = 45°$ and that the confidence interval is relatively narrow. This is because the kinematic distributions are very sensitive to changes in the CP-angle around $\alpha = 45°$, as seen in Fig. 6.3. The measurement of the CP-angle is especially challenging around the Standard Model value $\alpha = 0°$ because the matrix element is almost symmetric under a sign flip in α. For $\alpha = 0°$ and $\alpha = 90°$, the agreement with the hard-scattering result is still good, but slightly worse compared to $\alpha = 45°$.

To better understand the uncertainties introduced by the training of the Transfer-cINN, we can again make use of the Bayesian version of the network. To this end, we repeat the likelihood calculation for 10 network replicas sampled from the distribution over their trainable parameters. Again, we perform polynomial fits and show the resulting log-likelihoods in the lower panels of Fig. 6.7. The more challenging inference task for $\alpha = 0°$ and $\alpha = 90°$ is reflected by the larger uncertainty predicted by the Bayesian network. There is a very large uncertainty in the lower left panel around $\alpha = -10°$ because of the aforementioned degeneracy in the matrix element.

As the last check of our ML-MEM setup, we look at the calibration of the extracted CP-phase. While calibration plots are normally made by comparing the inferred α for a range of different truth values for α, we create approximate calibration plots where we compare the CP-angles extracted from the hard-scattering and reco-level likelihoods. To this end, we determine the minima and 68% confidence intervals from the negative log-likelihoods for 20 sets of 100 events. We compute the confidence intervals by assuming an approximately Gaussian likelihood distribution on each side of the minimum, such that the likelihood values at the two boundaries of the confidence interval are the same. This can lead to asymmetric error bars. We show such plots for $\alpha = 0°, 45°, 90°$ in Fig. 6.8. For all three truth angles, we find a good correlation between the hard-scattering level and the reco-level results. For the SM-value $\alpha = 0°$, some points have larger uncertainties at the reco-level because of the approximate degeneracy under a sign change in α. Still, even for this challenging case, the results are well-calibrated.

6.3.2 Hadronic top decay

In the case of the leptonic decay, every reco-level object can be clearly assigned to one particle in the final state of the hard process. The main challenge is the reconstruction of the missing neutrino momentum. This is no longer the case when we move to the hadronically decaying top. The networks then have to solve the increasingly complex jet combinatorics. There are at least four reco-level jets, which could come from the top decay, or from the light quark produced in the hard process. In addition, there can be jets from initial state radiation in our most challenging dataset.

Without ISR

We train the Transfer-cINN and Sampling-cINN with the same setup and hyperparameters as before. The only difference to the leptonic case is that the reco-level phase space is now higher-dimensional with $6 \cdot 4 - 2$ dimensions. Again, we show the extracted log-likelihoods for 400 events in Fig. 6.9, using a deterministic network in the upper panels and a Bayesian network in the lower panels. The results are comparable to those from the leptonic decay, showing that the network is able to resolve the jet combinatorics. We show the calibration in Fig. 6.10. Compared to the leptonic results in Fig. 6.8, our method finds the wrong sign of the CP-angle for some sets of events because of the approximate symmetry from Eq. (6.3). The absolute value is still well-calibrated.

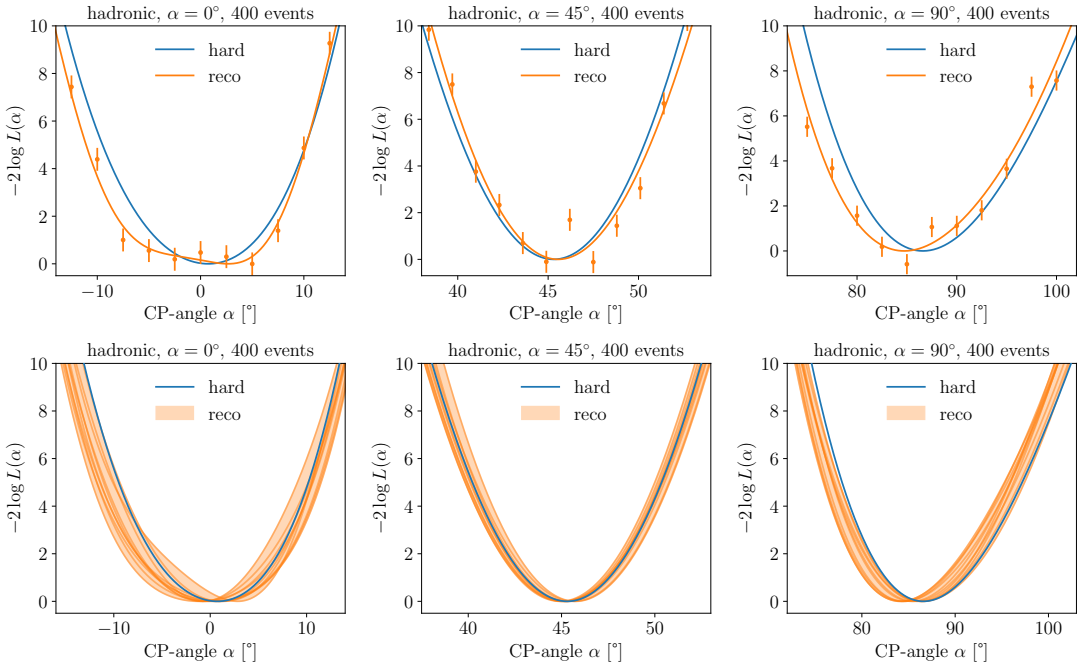


Figure 6.9: Likelihoods for the hadronic top decay as a function of the CP-angle α, extracted from 400 events for three assumed truth angles. For the Bayesian uncertainties we show the integrated likelihoods from 10 sampled networks.

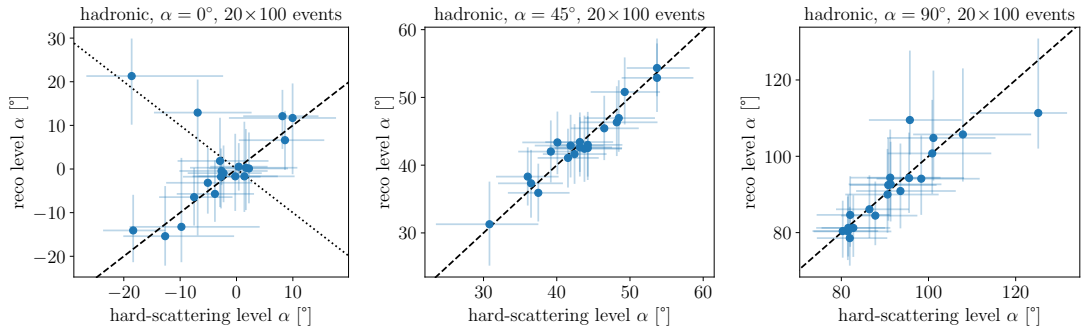


Figure 6.10: Calibration of the α-measurement from hadronic top decays, in terms of mean values and 68% confidence intervals extracted from 20 sets of 100 events at hard-scattering level and reco level.

With ISR

Next, we allow for initial state radiation (ISR) and train our networks with the same setup as before. We use a larger training dataset with 3.4M events to help the network extract the more complex kinematic patterns from the data. As shown in Fig. 6.3, the kinematic distribution of the hard forward jet is very sensitive to the CP-angle α. While final state radiation can lead to an additional jet from the top decay or a splitting of the forward jets, it does not change the event topology much. This is different for ISR jets, as they can look similar to the forward jet. Sometimes, they are misidentified as the forward jet by the network even though they are insensitive to the CP-angle. This can lead to a bias in the extracted likelihoods.

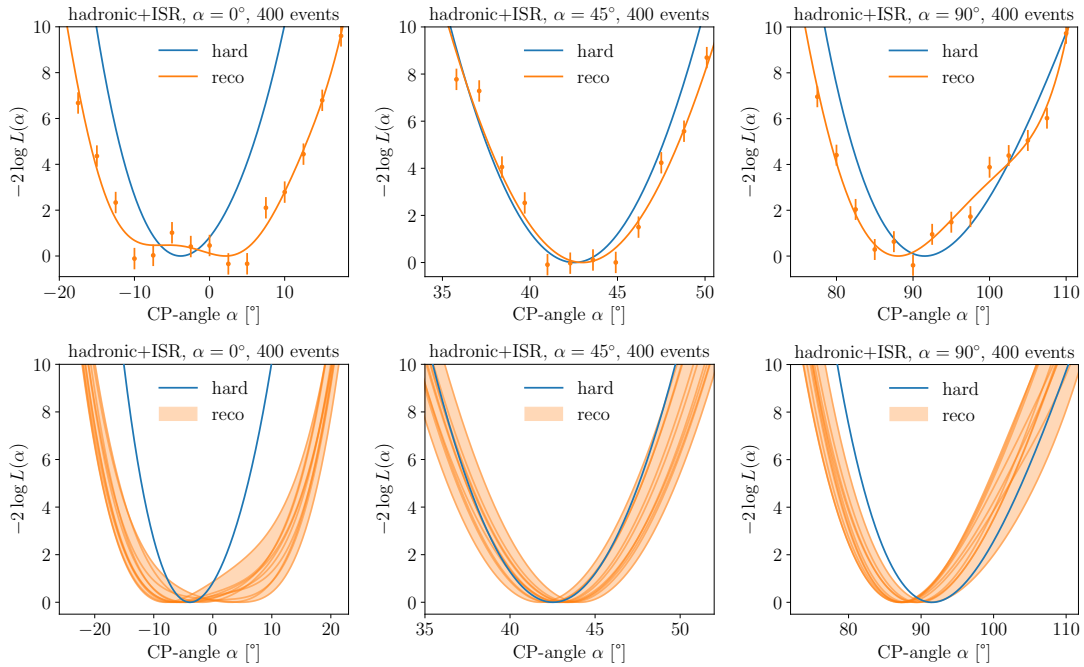


Figure 6.11: Likelihoods for the hadronic top decay, including ISR, as a function of the CP-angle α, extracted from 400 events for three assumed truth angles. For the Bayesian uncertainties we show the integrated likelihoods from 10 sampled networks.

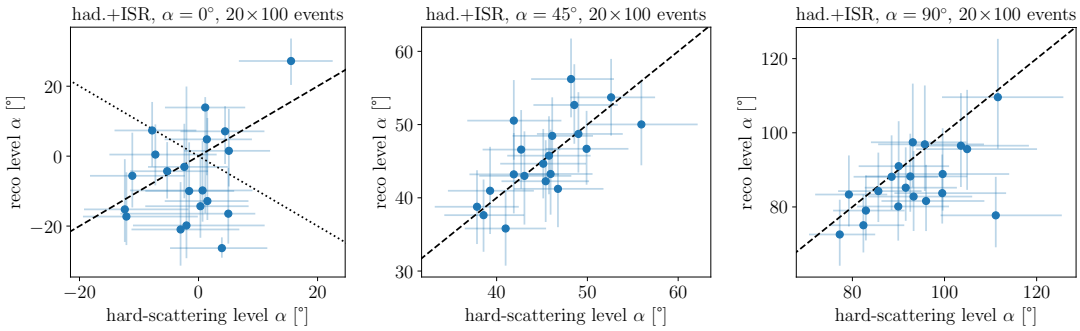


Figure 6.12: Calibration of the α-measurement from hadronic top decays with ISR, in terms of mean values and 68% confidence intervals extracted from 20 sets of 100 events at hard-scattering level and reco level.

We show the extracted likelihoods for 400 events in Fig. 6.11. While the results for $\alpha = 45°$ are still in good agreement with the hard-scattering truth, we can see that there is a bias towards lower CP-angles for $\alpha = 90°$, and a loss in sensitivity for $|\alpha| \lesssim 10°$ in the Standard Model case. This observation is consistent between the results from the deterministic Transfer-cINN in the upper panels and the Bayesian Transfer-cINN in the lower panels. Furthermore, we see that the uncertainty from the network training estimated by the Bayesian network is much larger compared to the results without ISR shown in Fig. 6.9.

These issues are confirmed by the calibration plots shown in Fig. 6.12. There is almost no sensitivity to the CP angle for $|\alpha| \lesssim 10°$. For $\alpha = 45°$ and $\alpha = 90°$, there is a clear correlation between the hard-scattering and reco-level α. It is however noisier than in the results without ISR, and the calibration plot confirms the bias towards lower CP-angles for $\alpha = 90°$.

6.4 Improved MEM setup

While the results shown in the previous section are promising, there are also several problems that need to be solved for the ML-based matrix element method to reach the required level of precision for an LHC analysis. First, the phase-space integration is relatively slow as it involves a large number of network evaluations. Second, trimmed means are used to stabilize the integration. This is effective, but can also lead to biased integration results. Lastly, we find that the parameter measurement becomes biased when ISR is included. In Sec. 6.4.2 we show that this is partially caused by phase-space dependent acceptance effects.

6.4.1 Improved integration

In this section, we continue to use the same network setup with two cINNs and the same hyperparameters as before to solve the integral in Eq. (6.30). To improve the speed and stability of the integration, we introduce a series of numerical improvements.

Single-pass integration over model parameters

In Sec. 6.3 we evaluated the integral separately for every value of the theory parameter α. We can optimize this by making use of the fact that the transfer probability does not depend on α, and the mapping for the importance sampling only has a small α-dependence. We can therefore reuse the phase-space samples $x_\text{hard} \sim q_\varphi(x_\text{hard}|x_\text{reco}, \alpha)$ and the corresponding values of $p_\theta(x_\text{reco}|x_\text{hard})$ to evaluate the differential cross section for multiple points in α. Furthermore, only parts of the differential cross section depend on α, like the matrix elements, whereas other parts only depend on x_hard, like the parton densities. An additional benefit of this method is that the integrand for a given sample x_hard is a smooth function of α. Consequently, the integral is a smooth function as well when the same samples x_hard are used for all α. This makes it unnecessary to perform a fit through noisy data points to get a smooth curve for the likelihood.

We replace multiple separate integrations with a single-pass integration for a given x_reco and a discrete set $\{\alpha\}$. The integration is then performed as follows:

1. For $j \in \{1, \dots, N\}$, draw $\alpha^{(j)}$ from $\{\alpha\}$ randomly;
2. Using the sampling network, sample $x_\text{hard}^{(j)} \sim q_\varphi(x_\text{hard}|x_\text{reco}, \alpha^{(j)})$;
3. Evaluate the transfer probability $p_\theta(x_\text{reco}|x_\text{hard}^{(j)})$ for each sample;
4. Evaluate the differential cross section $d\sigma(\alpha)/dx_\text{hard}^{(j)}$ for each sample $x_\text{hard}^{(j)}$ and α;
5. Compute the MC integral Eq. (6.30) for all α values at the same time

$$p(x_\text{reco}|\alpha) \approx \frac{1}{\sigma_\text{fid}(\alpha)} \frac{1}{N} \sum_{j=1}^{N} \frac{1}{q_\varphi(x_\text{hard}^{(j)}|x_\text{reco}, \alpha^{(j)})} \frac{\mathrm{d}\sigma(x_\text{hard}^{(j)}|\alpha)}{\mathrm{d}x_\text{hard}} p_\theta(x_\text{reco}|x_\text{hard}^{(j)}) \; . \tag{6.32}$$

As our example process has only two Feynman diagrams, the computational cost of the integration is dominated by network evaluations. As a result, the single-pass integration procedure leads to a large speedup.

Iterative integration

In Sec. 6.3 we used a fixed number of integration points for every reco-level event. However, we find that the integration converges faster for some events, and needs more samples for others. Possible reasons include that the integration is more challenging for some events, or that the peaks of the transfer probability and importance sampling distribution are not perfectly aligned. This results in a higher variance and a slower convergence of the integral. To guarantee a small integration error in such cases while keeping the overall integration time low, we compute the integral iteratively. To this end, we specify a minimal and maximal number of iteration as well as a threshold for the maximal relative uncertainty of the integration results for all values of α. The integration is then repeated with a fixed batch size until the combined uncertainty drops below the threshold or the maximal number of iterations is reached. For our example process we use a batch size of 10k, between two and 15 iterations, and a target uncertainty of 2%. Note that the uncertainties of the likelihood ratios shown in our Figures are much smaller than 2% because of the correlations between different α introduced by the single-pass integration.

Integration uncertainties

Because the result of the single-pass integration is a smooth function of α, the simple pointwise MC integration error is no longer sufficient to estimate the integration uncertainties. Due to the correlation between the likelihoods for different α, the uncertainty on the likelihood ratio should be much smaller than the uncertainty on the likelihood before normalization. We use bootstrapping to estimate the correlated integration uncertainty. To this end, we resample the integrand by randomly drawing M batches of N samples with replacement from our set of integrand samples $\{I^{(j)}(\alpha_i) \,|\, j = 1, \ldots, N\}$. We compute the integral by taking the mean over the N samples for each of the M replicas and then propagate these replicas through the downstream tasks. Finally, we estimate the uncertainties of the normalized negative log-likelihood by computing the standard deviation over the M replicas for every α_i.

We can use the same approach to extract the uncertainties from the training of the transfer network, estimated using a Bayesian neural network. Instead of performing a separate integration for every sample from the distribution of trainable network parameters like in Sec. 6.3, we extend our one-pass integration and reuse the same phase-space points x_{hard} for multiple transfer networks sampled from the Bayesian network distribution. This does not have a large effect on the integration performance because the same importance sampling distribution works well for different sampled networks. We combine this approach with the bootstrapping procedure described above by resampling the integrand and the transfer network parameters for every replica. Then, the integration and training uncertainties can be estimated in the same step.

Factorization of the differential cross section

In addition to the single-pass integration, we can further optimize the computation of the differential cross section for some processes. For our example process, we can see from Eq. (6.3) that the differential cross section can be written as

$$\begin{aligned} \frac{\mathrm{d}\sigma(x_{\text{hard}}|\alpha)}{\mathrm{d}x_{\text{hard}}} &= g_1 + \sin\alpha\, g_2 + \cos\alpha\, g_3 + \sin\alpha\cos\alpha\, g_4 + \sin^2\alpha\, g_5 \\ &\equiv \sum_i f_i(\alpha) g_i(x_{\text{hard}}) \,, \end{aligned} \tag{6.33}$$

i.e. the differential cross section factorizes into a phase-space-dependent part $g_i(x_{\text{hard}})$ and a parameter dependent part $f_i(\alpha)$. Similar factorization properties hold for SMEFT corrections where they are often referred to as operator morphing [230]. The MEM integration in Eq. (6.30) then becomes

$$p(x_{\text{reco}}|\alpha) = \frac{1}{\sigma_{\text{fid}}(\alpha)} \sum_i f_i(\alpha) \int \mathrm{d}x_{\text{hard}}\, g_i(x_{\text{hard}})\, p_\theta(x_{\text{reco}}|x_{\text{hard}}) \,, \tag{6.34}$$

and its Monte Carlo estimate is given by

$$\begin{aligned} p(x_\text{reco}|\alpha) &\approx \frac{1}{\sigma_\text{fid}(\alpha)} \frac{1}{N} \sum_{j=1}^{N} \frac{1}{q_\varphi(x_\text{hard}^{(j)}|x_\text{reco},\alpha^{(j)})} \frac{\mathrm{d}\sigma(x_\text{hard}^{(j)}|\alpha)}{\mathrm{d}x_\text{hard}} p_\theta(x_\text{reco}|x_\text{hard}^{(j)}) \\ &= \frac{1}{\sigma_\text{fid}(\alpha)} \sum_i f_i(\alpha) \frac{1}{N} \sum_{j=1}^{N} \frac{1}{q_\varphi(x_\text{hard}^{(j)}|x_\text{reco},\alpha^{(j)})} \, g_i(x_\text{hard}^{(j)}) \, p_\theta(x_\text{reco}|x_\text{hard}^{(j)}) \, , \end{aligned} \tag{6.35}$$

where $x_\text{hard}^{(j)} \sim q_\varphi(x_\text{hard}|x_\text{reco},\alpha^{(j)})$. Note that the MC estimate of the integral only preserves the exact functional form from Eq. (6.33) if $g_i(x_\text{hard})$ is evaluated with the same samples $x_\text{hard}^{(j)}$ for all i.

Importance sampling trained on transfer probability

So far the training of the Sampling-cINN was based on the assumption that the learned transfer probability closely approximate the true transfer probability. Consequently, we trained the proposal distribution directly on the truth distribution

$$q_\varphi(x_\text{hard}|x_\text{reco},\alpha) \approx p(x_\text{hard}|x_\text{reco},\alpha) \propto p(x_\text{reco}|x_\text{hard}) p_\text{fid}(x_\text{hard}|\alpha) \, . \tag{6.36}$$

The transfer probability is a sharply peaked function, so a small misalignment compared to the importance sampling distribution can lead to a large increase in the integral variance. This does not necessarily have a significant impact on the inference result itself, but slows down the convergence of the phase-space integral. Instead of training the Sampling-cINN on the real distribution, we can instead train it on the modeled distribution from Eq. (6.26),

$$q_\varphi(x_\text{hard}|x_\text{reco},\alpha) \approx p_\theta(x_\text{reco}|x_\text{hard}) p_\text{fid}(x_\text{hard}|\alpha) \, . \tag{6.37}$$

The training dataset consists of tuples $(\alpha, x_\text{hard}, x_\text{reco})$. We can modify it to follow the modeled distribution by replacing the reco-level momenta with samples

$$\tilde{x}_\text{reco} \sim p_\theta(x_\text{reco}|x_\text{hard}) \, , \tag{6.38}$$

where we use the surrogate network for the transfer function as a generative model. To increase the training statistics we re-sample $\tilde{x}_\text{reco}$ at the beginning of each epoch.

Vegas latent space refinement

Even when the Sampling-cINN is trained on the learned transfer probability, there are still some events with a large variance in the MEM integration. Adapting the importance sampling network during the integration, similar to the MadNIS approach from Ch. 5, would make the integration even slower. An alternative is to use Vegas, as introduced in Sec. 2.2.2, to refine the INN latent space. Because the α-dependence of the importance sampling is small, we can define a shared Vegas grid. We then use Vegas to transform our random latent space samples before we map them into phase space using the Sampling-cINN. We adapt the grid after every iteration of the integration by minimizing the variance for a point in the middle of the relevant α-interval. We then combine the results from multiple iterations weighted by the inverse variance. This

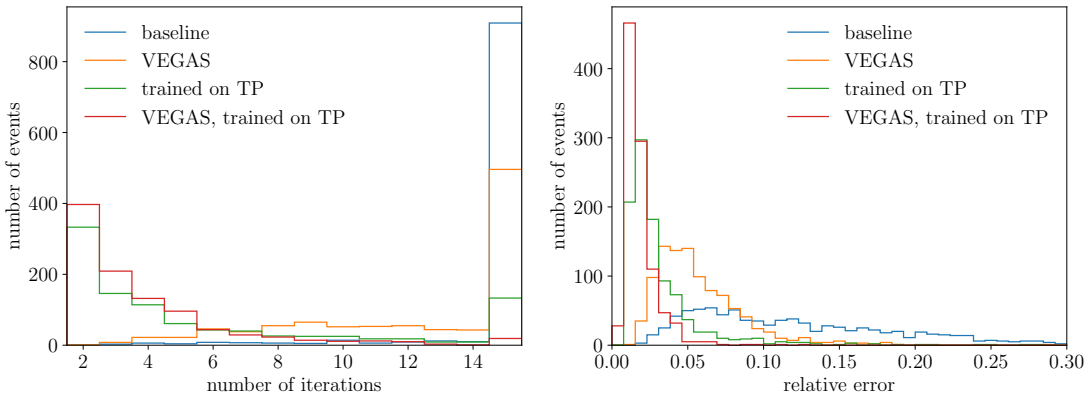


Figure 6.13: Integration performance with and without importance sampling trained on the transfer probability and VEGAS refinement. Left: number of iterations (10000 samples each) to reach the 2% target precision, with 2 to 15 iterations. Right: relative integration error after 10 iterations of 10000 samples each.

minimizes the overall variance and reduces the effect of early iterations where the grid is not yet well adapted.

Results

To benchmark the various improvements to the integration procedure, we use the same architecture and hyperparameters for the Transfer-cINN and Sampling-cINN as before, and train the networks on the challenging hadronic dataset including ISR. In all cases we use single-pass integration including a factorized differential cross section. We test the impact of training the Sampling-cINN on the transfer probability and of the VEGAS latent space refinement for 1000 SM events. We show the number of iterations needed to reach the target precision of 2% in the left panel of Fig. 6.13. While single-pass integration guarantees smooth likelihoods, it does not meet the target precision within 15 iterations for most events. This improves when it is combined with VEGAS refinement, and there are even larger improvements when the Sampling-cINN is trained on the learned transfer probability. When both methods are combined, the target precision is reached within 15 iterations for most events. These observations are confirmed by the relative integration error after 10 iterations shown in the right panel of Fig. 6.13. We will therefore use the combination of both methods for all the following results to ensure a fast and stable convergence of the phase-space integrals.

The purple line in the upper panels of Fig. 6.14 shows the log-likelihoods for 400 events, extracted using our improved integration method. The setup is otherwise similar to the one used for the results in Fig. 6.11. Compared to the previous results, the likelihoods are much smoother. We show the integration uncertainty as error bands, but they are barely visible because of the low error threshold and the single-pass integration. We confirmed that these uncertainties are consistent with the variations between multiple runs of the integrator. The key observation from Fig. 6.11 that there is a loss of sensitivity around the $\alpha = 0°$, is still true for the new results.

The higher integration speed allows us to show the log-likelihood for a large set of 10k events in the second row of Fig. 6.14. The higher number of events leads to much narrower

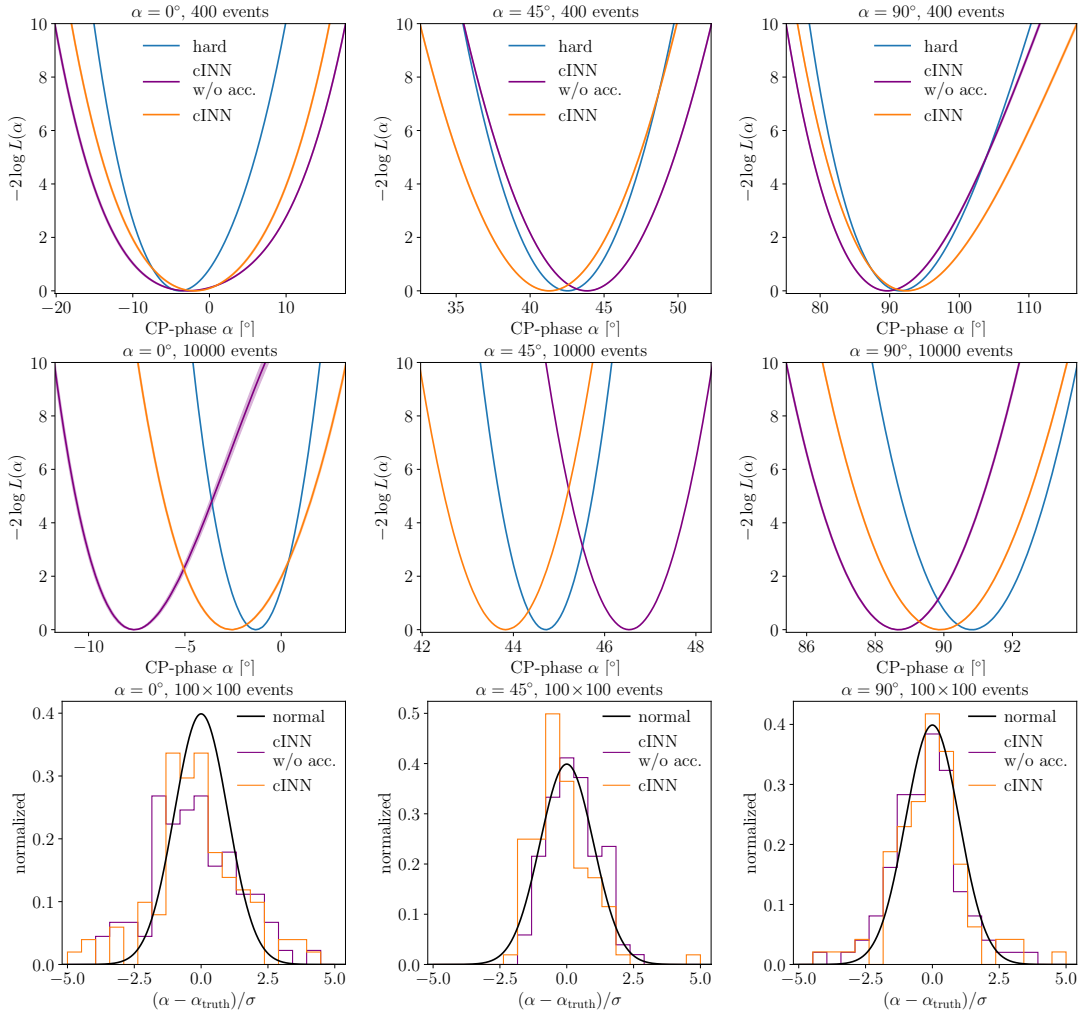


Figure 6.14: **cINN benchmark and learned acceptance:** likelihoods for different CP-angles. We use the same architecture as in Sec. 6.3, but with the improved integration. The purple curve shows the two-network cINN benchmark and the orange curve also includes the learned acceptance. From top to bottom: likelihoods for 400 events, 10000 events, and pulls.

confidence intervals. We can see that there is a significant deviation between the hard-process and reconstructed likelihoods for all three truth values of the CP-angle. These results show that the biased measurements were not caused by the unstable integration, so further improvements of our MEM setup are necessary.

The third row of Fig. 6.14 shows the distribution of the pull $(\alpha - \alpha_{\text{truth}})/\sigma$ for 100 sets of 100 events. α and σ are extracted from the likelihoods using the same method as for the calibration plots in Fig. 6.11. We can see that the uncertainty estimates are well-calibrated for $\alpha = 45°, 90°$, and slightly overconfident for $\alpha = 90°$.

6.4.2 Acceptance classifier

So far, we have assumed that we can neglect phase-space dependent acceptance effects. This assumption works well for simple cases like the leptonic dataset and the hadronic

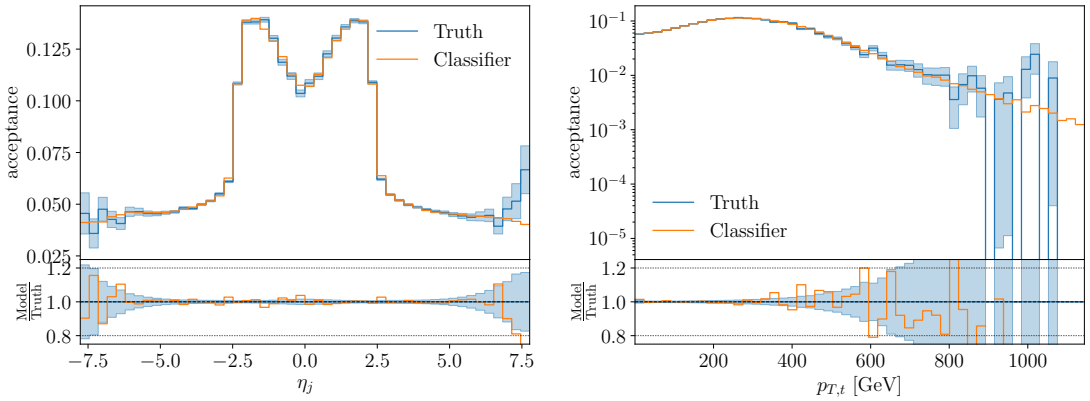


Figure 6.15: Truth and learned acceptance as a function of different kinematic observables.

dataset without ISR, as there is clear correspondence between hard-scattering and reco-level objects, and the transfer probability is relatively narrow. This means that hard cuts at the reconstruction level approximately lead to hard cuts of the hard-scattering momenta. The situation becomes more complicated when ISR is included. We show the acceptance probability for this case as a function of the hard-scattering phase space in Fig. 6.15. The forward jet is not detected if $|\eta_j| > 2.4$. Such events are sometimes still accepted when an ISR jet is tagged instead, reflected by the non-zero acceptance probability in this phase-space region. This also causes the increase in the fiducial cross section from the non-ISR-case to the ISR-case seen in Tab. 6.1. The transfer probability has to take into account the two possible origins of jets in the forward direction, and both regions of phase space contribute significantly to the MEM integral. We can see in Fig. 6.15 that there is a jump in the acceptance at $|\eta_j| = 2.4$, by almost a factor of three. There is also a significant variation of the acceptance as a function of other phase-space observables, like the p_T of the top. To correctly account for these effects, we have to include the acceptance function in our MEM integral.

To this end, we move from the two-network setup used in Secs. 6.3 and 6.4.1 to the full three-network setup introduced in Sec. 6.2. The MEM integral then has the form

$$p(x_\text{reco}|\alpha) = \frac{1}{\sigma_\text{fid}(\alpha)} \int \mathrm{d}x_\text{hard} \, \frac{\mathrm{d}\sigma(\alpha)}{\mathrm{d}x_\text{hard}} \, \epsilon_\psi(x_\text{hard}) \, p_\theta(x_\text{reco}|x_\text{hard}) \, . \tag{6.39}$$

To learn the acceptance function $\epsilon_\psi(x_\text{hard})$, we train a simple classifier on a dataset containing hard-scattering events with the acceptance information as a truth label. The hyperparameters are given in Tab. A.8. The classifier only takes a few minutes to train. Its results are in good agreement with the truth acceptance, as shown in Fig. 6.15.

We then evaluate the MEM integral including the learned acceptance and compare the likelihoods (orange lines) with the two-network baseline (purple lines) in Fig. 6.14. There are only small differences for the small set of 400 events. The likelihood is slightly narrower for $\alpha = 0°$, indicating a higher sensitivity. The differences become more clear for the large set of 10k events. There is a much smaller systematic bias for all three CP-angles compared to the results from the two-network baseline. We will use the three-network MEM setup for all the following results.

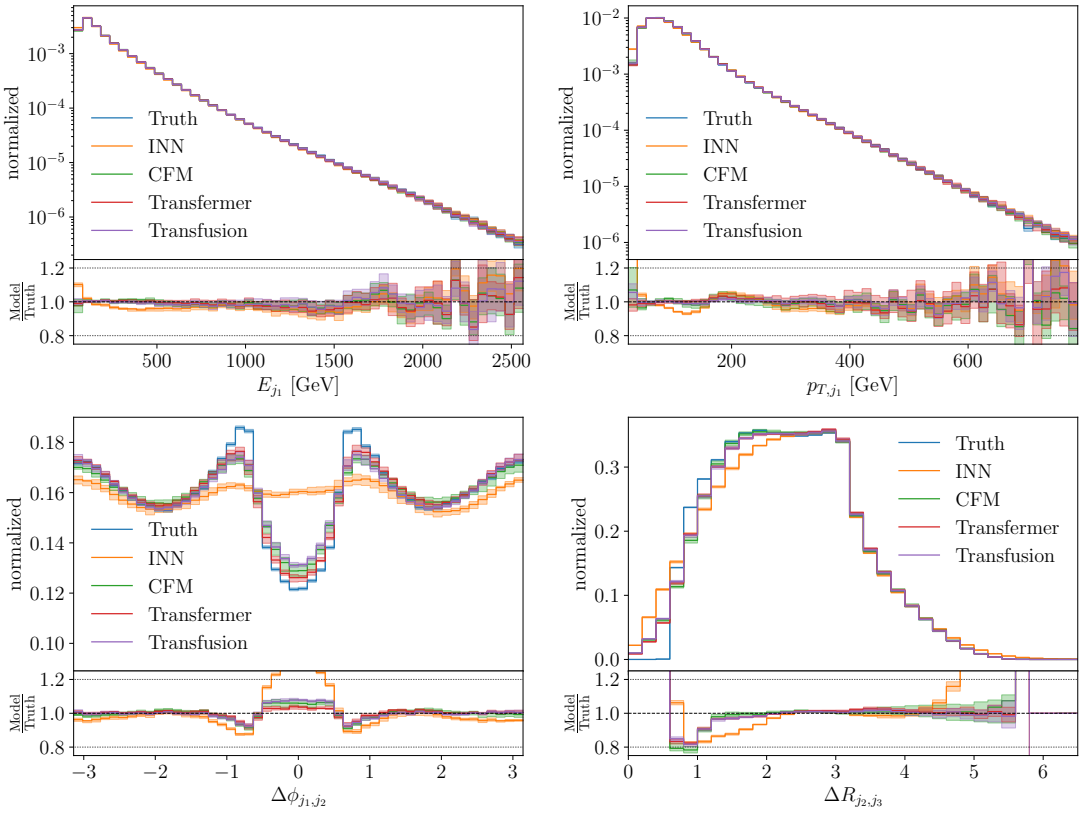


Figure 6.16: Reco-level distributions for different kinematic observables, obtained from the different generative transfer networks, conditioned on the hard-scattering momenta. Truth corresponds to the high-statistics test dataset.

6.5 Better transfer networks

The proper treatment of acceptance effects has considerably improved the accuracy of our MEM setup, but there is still some systematic bias left in the results shown in Fig. 6.14. To further reduce this bias, we need to improve the generative network used to model the transfer probability. We show reco-level distributions obtained by sampling from the learned transfer probability for different kinematic observables in Fig. 6.16. There are large deviations between the distribution from the Transfer-cINN (orange line) applied to hard-scattering events compared to the truth distribution (blue line), sometimes exceeding 20% relative error. We also show reco-level distributions for several alternative network architectures in Fig. 6.16. As seen in the figure, they perform much better than the Transfer-cINN. In the following, we will describe these architectures in detail.

6.5.1 Transfer diffusion

One major issue of INNs is that their expressivity is restricted by the invertibility constraint. Diffusion models are a more flexible architecture and have shown to reach higher levels of precision than INNs in phase-space sampling applications [64]. To be used as a transfer function it is crucial that the generative architecture has density estimation capability. For this reason, we choose conditional flow matching (CFM)

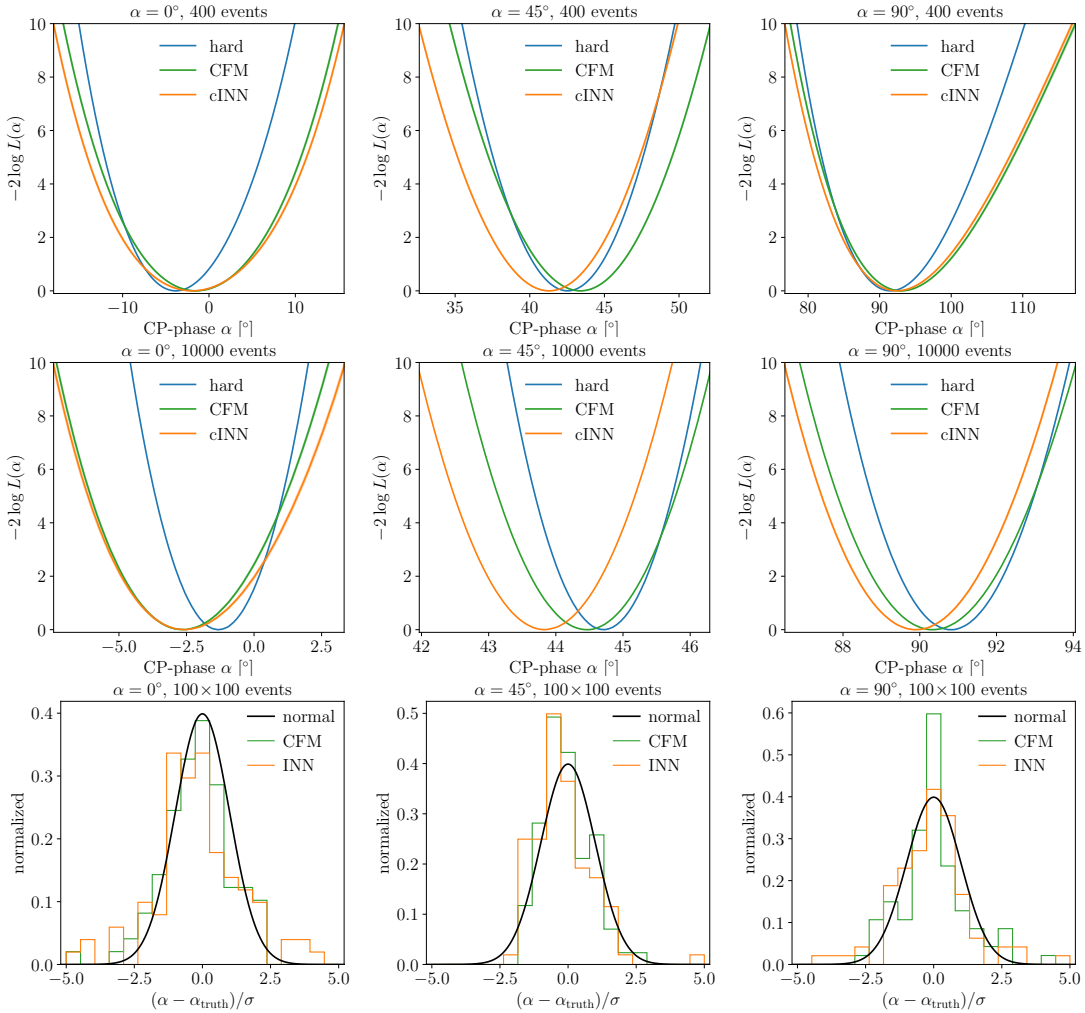


Figure 6.17: **Transfer-CFM:** likelihoods for different CP-angles. We compare the cINN baseline with a CFM diffusion network, both including the learned acceptance. From top to bottom: likelihoods for 400 events, 10000 events, and pulls.

networks [102–104], as introduced in Sec. 3.2.4. CFMs implement an invertible mapping with tractable Jacobian. Therefore, they can be used in place of the Transfer-cINN from Eq. (6.21) without further modifications.

We train the Transfer-CFM with the loss function from Eq. (3.58). Once trained, we apply an ODE solver to Eq. (3.52) to sample from the transfer probability, and use Eq. (3.55) for density estimation. The hyperparameters of our CFM network are given in Tab. A.9. To evaluate the likelihood with the required precision, we need $\mathcal{O}(100)$ evaluations of the velocity field encoded by a neural network. While sampling is relatively fast, we also need the gradients of the velocity field with respect to its inputs to compute the likelihood using Eq. (3.55). This makes the likelihood computation much slower compared to INNs, which is the main disadvantage of CFM-based transfer probabilities.

We show the likelihoods extracted with the Transfer-CFM in Fig. 6.17. The MEM setup is otherwise similar to the one used for the Transfer-cINN in Fig. 6.14. For 400 events, both architectures for the transfer probability work well. The extracted likelihoods are close to the optimal information from the hard process, and there is no visible systematic

bias. For the high-statistics case with 10k events, the CFM results are significantly better than the INN results. There is a slight increase in sensitivity for $\alpha = 0°$, and the systematic bias for $\alpha = 45°$ and $\alpha = 90°$ is significantly reduced. The extracted likelihoods are now close to the optimal information.

6.5.2 Combinatorics transformer

Next, we introduce a transformer [64, 184, 231] to improve the treatment of jet combinatorics [183], and combine it with cINNs and CFMs to be used as a density estimator. The transformer is used to generate a sequence of reco-level momenta from a sequence of hard-scattering momenta, similar to its original use for translation tasks [105]. The attention mechanism hereby takes care of encoding correlations between different particles and solving the jet combinatorics.

Transfermer

To encode the transfer probability, we follow the basic structure of a generative transformer described in Sec. 3.4.2. We make the transformer autoregressive at the level of reco-level momenta, and rely on a small and universal cINN conditioned on the output of the transformer $c^{(i)}$ to encode the likelihood for the momentum components of a single reco-level particle. The attention mechanism of the transformer encoder then models the correlations between hard-scattering momenta. The decoder models the correlations between the reco-level momenta as well as the input-output combinatorics. We illustrate the architecture in the left panel of Fig. 6.18.

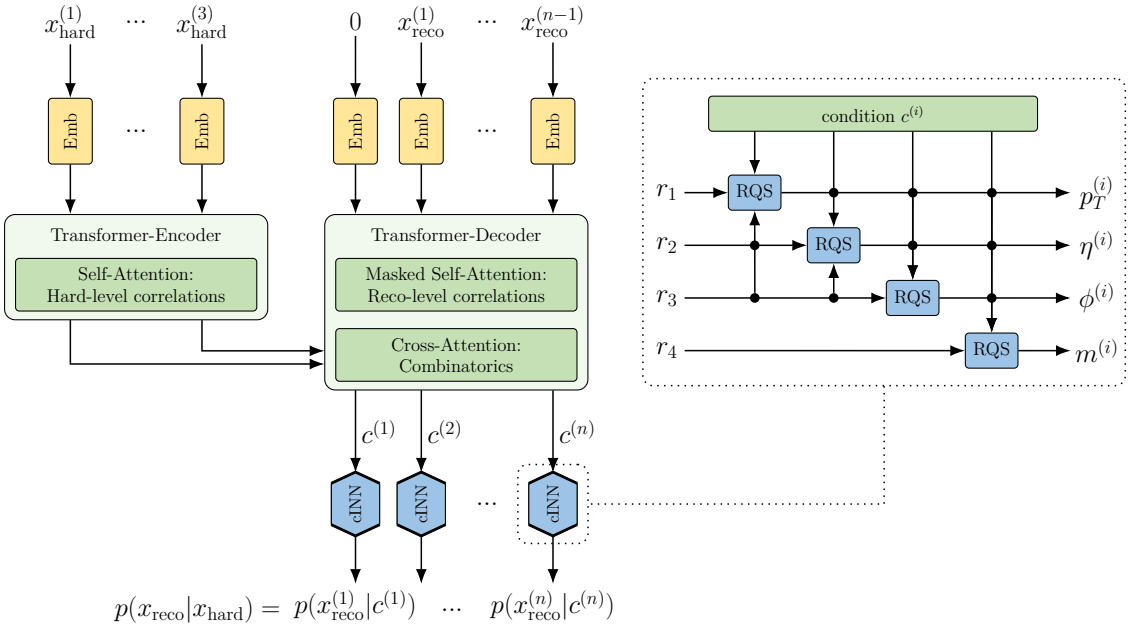


Figure 6.18: Left: transformer combined with cINN, encoding the transfer probability. Right: cINN used to learn individual momenta, where r is the usual latent space to parametrize a generative model.

We can use the autoregressive factorization together with the tractable likelihood of the cINNs for the individual particles to express the transfer probability from Eq. (6.45) as

$$p(x_{\text{reco}}|x_{\text{hard}}) = \prod_{i=1}^{n} p(x_{\text{reco}}^{(i)}|c^{(i)}(e_{\text{reco}}^{(0)}, \dots, e_{\text{reco}}^{(i-1)}, e_{\text{hard}})) \; , \tag{6.40}$$

where $c^{(i)}$ denotes the i-th output of the transformer. The autoregressive property is ensured by defining a starting token $e_{\text{reco}}^{(0)}$, shifting the inputs by one and using a triangular mask for the self-attention matrix. The likelihood of momentum $x_{\text{reco}}^{(i)}$ is then only conditioned on the previous reco-level momenta, but on all hard-scattering momenta. $e_{\text{reco}}^{(i)}$ and $e_{\text{hard}}^{(i)}$ denote the particle-wise embeddings of the momenta and their position in the p_T-ordered event. We build the embedding vectors by concatenating the momentum components and the one-hot encoded position. We then pad this vector with zeros until the embedding dimension of the transformer is reached. As the first operation applied to these embeddings within the transformer is a matrix multiplication, it is not necessary to apply any further processing steps like a single linear layer to the embeddings. As usual for autoregressive models, sampling from the transfer probability requires n Transfermer evaluations,

$$p(x_{\text{reco}}^{(i)}|x_{\text{hard}}) \equiv p(x_{\text{reco}}^{(i)}|c^{(i)}(e_{\text{reco}}^{(0)}, \dots, e_{\text{reco}}^{(i-1)}, e_{\text{hard}})) \quad \text{for } i = 1, \dots, n \; . \tag{6.41}$$

In contrast, the density from Eq. (6.40) can be evaluated with a single Transfermer evaluation as all reco-level momenta are known. This makes density estimation with the Tranfermer very fast, so the MEM integration remains computationally cheap.

As the last ingredient, we need to build a cINN to encode the particle-wise probability $p(x_{\text{reco}}^{(i)}|c^{(i)})$ in Eq. (6.41). Photon momenta only have three degrees of freedom because they are massless, while the mass is an additional degree of freedom for jets. To encode both types of reco-level objects in the same cINN we use the factorization

$$p(x_{\text{reco}}^{(i)}|c^{(i)}) = p(p_T^{(i)}, \eta^{(i)}, \phi^{(i)}|c^{(i)}) \cdot p(m^{(i)}|p_T^{(i)}, \eta^{(i)}, \phi^{(i)}, c^{(i)}) \; , \tag{6.42}$$

such that the mass component can be omitted during density estimation and generation without affecting the other three components. Because rational quadratic spline transformations (see Sec. 3.2.3) are very expressive [110], it is sufficient to transform every momentum component once, and condition it on the other components and the transformer output using a feed-forward network. This defines a minimal cINN architecture that is able to model the correlations between the momentum components. We illustrate the architecture in Fig. 6.18. To ensure the positivity and a roughly Gaussian distribution of the masses and transverse momenta, we train the network on normalized versions of $\log p_T$ and $\log m$. The cINN then maps these components to Gaussian latent spaces. We know that ϕ and η are only defined within an interval, the latter because of detector-level cuts. Hence, we map both components to uniform latent spaces, and use a periodic version of the RQ splines for ϕ [8].

Our implementation of the Transfermer is based on the standard PyTorch [196] transformer module and the cINN architecture described above. We train the cINN and the transformer jointly with a negative log-likelihood loss,

$$\mathcal{L} = -\log p_\theta(x_{\text{reco}}|x_{\text{hard}}) \; . \tag{6.43}$$

The hyperparameters are given in Tab. A.10. We show the extracted likelihoods for

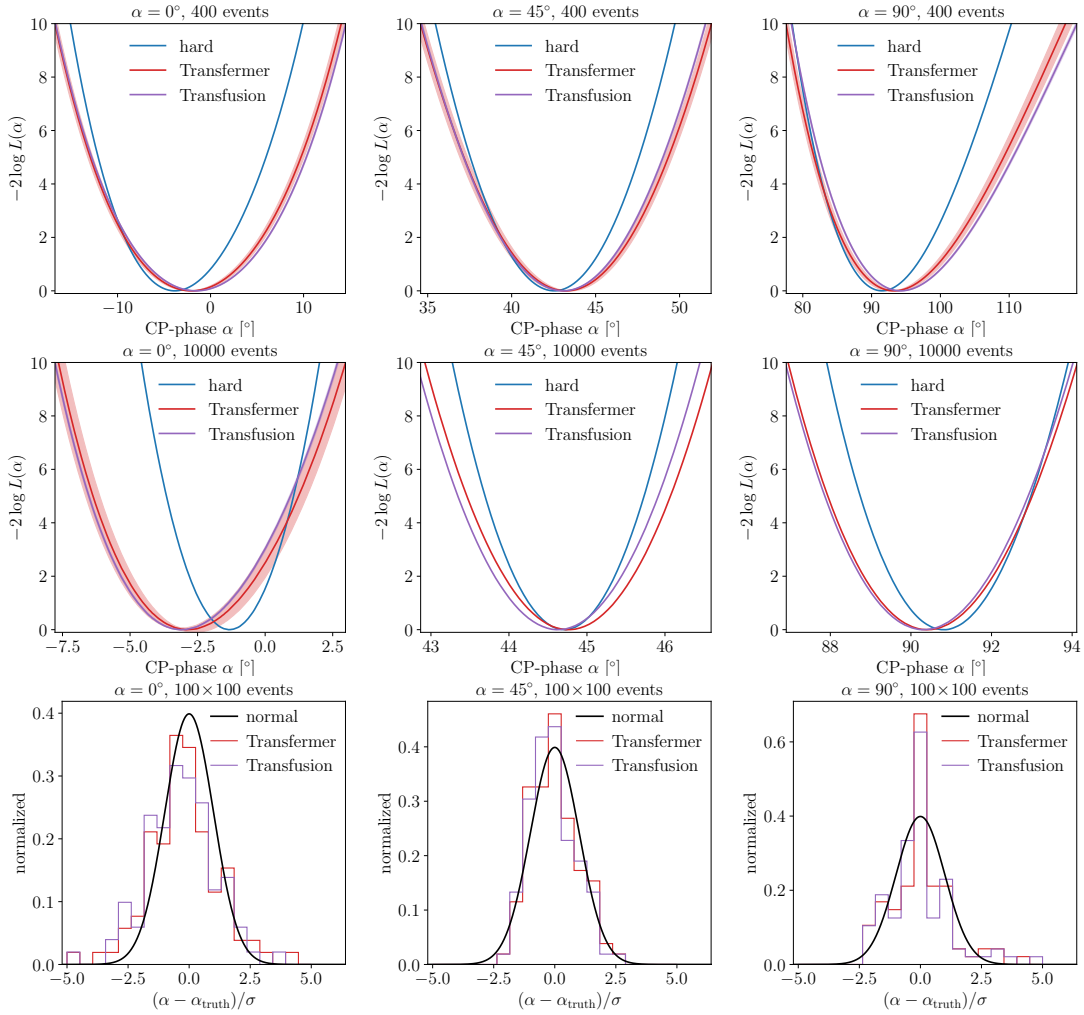


Figure 6.19: **Transfermer and Transfusion:** likelihoods for different CP-angles using a transformer for the transfer probability, combined with a cINN or a CFM network, respectively. From top to bottom: likelihoods for 400 events, 10000 events, and pulls. Only the Transfermer curve includes the training uncertainties estimated with the Bayesian network.

the Transfermer architecture in Fig. 6.19. The error bands in this plot are much larger than in the previous plots because we also included the systematic uncertainty from the training of the Transfermer, which we estimated with a Bayesian network and propagated using the procedure from Sec. 6.4.1. We omitted the Bayesian uncertainties for the other architectures as they lead to longer integration times. The accuracy is similar to the Transfer-CFM from Fig. 6.17, and there is again a significant reduction in bias compared to the cINN result from Fig. 6.14. The main advantage of the Transfermer is that it is as fast as the Transfer-cINN, while not being as restricted as a regular invertible architecture because of the flexibility of the transformer.

Transfusion

We saw a considerable improvement when the Transfer-cINN was replaced with the Transfer-CFM. Consequently, we should also ask the question whether replacing the small

cINN in the Transfermer with a CFM leads to further performance improvements. We refer to this combination of both architectures as Transfusion. We build the Transfusion network by keeping the autoregressive structure and masked self-attention from Fig. 6.18, and replace the cINN for the individual particle momenta with a small CFM. In analogy to Eq. (6.40), we then define a learnable velocity field for the i-th particle as a function of the transformer output and the time t,

$$v^{(i)}(x^{(i)}_{\text{reco}}(t), t | c^{(i)}(e^{(0)}_{\text{reco}}, \ldots, e^{(i-1)}_{\text{reco}}, e_{\text{hard}})) \; . \tag{6.44}$$

We build two different CFMs, one for the 3-dimensional on-shell momenta and a second one for the 4-dimensional jet momenta. This performs better than a shared network for both types of momenta where the mass is discarded for on-shell particles. The likelihoods can again be obtained by integrating the gradients of the velocity field, see Eq. (3.55). Note that the transformer output does not depend on the timestep t. As a consequence, it is sufficient to evaluate the transfermer once per autoregressive sampling step instead of evaluating its gradients in every timestep of the diffusion process. We train the Transfusion network with the hyperparameters given in Tab. A.11.

As seen in Fig. 6.19, the MEM likelihoods obtained from the Transfusion network are very similar to those from the Transfermer network. This means that the small cINN in the Transfermer is sufficiently expressive to model the distribution of the single-particle momenta. There is no additional benefit in replacing the cINN with a more flexible CFM, as the complicated task of extracting the correlations between the particles and solving the jet combinatorics is performed by the transformer. However, the Transfusion architecture is conceptually interesting because it can be modified to replace the autoregressive structure with a permutation-invariant structure [5]. Then, it does no longer rely on a specific ordering of the particles, similar to transformer-based point-cloud architectures like in Ref. [72].

Transfermer with variable jet number

So far, we have always ignored additional jets in the transfer probability because most generative architectures like cINNs or CFMs are constructed for a fixed dimension. We could solve this by training multiple networks for different multiplicities, or use a structure like in the precision generator from Sec. 4.2. It is much simpler to implement the variable

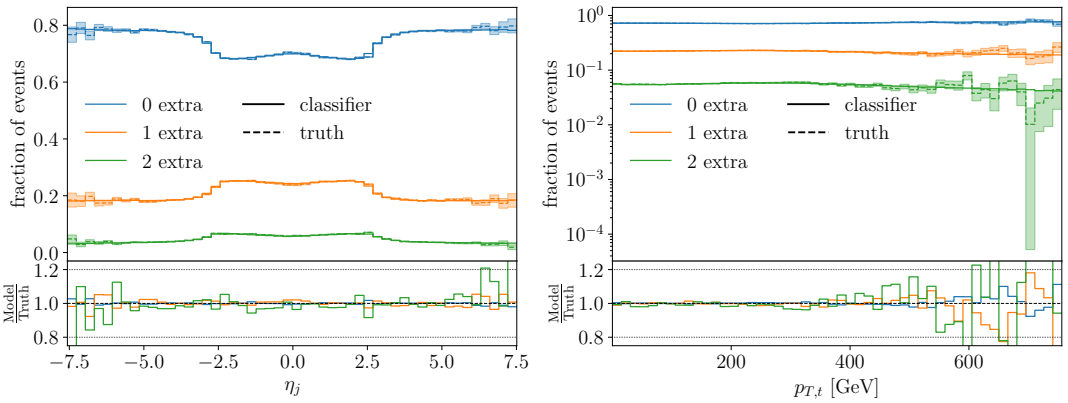


Figure 6.20: Probability of different multiplicities of additional jets as a function of hard-scattering phase space for different kinematic observables.

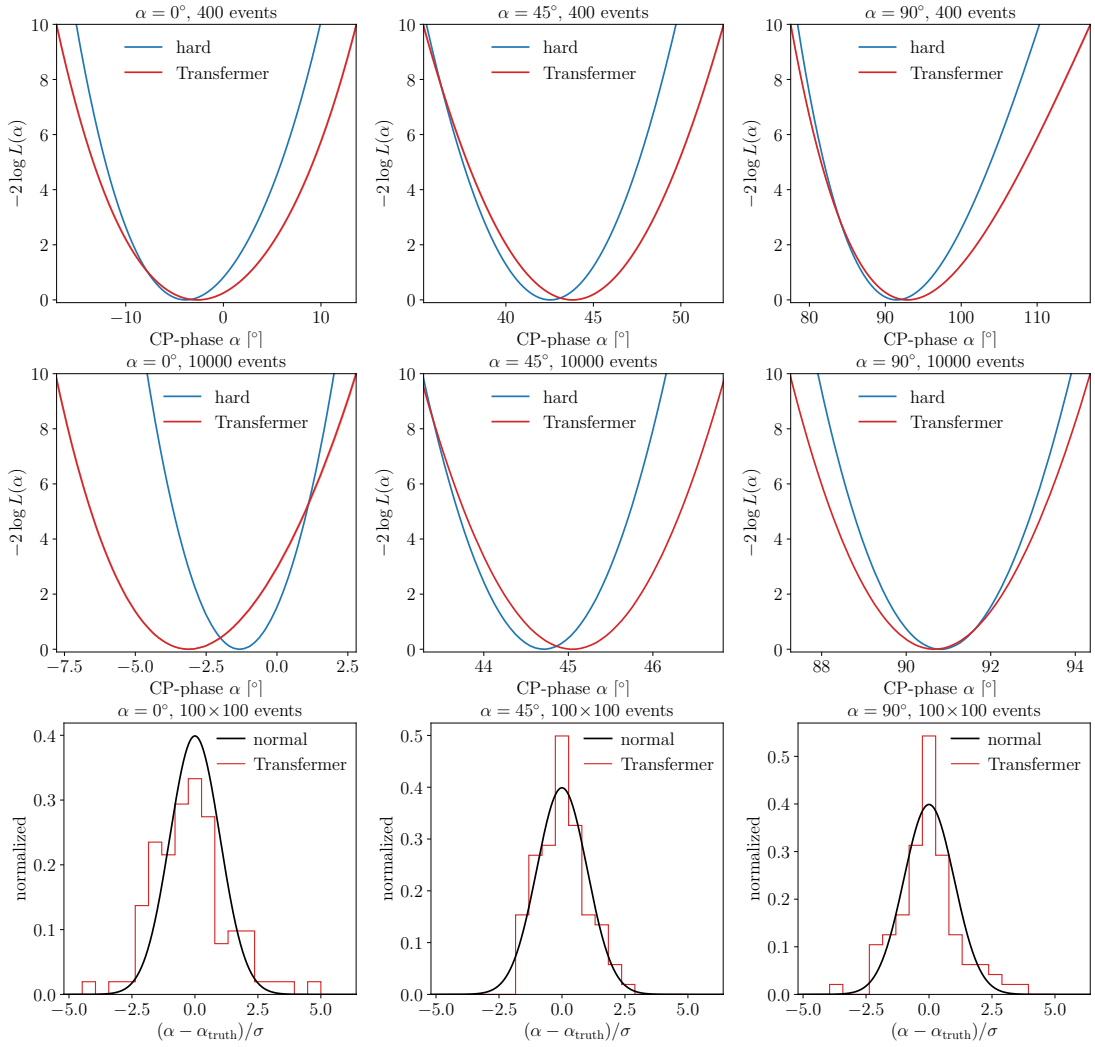


Figure 6.21: **Transfermer with variable jet numbers:** likelihoods for different CP-angles using the Transfermer with variable jet multiplicity as the transfer probability. From top to bottom: likelihood for 400 events, 10000 events, and pulls.

multiplicity for the Transfermer, as transformers are designed to work with variable sequence lengths. To this end, we split the inclusive transfer probability and evaluate it autoregressively,

$$\begin{aligned} p(x_\text{reco}, n|x_\text{hard}) &= p(n|x_\text{hard})\, p(x_\text{reco}|x_\text{hard}, n) \\ &= p(n|x_\text{hard}) \prod_{i=1}^{n} p(x_\text{reco}^{(i)}|c^{(i)}(e_\text{reco}^{(0)}, \dots, e_\text{reco}^{(i-1)}, e_\text{hard})) \ , \end{aligned} \tag{6.45}$$

where n is the number of final-state particles. This is similar to Eq. (6.40), except that we also need a model for the probability for the multiplicity n given the hard-scattering momenta x_hard. We pass the information about the multiplicity n to the transformer by appending it to the embedding of x_hard in one-hot encoded form. To sample from the transfer probability, we first sample the multiplicity n, followed by the usual autoregressive sampling of the momenta as described in Eq. (6.41). Note that the Transfermer also accepts a variable number of hard-scattering momenta without any

further changes. This makes it a good candidate for an extension of our machine-learned MEM to next-to-leading-order matrix elements.

To learn $p(n|x_{\text{hard}})$ we train a classifier with the hard-scattering momenta as inputs, the jet multiplicity as truth labels, and a categorical cross-entropy loss. We allow for up to two additional jets since a higher number of jets is only present for a negligible number of events. The hyperparameters are given in Tab. A.8. We show the probability for the different multiplicities as a function of different hard-scattering phase-space observables in Fig. 6.20. Like for the acceptance in Fig. 6.15, there is a drop in multiplicity for $|\eta| > 2.4$. Again, this is because ISR jets are tagged instead of forward jets from the hard process. For other observables like the transverse momentum of the top, the probability for different multiplicities is mostly flat.

We then run the MEM integration using Eq. (6.45) as the transfer probability, i.e. both the Transfermer and the multiplicity classifier have to be evaluated for every hard-scattering sample. The resulting likelihoods are shown in Fig. 6.21. They are similar or slightly worse than the results with a fixed multiplicity shown in Fig. 6.19. This indicates that the additional information from extra jets does not significantly increase the sensitivity for this specific process, while the training setup gets more complicated because of the larger number of momenta for some events. However, extracting information from additional jets could become relevant for other processes or the extension of the MEM to NLO matrix elements.

6.6 Conclusion

The matrix element method is a powerful inference method to measure fundamental Lagrangian parameters. It makes close to optimal use of the available information. This makes it attractive for LHC processes with very few events. To reach a high-level of precision, the MEM requires three key ingredients,

1. a precise model of the transfer-probability, encoding the effects of shower, hadronization, detector and reconstruction;
2. a flexible description of the acceptance probability as a function of phase space;
3. and precise phase-space mappings to make the integration over the hard-scattering phase space efficient.

We showed for the measurement of the CP phase in associated production of a Higgs and single top at the LHC that a combination of three neural networks provides the necessary speed and precision.

We started our study with a simplified setup where we neglected acceptance effects and used two cINNs to model the transfer function and sample hard-scattering phase space. We found that the sensitivity for events with a leptonic decay of the top was close to the hard-scattering truth. We saw similar results for hadronic decays when initial state radiation was neglected. However, after including ISR, jet combinatorics and acceptance effects became limiting factors. Next, we presented a series of numerical improvements that improved speed and accuracy of the phase-space integration. Furthermore, we added a classifier network to encode acceptance effects, reducing the measurement bias for the complicated case of hadronic decays with ISR jets.

Finally, we tested multiple generative architectures to improve the precision of the transfer probability. We showed that conditional flow matching models reach a much higher level of precision compared to cINNs. However, density estimation and therefore phase-space integration is much slower with diffusion-based transfer probabilities, which might eventually be solved using distillation techniques [232–234]. We then demonstrated how an autoregressive transformer combined with cINNs, called the Transfermer, improves the precision by solving the challenging jet combinatorics. We also showed that a combination of diffusion models and transformers, Transfusion, achieves a similar level of precision. Furthermore, we showed that these transformer-based architectures can be easily adapted to support variable numbers of jets, a requirement for future implementations of the MEM at next-to-leading-order precision [235–238]. The next step in the development of the ML-based matrix element method is to test these different architectures in the setting of an actual LHC analysis.

Chapter 7

Summary and outlook

The LHC is entering an era of precision measurements, and is producing vast amounts of data. The two key ingredients to ensure its success are fast and precise simulations, and improved analysis techniques. In this thesis, we presented multiple applications of generative networks that can help tackle these challenges.

In Chapter 4, we showed how invertible neural networks can be used as an end-to-end surrogate for the LHC simulation chain. For the example of a leptonically decaying Z with one to three QCD jets, we implemented an INN-based model that is able to generate events with a variable jet multiplicity. Furthermore, we showed how challenging features like the cut and collinear enhancement in the angular separation between jets can be addressed through appropriate transformations of the training data. Our generator achieved percent-level precision in important kinematic distributions.

When a neural network is used to replace parts of the LHC simulation chain, it is crucial to understand the uncertainties introduced by this approach. We investigated two methods to capture different parts of the uncertainty. First, we used a Bayesian version of our INN generator to extract the uncertainties introduced by the limited amount of training data. We confirmed that the uncertainties predicted by the BINN have the expected behavior, for instance in tails of distributions where only few training points were available, or for trainings on very small datasets. Second, we looked at the uncertainties from imperfect training and the lack of expressivity of the generative model. We used classifier networks to extract these uncertainties by learning the weight between the generated and truth distribution. We demonstrated how these weights can be used to improve the generated samples through reweighting, but also as a diagnostic tool to identify failure modes of the training, and as a performance metric to assess the quality of generative models. Lastly, we showed that the uncertainties captured by Bayesian networks and classifiers are in some cases related, and in other cases orthogonal. Hence, both approaches are important tools to understand the uncertainties introduced by generative networks.

Even when a large part of the LHC simulation chain is replaced with a generative network, this network still relies on Monte Carlo generated training data. In Chapter 5, we discussed how neural networks can be used to accelerate phase-space integration and event generation at the level of the hard process while guaranteeing exact sampling from the truth distribution. To this end, we replaced the established VEGAS algorithm used for adaptive importance sampling in most event generators with a normalizing flow, and showed how the multi-channel approach used to model complex phase-space distributions can be refined with neural networks. We validated our approach for simple toy distributions and found that the method performs best when it is used to refine pre-defined channel decompositions and mappings that are motivated by the target

function. Based on these results, we applied our method to the phase-space mappings, matrix elements and multi-channel weights provided by MG5aMC. This gives our method the name "MadNIS – MadGraph Neural Importance Sampling". We discussed several improvements of the method that go beyond a simple neural multi-channel importance sampling setup, including buffered training to accelerate the network training for costly matrix elements, Vegas initialization to give the training a better starting point, and stratified training and channel dropping to make it more stable and efficient. We benchmarked the MadNIS performance for several LHC processes and found that the combination of the various MadNIS features led to improvements in unweighting efficiency between five and ten for most processes. In the case of a difficult vector boson scattering process, we saw a gain of 15 in unweighting efficiency. Further, we showed that the method stays efficient even for processes with higher multiplicities.

We then moved from ML-based simulations to ML-based methods for precision analyses in Chapter 6. We showed how the differential cross section of a given process can be combined with three neural networks to build an ML-based version of the matrix element method. It allows for the measurement of theory parameters using all the available kinematic information. In particular, we trained (i) a generative network encoding the transfer probability from the effects of shower, hadronization, detector and reconstruction, (ii) a classifier encoding phase-space-dependent detector effects, and (iii) a normalizing flow for efficient phase-space integration. We tested our method for the associated production of a Higgs and single top at the LHC with an anomalous CP-phase. We started with a simplified version neglecting acceptance effects that was based on cINNs for the transfer probability and sampling networks. We demonstrated that it works well for the leptonic and hadronic decay channel of the top, as long as initial state radiation is neglected. We then showed how the systematic deviation that arises when ISR is included, can be solved by modeling phase-space-dependent acceptance effects. Furthermore, we discussed several methods to make the integration over hard-scattering phase space more precise and efficient. Even with these improvements, there were still systematic deviations in the results, caused by the lack of precision of the transfer network. We tested several different network architectures and found that replacing the Transfer-cINN with a diffusion network significantly improved the results, but also led to slower inference. Finally, we showed that a generative transformer solving the challenging jet combinatorics provides a fast and precise estimate of the transfer probability.

The work presented in this thesis opens the possibility for many applications and extensions. Our discussion of the use of generative models for event generation was focused on the aspect of speeding up parts of the LHC simulation chain. However, they could also be used as a compression tool. Instead of transferring a large number of events, a trained generative model could be used. This idea could not only be applied to Monte Carlo-generated events, but also to real LHC events, providing an efficient way to make measured data available for phenomenological studies without the need to reduce it to binned, low-dimensional summary statistics. Related applications of generative models have been presented in Refs. [239, 240].

The MadNIS phase-space sampler has almost progressed to a state where it could be applied for LHC event generation in practice. Remaining tasks are to optimize the interface between MadNIS and MG5aMC, extend the method to processes with more than one partonic initial state and variable flavors in the final state, and to find reliable settings that work for most LHC processes, including ones with large multiplicities. Furthermore, there are several complementary approaches to speed up the sampling of hard-scattering events. These could be combined with the MadNIS method. One

promising approach is to move matrix element computations to GPUs [241, 242]. In MadNIS, this could drastically decrease the time needed for the upfront training of the network. This training has to be done jointly for all channels and cannot be easily distributed to multiple machines, making it advantageous to move it to highly parallelized hardware. We have presented buffered training as a solution to reduce the number of computationally expensive matrix element evaluations. The same can also be achieved using fast surrogate networks [48, 49, 55, 243], for example to replace costly loop amplitude computations. Combined with MadNIS for phase-space sampling, this could further accelerate network training and event generation. Lastly, the matrix elements for the processes used in MadNIS and for the matrix element method were both evaluated at leading order in perturbation theory. Event generation including next-to-leading order QCD effects is already automated in MG5aMC [135], and ways to extend the MEM to NLO precision have been proposed [235–238]. Therefore, the next step for the corresponding machine learning frameworks will be to extend them to work with NLO matrix elements, and address the arising challenges like events with negative weights or costly loop integral computations.

We can conclude that generative networks are powerful tools to accelerate simulations and improve precision measurements at the LHC. They have the potential to strongly impact the way that we work with collider data. As the whole field of machine learning applications in particle physics is quickly progressing, it is likely that ML will become a standard tool for physics at the high-luminosity LHC and beyond.

Chapter A

Hyperparameters

The following tables show the hyperparameters for the various networks that we discussed in this thesis.

Precision event generation

hyperparameter	INN (Sec. 4.2)	BINN (Sec. 4.3)
LR scheduling	one-cycle	same
Starter LR	10^{-4}	10^{-5}
Maximum LR	10^{-3}	10^{-4}
Epochs	100	100
Batch size	1024	3072
ADAM β_1, β_2	0.9, 0.99	same
Coupling block	cubic spline	same
# spline bins	60	same
# coupling blocks	25	20
Layers per block	3	6
# generated events	2M	1M

Table A.1: Training setup and hyperparameters for the INN generators used in our different setups from Ch. 4.

Parameter	Events $Z + \{1, 2, 3\}$ jets
Optimizer	Adam
Learning rate	0.001
LR schedule	reduce on plateau
Decay factor	0.1
Decay patience (epochs)	5
Batch size	1024
Epochs	50
Number of layers	5
Hidden nodes	256
Dropout	10%
Activation function	leaky ReLU
Training samples	2.7M / 750k / 210k
Validation samples	300k / 80k / 20k
Testing samples	3.0M / 830k / 240k

Table A.2: Hyperparameters of the classifier network from Sec. 4.4 applied to the event generation dataset.

Neural importance sampling

Parameter	Value
Loss function	variance
Learning rate	0.001
LR schedule	inverse time decay
Decay rate	0.01
Batch size	128
Epochs	20
Batches per Epoch	100
Number of layers	3
Hidden nodes	16
Activation function	leaky ReLU

Table A.3: Hyperparameters of the multi-channel weight network for the 1-dimensional camel back, see Sec. 5.2.1.

Parameter	Value
Loss function	variance
Learning rate	0.0005 (0.001)
LR schedule	inverse time decay
Decay rate	0.02
Batch size	1024
Epochs	100
Batches per Epoch	500
Coupling blocks	affine
Permutations	soft
Blocks	6
Subnet hidden nodes	32 (16)
Subnet layers	3 (2)
CWnet layers	2
CWnet hidden nodes	16
Activation function	leaky ReLU

Table A.4: Hyperparameters of the INN and the channel weight network (CWnet) for the crossed ring. The numbers in parentheses indicate that a different setting was used for a ring mapping, see Sec. 5.2.2.

Parameter	Value
Loss function	variance
Learning rate	0.001
LR schedule	inverse time decay
Decay rate	0.01
Batch size	10000
Epochs	60
Batches per epoch	50
Coupling blocks	rational-quadratic splines
Permutations	exchange
Blocks	6
Subnet hidden nodes	16
Subnet layers	2
CWnet layers	2
CWnet hidden nodes	16
Activation function	leaky ReLU

Table A.5: Hyperparameters of the INN and the channel weight network (CWnet) for the integration of the Drell-Yan + Z' cross section, see Sec. 5.2.3.

Parameter	Value
Optimizer	Adam
Learning rate	0.001
LR schedule	Inverse decay
Final learning rate	0.0001
Batch size	$\min(200 \cdot n_c^{0.8}, 10000)$
Training length	88k batches
Permutations	Logarithmic decomposition [45]
Number of coupling blocks	$2\,\lceil \log_2 D \rceil$
Coupling transformation	RQ splines
Subnet hidden nodes	32
Subnet depth	3
CWnet parametrization	$(\log p_T, \eta, \phi)$
CWnet hidden nodes	64
CWnet depth	3
Activation function	leaky ReLU
Max. # of buffered channel weights	75
Buffer size	1000 batches
Channel dropping cutoff	0.001
Uniform training fraction r	0.1
VEGAS iterations	7 (7)
VEGAS bins	64 (128)
VEGAS samples per iteration	20k (50k)
VEGAS damping α	0.7 (0.5)

Table A.6: MADNIS hyperparameters for the results shown in Sec. 5.3. For the VEGAS parameters, the first value is used for the pre-training and the value in parentheses for the remaining runs.

ML for the matrix element method

Parameter	cINN
Blocks	20
Block type	Rational quadratic
Layers per block	5
Units per layer	256
Spline bins	30
Epochs (Bayesian)	100 (200)
Learning rate scheduling	One-cycle
Initial learning rate	$1 \cdot 10^{-4}$
Maximum learning rate	$3 \cdot 10^{-4}$
Batch size	1024
Training events	1.3M

Table A.7: Identical setup and hyper-parameters for the Transfer-cINN and the Unfolding-cINN used in Sec. 6.3.

Parameter	Acceptance	Multiplicities
Optimizer	Adam	
Learning rate	0.0001	
LR schedule	One-cycle	
Maximum learning rate	0.0003	
Batch size	1024	
Epochs	10	
Number of layers	6	
Hidden nodes	256	
Activation function	ReLU	
Preprocessing	p_T, η, ϕ, m	
Loss	Binary cross-entropy	Categorical cross-entropy
Training samples	5M	3.4M
Validation samples	500k	340k
Testing samples	4.5M	3.1M
Trainable parameters	266k	266k

Table A.8: Hyperparameters of the classifiers learning the acceptance $\epsilon(x_{\text{hard}})$ described in Sec. 6.4.2 (left) and the jet multiplicity used in Sec. 6.5.2 (right).

Parameter	Value
Optimizer	Adam
Learning rate	0.001
LR schedule	Cosine-annealing
Batch size	16384
Epochs	1000
Number of layers	8
Feed-forward dimension	512
Activation function	SiLU
Training samples	3.4M
Validation samples	340k
Testing samples	3.1M
Trainable parameters	3.2M
ODE solver method	Runge-Kutta 4
Solver step-size	0.05

Table A.9: Hyperparameters of the Transfer-CFM, see Sec. 6.5.1.

Parameter	Value
Optimizer	RAdam
Learning rate	0.0001
LR schedule	One-cycle
Maximum learning rate	0.0003
Batch size	1024
Epochs	200
Number of heads	8
Number of encoder layers	6
Number of decoder layers	8
Embedding dimension	64
Transformer feed-forward dimension	256
Number of subnet layers	5
Subnet hidden nodes	256
Subnet activation function	ReLU
RQS spline bins	16
Training samples	3.4M
Validation samples	340k
Testing samples	3.1M
Trainable parameters	2.6M

Table A.10: Hyperparameters of the Transfermer, see Sec. 6.5.2.

Parameter	Value
Optimizer	Adam
Learning rate	0.001
LR schedule	Cosine-annealing
Batch size	8192
Epochs	600
Number of heads	8
Number of encoder layers	6
Number of decoder layers	8
Embedding dimension	64
Transf. feed-forward dim	256
Number of layers CFM	6
Hidden nodes CFM	400
Activation function CFM	ReLU
Training samples	3.4M
Validation samples	340k
Testing samples	3.1M
Trainable parameters	3.5M
ODE solver method	Runge-Kutta, order 4
Solver step-size	0.05

Table A.11: Hyperparameters of the autoregressive Transfusion, see Sec. 6.5.2

Acknowledgments

First of all, I am grateful to my advisor Tilman Plehn for his support over the last six years, and for accepting me as a bachelor-, then as a master- and finally as a PhD student. I would like to thank him for giving me the opportunity to work on so many interesting projects, for letting me travel to lots of conferences and workshops with his almost infinite travel budget, and for giving up on trying to nano-manage me.

Furthermore, I would like to thank Björn Malte Schäfer for refereeing this thesis, as well as Stephanie Hansmann-Menzemer and Fred Hamprecht for completing my examination committee.

I am grateful to the research training group "Particle physics beyond the Standard Model" and the project "Model-Based AI: Physical Models and Deep Learning for Imaging and Cancer Treatment" of the Carl-Zeiss-Stiftung for funding the first two and the third year of my research, respectively.

Moreover, I would like to thank all the people who I collaborated with during the last three years. Special thanks go to Ramon Winterhalder for the great collaboration, for immediately responding to Skype messages at any time of the day, and for coming up with the name MadNIS; to Anja Butter for the collaboration on almost all of the projects in this thesis; to Nathan Huetsch for the great collaboration on the second MEM project; to Olivier Mattelaer for his hands-on help with bringing together ancient Fortran code in MadGraph and modern machine learning; to Armand Rousselot for building a bridge between physics and computer science; and to all of the bachelor and master students that I worked with over the course of my thesis: Sophia Vent, Sander Hummerich, Tobias Krebs, Simon Pijahn, Joran Köhler and Jona Ackerschott. Furthermore, I am grateful to the other collaborators that I have not mentioned yet for the successful work: Ranit Das, Luigi Favaro, Joshua Isaacson, Claudius Krause, Fabio Maltoni, Sascha Peitzsch, Till Martini and David Shih. Furthermore, I am grateful to the team at Schmelzpunkt for providing ice cream all year round, without which the work presented in this thesis would not have been possible.

I would like to thank Ramon Winterhalder, Ayodele Ore, Luigi Favaro, Nathan Huetsch, Jonas Spinner, Nina Elmer and Lorenz Vogel for helping me to make this thesis readable and nice-looking.

Most importantly, I would like to thank all the current and former members of the Heidelberg Pheno group for making the last three years such a fun time! In no particular order, I would like to thank Michel Luchmann for always being available for a coffee break; Emma Geoffray for teaching the group how to be a kind and happy researcher and for a great trip to New York; Nathalie Soybelman for bringing everyone together over pizza, beer, darts and Israeli pop; Luigi Favaro for countless darts matches, inviting Nina and me to visit him and his parents during our summer vacation, and for tolerating

my weird opinions on egg white in spaghetti carbonara; Benedikt Schosser for joining the pasta competition with Luigi and me; Lennart Röver for initiating lots of Spikeball games; Lorenz Vogel for typographic advice and for paying other people's food and drinks way too often; Jonas Spinner for always being enthusiastic to discuss about physics or machine learning; Tanmoy Modak for many dinner invitations with amazing Indian food; as well as Bob the evil paper roll, Marco Bellagente, Lennert Thormählen, Thorsten Buss, Jona Ackerschott, Nikita Schmal, Nathan Huetsch, Sofia Palacios Schweitzer, Claudius Krause, Florian Ernst, Maeve Madigan, Ayodele Ore, Giovanni de Crescenzo, Javier Villadamigo, Henning Bahl, Victor Breso and anyone else I unfortunately forgot to mention.

Very special thanks goes to Nina Elmer for the great time that we had together since I met you in the Heidelberg Pheno group! Thank you for your support over the last two years, for helping me to not spend too much time on work, and for making my life much better, both inside and outside of the institute.

Finally, I would like to thank my friends and family, and especially my parents, for their support throughout the years. Without you, this would not have been possible!

Bibliography

[1] A. Butter, T. Heimel, S. Hummerich, T. Krebs, T. Plehn, A. Rousselot and S. Vent, *Generative networks for precision enthusiasts*, SciPost Phys. **14**, 078 (2023), doi:10.21468/SciPostPhys.14.4.078, arXiv:2110.13632.

[2] A. Butter, T. Heimel, T. Martini, S. Peitzsch and T. Plehn, *Two Invertible Networks for the Matrix Element Method*, SciPost Phys. **15**, 094 (2023), doi:10.21468/SciPostPhys.15.3.094, arXiv:2210.00019.

[3] T. Heimel, R. Winterhalder, A. Butter, J. Isaacson, C. Krause, F. Maltoni, O. Mattelaer and T. Plehn, *MadNIS – Neural Multi-Channel Importance Sampling*, SciPost Phys. **15**, 141 (2023), doi:10.21468/SciPostPhys.15.4.141, arXiv:2212.06172.

[4] R. Das, L. Favaro, T. Heimel, C. Krause, T. Plehn and D. Shih, *How to Understand Limitations of Generative Networks*, SciPost Phys. **16**, 031 (2024), doi:10.21468/SciPostPhys.16.1.031, `https://scipost.org/10.21468/SciPostPhys.16.1.031`, arXiv:2305.16774.

[5] T. Heimel, N. Huetsch, R. Winterhalder, T. Plehn and A. Butter, *Precision-Machine Learning for the Matrix Element Method* (2023), arXiv:2310.07752.

[6] T. Heimel, N. Huetsch, F. Maltoni, O. Mattelaer, T. Plehn and R. Winterhalder, *The MadNIS Reloaded* (2023), arXiv:2311.01548.

[7] S. Badger *et al.*, *Machine learning and LHC event generation*, SciPost Phys. **14**, 079 (2023), doi:10.21468/SciPostPhys.14.4.079, arXiv:2203.07460.

[8] J. Ackerschott, R. K. Barman, D. Gonçalves, T. Heimel and T. Plehn, *Returning CP-Observables to The Frames They Belong* (2023), arXiv:2308.00027.

[9] G. Aad *et al.*, *Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC*, Phys. Lett. B **716**, 1 (2012), doi:10.1016/j.physletb.2012.08.020, arXiv:1207.7214.

[10] S. Chatrchyan *et al.*, *Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC*, Phys. Lett. B **716**, 30 (2012), doi:10.1016/j.physletb.2012.08.021, arXiv:1207.7235.

[11] G. Apollinari, O. Brüning, T. Nakamoto and L. Rossi, *High Luminosity Large Hadron Collider HL-LHC*, CERN Yellow Rep. pp. 1–19 (2015), doi:10.5170/CERN-2015-005.1, arXiv:1705.08830.

[12] A. Krizhevsky, I. Sutskever and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems **25** (2012).

[13] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, *Language models are few-shot learners*, Advances in neural information processing systems **33**, 1877 (2020).

[14] R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer, *High-resolution image synthesis with latent diffusion models*, In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695 (2022).

[15] J. M. Campbell *et al.*, *Event Generators for High-Energy Physics Experiments*, In *Snowmass 2021* (2022), arXiv:2203.11110.

[16] A. Butter *et al.*, *The Machine Learning landscape of top taggers*, SciPost Phys. **7**, 014 (2019), doi:10.21468/SciPostPhys.7.1.014, arXiv:1902.09914.

[17] B. Nachman and D. Shih, *Anomaly Detection with Density Estimation*, Phys. Rev. D **101**, 075042 (2020), doi:10.1103/PhysRevD.101.075042, arXiv:2001.04990.

[18] A. Hallin, J. Isaacson, G. Kasieczka, C. Krause, B. Nachman, T. Quadfasel, M. Schlaffer, D. Shih and M. Sommerhalder, *Classifying anomalies through outer density estimation*, Phys. Rev. D **106**, 055006 (2022), doi:10.1103/PhysRevD.106.055006, arXiv:2109.00546.

[19] J. A. Raine, S. Klein, D. Sengupta and T. Golling, *CURTAINs for your sliding window: Constructing unobserved regions by transforming adjacent intervals*, Front. Big Data **6**, 899345 (2023), doi:10.3389/fdata.2023.899345, arXiv:2203.09470.

[20] A. Hallin, G. Kasieczka, T. Quadfasel, D. Shih and M. Sommerhalder, *Resonant anomaly detection without background sculpting*, Phys. Rev. D **107**, 114012 (2023), doi:10.1103/PhysRevD.107.114012, arXiv:2210.14924.

[21] T. Golling, S. Klein, R. Mastandrea and B. Nachman, *Flow-enhanced transportation for anomaly detection*, Phys. Rev. D **107**, 096025 (2023), doi:10.1103/PhysRevD.107.096025, arXiv:2212.11285.

[22] D. Sengupta, S. Klein, J. A. Raine and T. Golling, *CURTAINs Flows For Flows: Constructing Unobserved Regions with Maximum Likelihood Estimation* (2023), arXiv:2305.04646.

[23] J. Brehmer, F. Kling, I. Espejo and K. Cranmer, *MadMiner: Machine learning-based inference for particle physics*, Comput. Softw. Big Sci. **4**, 3 (2020), doi:10.1007/s41781-020-0035-2, arXiv:1907.10621.

[24] S. Bieringer, A. Butter, T. Heimel, S. Höche, U. Köthe, T. Plehn and S. T. Radev, *Measuring QCD Splittings with Invertible Networks*, SciPost Phys. **10**, 126 (2021), doi:10.21468/SciPostPhys.10.6.126, arXiv:2012.09873.

[25] K. Datta, D. Kar and D. Roy, *Unfolding with Generative Adversarial Networks* (2018), arXiv:1806.00433.

[26] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn and R. Winterhalder, *How to GAN away Detector Effects*, SciPost Phys. **8**, 070 (2020), doi:10.21468/SciPostPhys.8.4.070, arXiv:1912.00477.

[27] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman and J. Thaler, *OmniFold: A Method to Simultaneously Unfold All Observables*, Phys. Rev. Lett. **124**, 182001 (2020), doi:10.1103/PhysRevLett.124.182001, arXiv:1911.09107.

[28] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, A. Rousselot, R. Winterhalder, L. Ardizzone and U. Köthe, *Invertible Networks or Partons to Detector and Back Again*, SciPost Phys. **9**, 074 (2020), doi:10.21468/SciPostPhys.9.5.074, arXiv:2006.06685.

[29] M. Backes, A. Butter, M. Dunford and B. Malaescu, *An unfolding method based on conditional Invertible Neural Networks (cINN) using iterative training* (2022), arXiv:2212.08674.

[30] M. Leigh, J. A. Raine, K. Zoch and T. Golling, *ν-flows: Conditional neutrino regression*, SciPost Phys. **14**, 159 (2023), doi:10.21468/SciPostPhys.14.6.159, arXiv:2207.00664.

[31] A. Shmakov, K. Greif, M. Fenton, A. Ghosh, P. Baldi and D. Whiteson, *End-To-End Latent Variational Diffusion Models for Inverse Problems in High Energy Physics* (2023), arXiv:2305.10399.

[32] S. Diefenbacher, G.-H. Liu, V. Mikuni, B. Nachman and W. Nie, *Improving Generative Model-based Unfolding with Schrödinger Bridges* (2023), arXiv:2308.12351.

[33] HEP ML Community, *A Living Review of Machine Learning for Particle Physics*.

[34] 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), doi:10.1016/j.cpc.2015.01.024, arXiv:1410.3012.

[35] E. Bothmann *et al.*, *Event Generation with Sherpa 2.2*, SciPost Phys. **7**, 034 (2019), doi:10.21468/SciPostPhys.7.3.034, arXiv:1905.09127.

[36] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H. S. Shao, T. Stelzer, P. Torrielli and M. Zaro, *The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations*, JHEP **07**, 079 (2014), doi:10.1007/JHEP07(2014)079, arXiv:1405.0301.

[37] J. Bellm *et al.*, *Herwig 7.0/Herwig++ 3.0 release note*, Eur. Phys. J. C **76**, 196 (2016), doi:10.1140/epjc/s10052-016-4018-8, arXiv:1512.01178.

[38] W. Kilian, T. Ohl and J. Reuter, *WHIZARD: Simulating Multi-Particle Processes at LHC and ILC*, Eur. Phys. J. C **71**, 1742 (2011), doi:10.1140/epjc/s10052-011-1742-y, arXiv:0708.4233.

[39] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens and M. Selvaggi, *DELPHES 3, A modular framework for fast simulation of a generic collider experiment*, JHEP **02**, 057 (2014), doi:10.1007/JHEP02(2014)057, arXiv:1307.6346.

[40] S. Agostinelli *et al.*, *GEANT4–a simulation toolkit*, Nucl. Instrum. Meth. A **506**, 250 (2003), doi:10.1016/S0168-9002(03)01368-8.

[41] *ATLAS HL-LHC Computing Conceptual Design Report* (2020).

[42] J. Bendavid, *Efficient Monte Carlo Integration Using Boosted Decision Trees and Generative Deep Neural Networks* (2017), arXiv:1707.00028.

[43] M. D. Klimek and M. Perelstein, *Neural Network-Based Approach to Phase Space Integration*, SciPost Phys. **9**, 053 (2020), doi:10.21468/SciPostPhys.9.4.053, arXiv:1810.11509.

[44] I.-K. Chen, M. D. Klimek and M. Perelstein, *Improved Neural Network Monte Carlo Simulation*, SciPost Phys. **10**, 023 (2021), doi:10.21468/SciPostPhys.10.1.023, arXiv:2009.07819.

[45] C. Gao, J. Isaacson and C. Krause, *i-flow: High-dimensional Integration and Sampling with Normalizing Flows*, Mach. Learn. Sci. Tech. **1**, 045023 (2020), doi:10.1088/2632-2153/abab62, arXiv:2001.05486.

[46] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale and S. Schumann, *Exploring phase space with Neural Importance Sampling*, SciPost Phys. **8**, 069 (2020), doi:10.21468/SciPostPhys.8.4.069, arXiv:2001.05478.

[47] C. Gao, S. Höche, J. Isaacson, C. Krause and H. Schulz, *Event Generation with Normalizing Flows*, Phys. Rev. D **101**, 076002 (2020), doi:10.1103/PhysRevD.101.076002, arXiv:2001.10028.

[48] K. Danziger, T. Janßen, S. Schumann and F. Siegert, *Accelerating Monte Carlo event generation – rejection sampling using neural network event-weight estimates*, SciPost Phys. **12**, 164 (2022), doi:10.21468/SciPostPhys.12.5.164, arXiv:2109.11964.

[49] T. Janßen, D. Maître, S. Schumann, F. Siegert and H. Truong, *Unweighting multijet event generation using factorisation-aware neural networks*, SciPost Phys. **15**, 107 (2023), doi:10.21468/SciPostPhys.15.3.107, arXiv:2301.13562.

[50] E. Bothmann, T. Childers, W. Giele, F. Herren, S. Hoeche, J. Isaacson, M. Knobbe and R. Wang, *Efficient phase-space generation for hadron collider event simulation*, SciPost Phys. **15**, 169 (2023), doi:10.21468/SciPostPhys.15.4.169, arXiv:2302.10449.

[51] F. Bishara and M. Montull, *Machine learning amplitudes for faster event generation*, Phys. Rev. D **107**, L071901 (2023), doi:10.1103/PhysRevD.107.L071901, arXiv:1912.11055.

[52] S. Badger and J. Bullock, *Using neural networks for efficient evaluation of high multiplicity scattering amplitudes*, JHEP **06**, 114 (2020), doi:10.1007/JHEP06(2020)114, arXiv:2002.07516.

[53] J. Aylett-Bullock, S. Badger and R. Moodie, *Optimising simulations for diphoton production at hadron colliders using amplitude neural networks*, JHEP **08**, 066 (2021), doi:10.1007/JHEP08(2021)066, arXiv:2106.09474.

[54] D. Maître and H. Truong, *A factorisation-aware Matrix element emulator*, JHEP **11**, 066 (2021), doi:10.1007/JHEP11(2021)066, arXiv:2107.06625.

[55] S. Badger, A. Butter, M. Luchmann, S. Pitz and T. Plehn, *Loop amplitudes from precision networks*, SciPost Phys. Core **6**, 034 (2023), doi:10.21468/SciPostPhysCore.6.2.034, arXiv:2206.14831.

[56] A. Dersy, M. D. Schwartz and X. Zhang, *Simplifying Polylogarithms with Machine Learning* (2022), arXiv:2206.04115.

[57] R. Winterhalder, V. Magerya, E. Villa, S. P. Jones, M. Kerner, A. Butter, G. Heinrich and T. Plehn, *Targeting multi-loop integrals with neural networks*, SciPost Phys. **12**, 129 (2022), doi:10.21468/SciPostPhys.12.4.129, arXiv:2112.09145.

[58] S. Otten, S. Caron, W. de Swart, M. van Beekveld, L. Hendriks, C. van Leeuwen, D. Podareanu, R. Ruiz de Austri and R. Verheyen, *Event Generation and Statistical Sampling for Physics with Deep Generative Models and a Density Information Buffer*, Nature Commun. **12**, 2985 (2021), doi:10.1038/s41467-021-22616-z, arXiv:1901.00875.

[59] B. Hashemi, N. Amin, K. Datta, D. Olivito and M. Pierini, *LHC analysis-specific datasets with Generative Adversarial Networks* (2019), arXiv:1901.05282.

[60] R. Di Sipio, M. Faucci Giannelli, S. Ketabchi Haghighat and S. Palazzo, *DijetGAN: A Generative-Adversarial Network Approach for the Simulation of QCD Dijet Events at the LHC*, JHEP **08**, 110 (2020), doi:10.1007/JHEP08(2019)110, arXiv:1903.02433.

[61] A. Butter, T. Plehn and R. Winterhalder, *How to GAN LHC Events*, SciPost Phys. **7**, 075 (2019), doi:10.21468/SciPostPhys.7.6.075, arXiv:1907.03764.

[62] Y. Alanazi, N. Sato, T. Liu, W. Melnitchouk, M. P. Kuchera, E. Pritchard, M. Robertson, R. Strauss, L. Velasco and Y. Li, *Simulation of electron-proton scattering events by a Feature-Augmented and Transformed Generative Adversarial Network (FAT-GAN)* (2020), arXiv:2001.11103.

[63] J. N. Howard, S. Mandt, D. Whiteson and Y. Yang, *Learning to simulate high energy particle collisions from unlabeled data*, Sci. Rep. **12**, 7567 (2022), doi:10.1038/s41598-022-10966-7, arXiv:2101.08944.

[64] A. Butter, N. Huetsch, S. P. Schweitzer, T. Plehn, P. Sorrenson and J. Spinner, *Jet Diffusion versus JetGPT – Modern Networks for the LHC* (2023), arXiv:2305.10475.

[65] C. Bierlich, P. Ilten, T. Menzo, S. Mrenna, M. Szewc, M. K. Wilkinson, A. Youssef and J. Zupan, *Towards a data-driven model of hadronization using normalizing flows* (2023), arXiv:2311.09296.

[66] J. Chan, X. Ju, A. Kania, B. Nachman, V. Sangli and A. Siodmok, *Fitting a deep generative hadronization model*, JHEP **09**, 084 (2023), doi:10.1007/JHEP09(2023)084, arXiv:2305.17169.

[67] L. de Oliveira, M. Paganini and B. Nachman, *Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis*, Comput. Softw. Big Sci. **1**, 4 (2017), doi:10.1007/s41781-017-0004-6, arXiv:1701.05927.

[68] A. Andreassen, I. Feige, C. Frye and M. D. Schwartz, *JUNIPR: a Framework for Unsupervised Machine Learning in Particle Physics*, Eur. Phys. J. C **79**, 102 (2019), doi:10.1140/epjc/s10052-019-6607-9, arXiv:1804.09720.

[69] E. Bothmann and L. Debbio, *Reweighting a parton shower using a neural network: the final-state case*, JHEP **01**, 033 (2019), doi:10.1007/JHEP01(2019)033, arXiv:1808.07802.

[70] K. Dohi, *Variational Autoencoders for Jet Simulation* (2020), arXiv:2009.04842.

[71] E. Buhmann, G. Kasieczka and J. Thaler, *EPiC-GAN: Equivariant Point Cloud Generation for Particle Jets*, SciPost Phys. **15**, 130 (2023), doi:10.21468/SciPostPhys.15.4.130, arXiv:2301.08128.

[72] M. Leigh, D. Sengupta, G. Quétant, J. A. Raine, K. Zoch and T. Golling, *PC-JeDi: Diffusion for Particle Cloud Generation in High Energy Physics* (2023), doi:10.21468/SciPostPhys.16.1.018, arXiv:2303.05376.

[73] V. Mikuni, B. Nachman and M. Pettee, *Fast point cloud generation with diffusion models in high energy physics*, Phys. Rev. D **108**, 036025 (2023), doi:10.1103/PhysRevD.108.036025, arXiv:2304.01266.

[74] E. Buhmann, C. Ewen, D. A. Faroughy, T. Golling, G. Kasieczka, M. Leigh, G. Quétant, J. A. Raine, D. Sengupta and D. Shih, *EPiC-ly Fast Particle Cloud Generation with Flow-Matching and Diffusion* (2023), arXiv:2310.00049.

[75] M. Paganini, L. de Oliveira and B. Nachman, *Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multilayer Calorimeters*, Phys. Rev. Lett. **120**, 042003 (2018), doi:10.1103/PhysRevLett.120.042003, arXiv:1705.02355.

[76] L. de Oliveira, M. Paganini and B. Nachman, *Controlling Physical Attributes in GAN-Accelerated Simulation of Electromagnetic Calorimeters*, J. Phys. Conf. Ser. **1085**, 042017 (2018), doi:10.1088/1742-6596/1085/4/042017, arXiv:1711.08813.

[77] M. Paganini, L. de Oliveira and B. Nachman, *CaloGAN : Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks*, Phys. Rev. D **97**, 014021 (2018), doi:10.1103/PhysRevD.97.014021, arXiv:1712.10321.

[78] M. Erdmann, L. Geiger, J. Glombitza and D. Schmidt, *Generating and refining particle detector simulations using the Wasserstein distance in adversarial networks*, Comput. Softw. Big Sci. **2**, 4 (2018), doi:10.1007/s41781-018-0008-x, arXiv:1802.03325.

[79] M. Erdmann, J. Glombitza and T. Quast, *Precise simulation of electromagnetic calorimeter showers using a Wasserstein Generative Adversarial Network*, Comput. Softw. Big Sci. **3**, 4 (2019), doi:10.1007/s41781-018-0019-7, arXiv:1807.01954.

[80] D. Belayneh *et al.*, *Calorimetry with Deep Learning: Particle Simulation and Reconstruction for Collider Physics*, Eur. Phys. J. C **80**, 688 (2020), doi:10.1140/epjc/s10052-020-8251-9, arXiv:1912.06794.

[81] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol and K. Krüger, *Getting High: High Fidelity Simulation of High Granularity Calorimeters with High Speed*, Comput. Softw. Big Sci. **5**, 13 (2021), doi:10.1007/s41781-021-00056-0, arXiv:2005.05334.

[82] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol and K. Krüger, *Decoding Photons: Physics in the Latent Space of a BIB-AE Generative Network*, EPJ Web Conf. **251**, 03003 (2021), doi:10.1051/epjconf/202125103003, arXiv:2102.12491.

[83] C. Krause and D. Shih, *Fast and accurate simulations of calorimeter showers with normalizing flows*, Phys. Rev. D **107**, 113003 (2023), doi:10.1103/PhysRevD.107.113003, arXiv:2106.05285.

[84] ATLAS Collaboration, *AtlFast3: the next generation of fast simulation in ATLAS*, Comput. Softw. Big Sci. **6**, 7 (2022), doi:10.1007/s41781-021-00079-7, arXiv:2109.02551.

[85] C. Krause and D. Shih, *Accelerating accurate simulations of calorimeter showers with normalizing flows and probability density distillation*, Phys. Rev. D **107**, 113004 (2023), doi:10.1103/PhysRevD.107.113004, arXiv:2110.11377.

[86] E. Buhmann, S. Diefenbacher, D. Hundhausen, G. Kasieczka, W. Korcari, E. Eren, F. Gaede, K. Krüger, P. McKeown and L. Rustige, *Hadrons, better, faster, stronger*, Mach. Learn. Sci. Tech. **3**, 025014 (2022), doi:10.1088/2632-2153/ac7848, arXiv:2112.09709.

[87] C. Chen, O. Cerri, T. Q. Nguyen, J. R. Vlimant and M. Pierini, *Analysis-Specific Fast Simulation at the LHC with Deep Learning*, Comput. Softw. Big Sci. **5**, 15 (2021), doi:10.1007/s41781-021-00060-4.

[88] V. Mikuni and B. Nachman, *Score-based generative models for calorimeter shower simulation*, Phys. Rev. D **106**, 092009 (2022), doi:10.1103/PhysRevD.106.092009, arXiv:2206.11898.

[89] ATLAS Collaboration, *Deep generative models for fast photon shower simulation in ATLAS* (2022), arXiv:2210.06204.

[90] C. Krause, I. Pang and D. Shih, *CaloFlow for CaloChallenge Dataset 1* (2022), arXiv:2210.14245.

[91] J. C. Cresswell, B. L. Ross, G. Loaiza-Ganem, H. Reyes-Gonzalez, M. Letizia and A. L. Caterini, *CaloMan: Fast generation of calorimeter showers with density estimation on learned manifolds*, In *36th Conference on Neural Information Processing Systems* (2022), arXiv:2211.15380.

[92] S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, C. Krause, I. Shekhzadeh and D. Shih, *L2LFlows: generating high-fidelity 3D calorimeter images*, JINST **18**, P10017 (2023), doi:10.1088/1748-0221/18/10/P10017, arXiv:2302.11594.

[93] H. Hashemi, N. Hartmann, S. Sharifzadeh, J. Kahn and T. Kuhr, *Ultra-High-Resolution Detector Simulation with Intra-Event Aware GAN and Self-Supervised Relational Reasoning* (2023), arXiv:2303.08046.

[94] A. Xu, S. Han, X. Ju and H. Wang, *Generative Machine Learning for Detector Response Modeling with a Conditional Normalizing Flow* (2023), arXiv:2303.10148.

[95] S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, K. Krüger, P. McKeown and L. Rustige, *New angles on fast calorimeter shower simulation*, Mach. Learn. Sci. Tech. **4**, 035044 (2023), doi:10.1088/2632-2153/acefa9, arXiv:2303.18150.

[96] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, W. Korcari, K. Krüger and P. McKeown, *CaloClouds: fast geometry-independent highly-granular calorimeter simulation*, JINST **18**, P11025 (2023), doi:10.1088/1748-0221/18/11/P11025, arXiv:2305.04847.

[97] M. R. Buckley, C. Krause, I. Pang and D. Shih, *Inductive CaloFlow* (2023), arXiv:2305.11934.

[98] S. Diefenbacher, V. Mikuni and B. Nachman, *Refining Fast Calorimeter Simulations with a Schrödinger Bridge* (2023), arXiv:2308.12339.

[99] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative adversarial nets*, In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 2672–2680. MIT Press, Cambridge, MA, USA (2014), arXiv:1406.2661.

[100] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan and S. Ganguli, *Deep unsupervised learning using nonequilibrium thermodynamics*, In *International conference on machine learning*, pp. 2256–2265. PMLR (2015).

[101] J. Ho, A. Jain and P. Abbeel, *Denoising diffusion probabilistic models*, Advances in neural information processing systems **33**, 6840 (2020).

[102] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel and M. Le, *Flow matching for generative modeling* (2023), arXiv:2210.02747.

[103] M. S. Albergo and E. Vanden-Eijnden, *Building normalizing flows with stochastic interpolants* (2023), arXiv:2209.15571.

[104] X. Liu, C. Gong and Q. Liu, *Flow straight and fast: Learning to generate and transfer data with rectified flow* (2022), arXiv:2209.03003.

[105] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, *Attention is all you need*, Advances in neural information processing systems **30** (2017).

[106] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed and B. Lakshminarayanan, *Normalizing flows for probabilistic modeling and inference*, The Journal of Machine Learning Research **22**, 2617 (2021), arXiv:1912.02762.

[107] L. Dinh, D. Krueger and Y. Bengio, *Nice: Non-linear independent components estimation* (2014), arXiv:1410.8516.

[108] L. Dinh, J. Sohl-Dickstein and S. Bengio, *Density estimation using real nvp* (2016), arXiv:1605.08803.

[109] L. Ardizzone, J. Kruse, C. Rother and U. Köthe, *Analyzing inverse problems with invertible neural networks*, International Conference on Learning Representations (2019), `https://openreview.net/forum?id=rJed6j0cKX`, 1808.04730.

[110] C. Durkan, A. Bekasov, I. Murray and G. Papamakarios, *Neural spline flows*, Advances in Neural Information Processing Systems **32**, 7511 (2019), `https://papers.nips.cc/paper/2019/hash/7ac71d433f282034e088473244df8c02-Abstract.html`, arXiv:1906.04032.

[111] D. MacKay, *Probable Networks and Plausible Predictions – A Review of Practical Bayesian Methods for Supervised Neural Networks*, Comp. in Neural Systems **6**, 4679 (1995), `http://www.inference.org.uk/mackay/network.pdf`.

[112] R. M. Neal, *Bayesian learning for neural networks*, Ph.D. thesis, Toronto (1995).

[113] Y. Gal, *Uncertainty in Deep Learning*, Ph.D. thesis, Cambridge (2016).

[114] A. Kendall and Y. Gal, *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?*, Proc. NIPS (2017), arXiv:1703.04977.

[115] S. Diefenbacher, E. Eren, G. Kasieczka, A. Korol, B. Nachman and D. Shih, *DCTRGAN: Improving the Precision of Generative Models with Reweighting*, JINST **15**, P11004 (2020), doi:10.1088/1748-0221/15/11/P11004, arXiv:2009.03796.

[116] G. P. Lepage, *A New Algorithm for Adaptive Multidimensional Integration*, J. Comput. Phys. **27**, 192 (1978), doi:10.1016/0021-9991(78)90004-9.

[117] G. P. Lepage, *VEGAS: AN ADAPTIVE MULTIDIMENSIONAL INTEGRATION PROGRAM* (1980).

[118] T. Ohl, *Vegas revisited: Adaptive Monte Carlo integration beyond factorization*, Comput. Phys. Commun. **120**, 13 (1999), doi:10.1016/S0010-4655(99)00209-X, arXiv:hep-ph/9806432.

[119] S. Brass, W. Kilian and J. Reuter, *Parallel Adaptive Monte Carlo Integration with the Event Generator WHIZARD*, Eur. Phys. J. C **79**, 344 (2019), doi:10.1140/epjc/s10052-019-6840-2, arXiv:1811.09711.

[120] G. P. Lepage, *Adaptive multidimensional integration: VEGAS enhanced*, J. Comput. Phys. **439**, 110386 (2021), doi:10.1016/j.jcp.2021.110386, arXiv:2009.05112.

[121] K. Kondo, *Dynamical Likelihood Method for Reconstruction of Events With Missing Momentum. 1: Method and Toy Models*, J. Phys. Soc. Jap. **57**, 4126 (1988), doi:10.1143/JPSJ.57.4126.

[122] K. Kondo, *Dynamical likelihood method for reconstruction of events with missing momentum. 2: Mass spectra for 2 → 2 processes*, J. Phys. Soc. Jap. **60**, 836 (1991), doi:10.1143/JPSJ.60.836.

[123] S. L. Glashow, *Partial Symmetries of Weak Interactions*, Nucl. Phys. **22**, 579 (1961), doi:10.1016/0029-5582(61)90469-2.

[124] S. Weinberg, *A Model of Leptons*, Phys. Rev. Lett. **19**, 1264 (1967), doi:10.1103/PhysRevLett.19.1264.

[125] A. Salam, *Weak and Electromagnetic Interactions*, Conf. Proc. C **680519**, 367 (1968), doi:10.1142/9789812795915_0034.

[126] D. J. Gross and F. Wilczek, *Ultraviolet Behavior of Nonabelian Gauge Theories*, Phys. Rev. Lett. **30**, 1343 (1973), doi:10.1103/PhysRevLett.30.1343.

[127] H. D. Politzer, *Reliable Perturbative Results for Strong Interactions?*, Phys. Rev. Lett. **30**, 1346 (1973), doi:10.1103/PhysRevLett.30.1346.

[128] M. E. Peskin and D. V. Schroeder, *An Introduction to quantum field theory*, Addison-Wesley, Reading, USA, ISBN 978-0-201-50397-5 (1995).

[129] J. D. Bjorken, *Asymptotic Sum Rules at Infinite Momentum*, Phys. Rev. **179**, 1547 (1969), doi:10.1103/PhysRev.179.1547.

[130] M. Constantinou *et al.*, *Parton distributions and lattice-QCD calculations: Toward 3D structure*, Prog. Part. Nucl. Phys. **121**, 103908 (2021), doi:10.1016/j.ppnp.2021.103908, arXiv:2006.08636.

[131] R. D. Ball *et al.*, *The path to proton structure at 1% accuracy*, Eur. Phys. J. C **82**, 428 (2022), doi:10.1140/epjc/s10052-022-10328-7, arXiv:2109.02653.

[132] G. Altarelli and G. Parisi, *Asymptotic Freedom in Parton Language*, Nucl. Phys. B **126**, 298 (1977), doi:10.1016/0550-3213(77)90384-4.

[133] Y. L. Dokshitzer, *Calculation of the Structure Functions for Deep Inelastic Scattering and e+ e- Annihilation by Perturbation Theory in Quantum Chromodynamics.*, Sov. Phys. JETP **46**, 641 (1977).

[134] V. N. Gribov and L. N. Lipatov, *Deep inelastic e p scattering in perturbation theory*, Sov. J. Nucl. Phys. **15**, 438 (1972).

[135] J. Alwall *et al.*, *The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations*, JHEP **07**, 079 (2014), doi:10.1007/JHEP07(2014)079, arXiv:1405.0301.

[136] S. Weinzierl, *Introduction to Monte Carlo methods* (2000), arXiv:hep-ph/0006269.

[137] S. Dittmaier and M. Roth, *LUSIFER: A LUcid approach to six FERmion production*, Nucl. Phys. B **642**, 307 (2002), doi:10.1016/S0550-3213(02)00640-5, arXiv:hep-ph/0206070.

[138] S. Plätzer, *RAMBO on diet* (2013), arXiv:1308.2922.

[139] R. Kleiss and R. Pittau, *Weight optimization in multichannel Monte Carlo*, Computer Physics Communications **83**, 141 (1994), doi:10.1016/0010-4655(94)90043-4.

[140] F. Berends, R. Pittau and R. Kleiss, *All electroweak four-fermion processes in electron-positron collisions*, Nuclear Physics B **424**, 308 (1994), doi:10.1016/0550-3213(94)90297-6.

[141] F. Berends, R. Pittau and R. Kleiss, *Excalibur — a Monte Carlo program to evaluate all four-fermion processes at lep 200 and beyond*, Computer Physics Communications **85**, 437 (1995), doi:10.1016/0010-4655(94)00138-R.

[142] F. Maltoni and T. Stelzer, *MadEvent: Automatic event generation with MadGraph*, JHEP **02**, 027 (2003), doi:10.1088/1126-6708/2003/02/027, arXiv:hep-ph/0208156.

[143] O. Mattelaer and K. Ostrolenk, *Speeding up MadGraph5_aMC@NLO*, Eur. Phys. J. C **81**, 435 (2021), doi:10.1140/epjc/s10052-021-09204-7, arXiv:2102.00773.

[144] R. Kleiss and R. Pittau, *Weight optimization in multichannel Monte Carlo*, Comput. Phys. Commun. **83**, 141 (1994), doi:10.1016/0010-4655(94)90043-4, arXiv:hep-ph/9405257.

[145] K. G. Wilson, *Confinement of Quarks*, Phys. Rev. D **10**, 2445 (1974), doi:10.1103/PhysRevD.10.2445.

[146] C. Bierlich *et al.*, *A comprehensive guide to the physics and usage of PYTHIA 8.3*, SciPost Phys. Codeb. **2022**, 8 (2022), doi:10.21468/SciPostPhysCodeb.8, arXiv:2203.11601.

[147] S. Catani and M. H. Seymour, *A General algorithm for calculating jet cross-sections in NLO QCD*, Nucl. Phys. B **485**, 291 (1997), doi:10.1016/S0550-3213(96)00589-5, [Erratum: Nucl.Phys.B 510, 503–504 (1998)], arXiv:hep-ph/9605323.

[148] S. Höche and S. Prestel, *The midpoint between dipole and parton showers*, Eur. Phys. J. C **75**, 461 (2015), doi:10.1140/epjc/s10052-015-3684-2, arXiv:1506.05057.

[149] B. Andersson, G. Gustafson and B. Soderberg, *A General Model for Jet Fragmentation*, Z. Phys. C **20**, 317 (1983), doi:10.1007/BF01407824.

[150] T. Sjostrand, *Jet Fragmentation of Nearby Partons*, Nucl. Phys. B **248**, 469 (1984), doi:10.1016/0550-3213(84)90607-2.

[151] A. Casher, H. Neuberger and S. Nussinov, *Chromoelectric Flux Tube Model of Particle Production*, Phys. Rev. D **20**, 179 (1979), doi:10.1103/PhysRevD.20.179.

[152] B. Andersson, G. Gustafson and T. Sjostrand, *Baryon Production in Jet Fragmentation and Υ Decay*, Phys. Scripta **32**, 574 (1985), doi:10.1088/0031-8949/32/6/003.

[153] G. Bewick *et al.*, *Herwig 7.3 Release Note* (2023), arXiv:2312.05175.

[154] J.-C. Winter, F. Krauss and G. Soff, *A Modified cluster hadronization model*, Eur. Phys. J. C **36**, 381 (2004), doi:10.1140/epjc/s2004-01960-8, arXiv:hep-ph/0311085.

[155] G. Aad *et al.*, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST **3**, S08003 (2008), doi:10.1088/1748-0221/3/08/S08003.

[156] S. Chatrchyan *et al.*, *The CMS Experiment at the CERN LHC*, JINST **3**, S08004 (2008), doi:10.1088/1748-0221/3/08/S08004.

[157] A. M. Sirunyan *et al.*, *Particle-flow reconstruction and global event description with the CMS detector*, JINST **12**, P10003 (2017), doi:10.1088/1748-0221/12/10/P10003, arXiv:1706.04965.

[158] M. Aaboud *et al.*, *Jet reconstruction and performance using particle flow with the ATLAS Detector*, Eur. Phys. J. C **77**, 466 (2017), doi:10.1140/epjc/s10052-017-5031-2, arXiv:1703.10485.

[159] S. Abdullin, P. Azzi, F. Beaudette, P. Janot and A. Perrotta, *The fast simulation of the CMS detector at LHC*, J. Phys. Conf. Ser. **331**, 032049 (2011), doi:10.1088/1742-6596/331/3/032049.

[160] M. Cacciari, G. P. Salam and G. Soyez, *The anti-k_t jet clustering algorithm*, JHEP **04**, 063 (2008), doi:10.1088/1126-6708/2008/04/063, arXiv:0802.1189.

[161] S. D. Ellis and D. E. Soper, *Successive combination jet algorithm for hadron collisions*, Phys. Rev. D **48**, 3160 (1993), doi:10.1103/PhysRevD.48.3160, arXiv:hep-ph/9305266.

[162] M. Wobisch and T. Wengler, *Hadronization corrections to jet cross-sections in deep inelastic scattering*, In *Workshop on Monte Carlo Generators for HERA Physics (Plenary Starting Meeting)*, pp. 270–279 (1998), arXiv:hep-ph/9907280.

[163] C. Nwankpa, W. Ijomah, A. Gachagan and S. Marshall, *Activation functions: Comparison of trends in practice and research for deep learning*, arXiv preprint arXiv:1811.03378 (2018).

[164] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization* (2014), arXiv:1412.6980.

[165] S. Kullback and R. A. Leibler, *On information and sufficiency*, The annals of mathematical statistics **22**, 79 (1951).

[166] D. J. Rezende and S. Mohamed, *Variational inference with normalizing flows*, Proceedings of the 32nd International Conference on International Conference on Machine Learning **37**, 1530 (2015), `http://proceedings.mlr.press/v37/rezende15.html`, arXiv:1505.05770.

[167] T. Müller, B. McWilliams, F. Rousselle, M. Gross and J. Novák, *Neural importance sampling*, ACM Transactions on Graphics (ToG) **38**, 1 (2019).

[168] G. Papamakarios, T. Pavlakou and I. Murray, *Masked autoregressive flow for density estimation*, Advances in neural information processing systems **30** (2017).

[169] A. Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. Driessche, E. Lockhart, L. Cobo, F. Stimberg *et al.*, *Parallel wavenet: Fast high-fidelity speech synthesis*, In *International conference on machine learning*, pp. 3918–3926. PMLR (2018).

[170] L. Ardizzone, C. Lüth, J. Kruse, C. Rother and U. Köthe, *Guided image generation with conditional invertible neural networks* (2019), arXiv:1907.02392.

[171] C. Durkan, A. Bekasov, I. Murray and G. Papamakarios, *Cubic-spline flows*, arXiv preprint arXiv:1906.02145 (2019).

[172] D. J. Rezende, G. Papamakarios, S. Racanière, M. S. Albergo, G. Kanwar, P. E. Shanahan and K. Cranmer, *Normalizing Flows on Tori and Spheres* (2020), arXiv:2002.02428.

[173] R. T. Q. Chen, Y. Rubanova, J. Bettencourt and D. Duvenaud, *Neural ordinary differential equations*, Advances in neural information processing systems **31** (2019), arXiv:1806.07366.

[174] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever and D. Duvenaud, *FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models*, arXiv e-prints arXiv:1810.01367 (2018), arXiv:1810.01367.

[175] C. Finlay, J.-H. Jacobsen, L. Nurbekyan and A. Oberman, *How to train your neural ode: the world of jacobian and kinetic regularization*, In *International conference on machine learning*, pp. 3154–3164. PMLR (2020).

[176] J. T. Springenberg, A. Klein, S. Falkner and F. Hutter, *Bayesian optimization with robust bayesian neural networks*, In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon and R. Garnett, eds., *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc. (2016).

[177] D. M. Blei, A. Kucukelbir and J. D. McAuliffe, *Variational inference: A review for statisticians*, Journal of the American statistical Association **112**, 859 (2017).

[178] M. Sharma, S. Farquhar, E. Nalisnick and T. Rainforth, *Do bayesian neural networks need to be fully stochastic?*, In *International Conference on Artificial Intelligence and Statistics*, pp. 7694–7722. PMLR (2023).

[179] M. Bellagente, M. Haussmann, M. Luchmann and T. Plehn, *Understanding Event-Generation Networks via Uncertainties*, SciPost Phys. **13**, 003 (2022), doi:10.21468/SciPostPhys.13.1.003, arXiv:2104.04543.

[180] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan and M. Shah, *Transformers in vision: A survey*, ACM computing surveys (CSUR) **54**, 1 (2022).

[181] T. Lin, Y. Wang, X. Liu and X. Qiu, *A survey of transformers*, AI Open (2022).

[182] H. Qu, C. Li and S. Qian, *Particle Transformer for Jet Tagging* (2022), arXiv:2202.03772.

[183] B. M. Dillon, G. Kasieczka, H. Olischlager, T. Plehn, P. Sorrenson and L. Vogel, *Symmetries, safety, and self-supervision*, SciPost Phys. **12**, 188 (2022), doi:10.21468/SciPostPhys.12.6.188, arXiv:2108.04253.

[184] T. Finke, M. Krämer, A. Mück and J. Tönshoff, *Learning the language of QCD jets with transformers*, JHEP **06**, 184 (2023), doi:10.1007/JHEP06(2023)184, arXiv:2303.07364.

[185] R. Fakoor, P. Chaudhari, J. Mueller and A. J. Smola, *Trade: Transformers for density estimation*, arXiv preprint arXiv:2004.02441 (2020).

[186] S. Bieringer, A. Butter, S. Diefenbacher, E. Eren, F. Gaede, D. Hundhausen, G. Kasieczka, B. Nachman, T. Plehn and M. Trabs, *Calomplification — the power of generative calorimeter models*, JINST **17**, P09028 (2022), doi:10.1088/1748-0221/17/09/P09028, arXiv:2202.07352.

[187] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman and T. Plehn, *GANplifying event samples*, SciPost Phys. **10**, 139 (2021), doi:10.21468/SciPostPhys.10.6.139, arXiv:2008.06545.

[188] I. Chahrour and J. D. Wells, *Comparing machine learning and interpolation methods for loop-level calculations*, SciPost Phys. **12**, 187 (2022), doi:10.21468/SciPostPhys.12.6.187, arXiv:2111.14788.

[189] S. Bollweg, M. Haußmann, G. Kasieczka, M. Luchmann, T. Plehn and J. Thompson, *Deep-Learning Jets with Uncertainties and More*, SciPost Phys. **8**, 006 (2020), doi:10.21468/SciPostPhys.8.1.006, arXiv:1904.10004.

[190] G. Kasieczka, M. Luchmann, F. Otterpohl and T. Plehn, *Per-Object Systematics using Deep-Learned Calibration*, SciPost Phys. **9**, 089 (2020), doi:10.21468/SciPostPhys.9.6.089, arXiv:2003.11099.

[191] R. Winterhalder, M. Bellagente and B. Nachman, *Latent Space Refinement for Deep Generative Models* (2021), arXiv:2106.00792.

[192] B. Nachman and R. Winterhalder, *Elsa: enhanced latent spaces for improved collider simulations*, Eur. Phys. J. C **83**, 843 (2023), doi:10.1140/epjc/s10052-023-11989-8, arXiv:2305.07696.

[193] S. Catani, F. Krauss, R. Kuhn and B. R. Webber, *QCD matrix elements + parton showers*, JHEP **11**, 063 (2001), doi:10.1088/1126-6708/2001/11/063, arXiv:hep-ph/0109231.

[194] M. Cacciari, G. P. Salam and G. Soyez, *FastJet User Manual*, Eur. Phys. J. C **72**, 1896 (2012), doi:10.1140/epjc/s10052-012-1896-2, arXiv:1111.6097.

[195] M. Cacciari, G. P. Salam and G. Soyez, *The anti-k_t jet clustering algorithm*, JHEP **04**, 063 (2008), doi:10.1088/1126-6708/2008/04/063, arXiv:0802.1189.

[196] A. Paszke *et al.*, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc. (2019), arXiv:1912.01703.

[197] L. N. Smith and N. Topin, *Super-convergence: Very fast training of neural networks using large learning rates*, In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006, p. 1100612. International Society for Optics and Photonics (2019).

[198] W. H. Press and G. R. Farrar, *Recursive stratified sampling for multidimensional monte carlo integration*, Computers in Physics **4**, 190 (1990), doi:10.1063/1.4822899.

[199] A. Buckley, J. Ferrando, S. Lloyd, K. Nordström, B. Page, M. Rüfenacht, M. Schönherr and G. Watt, *LHAPDF6: parton density access in the LHC precision era*, Eur. Phys. J. C **75**, 132 (2015), doi:10.1140/epjc/s10052-015-3318-8, arXiv:1412.7420.

[200] K. Cranmer, J. Brehmer and G. Louppe, *The frontier of simulation-based inference*, Proc. Nat. Acad. Sci. **117**, 30055 (2020), doi:10.1073/pnas.1912789117, arXiv:1911.01429.

[201] D0 Collaboration, *Measurement of the Top Quark Mass in the Dilepton Channel*, Phys. Rev. D **60**, 052001 (1999), doi:10.1103/PhysRevD.60.052001, arXiv:hep-ex/9808029.

[202] D0 Collaboration, *A precision measurement of the mass of the top quark*, Nature **429**, 638 (2004), doi:10.1038/nature02589, arXiv:hep-ex/0406031.

[203] CDF Collaboration, *Top quark mass measurement from dilepton events at CDF II with the matrix-element method*, Phys. Rev. D **74**, 032009 (2006), doi:10.1103/PhysRevD.74.032009, arXiv:hep-ex/0605118.

[204] F. Fiedler, A. Grohsjean, P. Haefner and P. Schieferdecker, *The Matrix Element Method and its Application in Measurements of the Top Quark Mass*, Nucl. Instrum. Meth. A **624**, 203 (2010), doi:10.1016/j.nima.2010.09.024, arXiv:1003.1316.

[205] A. Giammanco and R. Schwienhorst, *Single top-quark production at the Tevatron and the LHC*, Rev. Mod. Phys. **90**, 035001 (2018), doi:10.1103/RevModPhys.90.035001, arXiv:1710.10699.

[206] P. Artoisenet, V. Lemaitre, F. Maltoni and O. Mattelaer, *Automation of the matrix element reweighting method*, JHEP **12**, 068 (2010), doi:10.1007/JHEP12(2010)068, arXiv:1007.3300.

[207] J. Alwall, A. Freitas and O. Mattelaer, *The Matrix Element Method and QCD Radiation*, Phys. Rev. D **83**, 074010 (2011), doi:10.1103/PhysRevD.83.074010, arXiv:1010.2263.

[208] J. R. Andersen, C. Englert and M. Spannowsky, *Extracting precise Higgs couplings by using the matrix element method*, Phys. Rev. **D87**, 015019 (2013), doi:10.1103/PhysRevD.87.015019, arXiv:1211.3011.

[209] P. Artoisenet, P. de Aquino, F. Maltoni and O. Mattelaer, *Unravelling $t\bar{t}h$ via the Matrix Element Method*, Phys. Rev. Lett. **111**, 091802 (2013), doi:10.1103/PhysRevLett.111.091802, arXiv:1304.6414.

[210] C. Englert, O. Mattelaer and M. Spannowsky, *Measuring the Higgs-bottom coupling in weak boson fusion*, Phys. Lett. B **756**, 103 (2016), doi:10.1016/j.physletb.2016.02.074, arXiv:1512.03429.

[211] D. E. Ferreira de Lima, O. Mattelaer and M. Spannowsky, *Searching for processes with invisible particles using a matrix element-based method*, Phys. Lett. **B787**, 100 (2018), doi:10.1016/j.physletb.2018.10.044, arXiv:1712.03266.

[212] S. Brochet, C. Delaere, B. François, V. Lemaître, A. Mertens, A. Saggio, M. Vidal Marono and S. Wertz, *MoMEMta, a modular toolkit for the Matrix Element Method at the LHC*, Eur. Phys. J. C **79**, 126 (2019), doi:10.1140/epjc/s10052-019-6635-5, arXiv:1805.08555.

[213] CMS Collaboration, *Search for a Standard Model Higgs Boson Produced in Association with a Top-Quark Pair and Decaying to Bottom Quarks Using a Matrix Element Method*, Eur. Phys. J. C **75**, 251 (2015), doi:10.1140/epjc/s10052-015-3454-1, arXiv:1502.02485.

[214] ATLAS Collaboration, *Evidence for single top-quark production in the s-channel in proton-proton collisions at $\sqrt{s}$ =8 TeV with the ATLAS detector using the Matrix Element Method*, Phys. Lett. B **756**, 228 (2016), doi:10.1016/j.physletb.2016.03.017, arXiv:1511.05980.

[215] CMS Collaboration, *Measurement of Spin Correlations in $t\bar{t}$ Production using the Matrix Element Method in the Muon+Jets Final State in pp Collisions at $\sqrt{s}$ = 8 TeV*, Phys. Lett. B **758**, 321 (2016), doi:10.1016/j.physletb.2016.05.005, arXiv:1511.06170.

[216] A. V. Gritsan, R. Röntsch, M. Schulze and M. Xiao, *Constraining anomalous Higgs boson couplings to the heavy flavor fermions using matrix element techniques*, Phys. Rev. **D94**, 055023 (2016), doi:10.1103/PhysRevD.94.055023, arXiv:1606.03107.

[217] F. Bury and C. Delaere, *Matrix element regression with deep neural networks — Breaking the CPU barrier*, JHEP **04**, 020 (2021), doi:10.1007/JHEP04(2021)020, arXiv:2008.10949.

[218] M. R. Buckley and D. Goncalves, *Boosting the Direct CP Measurement of the Higgs-Top Coupling*, Phys. Rev. Lett. **116**, 091801 (2016), doi:10.1103/PhysRevLett.116.091801, arXiv:1507.07926.

[219] J. Ren, L. Wu and J. M. Yang, *Unveiling CP property of top-Higgs coupling with graph neural networks at the LHC*, Phys. Lett. B **802**, 135198 (2020), doi:10.1016/j.physletb.2020.135198, arXiv:1901.05627.

[220] B. Bortolato, J. F. Kamenik, N. Košnik and A. Smolkovič, *Optimized probes of CP -odd effects in the $t\bar{t}h$ process at hadron colliders*, Nucl. Phys. B **964**, 115328 (2021), doi:10.1016/j.nuclphysb.2021.115328, arXiv:2006.13110.

[221] H. Bahl, P. Bechtle, S. Heinemeyer, J. Katzy, T. Klingl, K. Peters, M. Saimpert, T. Stefaniak and G. Weiglein, *Indirect $\mathcal{CP}$ probes of the Higgs-top-quark interaction: current LHC constraints and future opportunities*, JHEP **11**, 127 (2020), doi:10.1007/JHEP11(2020)127, arXiv:2007.08542.

[222] T. Martini, R.-Q. Pan, M. Schulze and M. Xiao, *Probing the CP structure of the top quark Yukawa coupling: Loop sensitivity versus on-shell sensitivity*, Phys. Rev. D **104**, 055045 (2021), doi:10.1103/PhysRevD.104.055045, arXiv:2104.04277.

[223] D. Gonçalves, J. H. Kim, K. Kong and Y. Wu, *Direct Higgs-top CP-phase measurement with $t\bar{t}h$ at the 14 TeV LHC and 100 TeV FCC*, JHEP **01**, 158 (2022), doi:10.1007/JHEP01(2022)158, arXiv:2108.01083.

[224] R. K. Barman, D. Gonçalves and F. Kling, *Machine learning the Higgs boson-top quark CP phase*, Phys. Rev. D **105**, 035023 (2022), doi:10.1103/PhysRevD.105.035023, arXiv:2110.07635.

[225] H. Bahl and S. Brass, *Constraining $\mathcal{CP}$-violation in the Higgs-top-quark interaction using machine-learning-based inference*, JHEP **03**, 017 (2022), doi:10.1007/JHEP03(2022)017, arXiv:2110.10177.

[226] M. Kraus, T. Martini, S. Peitzsch and P. Uwer, *Exploring BSM Higgs couplings in single top-quark production* (2019), arXiv:1908.09100.

[227] P. Artoisenet *et al.*, *A framework for Higgs characterisation*, JHEP **11**, 043 (2013), doi:10.1007/JHEP11(2013)043, arXiv:1306.6464.

[228] L. Del Debbio, S. Forte, J. I. Latorre, A. Piccione and J. Rojo, *Unbiased determination of the proton structure function F(2)**p with faithful uncertainty estimation*, JHEP **03**, 080 (2005), doi:10.1088/1126-6708/2005/03/080, arXiv:hep-ph/0501067.

[229] G. Cowan, *Statistical data analysis*, Clarendon Press, ISBN 978-0-19-850156-5 (1998).

[230] 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), doi:10.1103/PhysRevD.98.052004, arXiv:1805.00020.

[231] V. Mikuni and F. Canelli, *ABCNet: An attention-based method for particle tagging*, Eur. Phys. J. Plus **135**, 463 (2020), doi:10.1140/epjp/s13360-020-00497-3, arXiv:2001.05311.

[232] T. Salimans and J. Ho, *Progressive distillation for fast sampling of diffusion models* (2022), arXiv:2202.00512.

[233] Y. Song, P. Dhariwal, M. Chen and I. Sutskever, *Consistency models* (2023), arXiv:2303.01469.

[234] E. Buhmann, F. Gaede, G. Kasieczka, A. Korol, W. Korcari, K. Krüger and P. McKeown, *CaloClouds II: Ultra-Fast Geometry-Independent Highly-Granular Calorimeter Simulation* (2023), arXiv:2309.05704.

[235] T. Martini and P. Uwer, *Extending the Matrix Element Method beyond the Born approximation: Calculating event weights at next-to-leading order accuracy*, JHEP **09**, 083 (2015), doi:10.1007/JHEP09(2015)083, arXiv:1506.08798.

[236] R. Baumeister and S. Weinzierl, *Matrix element method at next-to-leading order for arbitrary jet algorithms*, Phys. Rev. D **95**, 036019 (2017), doi:10.1103/PhysRevD.95.036019, arXiv:1612.07252.

[237] T. Martini and P. Uwer, *The Matrix Element Method at next-to-leading order QCD for hadronic collisions: Single top-quark production at the LHC as an example application*, JHEP **05**, 141 (2018), doi:10.1007/JHEP05(2018)141, arXiv:1712.04527.

[238] M. Kraus, T. Martini and P. Uwer, *Matrix Element Method at NLO for (anti-)k_T-jet algorithms*, Phys. Rev. D **100**, 076010 (2019), doi:10.1103/PhysRevD.100.076010, arXiv:1901.08008.

[239] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman, T. Plehn, D. Shih and R. Winterhalder, *Ephemeral Learning - Augmenting Triggers with Online-Trained Normalizing Flows*, SciPost Phys. **13**, 087 (2022), doi:10.21468/SciPostPhys.13.4.087, arXiv:2202.09375.

[240] H. Reyes-Gonzalez and R. Torre, *The NFLikelihood: an unsupervised DNNLikelihood from Normalizing Flows* (2023), arXiv:2309.09743.

[241] E. Bothmann, T. Childers, W. Giele, S. Höche, J. Isaacson and M. Knobbe, *A Portable Parton-Level Event Generator for the High-Luminosity LHC* (2023), arXiv:2311.06198.

[242] S. Hageboeck, T. Childers, W. Hopkins, O. Mattelaer, N. Nichols, S. Roiser, J. Teig, A. Valassi, C. Vuosalo and Z. Wettersten, *Madgraph5_aMC@NLO on GPUs and vector CPUs Experience with the first alpha release*, In *26th International Conference on Computing in High Energy & Nuclear Physics* (2023), arXiv:2312.02898.

[243] D. Maître and H. Truong, *One-loop matrix element emulation with factorisation awareness* (2023), arXiv:2302.04005.