Generative Machine Learning for Simulation-based Inference in High Energy Physics

Dissertation

Nathan Hütsch

Dissertation

submitted to the

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

for the degree of

Doctor of Natural Sciences

Put forward by

Nathan Hütsch

born in Frankfurt am Main, Germany

Oral examination: 04.07.2025

Generative Machine Learning for Simulation-based Inference in High Energy Physics

Referees: Dr. Anja Butter
Prof. Dr. Ullrich Köthe

Abstract

With the upcoming High-Luminosity LHC the volume of collider data will increase dramatically, leading to a new era of precision measurements. However, this also creates computational and methodological challenges. Established simulation and inference pipelines require significant upgrades to prevent them from becoming bottlenecks. This thesis investigates how generative machine learning can address these challenges. First, we investigate modern generative architectures, diffusion models and autoregressive transformers, for fast and accurate LHC event generation. We find that they can learn complex phase space distributions to percent-level precision, demonstrating their potential as surrogate simulators. Second, we advance the use of machine learning for the matrix element method, showing how generative networks can be used to encode the transfer probability and keep the phase space integration tractable. Finally, we explore high-dimensional, unbinned unfolding using generative models. We benchmark the performance of a range of methods on the same datasets and contribute several methodological advancements, including a transformer-enhanced diffusion model that achieves state-of-the-art precision.

Zusammenfassung

Mit dem bevorstehenden High-Luminosity LHC (HL-LHC) wird das Volumen der Kollisionsdaten drastisch zunehmen, was eine neue Ära von Präzisionsmessungen einläutet. Dies bringt jedoch auch erhebliche ressourcentechnische und methodische Herausforderungen mit sich. Die etablierten Simulations- und Inferenzstrategien müssen grundlegend weiterentwickelt werden, um nicht zum Engpass zu werden. In dieser Arbeit wird untersucht, wie generative Methoden des maschinellen Lernens zur Bewältigung dieser Herausforderungen beitragen können. Zunächst evaluieren wir moderne generative Architekturen, Diffusionsmodelle und autoregressive Transformer, im Hinblick auf eine schnelle und präzise Ereignissimulation am LHC. Wir zeigen, dass diese Modelle komplexe Phasenraumverteilungen mit einer Genauigkeit auf Prozentniveau erlernen können und damit als Ersatz-Simulatoren großes Potenzial besitzen. Im zweiten Teil erweitern wir den Einsatz von maschinellem Lernen für die Matrixelementmethode, indem wir zeigen, wie generative Netzwerke sowohl die Transferwahrscheinlichkeit modellieren als auch die Integration über den Phasenraum effizient gestalten können. Zuletzt widmen wir uns dem ungebinnten, hoch-dimensionalen Unfolding mittels generativer Modelle. Wir vergleichen die Leistung verschiedener Methoden auf denselben Datensätzen und leisten mehrere methodische Beiträge, darunter ein Diffusionsmodell mit Transformer-Architektur, das state-of-the-art Präzision erreicht.

Contents

Preface

The research presented in this thesis was conducted at the Institute for Theoretical Physics at Heidelberg University from September 2022 to April 2025. The contents of Chapters 4 to 7 are based on work in collaboration with other researchers and have been previously published as

[1] Anja Butter, Nathan Huetsch, Sofia Palacios Schweitzer, Tilman Plehn, Peter Sorrenson, Jonas Spinner
Jet Diffusion versus JetGPT – Modern Networks for the LHC
SciPost Phys. Core 8, 026 (2025) 078, arXiv:2305.10475 [hep-ph]

[2] Theo Heimel, Nathan Huetsch, Ramon Winterhalder, Tilman Plehn, Anja Butter
Precision-Machine Learning for the Matrix Element Method
SciPost Phys. 17, 129 (2024), arXiv:2310.07752 [hep-ph]

[3] N. Huetsch et al
The landscape of unfolding with machine learning
SciPost Phys. 18, 070 (2025), arXiv:2404.18807 [hep-ph]

[4] Anja Butter, Sascha Diefenbacher, Nathan Huetsch, Vinicius Mikuni, Benjamin Nachman, Sofia Palacios Schweitzer, Tilman Plehn
Generative Unfolding with Distribution Mapping
Submitted to SciPost Phys., arXiv:2411.02495 [hep-ph]

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

[5] Theo Heimel, Nathan Huetsch, Fabio Maltoni, Olivier Mattelaer, Tilman Plehn, Ramon Winterhalder
The MadNIS Reloaded
SciPost Phys. 17, 023 (2024), arXiv:2311.01548 [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 (SM) of particle physics [6–12] is one of the most successful scientific theories ever developed. It provides a comprehensive mathematical framework that describes all fundamental particles and unifies the description of three out of the four known fundamental interactions. Its predictions have been confirmed with extraordinary precision by countless experiments over more than five decades, through data collected at over a dozen particle colliders. However, despite its remarkable success, the Standard Model leaves some fundamental questions unanswered. The nature of dark matter [13,14] is yet to be understood, the same is true for the origin of neutrino masses [15,16], and the matter-antimatter asymmetry of the universe [17,18]. Finally, a full unified theory would also have to account for gravity.

The Large Hadron Collider (LHC), the largest machine ever built by humanity, was constructed to explore these open questions, providing unprecedented energy scales. Its first and foremost task, finding experimental evidence of the Higgs boson [19], was fulfilled in 2012 [20,21]. This was a tremendous achievement, it confirmed the electroweak symmetry breaking mechanism and thereby completed the Standard Model. However, by now this was more than a decade ago, and despite countless searches across a vast parameter and model space, no clear signs of new physics beyond the Higgs boson have been found.

With the upcoming High-Luminosity LHC (HL-LHC), particle physics is entering a new era characterized by an enormous increase in data volume. This big-data regime will enable an unprecedented level of precision measurements and significantly improve the sensitivity to potential new physics signals. However, established methods and tools are increasingly inadequate for handling such large-scale, complex datasets. The cost of the necessary simulations alone is projected to grow beyond all available computational resources, indicating that significant research efforts are required to prevent this from becoming a major bottleneck [22].
The traditional statistical tools used in high energy physics are often designed to test specific theory hypotheses or search for known signatures, such as the Higgs resonance. The future of LHC physics however lies in precision searches for subtle deviations from Standard Model predictions, if possible without strong prior assumptions about how these deviations look. Searching for unknown signals, possibly hidden in high-dimensional correlations, requires new inference techniques capable of extracting all available information from the data [23].

While the most visible examples of the deep learning revolution of the last decade are image and language generation, it has by now shown great potential to transform the sciences as well [24]. In particular generative machine learning models promise not just minor upgrades in existing workflows, but fundamentally new approaches to statistical inference [25, 26]. These models can learn complex, high-dimensional densities to percent-level precision, which enables scientists to forego simplifications and approximations that were necessary in the past. In particular sciences that rely on simulations are a natural fit for these new tools, since they bring both intractable likelihoods that need to be learned, and access to large and clean datasets [24]. For LHC physics, the prime example of a simulation-based science, this transformation comes just at the right time [27–29].

A major motivation for the use of generative machine learning in particle physics is preventing the cost of the simulation chain [30–36] from becoming a bottleneck in the high-luminosity era. Generative neural networks, such as normalizing flows [37, 38] and GANs [39], have successfully been used to learn complex phase space densities [40–49], demonstrating their capability as surrogate simulators. However, these models have not quite reached the required level of precision yet, in particular in modeling complex phase space correlations. In Chapter 4 of this thesis, we systematically study three modern generative architectures, two diffusion models [50, 51] and an autoregressive transformer [52]. We introduce the underlying mathematical frameworks in detail and, for each of the three models, develop Bayesian neural network versions to quantify uncertainties. Using toy models we carefully examine the learning patterns of the networks and finally we benchmark their performance on a dataset of Z+jets events. This work was the first use of Conditional Flow Matching (CFM) in a particle physics application, it has since become the leading generative model in the field.

Beyond simulation, generative machine learning enables new approaches to inference [24, 27]. Instead of using off-the-shelf techniques from the machine learning literature, these new methods should be developed with the specific challenges of LHC physics in mind. One example is the matrix element method (MEM) [53, 54], a classical analysis technique that leverages the analytical knowledge of the hard-scattering differential cross section. The intractability of the forward transfer density, together with the computational cost of the parton-level phase space integration have severely restricted the use of the MEM in the past. In chapter 5 we present a new ML-based framework for the MEM, that overcomes these limitations. Building on prior work [55, 56], we extend the setup to three networks, one density estimator for the transfer function, a second density estimator that serves as a neural importance sampler to keep the integration tractable, and a classifier that corrects for efficiency effects. For the challenging transfer function, we demonstrate how new generative networks building on CFMs and transformers can sort out the challenging jet combinatorics and learn the density to percent-level precision. Using a measurement of the CP-violating phase of the top Yukawa coupling as a benchmark, we demonstrate that the method can correctly infer the theory parameter with as little as 400 measured events.

Unfolding refers to the task of statistically inverting parts of the LHC simulation chain. If done well, it allows to propagate measured distributions up the chain, to be compared to theoretical predictions at an earlier stage [57–60]. Traditional unfolding algorithms however are restricted to low-dimensional, binned observables, severely limiting their usability. Generative models enable full-dimensional, unbinned unfolding [61–66], fundamentally transforming how differential cross sections can be measured and analyzed. In Chapters 6 and 7, we present original research that builds upon prior work on generative unfolding. We make progress on both the conceptual and ML-technical sides, taking a significant step towards applicability in real experiments.
In Chapter 6, we extensively benchmark a wide array of modern ML-based unfolding methods, for the first time systematically evaluating their performance on the same

datasets. We further propose a new transformer-based CFM model and demonstrate its superior performance. This setup, for the first time, allows to unfold the high-dimensional, multi-resonant $t\bar{t}$-phase space to percent-level precision.
Chapter 7 presents research on how distribution mapping methods can be used for generative unfolding [61, 67–69]. Established generative unfolding networks learn a mapping between a Gaussian latent space and the target unfolded phase space, conditioned on the reconstruction-level measurement. Distribution mapping networks however learn a direct mapping from the reconstruction-level distribution to the unfolded distribution. While conceptually appealing, this approach suffers from the fact that these mappings are only meaningful at distribution level, but do not preserve the event-wise conditional distributions. We explore the mathematical formulation of distribution mapping in detail and derive a modification that forces the network to preserve the correct single-event posterior distributions. Finally, using a newly simulated Z+2 jets dataset that mimics a recent ATLAS analysis [29], we showcase that this modification also increases the precision, bringing distribution mapping to a similar level of performance as the state-of-the-art CFM-based unfolding.

This thesis is structured as follows. Chapter 2 introduces the fundamentals of LHC physics relevant to this research. Chapter 3 provides an introduction to machine learning, focusing on the aspects and methods most relevant for particle physics. Chapters 4 discusses research on how diffusion models and autoregressive transformers can be used as surrogate simulators for the LHC simulation chain. Chapter 5 presents research on generative machine learning methods for the matrix element method. Chapters 6 and 7 focus on generative machine learning techniques for unbinned, full-dimensional unfolding, including thorough benchmarking and methodological advancements. Finally, Chapter 8 provides a summary and outlook.

Chapter 2

LHC physics

The Standard Model (SM) of particle physics [6–12] is a unified description of the known elementary particles and their interactions via the strong, electromagnetic, and weak forces. Mathematically it is formulated as a quantum field theory with the gauge symmetry group

$$SU(3)_C \times SU(2)_L \times U(1)_Y \; .$$

Here $SU(3)_C$ governs the strong interaction (Quantum Chromodynamics) and $SU(2)_L \times U(1)_Y$ describes the unified electroweak interaction. In the electroweak sector, the symmetry is spontaneously broken by the Higgs mechanism [19], which gives masses to the $W^\pm$ and Z bosons.

The SM defines two classes of particles. The first are fermions, spin-$\frac{1}{2}$ particles that constitute matter. Fermions are divided into quarks and leptons. Quarks carry both color and electroweak charge and interact via the strong, weak and electromagnetic forces. Leptons are color-neutral and can be sub-divided into charged leptons which interact via the weak and electromagnetic forces, and neutrinos, which are electrically neutral and interact only via the weak force. All fermions come in three generations with increasing masses.
The second class of particles in the Standard Model are bosons, integer spin particles that mediate interactions. The spin-1 gauge bosons are the force carriers, eight gluons for the strong interaction, the photon for the electromagnetic interaction, and the $W^\pm$ and Z bosons for the weak interaction. Finally the spin-0 Higgs boson is responsible for electroweak symmetry breaking and provides masses to the W and Z bosons as well as to fermions through the Higgs mechanism. The building blocks of the Standard Model are shown in Fig. 2.1.
A compact representation of the Standard Model is given by its Lagrangian,

$$\mathcal{L}_{\text{SM}} = -\frac{1}{4} G^a_{\mu\nu} G^{a\,\mu\nu} - \frac{1}{4} W^i_{\mu\nu} W^{i\,\mu\nu} - \frac{1}{4} B_{\mu\nu} B^{\mu\nu} + \bar{\psi}\, i\gamma^\mu D_\mu \psi + |D_\mu \phi|^2 - V(\phi) - y\,\bar{\psi}\,\phi\,\psi \; ,$$

where $G^a_{\mu\nu}$, $W^i_{\mu\nu}$, and $B_{\mu\nu}$ are the field strength tensors corresponding to the strong, weak, and hypercharge gauge fields, respectively. The fermion fields are denoted by ψ, and ϕ is the Higgs doublet. The covariant derivative D_μ encodes the gauge interactions. The Higgs potential $V(\phi)$ induces electroweak symmetry breaking, giving mass to the weak gauge bosons. The Yukawa term $y\,\bar{\psi}\,\phi\,\psi$ describes interactions between the Higgs field and the fermions and is responsible for generating their masses after symmetry breaking.

The Standard Model is an immensely successful theory of fundamental interactions. Developed half a century ago, it has been rigorously tested for decades at numerous

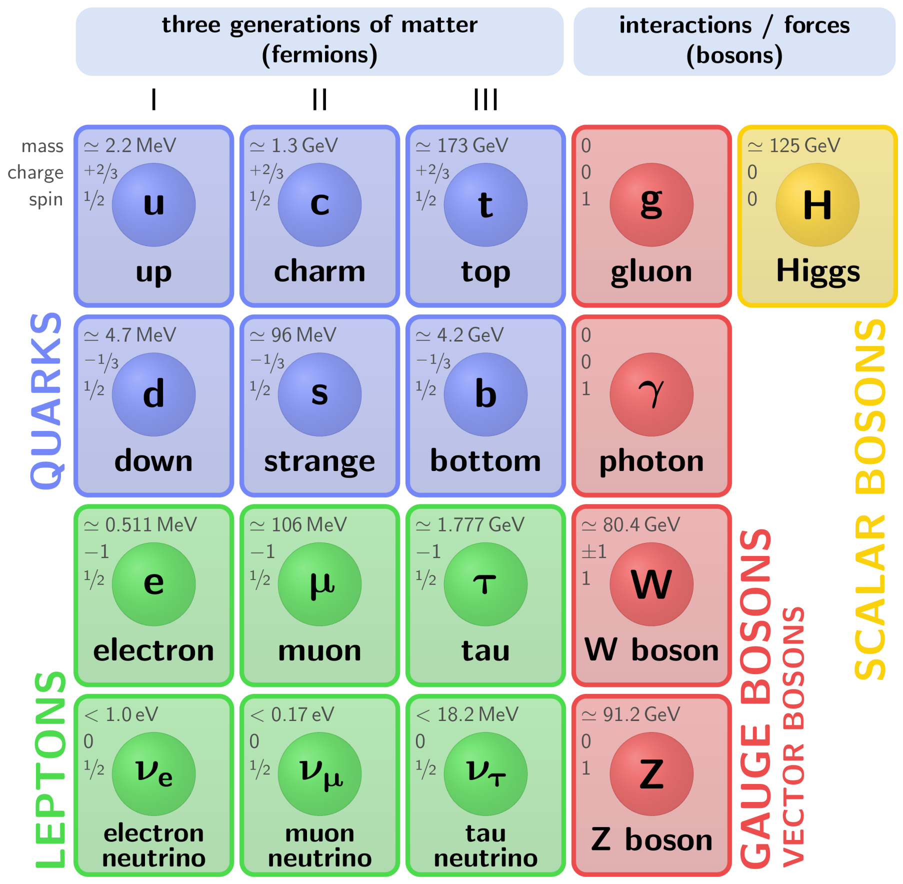

three generations of matter (fermions)
interactions / forces (bosons)
I
II
III
mass
charge
spin
≃ 2.2 MeV, +2/3, 1/2, u, up
≃ 1.3 GeV, +2/3, 1/2, c, charm
≃ 173 GeV, +2/3, 1/2, t, top
0, 0, 1, g, gluon
≃ 125 GeV, 0, 0, H, Higgs
≃ 4.7 MeV, −1/3, 1/2, d, down
≃ 96 MeV, −1/3, 1/2, s, strange
≃ 4.2 GeV, −1/3, 1/2, b, bottom
0, 0, 1, γ, photon
≃ 0.511 MeV, −1, 1/2, e, electron
≃ 106 MeV, −1, 1/2, μ, muon
≃ 1.777 GeV, −1, 1/2, τ, tau
≃ 80.4 GeV, ±1, 1, W, W boson
< 1.0 eV, 0, 1/2, νe, electron neutrino
< 0.17 eV, 0, 1/2, νμ, muon neutrino
< 18.2 MeV, 0, 1/2, ντ, tau neutrino
≃ 91.2 GeV, 0, 1, Z, Z boson
QUARKS
LEPTONS
GAUGE BOSONS
VECTOR BOSONS
SCALAR BOSONS

Figure 2.1: The Standard Model of particle physics

collider experiments—from early machines like SLAC and LEP to modern hadron colliders such as the Tevatron and the Large Hadron Collider (LHC). Its predictions have remained accurate across many orders of magnitude in energy. This success culminated in the discovery of the Higgs boson in 2012 at the LHC [20, 21], completing the particle content of the Standard Model and confirming the mechanism of electroweak symmetry breaking.

Nevertheless the Standard Model has well-known shortcomings. It successfully unifies three out of the four known fundamental interactions, but does not include gravity. It offers no viable candidate for dark matter, and it does not account for the small but non-zero neutrino masses inferred from oscillation experiments. Furthermore, it does not explain the observed matter–antimatter asymmetry in the universe.
These shortcomings motivate the search for physics beyond the Standard Model. While direct searches for new particles have historically been the dominant approach in particle colliders experiments, increasing emphasis is now placed on precision measurements of known processes. In this context, the Standard Model serves as a robust and indispensable foundation. It defines the baseline against which deviations are measured, and provides the theoretical framework for the simulation and interpretation of LHC data, as discussed in the following sections.
In the rest of this chapter, we briefly review the key ingredients of LHC physics. We begin with a brief overview of collider physics in Sec. 2.1, followed by a description of the LHC simulation chain in Sec. 2.2. These sections are based on Refs. [70–72]. Finally, we discuss traditional approaches to inference in LHC physics in Sec. 2.3.

[0]Figure taken from `https://tikz.net/sm_particles/`.

2.1 Collider physics

Collider experiments, such as those conducted at the Large Hadron Collider (LHC) in Geneva, probe our understanding of fundamental interactions by colliding particles at up to 13.6 TeV center-of-mass energy. The primary objective is to find deviations from the Standard Model that may indicate new physics. This requires a principled framework to translate the abstract Standard Model Lagrangian into measurable predictions. The central measurable quantities in collider physics are cross sections, a measure for the likelihood of specific particle interactions occurring at a given energy.

2.1.1 Cross sections

The cross section σ quantifies the probability that a specific scattering process will occur, normalized by flux. It can be expressed differentially in terms of the final-state particle momenta:

$$\mathrm{d}\sigma = \frac{1}{F}|\mathcal{M}|^2 \,\mathrm{d}\Phi_n \,, \tag{2.1}$$

where $|\mathcal{M}|^2$ is the squared scattering amplitude summed over initial and final spins and colors, F represents the incoming particle flux factor, and $\mathrm{d}\Phi_n$ is the differential n-particle phase-space element given by:

$$\mathrm{d}\Phi_n = (2\pi)^4 \delta^{(4)}\left(p_{\text{in}} - \sum_{i=1}^{n} p_i\right) \prod_{i=1}^{n} \frac{\mathrm{d}^3 p_i}{(2\pi)^3 2E_i} \,. \tag{2.2}$$

Here, p_i and E_i are the three-momenta and energies of the final-state particles, and p_{in} is the total initial four-momentum. Integrating over the full final-state phase space yields the total cross section:

$$\sigma = \int \frac{1}{F}|\mathcal{M}|^2 \,\mathrm{d}\Phi_n \,. \tag{2.3}$$

The connection between the theory Lagrangian and the measurable cross sections is in the squared matrix element $|\mathcal{M}|^2$, which encodes the transition probability from a given initial state into a given final state under the theory. It is calculated using perturbation theory, which expands scattering amplitudes in powers of the coupling constants defined by the Lagrangian. Interactions described by the Standard Model Lagrangian can be translated into scattering amplitudes via Feynman diagrams, which graphically represent particle interactions at each order in perturbation theory. Squaring and summing these amplitudes over spins and colors yields the matrix element $|\mathcal{M}|^2$.

Theoretical predictions of measurable cross sections typically start from calculations of partonic cross sections. Partonic cross sections describe fundamental scattering processes involving quarks and gluons, computed from perturbative QCD. They explicitly depend on the partonic center-of-mass energy $\hat{s}$ and momentum transfer Q^2. The simplest partonic cross section for a $2 \to 2$ scattering process (partons a, b into final-state partons c, d) at leading-order perturbation theory is:

$$\hat{\sigma}_{ab\to cd}(\hat{s}) = \frac{1}{16\pi\hat{s}^2} \int |\mathcal{M}_{ab\to cd}|^2 \,\mathrm{d}t \,, \tag{2.4}$$

where t is a Mandelstam variable defined as $t = (p_a - p_c)^2$, and the integration is performed over the allowed range of t.

Next-to-leading-order (NLO) and higher-order corrections involve additional loop diagrams and real emission processes. Including such corrections enhances the accuracy of predictions at the cost of significantly increasing computational complexity and introducing additional complications such as infrared and collinear divergences. These are handled using techniques like dimensional regularization and subtraction schemes.

The LHC is a proton-proton collider. The composite nature of the colliding protons complicates the calculation of measurable cross sections one step further. Unlike electron-positron colliders, where initial states are well-defined elementary particles, protons are bound states composed of quarks and gluons. The precise energy and identity of the interacting partons within the proton are not known. This is addressed via the introduction of parton distribution functions (PDFs), which are the subject of the next section.

2.1.2 Parton distribution functions

Parton distribution functions $f_i(x, Q^2)$ quantify the probability of finding a particular type i of parton (quark or gluon) inside the proton, carrying a specific fraction x of the proton's total momentum at a given energy resolution scale Q^2. They are normalized as

$$\sum_i \int_0^1 \mathrm{d}x \, x \, f_i(x, Q^2) = 1 \, . \tag{2.5}$$

The factorization theorem of QCD allows to separate short-distance physics, the perturbatively calculable partonic cross sections, from the non-perturbative, long-distance proton structure encoded in PDFs. Mathematically, this is expressed as:

$$\sigma_{pp\to X}(s, Q^2) = \sum_{i,j} \int_0^1 \mathrm{d}x_1 \mathrm{d}x_2 \, f_i(x_1, Q^2) \, f_j(x_2, Q^2) \, \hat{\sigma}_{ij\to X}(\hat{s}, Q^2) \, , \tag{2.6}$$

where $f_i(x, Q^2)$ denotes the PDF for parton i, and $\hat{\sigma}_{ij\to X}$ represents the partonic cross section calculable using perturbative QCD.

The evolution of the PDFs with the energy scale Q^2 can be calculated from first-principle QCD:

$$Q^2 \frac{\partial}{\partial Q^2} f_i(x, Q^2) = \frac{\alpha_s(Q^2)}{2\pi} \sum_j \int_x^1 \frac{\mathrm{d}y}{y} P_{ij}\left(\frac{x}{y}, \alpha_s(Q^2)\right) f_j(y, Q^2) \, , \tag{2.7}$$

this is called the Dokshitzer–Gribov–Lipatov–Altarelli–Parisi (DGLAP) equations [73–75]. Here $\alpha_s(Q^2)$ is the scale-dependent strong coupling and P_{ij} are universal splitting kernels [73], which describe how partons radiate other partons. While the scaling of the PDFs with the energy scale can be computed, the PDFs themselves cannot due to their non-perturbative nature. Instead, PDFs are extracted from global fits to experimental data, including measurements across a wide range of energies and kinematic regimes. The procedure involves parametrizing the PDFs at a chosen input scale and evolving them via the DGLAP equations, adjusting the parameters to best reproduce the observed cross sections. Precise fits of PDFs with faithful uncertainty quantification are an essential ingredient for precision measurements at the LHC. The NNPDF collaboration has successfully used neural networks for this task since 2009 [76].

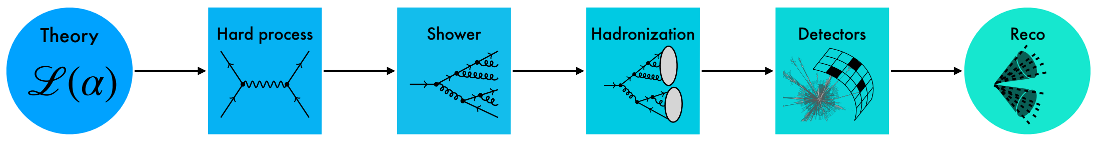


Figure 2.2: The LHC simulation chain. Figure adapted from Ref. [26]

2.2 The LHC simulation chain

In the previous sections, we introduced cross sections as the central theoretical quantities for collider physics. These can be calculated from the Standard Model Lagrangian using perturbation theory and parton distribution functions, yielding predictions for partonic scattering processes. However, parton-level cross sections are not directly accessible in experiments. Most of the particles produced in the hard scattering decay almost immediately, their lifetime is many orders of magnitudes below what would be necessary to reach the detector. This is especially true for the particles that are of greatest theoretical interest, such as the Higgs boson or the top quark. What is actually observed at the LHC are tracks in the tracking detectors and energy depositions in the calorimeters, resulting from the stable or long-lived decay and shower products of these short-lived particles.

This motivates the need for detailed simulations of the full collision process. The observable final state is the result of a complex sequence of physical processes, from the initial partonic interaction over showering and hadronization to the final interaction of stable particles with the detector. Each of these steps involves intrinsic quantum or statistical uncertainty — from the probabilistic branching of parton showers to the non-deterministic response of the detector. Even if every stage could be described analytically, which is not the case, the sheer number of possible configurations and stochastic interactions renders such a calculation intractable. High-precision simulations are therefore essential: they generate synthetic collider events that encode both the underlying physics and the practical limitations of the measurement process. These simulations form the foundation for interpreting experimental data, enabling both precision tests of the Standard Model and searches for new physics.

Simulating LHC events involves a chain of steps known as the LHC simulation chain, which can be roughly divided into four distinct steps as sketched in Fig. 2.2. It begins with the generation of the hard scattering process from a theory Lagrangian and proceeds through parton showering, hadronization, and detector simulation. Finally detector readouts are interpreted in the form of reconstructed particle kinematics. While simulations follow the forward chain from theory to observable events, data analysis conceptually follows the inverse direction, aiming to extract information about the underlying particle-level processes from measured data.

In the remainder of this section, we briefly review each step of the simulation chain. We begin with the generation of the hard-scattering event generation, then describe the modeling of QCD radiation through parton showers, the non-perturbative process of hadronization, and finally the simulation of the detector response.

2.2.1 Hard-scattering event generation

The hard-scattering process is where the short-distance partonic interactions take place. The underlying theory enters here directly in the form of the matrix elements, as discussed in Sec. 2.1. Event generators utilize perturbative calculations from the theory Lagrangian at a given order. Calculating cross sections then requires integrating the matrix elements over the high-dimensional phase space. Aside from a few textbook examples solving these integrals analytically is infeasible and Monte Carlo methods are employed to evaluate the integrals numerically .

A first, naive implementation of Monte Carlo integration approximates the phase space integral defined in Eq. (2.3) by uniformly sampling N points x_i from the phase space:

$$\hat{\sigma} = \frac{V}{N} \sum_{i=1}^{N} \frac{1}{F} |\mathcal{M}(x_i)|^2 \,, \tag{2.8}$$

where V denotes the total phase-space volume. While this estimator is unbiased, its statistical variance is given by

$$\mathrm{Var}[\hat{\sigma}] = \frac{V^2}{N} \cdot \frac{1}{F^2} \mathrm{Var}(|\mathcal{M}|^2) \,. \tag{2.9}$$

This variance is typically very large because the integrand $|\mathcal{M}|^2$ often features sharply peaked structures localized in small regions of the phase space. As a result, most uniformly sampled points contribute negligibly to the integral, leading to prohibitively slow convergence. To alleviate this issue, phase space generators employ importance sampling [77]. Instead of sampling points uniformly from phase space, one samples according to a proposal distribution $q(x)$. The integral estimator then takes the form

$$\hat{\sigma} = \frac{1}{N} \sum_{i=1}^{N} \frac{|\mathcal{M}(x_i)|^2}{F\, q(x_i)} \,, \tag{2.10}$$

where the points x_i are drawn from $q(x)$. Under mild conditions importance sampling leaves the expectation value of the estimator unchanged, but the variance becomes

$$\mathrm{Var}[\hat{\sigma}] = \frac{1}{N} \cdot \frac{1}{F^2} \mathrm{Var}\left(\frac{|\mathcal{M}|^2}{q} \right) \,. \tag{2.11}$$

Choosing a proposal distribution $q(x)$ that closely follows the shape of the integrand $|\mathcal{M}(x)|^2$ minimizes the variance and ensures convergence for a reasonable number of samples. Finding such a distribution is a hard problem in itself, the most common algorithm is `VEGAS` [78–81]. Recent work encoding the proposal distribution in an invertible neural networks shows very promising results [5, 82, 83].

A further advantage of the Monte Carlo approach is that it naturally includes event generation. Each sampled point in phase space represents a possible collision event with a weight given by

$$w_i = \frac{|\mathcal{M}(x_i)|^2}{q(x_i)} \,. \tag{2.12}$$

Since nature produces unweighted measurements, experimental analyses typically desire unweighted events, each representing equally probable outcomes of the collision. The most straight forward way of performing such an unweighting is by rejection sampling.

Each event is accepted with probability

$$p_{\mathrm{acc}}(x_i) = \frac{w_i}{w_{\mathrm{max}}} \; , \tag{2.13}$$

where w_{max} is the largest weight in the event set. Modern event generators typically employ slightly more complicated unweighting algorithms to avoid very low unweighting efficiencies due to single outlier events with large weights. After unweighting, the surviving set of unweighted events follows the probability distribution derived from the underlying scattering amplitude.

The most commonly used general-purpose event generators at the LHC are `Sherpa` [31] and `MadGraph5_aMC@NLO` [32]. Given user-specified initial and final states these generators automate the entire process; they generate the relevant Feynman diagrams, compute the matrix elements, perform the Monte Carlo integration, and generate weighted or unweighted parton-level events. In experimental analyses, it is common practice to simulate the same process using multiple event generators. This allows for cross-checks of theoretical assumptions, estimation of systematic uncertainties, and validation of implementation details.

The output of this stage represents purely partonic configurations. The generated events are passed on to the next stage of the simulation chain, the QCD showering. Most physics analyses at the LHC are targeting the hard scattering stage, for example by measuring some theory parameter α in the Lagrangian or searching for some new heavy resonance. From this perspective everything that happens after this stage is a nuisance that complicates measurements.

2.2.2 Parton showering

While hard-scattering event generation provides accurate predictions for a limited number of final state particles, scaling to higher multiplicities quickly becomes impractical due to the factorially growing number of diagrams. In addition, fixed-order perturbation theory breaks down in the soft and collinear regimes.

To include these emissions, parton shower simulators rely on factorization properties of QCD amplitudes. In the limit of soft and collinear emissions, $(n+1)$-parton cross sections factorize into n-parton cross sections and universal splitting kernels, $P_{a\to bc}(z)$. These quantify the probability of a parton a with momentum q splitting into two partons b and c, with momentum fractions z and $1-z$:

$$\mathrm{d}\sigma_{n+1} \approx \mathrm{d}\sigma_n \frac{\alpha_s}{2\pi} \frac{\mathrm{d}q^2}{q^2} \, \mathrm{d}z \, P_{a\to bc}(z) \; . \tag{2.14}$$

Parton showers are modeled as Markov processes by applying these splitting kernels iteratively, resulting in a probabilistic cascade of emissions. Two sources of parton showers have to be differentiated here. Final-state radiation emitted by outgoing partons after the hard interaction is well described by equation Eq. (2.14). For initial-state radiation (ISR) emitted by the incoming partons before the hard interaction we have to also account for the parton distribution functions discussed in Sec. 2.1, modifying the equation to

$$\mathrm{d}\sigma_{n+1} \approx \mathrm{d}\sigma_n \frac{\alpha_s}{2\pi} \frac{\mathrm{d}q^2}{q^2} \, \frac{f_b(x/z, q^2)}{f_a(x, q^2)} \, \mathrm{d}z \, P_{a\to bc}(z) \; . \tag{2.15}$$

The most commonly used software packages to simulate parton showers at the LHC are `Pythia` [33], `Herwig` [35], and `Sherpa` [31]. They generate emissions starting from the scale defined by the hard scattering and iteratively evolve downwards until reaching a non-perturbative cutoff scale, typically around 1 GeV. Once the energy scale of the emissions drops below this cutoff, a regime is reached where the strong coupling becomes too large for perturbative techniques to be applicable. The next step of the chain is the hadronization, the conversion of partons into color-neutral hadrons.

2.2.3 Hadronization

After the parton shower, events consist of a set of quarks and gluons. However, due to color confinement [84], only color-neutral hadrons are observed in the detector. The process that converts colored partons into hadrons is called hadronization.

Unlike hard scattering or parton showering, there is no first-principles method to compute hadronization from the QCD Lagrangian. It occurs in the non-perturbative regime of QCD, where the strong coupling becomes large. Lattice QCD methods can describe this regime in principle [85], but for now there exist no practical formulations to simulate the real-time dynamics of hadronization needed for collider event generation. Consequently, phenomenological models of hadronization are employed, with parameters tuned to experimental data.

Most widely used among these phenomenological approaches is the lund string fragmentation model [86, 87], implemented in `Pythia` [33]. The color field between separating color-anticolor partons is treated as a one-dimensional string whose energy grows linearly with the distance between partons. Once the energy becomes large enough, a string break occurs and a new quark-antiquark pair is formed from the vacuum. This process is repeated recursively, resulting in a sequence of string breaks that terminates once the remaining string energy is no longer sufficient to produce another hadron. An alternative approach is the cluster model, used by `Herwig` [35].

The most commonly used simulators for hadronization are `Pythia` [33] (string fragmentation model), `Herwig` [35] (cluster model) and `Sherpa` [31] (both models). Since hadronization is based on phenomenological models not derived from first principles, these models contain a number of free parameters that must be fitted to data. In the case of the string fragmentation model these parameters control the shape of the fragmentation function, the distribution of quark flavors, and the transverse momentum distribution of hadrons. A complication is that the string breaks that are directly controlled by the parameters are not measurable, only the final-state hadrons are experimentally observed. Different parameter sets can produce similar final-states, making the tuning inherently ambiguous. The resulting tunes are validated against observables not included in the fit, and the spread between different tunes has to be propagated to downstream analyses as a systematic uncertainty.

2.2.4 Detector effects

The output of the hadronization process, in both nature and simulation, are color-neutral and stable or long-lived particles that reach the detector. The final step that has to be modeled is the interaction of these particles with the detector material. What is actually measured in a collider experiment is not the particles directly, but indirect signals such as hits in tracking detectors and energy deposits in calorimeters [88, 89]. From these the particle types and kinematics have to be reconstructed. Detector simulations model this transformation from particle-level truth to measured signals. Their output can then be directly compared to measurements.

Detector models have to account for a number of factors to simulate realistic measurements. The finite resolution of the detector leads to a smearing of measured quantities such as energy and momentum. This resolution is not universal, it depends on both the particle type and kinematics. Further, the detector will generally not record all particles. Inefficiencies arise from several sources, such as incomplete angular and momentum coverage due to the detector geometry, or low energy particles falling below detection thresholds. An additional complication present in all measurements is pile-up, the overlapping of multiple collisions that occur within the same bunch crossing. Finally, additional systematic uncertainties arise from technical imperfections, for example malfunctioning detector elements, temperature fluctuations or electronic noise.

The gold standard software for detector simulations is `GEANT4` [34]. It performs a full simulation of all interactions of particles with detector components, utilizing a fine-grained model of the detector geometry and material. While `GEANT4` simulations are incredibly detailed and highly precise, they come at significant computational cost due to the number of modeled interactions. The full detector simulation can account for up to 90 percent of the computational budget of an analysis. Most phenomenological studies therefore utilize faster alternatives, which apply parametric smearing and efficiency functions to generator-level particles. The most commonly used public tool for this is `Delphes` [36]. While `Delphes` skips over most of the details of the detector simulation, it provides a reasonably precise approximation to `GEANT4` at a fraction of the cost. Developing surrogate simulators based on generative machine learning is a promising and active area of research [90–106].

2.3 Inference in LHC physics

In the last section we discussed the chain of simulation tools that enables us to propagate theoretical predictions from a theory Lagrangian all the way to detector signals. The inference task now consists in drawing conclusions about the underlying fundamental theory by comparing these predictions to measured data. The indirect nature of the measurements and the complexity of the simulation chain make this a challenging problem. In this section, we briefly discuss traditional approaches to inference at the LHC and their limitations. Based on these limitations we will motivate the development of new machine learning based inference techniques, which form the central subject of this thesis.

2.3.1 Reconstruction

In any LHC analysis the first step is the reconstruction of interpretable physical objects from raw detector readouts. This requires the combination of signals recorded in the various subsystems of the detector, charged particle tracks in the tracking detectors, energy deposits in the electromagnetic and hadronic calorimeters, and hits in the muon chambers. This information is processed by dedicated reconstruction algorithms that infer the types and kinematics of particles such as photons, electrons, muons, and hadrons. In ATLAS and CMS reconstruction is typically performed with particle flow algorithms [107,108], which reconstruct individual stable particles by optimally combining information from all detector components. The output is a set of particles characterized by four-momenta and additional identification information. These reconstructed objects are the basis for the rest of the analysis.

From the previous discussion on parton showering and hadronization we know that quarks and gluons are not directly observable in the detector. Instead of a naked parton what we see in the detector is a jet, a collimated spray of leptons and hadrons. In LHC physics these jets are among the most studied objects. They are reconstructed from the set of measured particles using dedicated jet clustering algorithms that work by grouping together particles based on their proximity in phase space. These algorithms typically define a pairwise distance measure and then proceed in an iterative way, successively grouping the two closest particles until hitting a stopping criterion. This is an inherently ill-posed problem that requires some design choices. The most commonly used algorithm is the anti-k_T algorithm [109] which produces conical jet shapes favored by experimental analyses. It utilizes the distance measure

$$d_{ij} = \min(p_{T,i}^{-2}, p_{T,j}^{-2}) \frac{\Delta R_{ij}^2}{R^2}, \tag{2.16}$$

where ΔR_{ij} is the angular distance in the rapidity–azimuth plane and R is the maximum jet radius. The clustering continues until all particles are assigned to jets. The jet four-momentum is then defined by summing the four-momenta of all its constituents. Jet clustering is a physically motivated compression from low-level particles to high-level objects that introduces a loss of information. For many physics questions the jet four-momentum together with a small set of jet substructure variables carries sufficient information.

Once reconstruction is completed a set of event selection cuts is applied. This serves to suppress backgrounds and focus the analysis on regions of phase space that are enriched in signal events. Typical selection cuts involve requirements on the number, type, and kinematics of reconstructed objects. As an example consider a search for a Higgs boson decaying to two photons. Requiring events to have exactly two isolated high-energy photons with an invariant mass near the Higgs mass cuts away a significant portion of the background. More complex analyses also reconstruct intermediate decay chains, a common example here is a semi-leptonically decaying $t\bar{t}$-pair ($pp \to t\bar{t} \to W^+b\,W^-\bar{b} \to \ell\nu\,jj\,b\bar{b}$).

2.3.2 Observables

The reconstruction and selection steps discussed in the previous section reduce the raw detector signals to a small set of reconstructed particles and jets characterized by their four-momenta and identification information. They form the starting point for defining physical observables, scalar quantities designed to highlight specific features of the signal process or enhance discrimination against background. Common choices are invariant masses, angular correlations and transverse momenta, more specialized choices are missing transverse energy or jet substructure variables.

In a typical LHC analysis, the final inference is performed with only a few observables, often just a single one. The measured distribution in the selected observable is binned and the resulting histogram is compared to theoretical predictions obtained from Monte Carlo simulations for various parameter values. Assuming Poisson statistics, the likelihood takes the form

$$\mathcal{L}(\alpha) = \prod_{i=1}^{N_{\text{bins}}} \frac{\mu_i(\alpha)^{n_i} e^{-\mu_i(\alpha)}}{n_i!} \,, \tag{2.17}$$

where n_i is the bin count in bin i and $\mu_i(\alpha)$ the expected count under theory parameter α. Statistical inference is performed on the basis of this binned Poisson likelihood, for example by maximizing it with respect to the theory parameter.

The expected bin counts $\mu_i(\alpha)$ are not known analytically and have to be estimated from a finite number of Monte Carlo simulations which introduces an inherent statistical uncertainty. Further, they are affected by the various sources of systematic uncertainty in the simulation chain that we discussed in the previous section, from parton density function uncertainty over hadronization tune to detector calibration. These uncertainties are typically modeled with nuisance parameters ν, and the likelihood is extended to $\mathcal{L}(\alpha, \nu)$. Inference then includes an additional step of profiling or marginalizing over ν.

Going from continuous, high-dimensional data to a one-dimensional histogram of a selected observable is a significant compression of the available information. Historically, this reduction has been motivated by both practical and physical considerations. On the practical side high-dimensional likelihood-based inference was computationally prohibitive and approximations had to be made. Further the amount of available data and simulations was often not sufficient to populate high-dimensional densities, especially for processes with a low cross section. On the physics side the reduction to 1D projections allowed for well-controlled, interpretable analyses. Up until very recently physicists always had a very clear idea what kind of signals they were looking for, which allowed for the design of sensitive observables. The most prominent and most recent example is the hunt for the Higgs boson where invariant mass of its decay products served as an excellent summary statistic.

However, in recent years both of these considerations have changed. The rapid progress in deep learning is transforming not only language and image processing, but also the way data is analyzed in the sciences [24]. New methods together with an increase in available computation budget enable more involved, higher dimensional analysis strategies. At the same time the physics program at the LHC has shifted from a targeted search for a well-known signal, the Higgs boson, to a more open-ended search for subtle deviations from standard model predictions. Finding small inconsistencies between theory and measurement requires more sensitive inference methods that extract all the available information from the data.

2.3.3 Unfolding

In the last section we reviewed how physics is learned from collider data by comparing measured and simulated distributions at the reconstruction level. This successful approach is behind most analyses at the LHC. An alternative approach is unfolding [57–60], the statistical inversion of (parts of) the simulation chain. Instead of propagating theoretical predictions all the way to the reconstruction level, unfolding propagates measured distributions up the simulation chain to be compared to theoretical predictions at an earlier stage.

Unfolding has several conceptual advantages over reconstruction level analyses. First, it decouples the probed physics from detector effects allowing the combination of data sets collected at different detectors. It further allows experiments to publish the data in at a stage where theorists, who are neither knowledgeable nor interested in detector specifics, can work with it. This in turn also substantially reduces future computational cost as it allows testing new theory hypotheses against the unfolded data without pushing them through the very costly detector simulation. And finally, having the ability to perform statistical inference at earlier levels of the simulation chain enables the use of new inference methods. In the most extreme case of parton-level unfolding the data is unfolded all the way up to the hard-scattering level where likelihoods are analytically tractable. However there are also drawbacks. Any unfolding algorithm introduces additional statistical and systematic uncertainties, and potentially biases when the response model or prior assumptions are incorrect. For most measurements, performing inference directly at the reconstruction level leads to more precise and robust results. Ultimately unfolding trades some precision for interpretability and reusability.

Formally we can define the unfolding problem as finding the particle-/parton-level distribution $p(x_\text{part})$ that gives rise to the measured distribution $p_\text{data}(x_\text{reco})$ under the forward model $p(x_\text{reco}|x_\text{part})$ encoded in our simulator:

$$p(x_\text{reco}) = \int dx_\text{part}\; p(x_\text{part})\; p(x_\text{reco}|x_\text{part}) \;\overset{!}{=}\; p_\text{data}(x_\text{reco}) \; . \tag{2.18}$$

This is a very hard statistical problem, especially if the distributions are high-dimensional. Further, the notion of "the" unfolded distribution is misleading as this is in general an ill-posed inverse problem that can admit multiple solutions compatible with the observed data. Finally, the forward model $p(x_\text{reco}|x_\text{part})$ is only implicitly accessible via simulations, the conditional density is not tractable analytically.

Traditional unfolding algorithms make the problem tractable by simplifying it to observable histograms. In the simplest form they invert the relation

$$r_i = \sum_j R_{ij} p_j \; , \tag{2.19}$$

where r_i are the histogram entries at reconstruction level, p_j is the particle-level spectrum and R_{ij} the response matrix that encodes the forward model. In the binned case the forward model quantifies the probability for a particle event in bin j to be reconstructed in detector bin i. A naive unfolding algorithm would proceed by estimating the response matrix R_{ij} from the simulated histograms at particle and reconstruction level, inverting it and applying the inverse mapping to the measured reconstruction level spectrum to obtain the unfolded spectrum. In practice naive inversion of the response matrix leads to unstable solutions with large statistical fluctuations. To mitigate this, unfolding algorithms introduce regularization to stabilize the inversion, popular examples are TUnfold [110] and iterative Bayesian unfolding [111].

While these methods work well for one- or two-dimensional histograms, they do not scale to higher dimensionality and are sensitive to the choice of binning and regularization scheme. The reduction of high-dimensional data to low-dimensional histograms restricts the applicability of unfolding techniques in modern analyses. The possibility to test future hypotheses on unfolded data is severely limited by only having access to one observable. To truly unlock this feature, the data has to be available unbinned and full-dimensional, allowing

Recent years have seen rapid progress in the development of machine learning-based methods for unfolding [3]. These ML-based approaches promise to overcome the limitations of traditional approaches, enabling unbinned and high-dimensional unfolding. These methods can be roughly categorized in generative methods [61–66] that aim to learn the inverse conditional probability $p(x_\text{part}|x_\text{reco})$ and classifier-based methods [63, 112] that aim to learn a reweighting from a Monte Carlo prior to the unfolded distribution. While first results are very promising [29], these new methods also introduce new challenges related to stability, uncertainty estimation, and integration into existing workflows. This is an active area of ongoing research and the topic of a significant part of this thesis.

Chapter 3

Machine Learning

The previous chapter reviewed the core aspects of LHC physics, from the Standard Model over the simulation chain to traditional approaches for inference. The ever growing amount of data collected at the LHC, combined with a shift towards precision searches for very subtle deviations from the standard model, brings these traditional approaches to their limits. This motivates the adoption of modern machine learning techniques, whose rapid progress in recent years comes at just the right time for LHC physics. In this chapter, we introduce the machine learning methods that are relevant for this thesis. We begin in Sec. 3.1 with the core concepts of machine learning, followed by a review of the most common learning tasks in Sec. 3.2. Finally, Sec. 3.3 discusses a number of recently proposed machine learning methods for simulation-based inference. This chapter is loosely based on Ref. [113].

3.1 Introduction

Machine learning is the study of algorithms that learn patterns from data and make predictions without explicitly programmed rules. At a high level, the framework can be summarized as defining a parametric function f_θ and adapting its parameters θ such that the function is suited for the task at hand. The learning process typically requires four ingredients: a flexible function class, a dataset of training points, a loss function that quantifies performance, and an optimization algorithm that finds a good set of parameters θ. The fitted function f_θ should generalize well, meaning it should produce correct predictions for new data points that were not seen during training.

Optimization

Learning from data is formulated as the minimization of a loss function, which measures the quality of the predicted outputs. For example in a typical supervised setting such as regression or classification the loss quantifies the discrepancy between the predicted output $f_\theta(x)$ and the true target y. We will discuss specific choices of loss functions towards the end of this section. The goal of the optimization is to find the set of parameters θ that minimizes the average loss on the training data set

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i) \; . \tag{3.1}$$

Apart from a few canonical examples like linear regression, this minimization cannot be solved in closed form. Instead, in modern machine learning the optimization is performed using gradient descent. Starting from some random initialization θ_0, the parameters are iteratively updated based on the gradient of the loss function

$$\theta_{t+1} = \theta_t - \eta \, \nabla_\theta \mathcal{L}(\theta_t) \;, \tag{3.2}$$

where η is called the learning rate. The choice of the learning rate is an important hyperparameter in practice. Too small learning rates can lead to slow convergence and a tendency to get stuck in suboptimal local minima of the loss landscape, whereas too large learning rates can lead to unstable training and a lack of precision in finding the exact minimum. Learning rate schedulers that decay the learning rate over the course of the training are often used to circumvent these issues.

In practice, naive gradient descent is often replaced by more advanced algorithms that improve convergence and training stability. The most widely used optimization algorithm in modern deep learning is Adam [114]. For each parameter θ, Adam keeps track of moving averages of the gradient m_t and the squared gradient v_t:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{with} \quad m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_t) \tag{3.3}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad \text{with} \quad v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\nabla_\theta \mathcal{L}(\theta_t)\right]^2 \;, \tag{3.4}$$

where β_1 and β_2 are the weights in the exponential moving averages. The moving average of the gradient determines the direction of the parameter update and the running average of the squared gradient is used to scale the learning rate

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \;, \tag{3.5}$$

where ϵ is a small constant added for numerical stability. Including first- and second-order momentum in the parameter updates smooths out noisy gradients and allows the algorithm to adapt the learning rate individually for each parameter.

Evaluating the loss and its gradient on the entire training dataset at each step is computationally infeasible. Instead, modern training algorithms employ stochastic gradient descent (SGD), where the full gradient is approximated using a small random subset (batch) of the training data

$$\mathcal{L}(\theta) \approx \frac{1}{B} \sum_{i=1}^{B} \ell(f_\theta(x_i), y_i) \;. \tag{3.6}$$

Besides being cheaper to compute, batch updates also introduce stochasticity into the optimization process, which can help escape local minima and improve generalization. The batch size is another important hyperparameter whose choice is a trade-off between computational efficiency and more stable gradient estimators.

Generalization and overfitting

Once the training has converged to the global minimum, a set of parameters is found such that the function f_θ optimally explains the training data, as measured by the loss function. However, the real measure of performance of a machine learning model is how well it generalizes to new, unseen data. A model that performs well on the training set but poorly on new data is of limited use. This problem is called overfitting, it occurs when the learned function f_θ captures noise and fluctuations in the training data instead of extracting the meaningful patterns. This problem is very common in modern deep learning. Neural networks are extremely flexible functions with millions or even billions of parameters. While this is generally a good thing as it allows them to fit arbitrarily complex patterns and solve hard problems, it also makes them very susceptible to overfitting.

To monitor and minimize overfitting, the dataset is usually split into disjoint training, validation, and test sets. The training set is used for the gradient-based optimization, while the validation set is used to track performance on unseen data. After training is completed, the parameter set that minimized the validation loss during training is used. It is common practice to train many models with different hyperparameter choices on the same problem to find the best-performing setting, typically measured by the validation loss. One subtlety to be aware of is that this can lead to an indirect fitting to the validation dataset. The held-out test set is then used to measure the generalization performance of the final model.

If a model is observed to be overfitting, a range of regularization strategies exist to address this. A natural fist step is to simply reduce the size or complexity of the model f_θ. Choosing an appropriate model size relative to the complexity of the problem and the amount of available data significantly reduces the risk of encountering severe overfitting. However, simply reducing the model size indefinitely can lead to the opposite problem of underfitting, where the model is no longer expressive enough to solve the task at hand. In deep learning, this trade-off can typically only be determined through trial and error. If a satisfactory setup cannot be found based on the model size alone, more advanced regularization techniques exist, such as weight decay, Dropout layers [115] and data augmentation.

The trade-off between flexibility and generalization is central to machine learning. In LHC physics applications, the availability of very large datasets, from both real experiments and simulations, reduces the risk of overfitting to small sample sizes and motivates the use of large and expressive models. Combined with the complex and high-dimensional nature of the data, this is one of the key reasons why machine learning methods are particularly well suited for LHC physics.

Neural networks

Artificial neural networks are the class of machine learning models that is behind the recent rapid progress of the field. The core idea is to build a flexible function by concatenating many simple, differentiable operations. The most common architecture is the fully connected feedforward neural network, also called a multilayer perceptron (MLP). The function f_θ is composed of successive layers where the output of each layer constitutes the input of the next

$$f_\theta(x) = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(x) \; . \tag{3.7}$$

Each layer $f^{(l)}$ consists of two parts, a linear transformation followed by a non-linear activation function:

$$f^{(l)}(x) = \sigma(W^{(l)}x + b^{(l)}) \; . \tag{3.8}$$

The learnable parameters of the model $\theta = \{W^{(l)}, b^{(l)}\}_{l=1}^{L}$ are the weight matrices $W^{(l)}$ and the bias vectors of the linear mappings. Note the crucial importance of the non-linear activations σ without whom the network would be a composition of linear mappings which in turn would be a linear mapping. The composition of linear operations with non-linear activations makes f_θ a flexible and highly expressive function. Further, as a concatenation of differentiable functions f_θ itself is also differentiable and can be trained via gradient based optimization as described in the last section:

$$\nabla_\theta \mathcal{L}(\theta) = \frac{\partial \mathcal{L}}{\partial f_\theta} \cdot \frac{\partial f_\theta}{\partial \theta} \ . \tag{3.9}$$

It turns out that simple and efficient activation functions work very well in practice. The most common choice in modern deep learning is the rectified linear unit (ReLU). It is possibly the simplest non-linear function imaginable, defined as the identity for positive inputs and 0 for negative inputs:

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

While the function is technically not differentiable in $x = 0$, this is not problematic in practice.

The final layer of a network is often equipped with a task-specific activation function, that differs from intermediate activations. In binary classification for example, the final activation is typically chosen to be a sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}} \ . \tag{3.11}$$

This choice enforces outputs in the interval $[0, 1]$ which can be interpreted as class probabilities. Another common example are exponential functions as final activations in regression tasks with only positive targets.

Besides linear layers and activation functions, modern networks often include additional layers. In the last section we already mentioned Dropout [115] layers as a regularization technique for neural networks. These layers randomly set a subset of inputs to zero, which encourages the network to learn more robust features and prevents it from simply memorizing the training data. Normalization layers such as batch normalization [116] standardize the inputs across a batch which was empirically found to accelerate convergence and stabilize training.

Beyond fully connected networks, more specialized architectures have been developed for specific application areas [113]. Convolutional neural networks (CNNs) exploit spatial locality which makes them well-suited for image processing task [117]. Recurrent neural networks (RNNs) were designed for sequential data, they maintain an internal state that is updated with each new observation [118]. While these architectures were dominant in their respective domains for some time, in recent years the literature has shifted focus towards transformer architectures [52]. Originally developed for language modeling, transformers have since become state-of-the-art in nearly all data domains. We will explore transformers in more detail in Sec. 4.

3.2 Learning tasks

So far, we introduced the foundations of machine learning. In this section, we briefly review the three types of learning tasks that are relevant for this thesis. We begin with classification and regression, the two central supervised learning tasks. The goal is to predict a label or a continuous target variable given input data. We then discuss generative modeling and density estimation, two closely related unsupervised learning tasks. Instead of predicting some target given an input, generative learning aims at modeling the underlying data distribution.

3.2.1 Classification

In classification tasks, the goal is to predict a discrete class label given an input x. We will focus the discussion on binary classification $y \in \{0, 1\}$, which is relevant for many particle physics problems such as acceptance classification. The discussed concepts can be generalized to multi-class classification $y \in \{1, \dots, C\}$.

For the sake of this discussion we define $f_\theta : \mathbb{R}^d \to [0, 1]$ as a function that maps input features to a probability estimate. In practice this is achieved with neural networks with a sigmoid final activation function, as discussed in the last section. The canonical choice of loss function for a binary classification task is the binary cross-entropy

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log f_\theta(x_i) + (1 - y_i) \log(1 - f_\theta(x_i)) \right] \ , \tag{3.12}$$

which is equivalent to the negative log-likelihood under a Bernoulli model.

Although machine learning models are often viewed as black boxes, in this setup we can rigorously derive what the classifier is learning. To that end, consider the continuous limit of Eq. (3.12)

$$\mathcal{L}[f_\theta] = -\int \mathrm{d}x \, \mathrm{d}y \; p(x, y) \left[y \log f_\theta(x) + (1 - y) \log(1 - f_\theta(x)) \right] \ . \tag{3.13}$$

During optimization we minimize this loss functional with respect to the learned function $f_\theta(x)$. Taking the the functional derivative of $\mathcal{L}[f]$ with respect to f yields

$$\frac{\delta \mathcal{L}[f_\theta]}{\delta f_\theta(x)} = -\int \mathrm{d}y \; p(y|x) \left[\frac{y}{f_\theta(x)} - \frac{1 - y}{1 - f_\theta(x)} \right] \tag{3.14}$$

Setting the derivative to zero and solving for f_θ we find

$$f_\theta^*(x) = \int \mathrm{d}y \; y \, p(y|x) = \mathbb{E}[y|x] = p(y = 1|x) \ . \tag{3.15}$$

An optimal classifier trained with binary cross-entropy learns the posterior class probability. This justifies a probabilistic interpretation of its outputs.

Likelihood ratios

While classification itself is already an important task in particle physics, in recent years an alternative usage of classifier networks has gained a lot of attention. It is based on the observation that an ideal classifier in the sense of Eq. (3.15) can be used to approximate the likelihood ratio between the classes

$$\frac{f_\theta^*(x)}{1 - f_\theta^*(x)} = \frac{p(y=1|x)}{p(y=0|x)} = \frac{p(x|y=1)}{p(x|y=0)} \; , \tag{3.16}$$

where we have assumed equal class priors for simplicity. Likelihood ratios between distributions are a very powerful tool, so this remarkably simple result has been put to use in a long list of problems including unfolding [63, 112] and simulation-based inference [24, 28]. It will also repeatedly appear throughout this thesis.

3.2.2 Regression

In regression tasks, the goal is to predict a continuous-valued target variable y given an input x, using a flexible function $f_\theta(x)$ trained on a dataset $\{(x_i, y_i)\}_{i=1}^N$. The canonical loss function for regression is the mean squared error (MSE)

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N \left(f_\theta(x_i) - y_i \right)^2 \; . \tag{3.17}$$

Like the BCE loss in classification, the MSE also allows for a probabilistic interpretation. It is the negative log-likelihood under a homoscedastic Gaussian model

$$y \sim \mathcal{N}(f_\theta(x), \sigma^2) \; . \tag{3.18}$$

We again consider the continuous version of the loss functional

$$\mathcal{L}[f_\theta] = \int \mathrm{d}x \, \mathrm{d}y \; p(x,y) \left(f_\theta(x) - y \right)^2 \tag{3.19}$$

to understand what function is learned by minimizing the MSE loss. Taking the functional derivative with respect to f_θ yields

$$\frac{\delta \mathcal{L}[f_\theta]}{\delta f_\theta(x)} = 2 \int \mathrm{d}y \; p(y|x) (f_\theta(x) - y) \; . \tag{3.20}$$

Setting the derivative to zero we find the unique minimizer to be

$$f_\theta^*(x) = \int \mathrm{d}y \; y \, p(y|x) = \mathbb{E}[y|x] \; . \tag{3.21}$$

A regression network trained with MSE loss yields a function that approximates the conditional mean of the target variable given the input.

A widely used extension of this setup is the heteroskedastic loss. It adapts the underlying Gaussian model to

$$y \sim \mathcal{N}(f_\theta(x), \sigma_\theta(x)^2) \; . \tag{3.22}$$

This forces the network to not only learn a mean prediction, but also a pointwise uncertainty estimate on its own prediction. In practice, this is implemented by giving the network two outputs per input, one for the mean and one for the (log) variance, and including an additional loss term.

3.2.3 Generation and density estimation

While classification and regression focus on predicting a target variable given input data, generative modeling and density estimation aim to model the underlying distribution of the data itself. Typically we have access to a dataset of events $x \sim p(x)$, but no truth values for the density making this an unsupervised learning task. Depending on the application the goal is either to learn the underlying probability distribution explicitly (density estimation), or to generate new samples that follow the same distribution as the dataset (generation). These two problems are closely related and many methods, though not all, solve both at once. There are many use cases for generative learning in LHC physics, from fast surrogate simulators over likelihood-based inference methods to statistical inversion. It is at the heart of the research presented in the later chapters of this thesis.

A wide range of generative machine learning methods have been developed, all of which share the same underlying design principle: they learn a mapping between a simple latent distribution and the complex target distribution

$$z \sim p_\text{latent}(z) \quad \longleftrightarrow \quad x \sim p_\theta(x) \approx p_\text{data}(x) \; , \tag{3.23}$$

where latent distribution is typically a standard multi-dimensional Gaussian,

$$p_\text{latent}(z) = \mathcal{N}(z; 0, 1) \; . \tag{3.24}$$

The mathematical formulation of this mapping and how it is fitted to data are what differentiates the methods from one another. Two types of generative methods that have been very successful in particle physics applications will play a central role in this thesis, Invertible Neural Networks (INNs) [119–121] and diffusion models [50, 51, 122, 123]. We will give a short introduction to INNs in Sec. 3.2.3. The development of diffusion models for LHC physics is a central part of the original research presented in this thesis, they will be introduced in depths in Ch. 4.

Invertible neural networks

Invertible neural networks belong to the class of normalizing flows, generative networks that define the mapping between the latent and phase space as an invertible function

$$z \sim p_\text{latent}(z) \quad \underset{\leftarrow G_\theta^{-1}(x)}{\overset{G_\theta(z) \rightarrow}{\longleftrightarrow}} \quad x \sim p_\theta(x) \; . \tag{3.25}$$

The bijection allows to write the model density using the change of variables formula as

$$p_\theta(x) = p_\text{latent}\big(G_\theta^{-1}(x)\big) \left| \det \left(\frac{\partial G_\theta^{-1}(x)}{\partial x} \right) \right| \; . \tag{3.26}$$

With direct access to the model likelihood, the network can be trained via maximum likelihood estimation

$$\mathcal{L}_\text{INN} = -\big\langle \log p_\theta(x) \big\rangle_{x \sim p_\text{data}} \; . \tag{3.27}$$

While conceptually very simple, this approach requires a bijective map that is flexible enough to model complex transformations, while still allowing for efficient computation of the Jacobian determinant. In INNs this complex invertible map is constructed as a chain of simple invertible transformation blocks. This follows the same design idea as neural networks, building complex functions by concatenating many simple operations. The INNs used for the work presented in this thesis employ spline blocks [124].

Invertible neural networks are particularly well suited for scientific applications because they combine generative modeling and tractable likelihood estimation in a single framework. Their bijective structure allows sampling from complex distributions as well as evaluation of exact likelihoods via the change-of-variables formula. Finally, thanks to their efficient sampling and fast likelihood evaluation, invertible neural networks are well suited for time-sensitive applications such as neural importance sampling [5, 82, 83].

3.3 Simulation-based inference

In particle physics, as in many scientific disciplines, inference must be performed in a setting where the likelihood function $p(x|\theta)$ is intractable, but indirectly accessible via simulators. These simulators can be used to generate samples $x \sim p(x|\theta)$ for given parameters θ. This problem is known as likelihood-free inference or simulation-based inference (SBI). Formally speaking, the simulator encodes a joint distribution $(\theta, x) \sim p(\theta, x)$ over parameters and observations, but only forward samples are available

$$\theta \sim p(\theta) \; , \quad x \sim p(x|\theta) \; . \tag{3.28}$$

The task is then to infer the parameters for a given measurement of true data. There are three common approaches to statistical inference in this setting: estimating the posterior $p(\theta|x)$, the likelihood $p(x|\theta)$, or the likelihood ratio $\frac{p(x|\theta)}{p(x|\theta_0)}$ to some reference point in parameter space.

Traditional statistical inference methods include Markov Chain Monte Carlo (MCMC), variational inference, and Approximate Bayesian Computation (ABC). While these methods have been successfully used in science for many years, they suffer from poor scalability to high-dimensional data, low sample efficiency, and the need for handcrafted summary statistics. Moreover, these classical methods are typically non-amortized—they must perform a separate, costly inference procedure for every new observation. With upgrades on both the experimental and computational side, particle physics is currently transitioning into a big data regime where these methods break down.

Recent advances in machine learning have produced a new generation of simulation-based inference methods [24]. These SBI algorithms use simulated datasets and powerful neural networks to learn surrogate models for the likelihood, likelihood ratio, or posterior distribution. A key advantage of these techniques is amortization: once trained, the network can infer parameters for new observations with negligible additional cost.

In the remainder of this section, we will briefly review the three main approaches neural likelihood ratio estimation (NLRE), neural likelihood estimation (NLE), and neural posterior estimation (NPE). Each of these approaches approximates a different statistical quantity using a slightly different ML implementation. For a detailed review on the topic see Ref. [24] and the references therein.

Neural likelihood ratio estimation

Neural likelihood ratio estimation (NLRE) [125,126] aims to approximate the likelihood ratio

$$r(x;\theta_0,\theta_1) = \frac{p(x|\theta_1)}{p(x|\theta_0)} \; , \tag{3.29}$$

which, according to the Neyman-Pearson lemma, is the optimal test statistic to decide between the parameter settings θ_0 and θ_1. The crucial insight in NLRE is that this likelihood ratio can be extracted from a binary classifier, without having access to the individual likelihoods. As discussed in Sec. 3.2, an optimal classifier trained to distinguish between samples from $p(x|\theta_0)$ and $p(x|\theta_1)$ with binary cross-entropy learns the posterior class probability. From this, the likelihood ratio can be recovered via

$$r(x;\theta_0,\theta_1) = \frac{f(x)}{1-f(x)} \; . \tag{3.30}$$

While this setup estimates the likelihood ratio between two fixed parameter settings, many inference tasks require evaluating $r(x;\theta,\theta_0)$ for arbitrary θ. A common generalization, known as likelihood-to-evidence ratio estimation, enables this by learning a classifier over the full parameter space. It works by training a classifier $f(\theta,x)$ between the joint distribution $p(\theta,x)$ and the product of the marginal distributions $p(\theta)p(x)$. Samples from $p(\theta,x) = p(\theta)p(x|\theta)$ can be generated using the simulator, whereas samples from $p(\theta)p(x)$ can be trivially generated by random shuffling the θ entries in the simulated samples to break the correlation between θ and x. Note that the network is now a function of both the observed data and the parameter. The learned classifier can then be used to calculate the ratio

$$\frac{f^*(x,\theta)}{1-f^*(x,\theta)} = \frac{p(x,\theta)}{p(x)p(\theta)} = \frac{p(x|\theta)p(\theta)}{p(x)p(\theta)} = \frac{p(x|\theta)}{p(x)} \; . \tag{3.31}$$

Access to this likelihood-to-evidence ratio enables statistical inference in two different ways. Maximizing this ratio with respect to θ is equivalent to maximum likelihood estimation because the denominator is independent of θ. Alternatively taking the ratio of $\frac{p(x|\theta_0)}{p(x)}$ and $\frac{p(x|\theta_1)}{p(x)}$ gives access to the likelihood ratio $\frac{p(x|\theta_1)}{p(x|\theta_0)}$ for arbitrary parameter choices θ_0 and θ_1.

Neural likelihood estimation

Neural likelihood estimation (NLE) [127,128] aims to approximate the likelihood function $p(x|\theta)$ directly with a neural density estimator. This is conceptually different from likelihood ratio estimation, which learns a discriminative function between two likelihoods. Instead, NLE employs generative machine learning to model the full conditional distribution over observations given parameters. The trained model then constitutes a machine learning surrogate for the forward simulator.

Once the model is trained and real data x is measured, the surrogate likelihood enables a variety of inference strategies. For instance, maximum likelihood estimation can be performed by finding the parameter set θ_{MLE} that maximizes the likelihood of the observed data x

$$\theta_{\text{MLE}} = \arg\max_{\theta} \, p_\phi(x|\theta) \; . \tag{3.32}$$

Since neural networks are by construction differentiable, the maximum can be searched via gradient ascent. Alternatively, Bayesian inference can be performed by plugging the learned likelihood into Bayes' theorem

$$p(\theta|x) \propto p(x|\theta)\, p(\theta) \; , \tag{3.33}$$

and exploring the posterior using Markov Chain Monte Carlo.

In theory, neural likelihood estimation is strictly more powerful than neural likelihood ratio estimation since having access to likelihoods automatically also gives access to likelihood ratios. However, as a rule of thumb classifiers are generally easier to train than density estimators, so the choice between the two methods is a trade-off. Learning a reliable and precise surrogate for $p(x|\theta)$ is a very difficult task, in particular if the observation space is high-dimensional.

A related, particle-physics-specific inference method is the matrix element method [53,54]. It makes use of the fact that in LHC physics the reconstruction level likelihood $p(x_\text{reco}|\theta)$ can often be factorized as

$$p(x_\text{reco}|\theta) = \int dx_\text{hard}\; p(x_\text{hard}|\theta)\; p(x_\text{reco}|x_\text{hard}) \; , \tag{3.34}$$

where the dependence on the theory parameter enters only in the parton level hard-scattering likelihood $p(x_\text{hard}|\theta)$. Crucially, this factor can be analytically calculated from the underlying quantum field theory. The forward transfer function $p(x_\text{reco}|x_\text{hard})$ however is not known and must be approximated. Historically, the need for hand-crafted transfer functions has limited the use of the matrix element method to very specific settings. In Ch. 5 we will discuss how to overcome this limitation by encoding this transfer function in a generative neural network. Compared to standard neural likelihood estimation, this approach has the advantage that it makes use of the analytically tractable, θ-dependent part of the simulation chain. Only the intractable, θ-independent part is approximated with a neural network, significantly reducing the complexity of the learning task.

Neural posterior estimation

Neural posterior estimation (NPE) [129–132] approximates the posterior distribution $p(\theta|x)$ directly with a neural conditional density estimator. On the technical level, it tries to solve exactly the inverse task of neural likelihood estimation. It is particularly well suited for Bayesian inference settings where the posterior distribution over the parameter space is the final inference target.

To train the model, samples are drawn from the joint distribution $(\theta, x) \sim p(\theta)\, p(x|\theta)$. These samples are used to train a neural network to approximate the posterior via conditional density estimation. Whereas in NLE the network inputs are the parameters θ and the outputs the density over observations $p(x|\theta)$, in NPE the roles are reversed. Once trained, the model can be evaluated on new observations x_0 to produce an approximate posterior $p(\theta \mid x_0)$. Since the model is amortized, this requires only a forward pass through the network, making it orders of magnitude more efficient than traditional methods like MCMC.

The fact that NPE gives direct access to the posterior distribution has an important implication. According to Bayes' theorem every posterior is proportional to a prior

$$p(\theta|x) = \frac{p(\theta)\, p(x|\theta)}{p(x)} \,. \tag{3.35}$$

When using NPE, the prior $p(\theta)$ is the distribution used to generate the training samples for the network. While Bayesian inference with a well-motivated prior is the method of choice in some fields, LHC physics is generally a frequentist field that sees prior dependence as undesirable. This issue can be alleviated through the use of iterative methods, where the posterior estimated in one round is used as the prior for the next.

Chapter 4

Precision event generation with diffusion models and transformers

The research presented in this chapter is based on work in collaboration with Anja Butter, Sofia Palacios Schweitzer, Tilman Plehn, Peter Sorrenson, and Jonas Spinner, and has been previously published in Ref. [1]. The content is similar or identical to the content of this article.

The future of LHC physics lies in a systematic and comprehensive understanding of all aspects of the recorded data in terms of fundamental theory. This can be achieved through simulation-based analyses, applying and adapting modern data science methods. As obvious from the name, this method relies on a fast and precise simulation chain, starting with the hard scattering evaluated for a given Lagrangian, to fast detector simulations. Because LHC physics is driven by fundamental questions, these simulations have to be based on first principles, mere modeling would not allow us to extract relevant or interesting information from the data. Moreover, for theory predictions to not become a limiting factor to the entire LHC program, this simulation chain has to be (i) precise, (ii) fast, and (iii) flexible.

Modern machine learning (ML) has shown great potential to transform LHC analyses and simulations [25, 26]. The leading ML-tools for LHC simulations are, obviously, generative networks, which combine unsupervised density estimation over phase space with a fast sampling into the learned density. The list of tasks where (generative) neural networks can improve LHC simulations is long [25]. It starts with phase space integration [133, 134] and phase space sampling [82, 135–138] of ML-encoded fast transition amplitudes [139, 140]. More advanced tasks include event subtraction [141], event unweighting [142, 143], or super-resolution enhancement [144, 145]. Prototypical applications which allow for a systematic evaluation of the network performance are NN-event generators [40–45], NN-parton showers [146–151], or detector simulations [90–106]. Even when trained on first-principle simulations, such fast generators are easy to handle, efficient to ship, and powerful in amplifying statistical limitations of the training samples [152, 153].

Classical generative network architectures include variational autoencoders (VAEs) and generative adversarial networks (GANs). Both of them can generate high-dimensional data, assuming that the intrinsic dimensionality of the problem is much smaller than the apparent dimensionality of its representation. However, both have not been shown to fulfill the precision requirements of the LHC. Precise density estimation points to bijective generative networks, for instance normalizing flows [37, 46–49] and their invertible network

(INN) variant [119–121], which are limited to lower-dimensional sampling but sufficient at least for the hard process at the LHC.

LHC studies are consistently showing promising results for normalizing flows[1], including transformative tasks, like probabilistic unfolding [61–65], inference from high-dimensional data [154], or the matrix element method [56]. One reason why INNs have established a new level of stability, control and uncertainty estimation, is the combination with Bayesian neural network (BNN) concepts [155–161], discriminator-training and reweighting [162, 163], and conditional training on augmented data. In the spirit of explainable AI, Bayesian generative networks allow us to understand how networks learn a given phase space distribution, in the case of INNs very similar to a traditional fit [161]. They systematically improve the precision of the underlying density estimation and track the effects from statistical and systematic limitations of the training data [45]. In this study we will first compare the successful INNs with new diffusion networks [50, 51, 122, 164, 165] on the task of unconditional phase space generation. This allows us to benchmark their performance as surrogate simulators [45, 152, 153], as well as prepare their future usage for conditional tasks such as probabilistic unfolding [61–65].

An aspect of neural networks which is often overlooked is that in precision LHC simulations the intrinsic dimension of the physics problem and the apparent dimension of its phase space are similar; for this dimensionality we need to encode all correlations [166, 167]. This implies that the network size, its training effort, and its performance tend to scale poorly with the number of particles and suffer from the curse of dimensionality. This is the motivation to also include an autoregressive [168, 169] transformer [170] in our study of modern generative networks.

In this paper we will introduce two new different diffusion models for particle physics applications in Secs. 4.1.1 and 4.1.2. We then introduce a new autoregressive, eventually pre-trained, transformer architecture (JetGPT) with an improved dimensionality scaling in Sec. 4.1.3. For all three networks we develop new Bayesian versions, to control their learning patterns and the uncertainty in the density estimation step. In Sec. 4.2 we illustrate all three models for two toy models, a two-dimensional linear ramp and a Gaussian ring. Finally, in Sec. 4.3 we use all three networks to generate Z+jets events for the LHC, the same reference process as used for uncertainty-aware INNs in Ref. [45]. This standard application allows us to quantify the advantages and disadvantages of the three new architectures and compare them to the INN benchmark.

4.1 Novel generative networks

At the LHC, generative networks are used for many simulation and analysis tasks, typically to describe virtual or real particles over their correlated phase space. The number of particles ranges from few to around 50, described by their energy and three momentum directions, sometimes simplified through on-shell conditions. Typical generative models for the LHC then map simple latent distributions to a phase space distribution encoded in the training data,

$$r \sim p_\text{latent}(r) \quad \longleftrightarrow \quad x \sim p_\theta(x|\theta) \approx p_\text{data}(x) \; . \tag{4.1}$$

[1]Note that in these applications autoregressive flows do *not* outperform advanced coupling layers.

The last step represents the network training, for instance in terms of a variational approximation of $p_\text{data}(x)$. The latent distribution is typically a standard multi-dimensional Gaussian,

$$p_\text{latent}(r) = \mathcal{N}(r; 0, 1) \; . \tag{4.2}$$

We focus on the case where the dimensionalities of the latent space r and the phase space x are identical, and there is no lower-dimensional latent representation. For these kinds of dimensionalities, bijective network architectures are promising candidates to encode precision correlations. For strictly symmetric bijective networks like INNs the forward and backward directions are inverse to each other, and the network training and evaluation is symmetric. However, this strict symmetry is not necessary to generate LHC events or configurations.

The success of normalizing flows or INNs for this task motivates a study of so-called diffusion or score-based models as an alternative. We will introduce two different models in Sec. 4.1.1 and 4.1.2, one with a discrete and one with a continuous time evolution. The main question concerning such diffusion models in LHC physics is if their precision matches the INNs, how we can benefit from their superb expressivity, and if those benefits outweigh the slower evaluation.

A major challenge for all network applications in LHC physics is the required precision in all correlations, and the corresponding power-law scaling with the number of phase space dimensions. This scaling property leads us to introduce and test an autoregressive transformer in Sec. 4.1.3. Again, the question is how precise and how expressive this alternative approach is and if the benefits justify the extra effort in setup and training.

Because fundamental physics applications require full control and a conservative and reliable uncertainty estimation of neural networks, we will develop Bayesian versions of all three generative models. This allows us to control the uncertainty in the density estimation and to derive an intuition how the different networks learn the phase space distribution of the data.

4.1.1 Denoising Diffusion Probabilistic Model

Architecture

Denoising Diffusion Probabilistic Models (DDPM) [50] transform a model density by gradually adding Gaussian noise. This setup guarantees that the network links a non-trivial physics distribution to a Gaussian noise distribution, as illustrated in Eq. (4.1). The task of the reverse, generative process is to to denoise this diffused data. The structure of diffusion models considers the transformation in Eq. (4.3) a time-dependent process with $t = 0 \,...\, T$,

$$p_\theta(x_0|\theta) \quad \xleftrightarrow[\leftarrow\text{backward}]{\text{forward}\rightarrow} \quad p_\text{latent}(x_T) \; . \tag{4.3}$$

The DDPM discretizes the time series in Eq. (4.3) in the forward direction and encodes is it into a neural network for the backward direction. We start with the forward process, which turns the physical distribution into noise. The corresponding joint distribution is factorized into discrete steps,

$$\begin{aligned} p(x_1, ..., x_T|x_0) &= \prod_{t=1}^{T} p(x_t|x_{t-1}) \\ \text{with} \qquad p(x_t|x_{t-1}) &= \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t) \; . \end{aligned} \tag{4.4}$$

Each conditional step $p(x_t|x_{t-1})$ adds noise with variance β_t around the mean $\sqrt{1-\beta_t}x_{t-1}$. The combination of x_t as a variable and the mean proportional to x_{t-1} implies that the successive steps can be combined as Gaussian convolutions and give the closed form

$$\begin{aligned} p(x_t|x_0) &= \int \prod_{i=1}^{t-1} dx_i \; p(x_t|x_{t-1}) p(x_i|x_{i-1}) \\ &= \mathcal{N}(x_t; \sqrt{1-\bar{\beta}_t}x_0, \bar{\beta}_t) \qquad \text{with} \qquad 1-\bar{\beta}_t = \prod_{i=1}^{t}(1-\beta_i) \; . \end{aligned} \tag{4.5}$$

The scaling of the mean with $\sqrt{1-\beta_t}$ prevents the usual addition of the variances and instead stabilizes the evolution of the Gaussian over the time series. The variance can be adapted through a schedule, where $\bar{\beta}_t \to 1$ for $t \to T$ should be guaranteed. As suggested in Ref. [50] we choose a linear increase with $\beta_1 = 10^{-7}T$ and $\beta_T = 2 \cdot 10^{-5}T$.

As a first step towards reversing the forward diffusion, we apply Bayes' theorem on each slice defined in Eq. (4.4),

$$p(x_{t-1}|x_t) = \frac{p(x_t|x_{t-1})p(x_{t-1})}{p(x_t)} \; . \tag{4.6}$$

However, a closed-form expression for $p(x_t)$ only exists if conditioned on x_0, as given in Eq. (4.5). Using $p(x_t|x_{t-1}, x_0) = p(x_t|x_{t-1})$ we can instead compute the conditioned forward posterior as a Gaussian

$$\begin{aligned} p(x_{t-1}|x_t, x_0) &= \frac{p(x_t|x_{t-1})p(x_{t-1}|x_0)}{p(x_t|x_0)} = \mathcal{N}(x_{t-1}; \hat{\mu}_t(x_t, x_0), \hat{\beta}_t) \\ \text{with} \quad \hat{\mu}(x_t, x_0) &= \frac{\sqrt{1-\bar{\beta}_{t-1}}\beta_t}{\bar{\beta}_t}x_0 + \frac{\sqrt{1-\beta_t}\bar{\beta}_{t-1}}{\bar{\beta}_t}x_t \quad \text{and} \quad \hat{\beta}_t = \frac{\bar{\beta}_{t-1}}{\bar{\beta}_t}\beta_t \; . \end{aligned} \tag{4.7}$$

The actual reverse process starts with Gaussian noise and gradually transforms it into the phase-space distribution through the same discrete steps as Eq. (4.4), without knowing x_0 a priori. The corresponding generative network needs to approximate Eq. (4.6) for each step. We start by defining our modeled phase-space distribution

$$p_\theta(x_0|\theta) = \int dx_1...dx_T \; p(x_0, ..., x_T|\theta) \; , \tag{4.8}$$

and assume that the joint probability is again given by a chain of independent Gaussians,

$$\begin{aligned} p(x_0, ..., x_T|\theta) &= p_{\text{latent}}(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \\ \text{with} \quad p_\theta(x_{t-1}|x_t) &= \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_\theta^2(x_t, t)) \; . \end{aligned} \tag{4.9}$$

Here, μ_θ and σ_θ are learnable parameters describing the individual conditional probability slices $x_t \to x_{t-1}$. It turns out that in practice we can fix $\sigma_\theta^2(x_t, t) \to \sigma_t^2$ [50]. We will see that the advantage of the discrete diffusion model is that we can compare a Gaussian posterior, Eq. (4.7), with a reverse, learned Gaussian in Eq. (4.9) for each step.

Loss function

Ideally, we want to train our model by maximizing the posterior $p_\theta(\theta|x_0)$, however, this is not tractable. Using Bayes' theorem and dropping regularization and normalization terms this is equivalent to minimizing the corresponding negative log likelihood in Eqs. (4.8) and (4.9),

$$\begin{aligned}
&\Big\langle -\log p_\theta(x_0|\theta) \Big\rangle_{p_\text{data}} \\
&= -\int dx_0 \; p_\text{data}(x_0) \; \log \left(\int dx_1 ... dx_T \; p_\text{latent}(x_T) \prod_{t=1}^T p_\theta(x_{t-1}|x_t) \right) \\
&= -\int dx_0 \; p_\text{data}(x_0) \; \log \left(\int dx_1 ... dx_T \; p_\text{latent}(x_T) p(x_1, ..., x_T|x_0) \prod_{t=1}^T \frac{p_\theta(x_{t-1}|x_t)}{p(x_t|x_{t-1})} \right) \\
&= -\int dx_0 \; p_\text{data}(x_0) \; \log \left\langle p_\text{latent}(x_T) \prod_{t=1}^T \frac{p_\theta(x_{t-1}|x_t)}{p(x_t|x_{t-1})} \right\rangle_{p(x_1,...,x_T|x_0)} \qquad (4.10)
\end{aligned}$$

In the first step, we insert a one into our loss function by dividing Eq. (4.4) with itself. Using Jensen's inequality $f(\langle x \rangle) \leq \langle f(x) \rangle$ for convex functions we find

$$\begin{aligned}
\Big\langle -\log p_\theta(x_0|\theta) \Big\rangle_{p_\text{data}} &\leq -\int dx_0 \; p_\text{data}(x_0) \; \left\langle \log \left(p_\text{latent}(x_T) \prod_{t=1}^T \frac{p_\theta(x_{t-1}|x_t)}{p(x_t|x_{t-1})} \right) \right\rangle_{p(x_1,...,x_T)|x_0)} \\
&= -\left\langle \log \left(p_\text{latent}(x_T) \prod_{t=1}^T \frac{p_\theta(x_{t-1}|x_t)}{p(x_t|x_{t-1})} \right) \right\rangle_{p(x_0,...,x_T)} \\
&= \left\langle -\log p_\text{latent}(x_T) - \sum_{t=2}^T \log \frac{p_\theta(x_{t-1}|x_t)}{p(x_t|x_{t-1})} - \log \frac{p_\theta(x_0|x_1)}{p(x_1|x_0)} \right\rangle_{p(x_0,...,x_T)} \\
&\equiv \mathcal{L}_\text{DDPM} \; . \qquad (4.11)
\end{aligned}$$

As suggested above, we would like to compare each intermediate learned latent distribution $p_\theta(x_{t-1}|x_t)$ to the real posterior distribution $p(x_{t-1}|x_t, x_0)$ of the forward process. To reverse the ordering of the forward slice we use Bayes' theorem,

$$\begin{aligned}
\mathcal{L}_\text{DDPM} &= \left\langle -\log p_\text{latent}(x_T) - \sum_{t=2}^T \log \frac{p_\theta(x_{t-1}|x_t) p(x_{t-1}|x_0)}{p(x_{t-1}|x_t, x_0) p(x_t|x_0)} - \log \frac{p_\theta(x_0|x_1)}{p(x_1|x_0)} \right\rangle_{p(x_0,...,x_T)} \\
&= \left\langle -\log p_\text{latent}(x_T) - \sum_{t=2}^T \log \frac{p_\theta(x_{t-1}|x_t)}{p(x_{t-1}|x_t, x_0)} - \log \frac{p(x_1|x_0)}{p(x_T|x_0)} - \log \frac{p_\theta(x_0|x_1)}{p(x_1|x_0)} \right\rangle_{p(x_0,...,x_T)} \\
&= \left\langle -\log \frac{p_\text{latent}(x_T)}{p(x_T|x_0)} - \sum_{t=2}^T \log \frac{p_\theta(x_{t-1}|x_t)}{p(x_{t-1}|x_t, x_0)} - \log p_\theta(x_0|x_1) \right\rangle_{p(x_0,...,x_T)} \\
&= \sum_{t=2}^T \Big\langle \text{KL}[p(x_{t-1}|x_t, x_0), p_\theta(x_{t-1}|x_t)] \Big\rangle_{p(x_0,x_t)} + \Big\langle -\log p_\theta(x_0|x_1) \Big\rangle_{p(x_0,...,x_T)} + \text{const}
\end{aligned}$$

$$\approx \sum_{t=2}^{T} \Big\langle \text{KL}[p(x_{t-1}|x_t, x_0), p_\theta(x_{t-1}|x_t)] \Big\rangle_{p(x_0,x_t)} \tag{4.12}$$

Now, the KL-divergence compares the forward Gaussian step of Eq. (4.7) with the reverse, learned Gaussian in Eq. (4.9). The second sampled term will always be negligible compared to the first $T-1$ terms. The KL-divergence between two Gaussian, with means $\mu_\theta(x_t, t)$ and $\hat{\mu}(x_t, x_0)$ and standard deviations σ_t^2 and $\hat{\beta}_t$, has the compact form

$$\mathcal{L}_\text{DDPM} = \sum_{t=2}^{T} \Big\langle \frac{1}{2\sigma_t^2} \left|\hat{\mu} - \mu_\theta\right|^2 \Big\rangle_{p(x_0,x_t)} + \text{const.} \tag{4.13}$$

This implies that μ_θ approximates $\hat{\mu}$. The sampling follows $p(x_0, x_t) = p(x_t|x_0)\, p_\text{data}(x_0)$. We numerically evaluate this loss using the reparametrization trick on Eq. (4.5)

$$\begin{aligned} & x_t(x_0, \epsilon) = \sqrt{1-\bar{\beta}_t} x_0 + \sqrt{\bar{\beta}_t} \epsilon \qquad \text{with} \qquad \epsilon \sim \mathcal{N}(0,1) \\ \Leftrightarrow \qquad & x_0(x_t, \epsilon) = \frac{1}{\sqrt{1-\bar{\beta}_t}} \left(x_t - \sqrt{\bar{\beta}_t} \epsilon \right) . \end{aligned} \tag{4.14}$$

These expressions provide, for example, a closed form for $\hat{\mu}(x_t, x_0)$, but in terms of x_t and ϵ,

$$\hat{\mu}(x_t, \epsilon) = \frac{1}{\sqrt{1-\beta_t}} \left(x_t(x_0, \epsilon) - \frac{\beta_t}{\sqrt{\bar{\beta}_t}} \epsilon \right) . \tag{4.15}$$

For the reverse process we choose the same parametrization, but with a learned $\epsilon_\theta(x_t, t)$,

$$\mu_\theta(x_t, t) \equiv \hat{\mu}(x_t, \epsilon_\theta) = \frac{1}{\sqrt{1-\beta_t}} \left(x_t - \frac{\beta_t}{\sqrt{\bar{\beta}_t}} \epsilon_\theta(x_t, t) \right) . \tag{4.16}$$

Inserting both expressions into Eq. (4.13) gives us

$$\mathcal{L}_\text{DDPM} = \sum_{t=2}^{T} \Big\langle \frac{1}{2\sigma_t^2} \frac{\beta_t^2}{(1-\beta_t)\bar{\beta}_t} \left| \epsilon - \epsilon_\theta \left(\sqrt{1-\bar{\beta}_t} x_0 + \sqrt{\bar{\beta}_t} \epsilon, t \right) \right|^2 \Big\rangle_{x_0 \sim p_\text{data}, \epsilon \sim \mathcal{N}(0,1)} . \tag{4.17}$$

The sum over t can be evaluated numerically as a sampling. We chose our model variance $\sigma_t^2 \equiv \hat{\beta}_t$ to follow our true variance. Often, the prefactor in this form for the loss is neglected in the training, but as we need a likelihood loss for the Bayesian setup and no drop in performance was observed, we keep it.

The DDPM model belongs to the broad class of score-based models, and Eq. (4.13) can also be reformulated for the model to predict the score $s(x_t, t) = \nabla_{x_t} \log p(x_t)$ of our latent space at time t. It can be shown that $s_\theta(x_t, t) = -\epsilon_\theta(x_t, t)/\sigma_t$ [171].

Training and sampling

The training algorithm for the DDPM is illustrated in Fig. 4.1. For a given phase-space point $x_0 \sim p_\text{data}(x_0)$ drawn from our true phase space distribution we draw a time step $t \sim \mathcal{U}(1, T)$ from a uniform distribution as well as Gaussian noise $\epsilon \sim \mathcal{N}(0,1)$ at each iteration. Given Eq. (4.15) we can then calculate our diffused data after t time steps x_t,

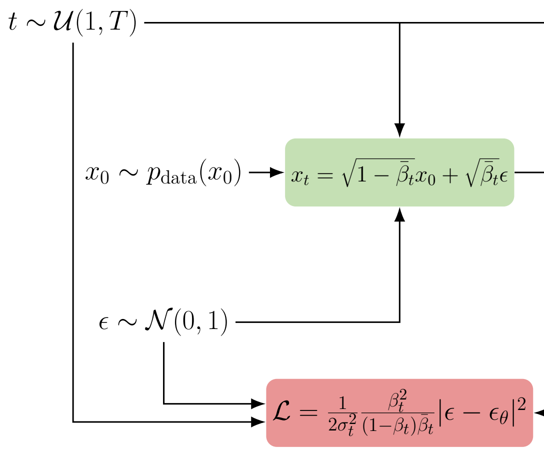

$t \sim \mathcal{U}(1,T)$
$x_0 \sim p_\text{data}(x_0)$
$x_t = \sqrt{1-\bar\beta_t}x_0 + \sqrt{\bar\beta_t}\epsilon$
DDPM
$\epsilon \sim \mathcal{N}(0,1)$
ϵ_θ
$\mathcal{L} = \frac{1}{2\sigma_t^2}\frac{\beta_t^2}{(1-\beta_t)\bar\beta_t}|\epsilon-\epsilon_\theta|^2$

Figure 4.1: DDPM training algorithm, following Ref. [50], with the loss derived in Eq. (4.17).

which is fed to the DDPM network together with our condition t. The network encodes ϵ_θ and we compare this network prediction with the true Gaussian noise ϵ multiplied by a t-dependent constant as given in the likelihood loss of Eq. (4.17). Note that we want to ensure that our network sees as many different time steps t for many different phase-space points x_0 as necessary to learn the step-wise reversed diffusion process, which is why we use a relatively simple residual dense network architecture, which is trained over many epochs.

The sampling algorithm for the DDPM is shown in Fig. 4.2. We start by feeding our network our final timestep T and $x_T \sim p_\text{latent}(x_T)$ drawn from our Gaussian latent space distribution. With the predicted ϵ_θ and drawn Gaussian noise $z_{T-1} \sim \mathcal{N}(0,1)$ we can then calculate x_{T-1}, which is a slightly less diffused version of x_T. This procedure is repeated until reaching our phase-space and computing x_0, where no additional Gaussian noise is added. Note that during sampling the model needs to predict ϵ_θ T times, making the sampling process slower than for classic generative networks like VAEs, GANs, or INNs.

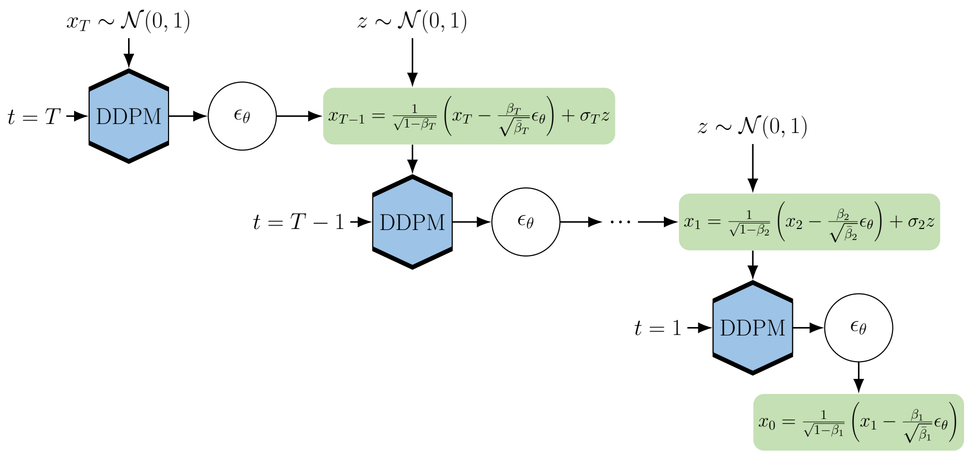

$x_T \sim \mathcal{N}(0,1)$
$z \sim \mathcal{N}(0,1)$
$t = T$
DDPM
ϵ_θ
$x_{T-1} = \frac{1}{\sqrt{1-\beta_T}}\left(x_T - \frac{\beta_T}{\sqrt{\bar\beta_T}}\epsilon_\theta\right) + \sigma_T z$
$z \sim \mathcal{N}(0,1)$
$t = T-1$
DDPM
ϵ_θ
$\cdots$
$x_1 = \frac{1}{\sqrt{1-\beta_2}}\left(x_2 - \frac{\beta_2}{\sqrt{\bar\beta_2}}\epsilon_\theta\right) + \sigma_2 z$
$t = 1$
DDPM
ϵ_θ
$x_0 = \frac{1}{\sqrt{1-\beta_1}}\left(x_1 - \frac{\beta_1}{\sqrt{\bar\beta_1}}\epsilon_\theta\right)$

Figure 4.2: DDPM sampling algorithm, following Ref. [50].

Likelihood extraction

To calculate the model likelihood we can use Eq. (4.8) or its sampled estimate,

$$p_\theta(x_0|\theta) = \Big\langle p_\theta(x_0|x_1) \Big\rangle_{p(x_1,\ldots,x_T|\theta)} \,, \tag{4.18}$$

but this is very inefficient. The problem is that $p_\theta(x_0|x_1)$ is a narrow distribution, essentially zero for almost all sampled x_1. We can improve the efficiency by importance sampling and use instead

$$\begin{aligned} p_\theta(x_0|\theta) &= \left\langle \frac{p(x_1,\ldots,x_T|\theta)}{p(x_1,\ldots,x_T|x_0)} p_\theta(x_0|x_1) \right\rangle_{p(x_1,\ldots,x_T|x_0)} \\ &= \left\langle \frac{p(x_0,\ldots,x_T|\theta)}{p(x_1,\ldots,x_T|x_0)} \right\rangle_{p(x_1,\ldots,x_T|x_0)} \,. \end{aligned} \tag{4.19}$$

This samples a diffusion process starting from x_0 and into the latent space, meaning that it represents a likely forward and backward path. This means the integrand should not just be zero most of the time.

Bayesian DDPM

The key step in the training of generative networks is the density estimation over phase space, from which the network then samples. Like any neural network task this density estimation comes with uncertainties, for instance from a limited amount of training data, a lack of model flexibility, or even training data which we know cannot be trusted. This means that the density estimation step of the generative network should assign an uncertainty to the extracted phase space density, ideally in form of a second map over the target phase space. This problem has been tackled for bijective normalizing flows through a Bayesian network extension [161], which can be combined with other measures, like conditional training on augmented data [45].

The idea behind Bayesian networks is to train network weights as distributions and evaluate the network by sampling over these distributions. This will provide a central value and an uncertainty for the numerically defined network output [155–157].[2] Because general MCMC-methods are expensive for large networks, we use variational inference [172] to learn Gaussian approximations for each weight distribution. Because of the non-linear nature of the network this does not mean that the network output has to come with a Gaussian uncertainty [160].

We repeat the main steps in deriving the Bayesian loss for any neural network approximating, for instance, a density map $\rho(x) \approx \rho_\theta(x)$ following Ref. [23]. The expectation value is defined as

$$\langle\, \rho \,\rangle(x) \equiv \langle\, \rho \,\rangle = \int d\rho \; \rho \; p(\rho) \qquad \text{with} \qquad p(\rho) = \int d\theta \; p(\rho|\theta) \; p(\theta|x_\text{train}) \,, \tag{4.20}$$

where we omit the x-dependence. We use the variational approximation to approximate

$$p(\rho) = \int d\theta \; p(\rho|\theta) \; p(\theta|x_\text{train}) \approx \int d\theta \; p(\rho|\theta) \; q(\theta) \,, \tag{4.21}$$

[2]We cannot emphasize often enough that Bayesian networks for uncertainty quantification have nothing to do with Bayesian inference.

where $q(\theta)$ is also a function of x. The variational approximation step requires us to minimize

$$\begin{aligned}\mathcal{L}_\text{BNN} = \text{KL}[q(\theta), p(\theta|x_\text{train})] &= \left\langle \log \frac{q(\theta)}{p(\theta|x_\text{train})} \right\rangle_q \\ &= \int d\theta\; q(\theta)\; \log \frac{q(\theta)}{p(\theta|x_\text{train})} \\ &= - \int d\theta\; q(\theta)\; \log p(x_\text{train}|\theta) + \text{KL}[q(\theta), p(\theta)] + \text{const}\;, \end{aligned} \tag{4.22}$$

where we use Bayes' theorem to transform the untractable $p(\theta|x_\text{train})$, introducing the prior $p(\theta)$ for the network weights. This so-called ELBO loss combines a likelihood loss with a regularization term, their relative size fixed by Bayes' theorem.

It turns out that for sufficiently deep networks we can choose $q(\theta)$ as uncorrelated Gaussians per network weight [157], such that the training parameters are a set of means and standard deviations for each network weight. Compared to the deterministic network, its Bayesian version is twice the size, but automatically regularized, keeping the additional numerical effort minimal. While $p(\theta)$, also chosen as a Gaussian, is formally defined as a prior, we emphasize that in our case the step from the prior to the posterior has nothing to do with Bayesian inference. The Gaussian width of $p(\theta)$ can be treated as a network hyperparameter and varied to improve the numerical performance. We typically find that the result is stable under varying the width by several orders of magnitude, and width one works well.

The derivation of Eq. (4.22) can be easily extended to the density estimation step of a generative networks, in the same way as for the Bayesian INN [161]. The Bayesian DDPM loss follows from the deterministic likelihood loss in Eqs. (4.11) and (4.17) by adding a sampling over $\theta \sim q(\theta)$ and the regularization term,

$$\mathcal{L}_\text{B-DDPM} = \Big\langle \mathcal{L}_\text{DDPM} \Big\rangle_{\theta\sim q} + \text{KL}[q(\theta), p(\theta)]\;. \tag{4.23}$$

Switching a deterministic network into its Bayesian version includes two steps, (i) swap the deterministic layers to the corresponding Bayesian layers, and (ii) add the regularization term to the loss. For the latter, one complication arises. We estimate the complete loss from a dataset including N events in M batches, which means the likelihood term is summed and then normalized over M batches, while the regularization term comes with the complete prefactor $1/N$.

To evaluate the Bayesian network we need to again sample over the network weight distribution. This way we guarantee that the uncertainty of the network output can have any functional form. The number of samplings for the network evaluations can be chosen according to the problem. We choose 30 for all problems discussed in this work. To compare the Bayesian network output with a deterministic network output we can either go into the limit $q(\theta) \to \delta(\theta - \theta_0)$ or only evaluate the means of the network weight distributions.

Our network is implemented in PyTorch [173] and uses Adam [114] as optimizer. All hyperparameters are given in Tab. A.1. As already mentioned we use a simple residual network which consists of multiple fully connected dense layers with SiLU activation functions. Within our setup a significant increase in performance is achieved when initializing the weights of the last layer in each block to zero.

4.1.2 Conditional Flow Matching

Architecture

As an alternative, we study Conditional Flow Matching (CFM) [51, 164, 165]. Like the DDPM, it uses a time evolution to transform phase space samples into noise, so the reverse direction can generate samples as outlined in Eq. (4.3). Instead of a discrete chain of conditional probabilities, the time evolution of samples in the CFM framework follows a continuous ordinary differential equation (ODE)

$$\frac{dx(t)}{dt} = v(x(t), t) \qquad \text{with} \qquad x(t=0) = x_0 \; , \tag{4.24}$$

where $v(x(t), t)$ is called the velocity field of the process. This velocity field can be linked to a probability density $p(x, t)$ with the continuity equation

$$\frac{\partial p(x,t)}{\partial t} + \nabla_x \left[p(x,t) v(x,t) \right] = 0 \; . \tag{4.25}$$

These two equations are equivalent in the sense that for a given probability density path $p(x, t)$ any velocity field $v(x, t)$ describing the sample-wise evolution Eq. (4.24) will be a solution of Eq. (4.25), and vice versa. Our generative model employs $p(x, t)$ to transforms a phase space distribution into a Gaussian latent distribution

$$p(x,t) \to \begin{cases} p_\text{data}(x) & t \to 0 \\ p_\text{latent}(x) = \mathcal{N}(x; 0, 1) & t \to 1 \; . \end{cases} \tag{4.26}$$

The associated velocity field will allow us to generate samples by integrating the ODE of Eq. (4.24) from $t = 1$ to $t = 0$.

As for the DDPM, we start with a diffusion direction. We define the time evolution from a phase space point x_0 to the standard Gaussian as

$$x(t|x_0) = (1-t)x_0 + t\epsilon \to \begin{cases} x_0 & t \to 0 \\ \epsilon \sim \mathcal{N}(0,1) & t \to 1 \; , \end{cases} \tag{4.27}$$

following a simple linear trajectory [165], after not finding better results with other choices. For given x_0 we can generate $x(t|x_0)$ by sampling

$$p(x, t|x_0) = \mathcal{N}(x; (1-t)x_0, t) \; . \tag{4.28}$$

This conditional time evolution is similar to the DDPM case in Eq. (4.5), and it gives us the complete probability path

$$p(x,t) = \int dx_0 \; p(x,t|x_0) \; p_\text{data}(x_0) \; . \tag{4.29}$$

It fulfills the boundary conditions in Eq. (4.26),

$$p(x,0) = \int dx_0 \; p(x,0|x_0) \; p_\text{data}(x_0) = \int dx_0 \; \delta(x - x_0) \; p_\text{data}(x_0) = p_\text{data}(x)$$

$$p(x,1) = \int dx_0 \; p(x,1|x_0) \; p_{\text{data}}(x_0) = \mathcal{N}(x;0,1) \int dx_0 \; p_{\text{data}}(x_0) = \mathcal{N}(x;0,1) \; . \qquad (4.30)$$

From this probability density path we need to extract the velocity field. We start with the conditional velocity, associated with $p(x,t|x_0)$, and combine Eq. (4.24) and (4.27) to

$$v(x(t|x_0),t|x_0) = \frac{d}{dt} \left[(1-t)x_0 + t\epsilon \right] = -x_0 + \epsilon \; . \qquad (4.31)$$

The linear trajectory leads to a time-constant velocity, which solves the continuity equation for $p(x,t|x_0)$ by construction. We exploit this fact to find the unconditional $v(x,t)$

$$\begin{aligned} \frac{\partial p(x,t)}{\partial t} &= \int dx_0 \; \frac{\partial p(x,t|x_0)}{\partial t} \; p_{\text{data}}(x_0) \\ &= -\int dx_0 \; \nabla_x \left[v(x,t|x_0) p(x,t|x_0) \right] \; p_{\text{data}}(x_0) \\ &= -\nabla_x \left[p(x,t) \int dx_0 \; \frac{v(x,t|x_0)p(x,t|x_0)p_{\text{data}}(x_0)}{p(x,t)} \right] \\ &= -\nabla_x \left[p(x,t) v(x,t) \right] \; , \end{aligned} \qquad (4.32)$$

by defining

$$v(x,t) = \int dx_0 \; \frac{v(x,t|x_0)p(x,t|x_0)p_{\text{data}}(x_0)}{p(x,t)} \; . \qquad (4.33)$$

While the conditional velocity in Eq. (4.31) describes a trajectory between a normal distributed and a phase space sample x_0 that is specified in advance, the aggregated velocity in Eq. (4.33) can evolve samples from p_{data} to p_{latent} and vice versa.
Like the DDPM model, the CFM model can be linked to score-based diffusion models, [164] derive a general relation between the velocity field and the score of a diffusion process that for our linear trajectory reduces to $s(x,t) = -\frac{1}{t}(x + (1-t)v(x,t))$.

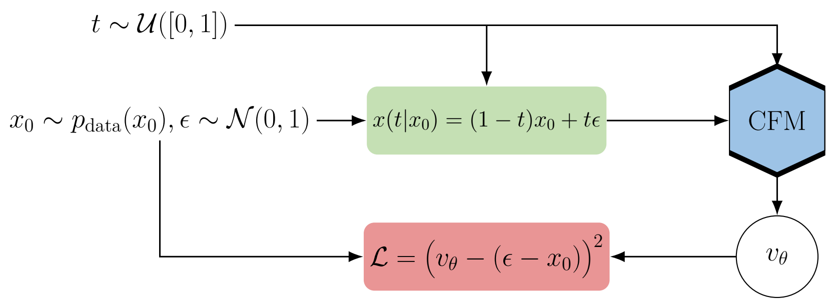
Loss function

Encoding the velocity field in Eq. (4.33) is a simple regression task, $v(x,t) \approx v_\theta(x,t)$. The straightforward choice for the loss is the mean squared error,

$$\left\langle \left[v_\theta(x,t) - v(x,t) \right]^2 \right\rangle_{t,x\sim p(x,t)} = = \left\langle v_\theta(x,t)^2 \right\rangle_{t,x\sim p(x,t)} - \left\langle 2 v_\theta(x,t) v(x,t) \right\rangle_{t,x\sim p(x,t)} + \text{const} \; , \qquad (4.34)$$

where the time is sampled uniformly over $t \in [0,1]$. While we would want to sample x from the probability path given in Eq. (4.29) and learn the velocity field given in Eq. (4.33), neither of those is tractable. However, it would be easy to sample from the conditional path in Eq. (4.28) and calculate the conditional velocity in Eq. (4.31). We rewrite the above loss in terms of the conditional quantities, so the first term becomes

$$\begin{aligned} \left\langle v_\theta(x,t)^2 \right\rangle_{t,x\sim p(x,t)} &= \left\langle \int dx v_\theta(x,t)^2 \int dx_0 \; p(x,t|x_0) p_{\text{data}}(x_0) \right\rangle_t \\ &= \left\langle v_\theta(x,t)^2 \right\rangle_{t,x_0\sim p_{\text{data}},x\sim p(x,t|x_0)} \end{aligned}$$



$t \sim \mathcal{U}([0,1])$
$x_0 \sim p_\text{data}(x_0), \epsilon \sim \mathcal{N}(0,1)$
$x(t|x_0) = (1-t)x_0 + t\epsilon$
CFM
v_θ
$\mathcal{L} = \left(v_\theta - (\epsilon - x_0)\right)^2$

Figure 4.3: CFM training algorithm, with the loss derived in Eq. (4.37).

$$= \Big\langle v_\theta(x(t|x_0),t)^2 \Big\rangle_{t,x_0\sim p_\text{data},\epsilon} \tag{4.35}$$

Using Eq. (4.33) we can rewrite the second loss term as

$$\begin{aligned}
-2\Big\langle v_\theta(x,t)v(x,t)\Big\rangle_{t,x\sim p(x,t)} &= -2\Big\langle \int dx\; p(x,t)v_\theta(x,t)\; \frac{\int dx_0 v(x,t|x_0)p(x,t|x_0)p_\text{data}(x_0)}{p(x,t)} \Big\rangle_t \\
&= -2\Big\langle \int dx dx_0\; v_\theta(x,t)\; v(x,t|x_0)\; p(x,t|x_0)\; p_\text{data}(x_0)\Big\rangle_t \\
&= -2\Big\langle v_\theta(x,t)\; v(x,t|x_0)\Big\rangle_{t,x_0\sim p_\text{data},x\sim p(x,t|x_0)} \\
&= -2\Big\langle v_\theta(x(t|x_0),t)\; v(x(t|x_0),t|x_0)\Big\rangle_{t,x_0\sim p_\text{data},\epsilon} \, .
\end{aligned} \tag{4.36}$$

The (conditional) Flow Matching loss of Eq. (4.34) then becomes

$$\begin{aligned}
\mathcal{L}_\text{CFM} &= \Big\langle [v_\theta(x(t|x_0),t) - v(x(t|x_0),t|x_0)]^2 \Big\rangle_{t,x_0\sim p_\text{data},\epsilon} \\
&= \Big\langle \left[v_\theta(x(t|x_0),t) - \frac{dx(t|x_0))}{dt}\right]^2 \Big\rangle_{t,x_0\sim p_\text{data},\epsilon} \\
&= \Big\langle [v_\theta((1-t)x_0 + t\epsilon, t) - (\epsilon - x_0)]^2 \Big\rangle_{t,x_0\sim p_\text{data},\epsilon} \, .
\end{aligned} \tag{4.37}$$

Training and Sampling

The CFM training is illustrated in Fig. 4.3. At each iteration we sample a data point $x_0 \sim p_\text{data}(x_0)$ and a normal distributed $\epsilon \sim \mathcal{N}(0,1)$ as starting and end points of a trajectory, as well as a time $t \sim \mathcal{U}([0,1])$. We then compute $x(t|x_0)$ following Eq. (4.27) and the associated conditional velocity $v(x(t|x_0),t|x_0)$ following Eq. (4.31). The point $x(t|x_0)$ and the time t are passed to a neural network which encodes the conditional velocity field $v_\theta(x,t) \approx v(x,t|x_0)$. One property of the training algorithm is that the same network input, a time t and a position $x(t|x_0)$, can be produced by many different trajectories, each with a different conditional velocity. While the network training is based on a wide range of possible trajectories, the CFM loss in Eq. (4.37) ensures that sampling over many trajectories returns a well-defined velocity field.

Once the CFM model is trained, the generation of new samples is straightforward. We start by drawing a sample from the latent distribution $x_1 \sim p_\text{latent} = \mathcal{N}(0,1)$ and calculate its time evolution by numerically solving the ODE backwards in time from $t = 1$ to $t = 0$

$$\begin{aligned} &\frac{d}{dt}x(t) = v_\theta(x(t),t) \qquad \text{with} \quad x_1 = x(t=1) \\ \Rightarrow \qquad &x_0 = x_1 - \int_0^1 v_\theta(x,t)dt \equiv G_\theta(x_1)\;, \end{aligned} \tag{4.38}$$

We use the `scipy.solve_ivp` function with default settings for this. Under mild regularity assumptions this solution defines a bijective transformation between the latent space sample and the phase space sample $G_\theta(x_1)$, similar to an INN.

Likelihood extraction

The CFM model also allows to calculate phase space likelihoods. Making use of the continuity equation we can write

$$\begin{aligned} \frac{dp(x,t)}{dt} &= \frac{\partial p(x,t)}{\partial t} + \nabla_x p(x,t)\; v(x,t) \\ &= \frac{\partial p(x,t)}{\partial t} + \nabla_x \left[p(x,t)v(x,t)\right] - p(x,t)\nabla_x v(x,t) \\ &= -p(x,t)\nabla_x v(x,t)\;. \end{aligned} \tag{4.39}$$

Its solution is

$$\frac{p(x_1,1)}{p(x_0,0)} \equiv \frac{p_\text{latent}(G_\theta^{-1}(x_0))}{p_\theta(x_0|\theta)} = \exp\left(-\int_0^1 dt \nabla_x v(x(t),t)\right)\;, \tag{4.40}$$

and we can write in the usual INN notation [23]

$$\begin{aligned} p_\theta(x_0|\theta) &= p_\text{latent}(G_\theta^{-1}(x_0))\left|\det\frac{\partial G_\theta^{-1}(x_0)}{\partial x_0}\right| \\ \Rightarrow \qquad \left|\det\frac{\partial G_\theta^{-1}(x_0)}{\partial x_0}\right| &= \exp\left(\int_0^1 dt \nabla_x v_\theta(x(t),t)\right)\;. \end{aligned} \tag{4.41}$$

Calculating the Jacobian requires integrating over the divergence of the learned velocity field. This divergence can be calculated using automatic differentiation approximately as fast as n network calls, where n is the data dimensionality.

Bayesian CFM

Finally, we also turn the CFM into a Bayesian generative model, to account for the uncertainties in the underlying density estimation [161]. From the Bayesian DDPM we know that this can be achieved by promoting the network weights from deterministic values to, for instance, Gaussian distributions and using variational approximation for the training [155–157, 172]. For the Bayesian INN or the Bayesian DDPM the loss is a sum of the likelihood loss and a KL-divergence regularization, Eq. (4.23). Unfortunately, the CFM loss in Eq. (4.37) is not a likelihood loss. To construct a Bayesian CFM loss we therefore combine it with Bayesian network layers and a free KL-regularization,

$$\mathcal{L}_\text{B-CFM} = \Big\langle \mathcal{L}_\text{CFM} \Big\rangle_{\theta\sim q(\theta)} + c\,\text{KL}[q(\theta),p(\theta)]. \tag{4.42}$$

While for a likelihood loss the factor c is fixed by Bayes' theorem, in the CFM case it is a free hyperparameter. We find that the network predictions and their associated uncertainties are very stable when varying it over several orders of magnitude.

Our network is implemented in PyTorch [173] and uses Adam [114] as optimizer. All hyperparameters are given in Tab. A.2. We employ a simple network consisting of fully connected dense layers with SiLU activation functions. Given limited resources, simple and fast networks trained for a large number of iterations produces the best results. For the LHC events we used two blocks of dense layers connected by a residual connection. In our setup dropout layers lead to significantly worse results, while normalization layers have no visible impact on the results. We find that the training of CFM models can be very noisy, using a large batch size can help to stabilize this.

In general, training diffusion models requires a relatively large number of epochs, as indicated in Tabs. A.1 and A.2. A key result of our study is to use a cosine-annealing learning rate scheduler for the CFMs and one-cycle scheduling for the DDPM, as well as significantly downsizing the models compared to INNs, to allow for more training epochs. For the entire hyperparameter setup, our B-DDPM implementation turns out to be slightly more sensitive than the B-CFM.

4.1.3 Autoregressive Transformer

Architecture

A distinct shortcoming of traditional generative models like GANs, INNs, and diffusion models is that they learn the correlations in all phase space directions simultaneously. This leads to a power-law scaling for instance of the training effort for a constant precision in the learned correlations [166]. The autoregressive transformer (AT) [52, 174] instead interprets the phase space vector $x = (x_1, ...x_n)$ as a sequence of elements x_i and factorizes the joint n-dimensional probability into n probabilities with a subset of conditions,

$$p_\theta(x|\theta) = \prod_{i=1}^{n} p(x_i|x_1, ..., x_{i-1}) \approx p_{\text{data}}(x) , \tag{4.43}$$

as illustrated in Fig. 4.4. This autoregressive approach improves the scaling with the phase space dimensionality in two ways. First, each distribution $p(x_i|x_1, ...x_{i-1})$ is easier to learn than a distribution conditional on the full phase space vector x. Second, we can use our physics knowledge to group challenging phase space directions early in the sequence $x_1, ..., x_n$.

The network learns the conditional probabilities over phase space using a representation

$$p(x_i|\omega^{(i-1)}) = p(x_i|x_1, ...x_{i-1}) , \tag{4.44}$$

where the parameters $\omega^{(i-1)}$ encode the conditional dependence on $x_1, ...x_{i-1}$. A naive choice are binned probabilities $w_j^{(i-1)}$ per phase space direction,

$$p(x_i|\omega^{(i-1)}) = \sum_{\text{bins } j} w_j^{(i-1)} \mathbb{1}^{(j)}(x_i) , \tag{4.45}$$

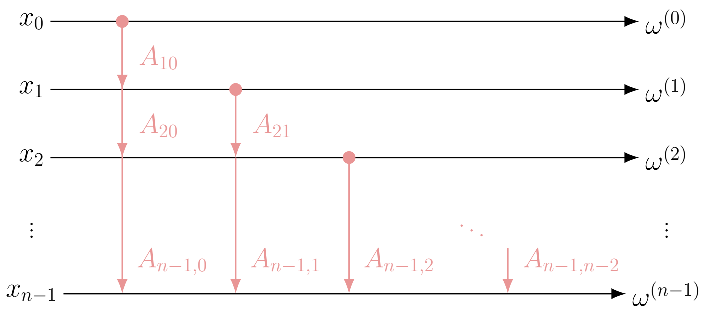


Figure 4.4: Autoregressive approach to density estimation. The attention matrix A_{ij} defined in Eq. (4.50) encodes information between components x_i. We introduce an auxiliary condition $x_0 = 0$ for the first phase space component x_1.

where $\mathbb{1}^{(j)}(x)$ is one for x inside the bin j and zero outside. A more flexible and better-scaling approach is a Gaussian mixture,

$$p(x_i|\omega^{(i-1)}) = \sum_{\text{Gaussian } j} w_j^{(i-1)} \mathcal{N}(x_i; \mu_j^{(i-1)}, \sigma_j^{(i-1)}) \; . \qquad (4.46)$$

It generalizes the fixed bins to a set of learnable means and widths.

Our architecture closely follows the Generative Pretrained Transformer (GPT) models [174], illustrated in Fig. 4.5. The network takes a sequence of x_i as input and evaluates them all in parallel. We use a linear layer to map each value x_i in a d-dimensional latent space, denoted as $x_{i\alpha}$. The network consists of a series of TransformerDecoder blocks, combining a self-attention layer with a standard feed-forward network. Finally, a linear layer maps the latent space onto the representation $\omega^{(i-1)}$ of the conditions.

Equations (4.45) and (4.46) do not provide an actual structure correlating phase space regions and phase space directions. This means the transformer needs to construct an appropriate basis and correlation pattern by transforming the input x into an x', with the same dimension as the input vector and leading to the ω representation. Its goal is to construct a matrix A_{ij} that quantifies the relation or similarity of two embedded phase space components $x_{i\alpha}$ and $x_{j\alpha}$. We construct the single-headed self-attention [52] of an input x in three steps.

1. Using the conventions of the first layer, we want to measure the relation between x_i and a given x_j, embedded in the d-dimensional latent space. Replacing the naive scalar product $x_{i\alpha} x_{j\alpha}$, we introduce learnable latent-space transformations $W^{Q,K}$

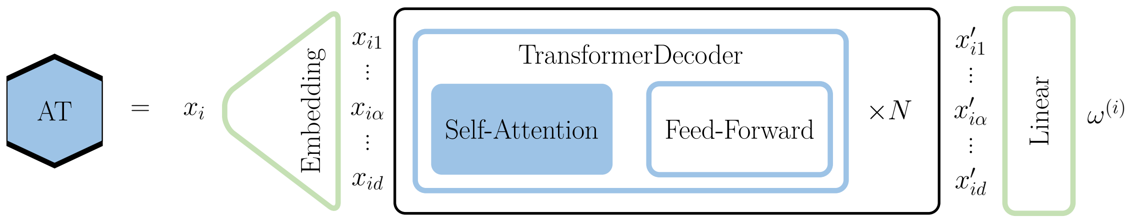


Figure 4.5: Architecture of the autoregressive transformer. All phase space components x_i are evaluated in parallel, see Fig. 4.4.

to the elements

$$q_{i\alpha} = W^Q_{\alpha\beta} x_{i\beta} \qquad \text{and} \qquad k_{j\alpha} = W^K_{\alpha\beta} x_{j\beta} \; , \tag{4.47}$$

and use the directed scalar product

$$A_{ij} \sim q_{i\alpha} k_{j\alpha} \tag{4.48}$$

to encode the relation of x_j with x_i through k_j and q_i. While the scalar product is symmetric, the attention matrix does not have to be, $A_{ij} \neq A_{ji}$. These global transformations allow the transformer to choose a useful basis for the scalar product in latent space.

2. The first problem with A_{ij} given in Eq. (4.48) is that it grows with the latent space dimensionality, so it turns out to be useful to replace it by $A_{ij} \to A_{ij}/\sqrt{d}$. More importantly, we want all entries j referring to a given i to be normalized,

$$A_{ij} \in [0,1] \qquad \text{and} \qquad \sum_j A_{ij} = 1 \; . \tag{4.49}$$

This leads us to the definition

$$A_{ij} = \text{Softmax}_j \frac{q_{i\alpha} k_{j\alpha}}{\sqrt{d}} \qquad \text{with} \qquad \text{Softmax}_j(x_j) = \frac{e^{x_j}}{\sum_k e^{x_k}} \; . \tag{4.50}$$

Similar to the adjacency matrix of a graph, this attention matrix quantifies how closely two phase space components are related. Our autoregressive setup sketched in Fig. 4.4 requires us to set

$$A_{ij} = 0 \qquad \text{for} \quad j > i \; . \tag{4.51}$$

3. Now that the network has constructed a basis to evaluate the relation between two input elements x_i and x_j, we use it to update the actual representation of the input information. We combine the attention matrix A_{ij} with the input data, but again transformed in latent space through a learnable matrix W^V,

$$\begin{aligned} v_{j\alpha} = W^V_{\alpha\beta} x_{j\beta} \quad \Rightarrow \quad x'_{i\alpha} &= A_{ij} v_{j\alpha} \\ &= \text{Softmax}_j \left(\frac{W^Q_{\delta\gamma} x_{i\gamma} W^K_{\delta\sigma} x_{j\sigma}}{\sqrt{d}} \right) W^V_{\alpha\beta} x_{j\beta} \; . \end{aligned} \tag{4.52}$$

In this form we see that the self-attention vector x' just follows from a general basis transformation with the usual scalar product, but with an additional learned transformation for every input vector.

The self-attention can be stacked with other structures like a feed-forward network, to iteratively construct an optimal latent space representation. This can either be identified with the final output $\omega^{(i)}$ or linked to this output through a simple linear layer. To guarantee a stable training of this complex structure, we evaluate the self-attention as an ensemble, defining a multi-headed self-attention. In addition, we include residual connections, layer normalization, and dropout just like the GPT model. Because the sum over j in Eq. (4.52) leads to permutation equivariance in the phase space components, we break it by providing explicit positional information through a linear layer that takes

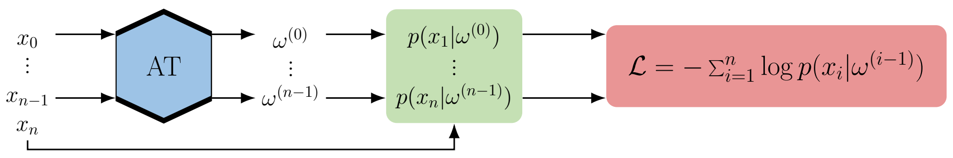


Figure 4.6: Training algorithm for the autoregressive transformer.

the one-hot encoded phase space position i as input. This positional embedding is then added to the latent representation $x_{i\alpha}$.

Training and sampling

The training of the autoregressive transformer is illustrated in Fig. 4.6. We start with an universal $x_0 = 0$ in $p(x_1|\omega^{(0)})$ for all events. The transformer encodes all parameters ω needed for $p(x_i|\omega^{(i-1)})$ in parallel. The chain of conditional likelihoods for the realized values x_i gives the full likelihood $p_\theta(x|\theta)$, which in turn can be used for the loss function

$$\begin{aligned}\mathcal{L}_{\text{AT}} &= \Big\langle -\log p_\theta(x|\theta) \Big\rangle_{x\sim p_{\text{data}}} \\ &= \sum_{i=1}^n \Big\langle -\log p(x_i|\omega^{(i-1)}) \Big\rangle_{x\sim p_{\text{data}}} .\end{aligned} \tag{4.53}$$

The successive transformer sampling is illustrated in Fig. 4.7. For each component, $\omega^{(i-1)}$ encodes the dependence on the previous components $x_1, ..., x_{i-1}$, and correspondingly we sample from $p(x_i|\omega^{(i-1)})$. The parameters $\omega^{(0)}, ...\omega^{(i-2)}$ from the sampling of previous components are re-generated in each step, but not used further. This way the event generation is less efficient than the likelihood evaluation during training, because it cannot be parallelized.

Bayesian version

As any generative network, we bayesianize the transformer by drawing its weights from a set of Gaussians $q(\theta)$ as defined Eq. (4.21). In practice, we replace the deterministic layers

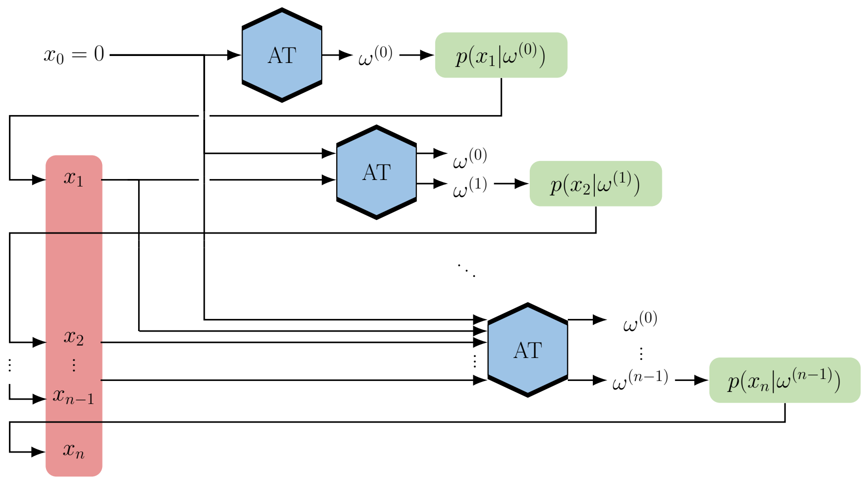


Figure 4.7: Sampling algorithm for the autoregressive transformer.

of the transformer by Bayesian layers and add the KL-regularization from Eq. (4.22) to the likelihood loss of the transformer, Eq. (4.53)

$$\mathcal{L}_{\text{B-AT}} = \Big\langle \mathcal{L}_{\text{AT}} \Big\rangle_{\theta\sim q(\theta)} + \text{KL}[q(\theta), p(\theta)]. \tag{4.54}$$

For large generative networks, we encounter the problem that too many Bayesian weights destabilize the network training. While a deterministic network can switch of unused weights by just setting them to zero, a Bayesian network can only set the mean to zero, in which case the Gaussian width will approach the prior $p(\theta)$. This way, excess weights can contribute noise to the training of large networks. This problem can be solved by adjusting the hyperparameter describing the network prior or by only bayesianizing a fraction of the network weights. In both cases it is crucial to confirm that the uncertainty estimate from the network is on a stable plateau. For the transformer we find that the best setup is to only bayesianizing the last layer.

To implement the autoregressive transformer we use PyTorch [173] with the RAdam [175] optimizer. All hyperparameters are given in Tab. A.3. We propose to couple the number of parameters m in the parametrization vector $\omega^{(i-1)}$ to the latent space dimensionality d, because the latent space dimensionality naturally sets the order of magnitude of parameters that the model can predict confidently.

4.2 Toy models and Bayesian networks

Before we can turn to the LHC phase space as an application to our novel generative models, we study their behavior for two simple toy models, directly comparable to Bayesian INNs [161]. These toy models serve two purposes: first, we learn about the strengths and the challenges of the different network architectures, when the density estimation task is simple and the focus lies on precision. Second, the interplay between the estimation of the density and its uncertainty over phase space allows us to understand how the different network encode the density. We remind ourselves that an INN just works like a high-dimensional fit to the correlated 2-dimensional densities [161].

Denoising Diffusion Probabilistic Model

Our first toy example is a normalized ramp, linear in one direction and flat in the second,

$$p_{\text{ramp}}(x_1, x_2) = 2x_2\ . \tag{4.55}$$

The network input and output are unweighted events. The hyperparameters of each model are given in Tabs. A.1, A.2, and A.3. A training dataset of 600k events guarantees that for our setup and binning the statistical uncertainty on the phase space density is around the per-cent level. To show one-dimensional Bayesian network distributions we sample the x_i-direction and the θ-space in parallel [45, 161]. This way the uncertainty in one dimension is independent of the existence and size of other dimensions.

Starting with the DDPM we show the non-trivial one-dimensional distributions in Fig. 4.8. In the left panel we see that the network learns the underlying phase space density well, but not quite at the desired per-cent precision. The uncertainty from the B-DDPM captures remaining deviations, if anything, conservatively. In the right panel we see that

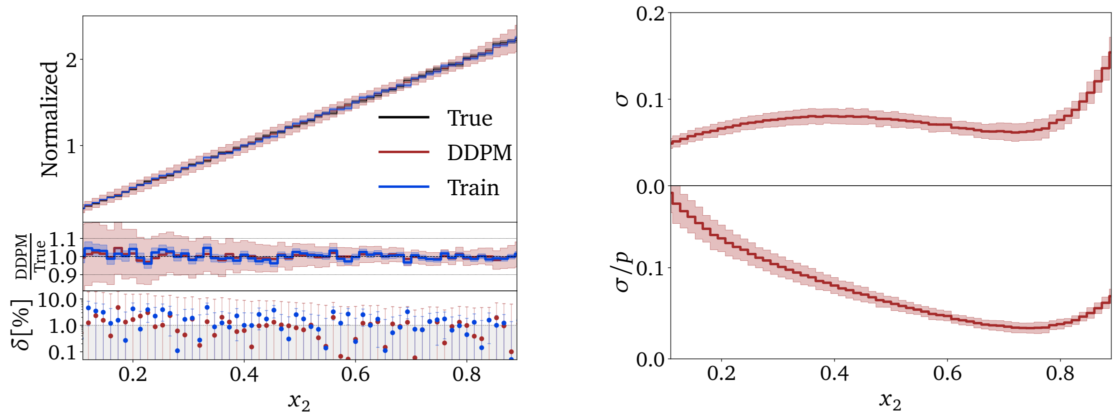


Figure 4.8: Ramp distribution from the DDPM. We show the learned density and its B-DDPM uncertainty (left) as well as the absolute and relative uncertainties with a range given by 10 independent trainings (right). We use $\delta = |\text{Model} - \text{Truth}|/\text{Truth}$.

the absolute uncertainty has a minimum around $x_1 = 0.7$, similar to the behavior of the Bayesian INN and confirmed by independent trainings. We can understand this pattern by looking at a constrained fit of the normalized density

$$p(x_2) = ax_2 + b = a\left(x_2 - \frac{1}{2}\right) + 1 \qquad \text{with} \qquad x_2 \in [0,1]\ . \tag{4.56}$$

A fit of a then leads to an uncertainty in the density of

$$\sigma \equiv \Delta p \approx \left|x_2 - \frac{1}{2}\right| \; \Delta a\ , \tag{4.57}$$

just using simple error propagation. The minimum in the center of the phase space plan can be interpreted as the optimal use of correlations in all directions to determine the local density.

For the DDPM the minimum is not quite at $x_2 = 0.5$, and the uncertainty as a function of x_2 is relatively flat over the entire range. Because of the statistically limited training

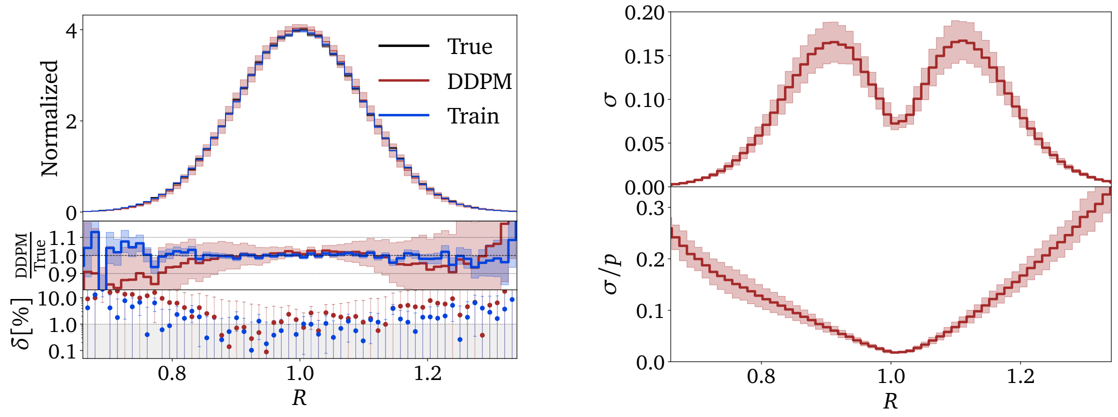


Figure 4.9: Gaussian ring distribution from the DDPM. We show the learned density and its B-DDPM uncertainty (left) as well as the absolute and relative uncertainties with a range given by 10 independent trainings (right).

sample, the network output comes with a relatively large uncertainty towards $x_2 = 0$. For larger x_2-values, the gain in precision and uncertainty is moderate. For $x_2 > 0.75$ the absolute and relative uncertainties increase, reflecting the challenge to learn the edge at $x_2 = 1$. These results are qualitatively similar, but quantitatively different from the INN case, which benefits more from the increase in training data and correlations for $x_2 = 0.1 \dots 0.5$.

The second toy example is a Gaussian ring, or a Gaussian sphere in two dimensions,

$$p_\text{ring}(x_1, x_2) = \mathcal{N}\left(\sqrt{x_1^2 + x_2^2}; 1, 0.1\right). \tag{4.58}$$

The DDPM result are shown in Fig. 4.9. The precision on the density is significantly worse than for the ramp, clearly missing the per-cent mark. The agreement between the training data and the learned density is not quite symmetric, reflecting the fact that we train and evaluate the network in Cartesian candidates but show the result in R. Especially for large radii, the network significantly overestimates the tail, a failure mode which is covered by the predictive uncertainty only for $R \lesssim 1.3$. In the right panels of Fig. 4.9 the main feature is a distinct minimum in the uncertainty around the mean of the Gaussian. As for the ramp, this can be understood from error propagation in a constrained fit. If we assume that the network first determines a family of functions describing the radial dependence, in terms of a mean and a width, the contribution from the mean vanishes at $R = 1$ [161]. Alternatively, we can understand the high confidence

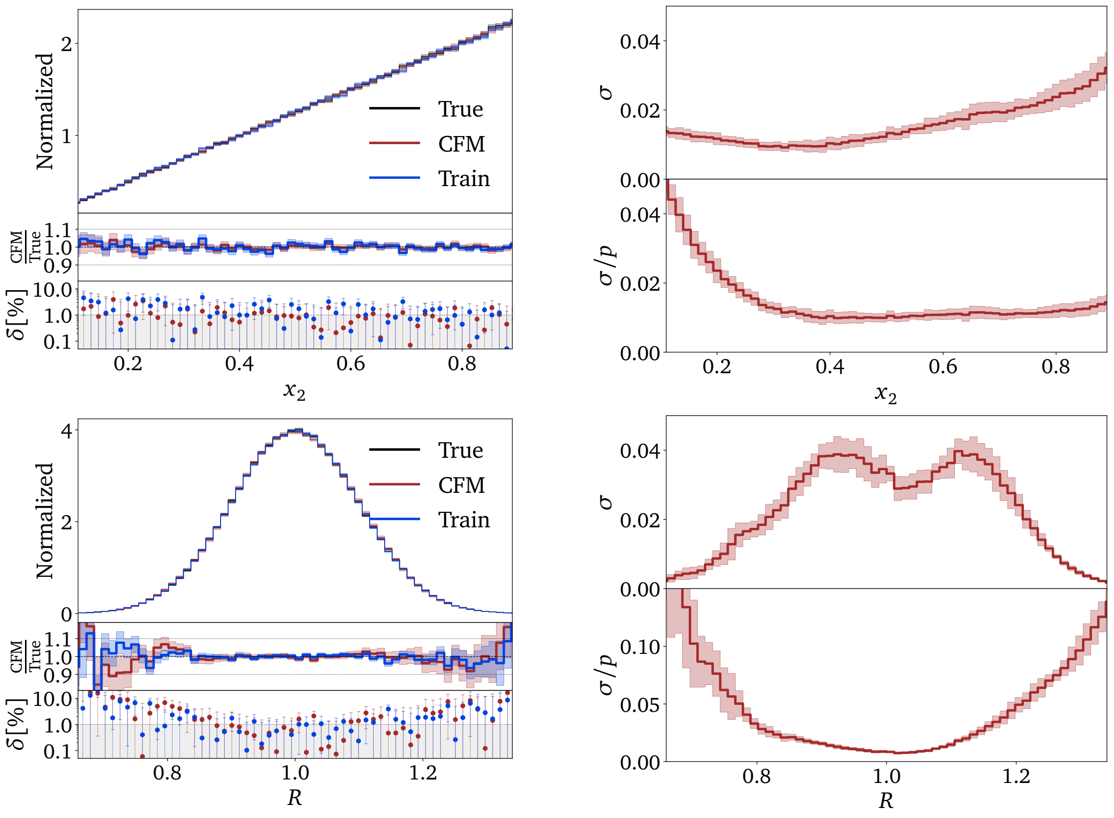


Figure 4.10: Ramp (upper) and Gaussian ring (lower) distributions from the CFM. We show the learned density and its B-CFM uncertainty (left) as well as the absolute and relative uncertainties with a range given by 10 independent trainings (right).

of the network through the availability of many radial and angular correlations in this phase space region.

Conditional Flow Matching

To confirm that the diffusion architecture is behind the DDPM features, we repeat our study with the CFM model in Fig. 4.10. The main difference to the DDPM is that the agreement between the learned and the training densities is now at the per-cent level, for the ramp and for the Gaussian ring. This shows that diffusion models are indeed able to learn a phase space density with the same precision and stability as normalizing flows or INNs. As before, the predictive uncertainty from the B-CFM model is conservative for the entire phase space of the ramp, but it fails in the exponentially suppressed tail of the Gaussian ring for $R \gtrsim 1.3$. We emphasize that as a function of R this problem is clearly visible when we increase R to the point where $\sigma(R) = \mathcal{O}(p(R))$.

Looking at the pattern of the predicted uncertainty σ in x_2 and in R, we see a similar behavior as for the INN and for the DDPM. As for the DDPM, the minimum in the middle of the ramp is flatter than for the INN, and its position has moved to $x_2 \approx 0.3$. For the radial distribution of the Gaussian ring there is the usual minimum on the peak.

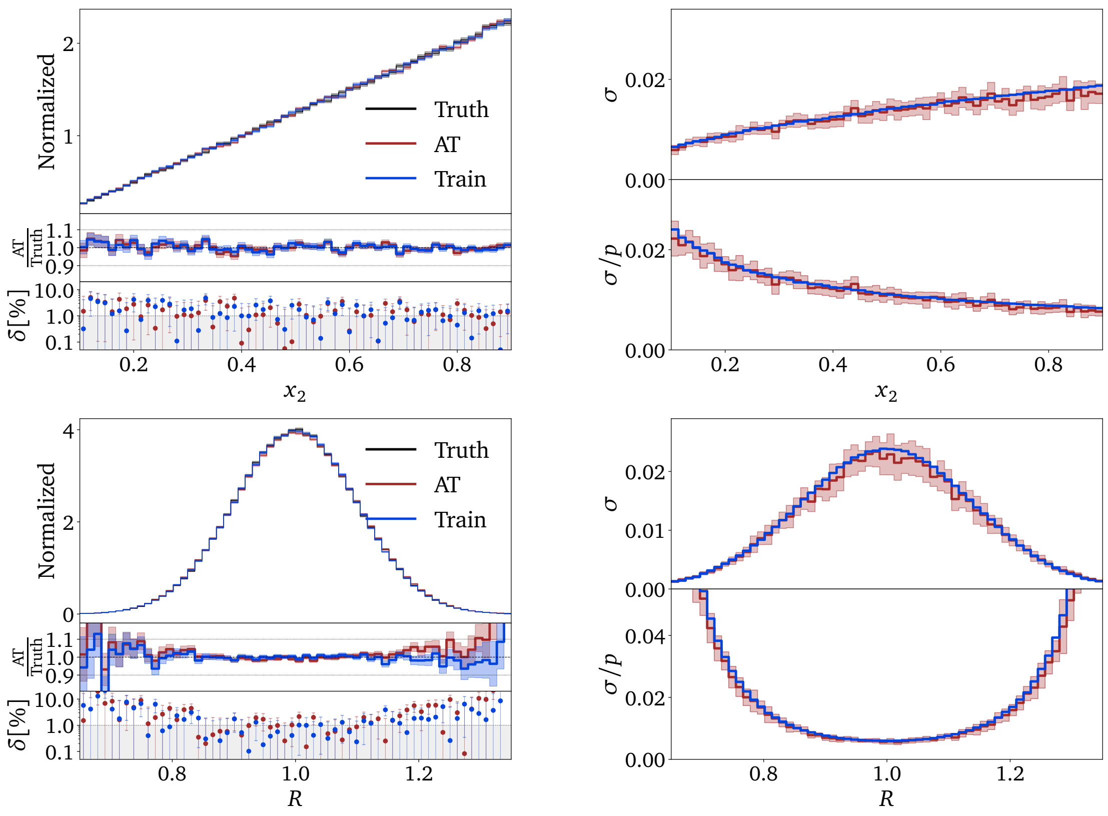


Figure 4.11: Ramp (upper) and Gaussian ring (lower) distribution from the autoregressive transformer with a binned likelihood. We show the learned density and its Bayesian network uncertainty (left) as well as the absolute and relative uncertainties with a range given by 10 independent trainings, compared to the statistical uncertainty of the training data in blue (right).

Summarizing our findings for the two diffusion models, they behave similar but not identical to the INN. For all of them, the relation between the density and its uncertainty shows patterns of a constrained fit, suggesting that during the the training the networks first determine a class of suitable models and then adjust the main features of these models, like the slope of a ramp or the position and width of a Gaussian ring.

Autoregressive Transformer

Finally, we target the two-dimensional ramp, Eq. (4.55), and the Gaussian ring, Eq. (4.58) with the transformer. In Fig. 4.11 we start with a simple representation of the phase space density using 64 bins. In this naive setup the densities of the ramp and the Gaussian ring are described accurately, within our per-cent target range. The largest deviations appear in the tails of the Gaussian ring, but remain almost within the statistical limitations of the training data.

Unlike for the INN and the diffusion models, the uncertainty in the right panels of Fig. 4.11 does not show any real features for the ramp or the Gaussian ring. This shows that the transformer does not use a fit-like density estimation and does not benefit from the increased correlations in the center of phase space. Both of these aspects can be understood from the model setup. First, the autoregressive structure never allows the

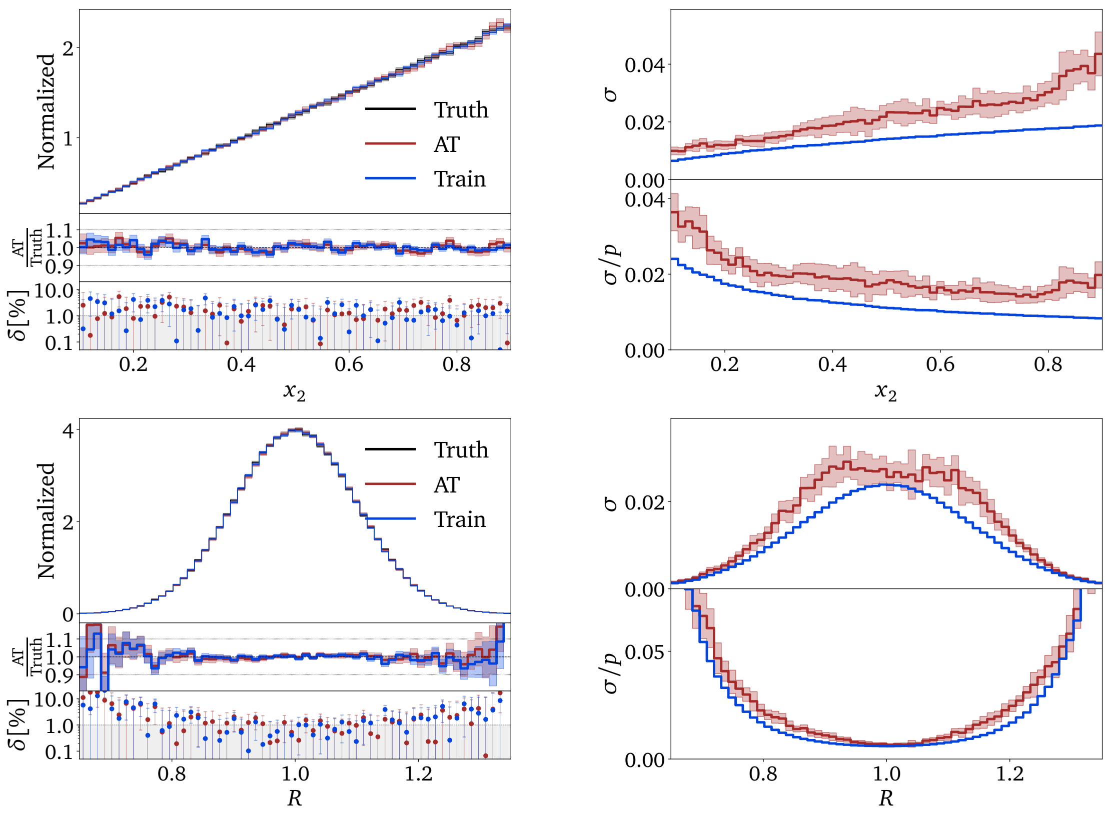


Figure 4.12: Ramp (upper) and Gaussian ring (lower) distribution from the autoregressive transformer with a Gaussian mixture likelihood. We show the learned density and its Bayesian network uncertainty (left) as well as the absolute and relative uncertainties with a range given by 10 independent trainings, compared to the statistical uncertainty of the training data in blue (right).

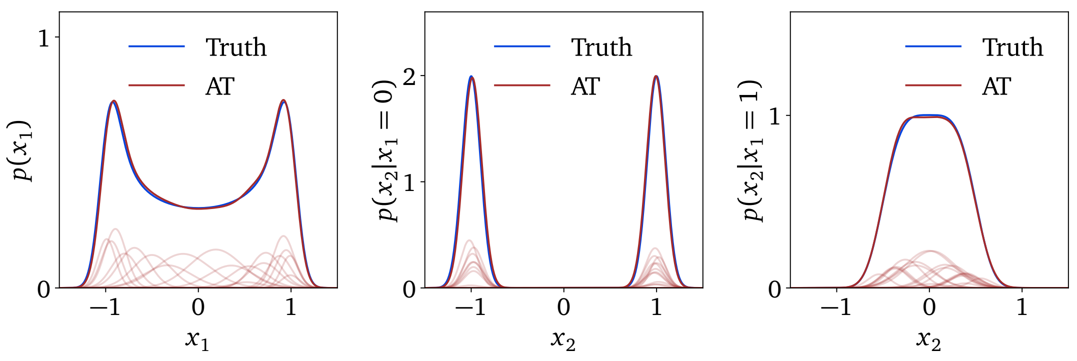


Figure 4.13: Conditional likelihoods for the Gaussian ring. We show the full Gaussian mixture as well as the 21 individual Gaussians, compared to the truth distribution.

transformer to see the full phase space density and encode global (symmetry) patterns; second, the main motivation of the transformer is to improve the power-law scaling with the dimensionality of all possible correlations and only focus on the most relevant correlations at the expense of the full phase space coverage.

In Fig. 4.12 we show the same results for a mixture of 21 Gaussians. For this small number of dimensions the advantage over the binned distribution is not obvious. The main problem appears at the upper end of the ramp, where there exists enough training data to determine a well-suited model, but the poorly-suited GMM just fails to reproduce the flat growth towards the sharp upper edge and introduces a significant artifact, just covered by the uncertainty. For the Gaussian ring the GMM-based transformer is also less precise than the binned version, consistent with the lower resolution in the 2-dimensional model.

The uncertainty predicted by the Bayesian transformer is typically smaller than for diffusion models. We therefore add the statistical uncertainty of the training data to the right panels of Figs. 4.11 and 4.12, providing a lower bound on the uncertainty. In both cases, the uncertainty of the Bayesian transformer conservatively tracks the statistical uncertainty of the training data.

Finally, in Fig. 4.13 we illustrate the unique way in which the GMM-based transformer reconstructs the density for the Gaussian ring successively. In the left panel, we show $p_\theta(x_1)$ after the first autoregressive step, constructed out of 21 learned Gaussians. The peaks at ± 1 arise from the marginalization along the longest line of sight. The marginalization also distorts the form of the Gaussians, which are distributed along the ring. The density after the second autoregressive step, $p_\theta(x_2|x_1)$, is conditioned on the first component. In the second panel we show $p_\theta(x_2|x_1 = 0)$ with sharp peaks at ± 1 because the event has to be at the edge of the ring. The Gaussians building the left and right peak are distributed roughly equally. On the other hand, $p_\theta(x_2|x_1 = 1)$ has a broad plateau in the center, again from the x_1-condition.

4.3 LHC events

Most generative network tasks at the LHC are related to learning and sampling phase space densities, for instance event generation at the parton or reconstruction level, the

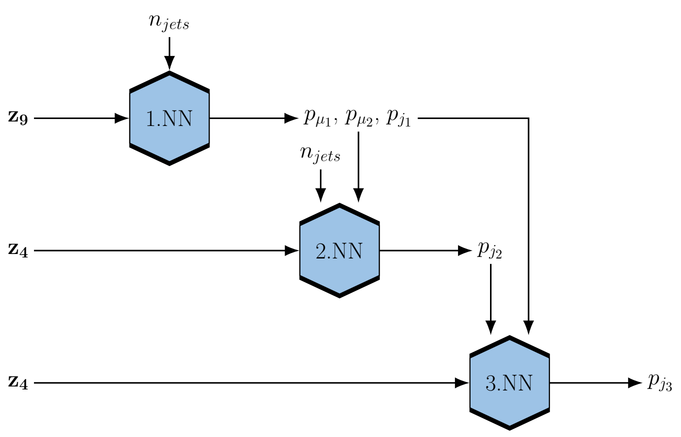


Figure 4.14: Conditional Sampling Architecture.

description of detector effects at the reconstruction level, the computation of event-wise likelihoods in the matrix element method, or the inversion and unfolding of reconstructed events. This is why we benchmark our new networks on a sufficiently challenging set of LHC events. Following Ref. [45] we choose the the production of leptonically decaying Z-bosons, associated with a variable number of QCD jets,

$$pp \to Z_{\mu\mu} + \{1, 2, 3\} \text{ jets} . \tag{4.59}$$

The network has to learn the sharp Z-peak as well as correlated phase space boundaries and features in the jet-jet correlations. We generate the training dataset of 5.4M events (4.0M + 1.1M + 300k) using Sherpa2.2.10 [31] at 13 TeV, including ISR and parton shower with CKKW merging [176], hadronization, but no pile-up. The jets are defined by Fastjet3.3.4 [177] using the anti-k_T algorithm [109] and applying the basic cuts

$$p_{T,j} > 20 \text{ GeV} \qquad \text{and} \qquad \Delta R_{jj} > 0.4 . \tag{4.60}$$

The jets and muons are each ordered in transverse momentum. Our phase space dimensionality is three per muon and four per jet, i.e. 10, 14, and 18 dimensions. Momentum conservation is not guaranteed, because some final-state particles might escape for instance the jet algorithm. However, the physically relevant phase space dimensionality is reduced to 9, 13, and 17 by removing the global azimuthal angle.

Our data representation includes a minimal preprocessing. Each particle is represented by

$$\{ \, p_T, \eta, \phi, m \, \} . \tag{4.61}$$

Given Eq. (4.60), we provide the form $\log(p_T - p_{T,\text{min}})$, leading to an approximately Gaussian shape. All azimuthal angles are given relative to the leading muon, and the transformation into $\text{artanh}(\Delta\phi/\pi)$ again leads to an approximate Gaussian. The jet mass is encoded as $\log m$. Finally, we centralize and normalize each phase space variable as $(q_i - \bar{q}_i)/\sigma(q_i)$ and apply a whitening/PCA transformation separately for each jet multiplicity for the two diffusion models.

Denoising Diffusion Probabilistic Model

The additional challenge for Z+jets event generation is the variable number of jets, which we tackle with a conditional evaluation [45], illustrated in Fig. 4.14. The training is independent for the three jet multiplicities. We start by giving the information for the $Z + 1$-jet sub-process, 12 phase space dimensions, to a first network. It is supplemented with the one-hot encoded jet count. The second network then receives the 4-momentum of the second jet as an input, and the $Z + 1$-jet information additionally to the jet count as a condition. Analogously, the third network learns the third jet kinematics conditioned on the $Z + 2$-jet information. For democratic jets this conditioning would be perfect, but since we order the jets in p_T it has to and does account for the fact that for higher jet multiplicities the interplay between partonic energy and jet combinatorics leads to differences in the spectra of the leading jets at a given multiplicity.

As discussed in Sec. 4.1.1 time is a crucial condition for the DDPM network, and we embed it into the full conditioning of the LHC setup as a high-dimensional latent vector linked by a linear layer. We also add a second block to our network architecture, where the conditions are fed to each block individually. The amount of training data is different for the different jet multiplicities and corresponding networks. As shown in Tab. A.1, the first network uses the full 3.2M events, the second 850k events with at least two jets, and the third network 190k events with three jets. This hierarchy is motivated by the way the chain of conditional networks add information and also by the increasing cost of producing the corresponding training samples. We could balance the data during training, but for the B-DDPM model this leads to a slight performance drop. We compensate the lack of training data by increasing the number of epochs successively from 1000 to 10000.

Going from toy models to LHC events, we increase the number of blocks to two, which improves the performance. The reason is that we attach the condition to the input at the beginning of each block, so the second block reinforces the condition. Going to even more blocks will slightly improve the performance, but at the expense of the training time.

In Fig. 4.15 we show a set of kinematic distributions for different jet multiplicities, including the jet-inclusive scalar sum of the up to three $p_{T,j}$. These distributions will be the same for all three network in this paper and can be compared directly to the Bayesian INN results in Fig. 11 of Ref. [45], serving as a precision baseline. Starting with the almost featureless p_T-distributions in the left panels, we see that for all three distributions the deviation from the truth, given by high-statistics training data, is similar for the actual training data and for the DDPM-generated events. The network really extracts all available information from the training data combined with its fit-like implicit bias. For sufficient training statistics, the precision on the phase space density as a function of p_T is below the per-cent level, easily on par with the INN baseline. For a given jet multiplicity this precision drops with increasing p_T and correspondingly decreasing training data, an effect that is correctly and conservatively modeled by the uncertainty estimate of the B-DDPM. Combining all n-jet samples into one observable is no problem for the network and does not lead to any artifacts.

In the right panels of Fig. 4.15 we show the most challenging phase space correlations. We start with the Z-peak, which governs most of the events, but requires the network to learn a very specific phase space direction very precisely. Here, the agreement between the true density and the DDPM result drops to around 10% without any additional phase space mapping, similar to the best available INN. The deviation is not covered by the Bayesian network uncertainty, because it arises from a systematic failure of the network

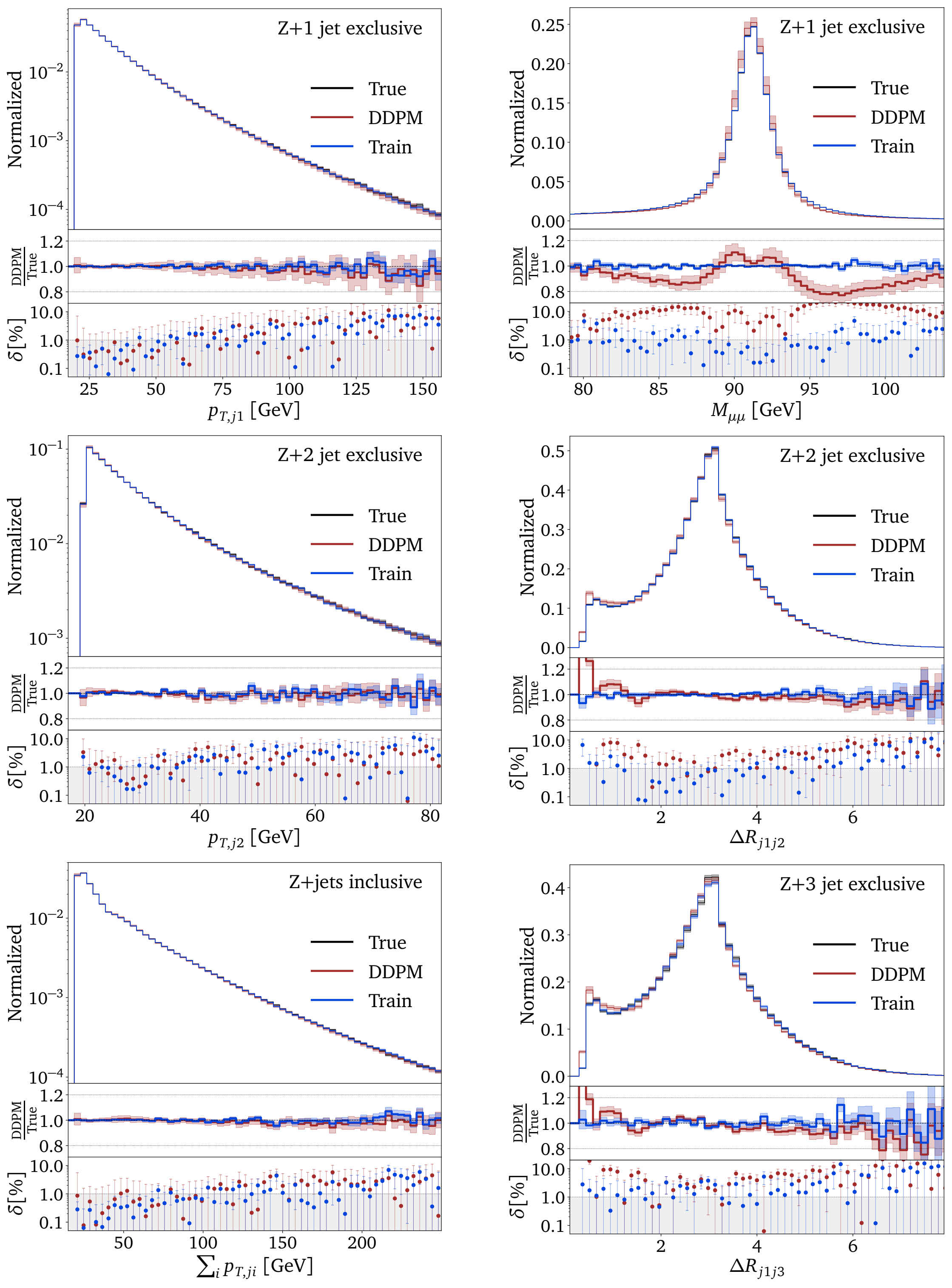


Figure 4.15: Bayesian DDPM densities and uncertainties for $Z + 1$ jet (upper), $Z + 2$ jets (center), and $Z + 3$ jets (lower) from combined $Z+$ jets generation. The uncertainty on the training data is given by bin-wise Poisson statistics. The network architecture is given in Tab. A.1. For a comparison with the INN we refer to Fig. 11 of Ref. [45].

in the phase space resolution, induced by the network architecture. However, this effect is less dramatic than it initially looks when we notice that the ratio of densities just describes the width of the mass peak being broadened by around 10%. If needed, it can be easily corrected by an event reweighting of the Z-kinematics. Alternatively, we can change the phase space parametrization to include intermediate particles, but most likely at the expense of other observables.

Next, we study the leading challenge of ML-event generators, the jet-jet correlations and specifically the collinear enhancement right at the hard jet-separation cut of $\Delta R_{jj} > 0.4$. Three aspects make this correlation hard to learn: (i) this phase space region is a subleading feature next to the bulk of the distribution around $\Delta R_{jj} \sim \pi$; (ii) it includes a sharp phase space boundary, which density estimators will naturally wash out; and (iii), the collinear enhancement needs to be described correctly, even though it appears right at the phase space boundary. Finally, for this correlation the conditional setup and the Bayesian extension are definitely not helpful.

What helps for this correlation is the so-called magic transformation introduced in Ref. [45]. It scales the ΔR_{jj}-direction in phase space such that the density in this phase space direction becomes a monotonous function. While from a classic Monte Carlo perspective the benefits of this transformation are counter-intuitive, from a a fit-like perspective the magic transformation can simplify the class of function which the network then adapts to the data, as shown for the toy models in the previous section. This argument is confirmed by the fact that for our diffusion networks this transformation is helpful, just like for the INN, but for the transformer it is not needed. Both, for the 2-jet and the 3-jet sample we see that with the magic transformation the DDPM learns the ΔR_{jj} features, but at the same 10% level as the INN and hence missing our 1% target. The Bayesian uncertainty estimate increases in this phase space region as well, but it is not as conservative as for instance in the p_T-tails.

The challenge of current diffusion networks, also the DDPM, is the evaluation speed. For each additional jet we need to call our network 1000 times, so sampling 3-jet events takes three times as long as sampling 1-jet events. However, none of the networks presented in this study are tuned for generation speed, the only requirement for a limited hyperparameter scan is the precision baseline given by the INN.

Conditional Flow Matching

For the CFM diffusion network we follow the same conditional setup as for the DDPM and the INN to account for the variable number of jets. The network is described in Tab. A.2, unlike for the DDPM the three networks do not have the same size, but the first network with its 9 phase space dimensions is larger. Also the number of epochs increases from 1000 to 10000 going to the 3-jet network. For the CFM we combine the embedding of the time and the conditioning on the lower jet multiplicities. We find the best results when encoding time, the kinematic condition, and the actual network input separately into same-sized latent vectors with independent linear layers. Then all three are concatenated and given to the network.

The kinematic distributions generated by the CFM are shown in Fig. 4.16. Again, the transverse momentum spectra are learned with high precision, with decreasing performance in the tails, tracked correctly by the Bayesian network uncertainty. The correlation describing the Z-peak is now modeled as well as the bulk of the single-particle distributions, a significant improvement over the INN baseline [45]. For the most challenging ΔR_{jj} distributions the CFM uses the same magic transformation as the DDPM and achieves comparable precision. This means that while there might possibly be a slight benefit to our CFM implementation with an ODE approach to the discrete time evolution in terms of precision, our level of network optimization does not allow us to attribute this difference to luck vs network architecture. Similarly, in the current implementation the CFM generation is about an order of magnitude faster than the DDPM generation, but this can mostly be attributed to the linear trajectory and the extremely efficient ODE solver.

Autoregressive Transformer

For the third network, a generative transformer, we already know from Sec. 4.2 that it learns and encodes the phase space density different from normalizing flows or diffusion networks. A key structural difference for generating LHC events is that the transformer can generate events with different jet multiplicities using the same network. The one-hot-encoded jet multiplicity is provided as an additional condition for the training. The autoregressive structure can work with sequences of different length, provided there is an appropriate way of mapping the sequences onto each other. For the LHC events we enhance the sensitivity to the angular jet-jet correlations through the ordering

$$\left(\ (\phi,\eta)_{j_{1,2,3}},\ (p_T,\eta)_{\mu_1},\ (p_T,\phi,\eta)_{\mu_2},\ (p_T,m)_{j_{1,2,3}}\ \right)\ . \tag{4.62}$$

While the Bayesian transformer does learn the angular correlations also when they appear at the end of the sequence, this ordering provides a significant boost to the network's precision. For the transformer training, we want the features of the 3-jet to be well represented in the set of vectors defined in Eq. (4.62). To train on equal numbers of events with one, two, and three jets, we sample 1-jet and 2-jet events randomly at the beginning of each epoch. The loss is first evaluated separately for each jet multiplicity, and then averaged for the training update.

In Fig. 4.17 we show the standard set of kinematic observables for the autoregressive transformer based on a Gaussian mixture model, with the architecture given in Tab. A.3. Just like the two diffusion models, and the INN, it learns the different p_T-distributions with a precision close to the statistics of the training data. Sampling a variable number of jets with the multiplicity as a condition leads to no additional complication.

Looking at the correlations in the right panels, the Z-mass now comes with an increased width and a shift. This is, in part, an effect of the ordering of the input variables, where the lepton information comes after the angular information on the jets. The benefit of this ordering can be seen in the ΔR_{jj} distributions, which are reproduced at the per-cent precision without any additional effort. This is true for $\Delta R_{j_1 j_2}$ and $\Delta R_{j_1 j_3}$, reflecting the democratic ordering and training dataset. The sharp phase space boundary at $\Delta R_{jj} = 0.4$ can be trivially enforced during event generation.

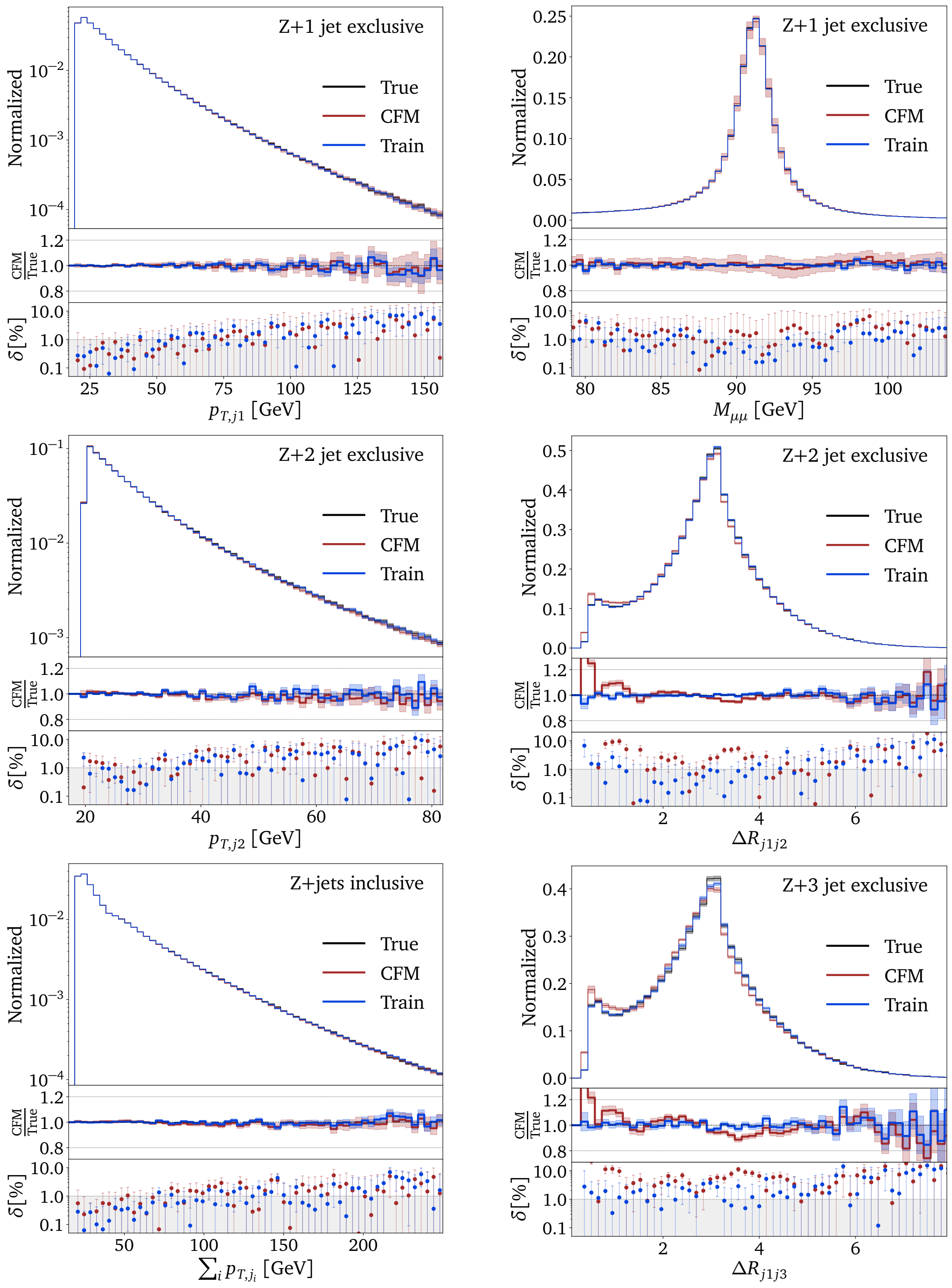


Figure 4.16: Bayesian CFM densities and uncertainties for $Z + 1$ jet (upper), $Z + 2$ jets (center), and $Z + 3$ jets (lower) from combined $Z+$ jets generation. The uncertainty on the training data is given by bin-wise Poisson statistics. The network architecture is given in Tab. A.2. For a comparison with the INN we refer to Fig. 11 of Ref. [45].

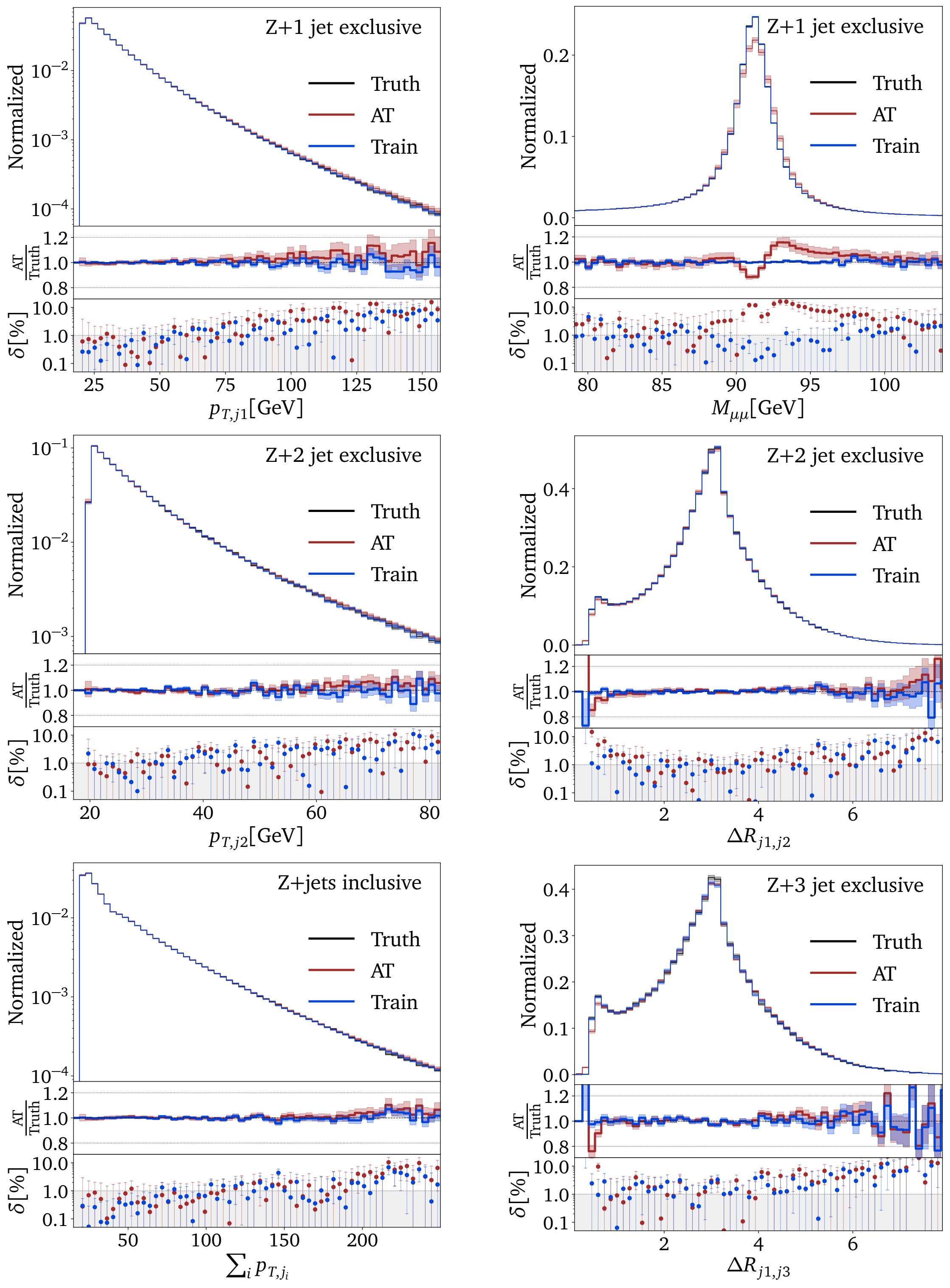


Figure 4.17: Bayesian autoregressive transformer densities and uncertainties for Z+1 jet (upper), Z+2 jets (center), and Z+3 jets (lower) from combined Z+ jets generation. The uncertainty on the training data is given by bin-wise Poisson statistics. The network architecture is given in Tab. A.3. For a comparison with the INN we refer to Fig. 11 of Ref. [45].

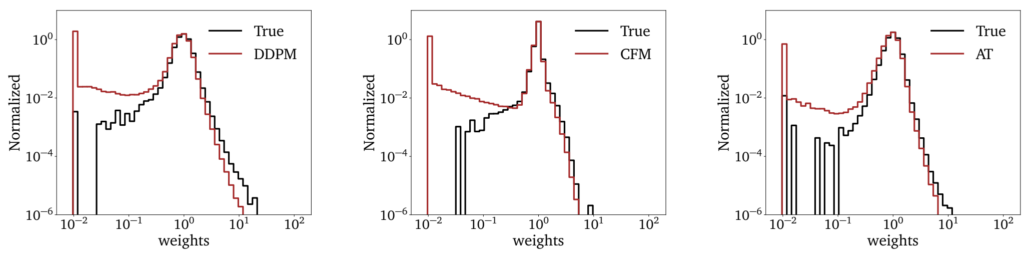


Figure 4.18: Classifier weight distribution for each of the three networks evaluated on Z+2j events.

4.4 Quantitative evaluation of generators

Following Ref. [178] we evaluate the quality of the networks by training binary classifiers to distinguish between generated and true samples. The output $C(x)$ of a well-trained classifier gives access to the likelihood ratio between the true and the model density via

$$w(x) = \frac{p_\text{true}(x)}{p_\text{model}(x)} = \frac{C(x)}{1 - C(x)} \; . \tag{4.63}$$

This is done individually for all three models, DDPM, CFM and AT, following the training procedure as discussed in [178]. The corresponding weight distributions are shown in Fig. 4.18. For all three models the weight distribution shows a similar structure. The overwhelming majority of events is clustered around weights of one, indicating a good agreement between the model and the true distribution. The weak tail to the right shows that no phase space region is systematically underpopulated. Lastly, all three models show a peak in the overflow bin to the left, indicating a clear failure mode. Low values of $w(x)$ correspond to regions where p_true is approaching zero whereas p_model is not. We checked that this excess is due to the already discussed mismodeling around the hard cut-off $\Delta R_{j1j2} = 0.4$.

4.5 Outlook

Generative neural networks are revolutionizing many aspects of our lives, and LHC physics is no exception. Driven by large datasets and precise first-principle simulations, LHC physics offers a wealth of opportunities for modern machine learning, in particular generative networks [26]. Here, classic network architectures have largely been surpassed by normalizing flows, especially its INN variant, but cutting-edge new approaches are extremely promising. Diffusion networks should provide an even better balance between expressivity and precision in the density estimation. Autoregressive transformers should improve the scaling of network size and training effort with the phase space dimensionality.

In this paper we have provided the first comprehensive study of strengths and weaknesses of these new architectures for an established LHC task. We have chosen two fundamentally different approaches to diffusion networks, where the DDPM learns the time evolution in terms of discrete steps, while the CFM encodes the continuous time evolution into in a differential equation. The autoregressive JetGPT transformer follows the standard

GPT architecture, where for our relatively simple setup we get away without actual pretraining.

For each architecture we have first implemented a Bayesian network version, which allows us to understand the different ways they approach the density estimation. While the diffusion networks first identify classes of functions and then adapt them to the correlations in phase space, much like the INN [161], the transformer learns patterns patch-wise and dimension by dimension.

Next, we have applied all three networks to the generation of Z+jets events, with a focus on the conditional setup for variable jet multiplicities and the precision in the density estimation [45]. The most challenging phase space correlations are the narrow Z-peak and the angular jet–jet separation combined with a collinear enhancement.

Our two diffusion models are, conceptually, not very far from the INNs. We have found that they face the same difficulties, especially in describing the collinear jet–jet correlation. Just like for the INN, the so-called magic transformation [45] solved this problem. Both diffusion networks provided an excellent balance between expressivity and precision, at least on part with advanced INNs. This included the density estimation as well as the uncertainty map over phase space. The main advantage of the CFM over the DDPM was a significantly faster sampling for our current implementation, at the level of the INN or the transformer. In contrast, the DDPM model is based on a proper likelihood loss, with all its conceptual and practical advantages for instance when bayesianizing it. Both networks required long training, but fewer network parameters than then INN. We emphasize that ML-research on diffusion models it far from done, so all differences between the two models found in this paper should be considered with a grain of salt.

Finally, we have adapted the fundamentally different GPT architecture to LHC events. Its autoregressive setup provided a different balance between learning correlations and scaling with the phase space dimension, and it has never been confronted with the precision requirements of the LHC. The variable numbers of particles in the final state was implemented naturally and without an additional global conditioning. Our transformer is based on a Gaussian mixture model for the phase space coverage, and we have used the freedom of ordering phase space dimensions in the conditioning chain to emphasize the most challenging correlations. This has allowed the transformer to learn the jet–jet correlations better than the INN or the diffusion models, but at the expense of the description of the Z-peak. The generation time of the transformer is comparable with the fast INN.

Altogether, we have found that new generative network architectures have the potential to outperform even advanced normalizing flows and INNs. However, diffusion models and autoregressive transformers come with their distinct set of advantages and challenges. Given the result of our study we expect significant progress in generative network applications for the LHC, whenever the LHC requirements in precision, expressivity, and speed can be matched by one of the new architectures.

Chapter 5

Precision-Machine Learning for the Matrix Element Method

The research presented in this chapter is based on work in collaboration with Theo Heimel, Ramon Winterhalder, Tilman Plehn and Anja Butter, and has been previously published in Ref. [2]. The content is similar or identical to the content of this article.

Optimal analyses are the key challenge for the current and future LHC program, including specific model-based as well simulation-based search strategies. A classic method is the matrix element method (MEM), developed for the top physics program at the Tevatron [53, 54]. It derives its optimality from the Neyman-Pearson lemma and the fact that all information for a given hypothesis is encoded in the differential cross section. In the MEM, we compute likelihood ratios for individual events, such that the log-likelihood ratio of an event sample is the sum of the event-wise log-likelihood ratios. A combination of events to a kinematic distribution is not necessary [179].

The MEM was first used in the top mass measurement [180–183] and the discovery of the single-top production process [184] at the Tevatron. At the LHC, there exist several studies [185–191] and analysis applications [190, 192–195]. The critical challenge to MEM analyses is the integration over all possible parton-level configurations which could lead to the analyzed observed events. It can be solved by using modern machine learning (ML) for a fast and efficient combination of simulation and integration [55, 56]. A related ML approach to likelihood extraction is the classifier-based estimation of likelihood ratios [125].

We present a comprehensive simulation and integration framework for the MEM, based on modern machine learning [23, 26]. It makes extensive use of generative networks, which are transforming LHC simulations and analyses just like any other part of our lives. This starts with phase-space integration and sampling [82, 134–138] and continues with more LHC-specific tasks like event subtraction [141], event unweighting [142, 143], loop integrations [196], or super-resolution enhancement [144, 145]. At the LHC, generative networks generally work in interpretable physics phase spaces, for example scattering events [1, 40–45], parton showers [105, 147, 149–151, 197–199], and detector simulations [90, 91, 93–104, 106, 200–208]. These networks can be trained on first-principle simulations and are easy to handle, efficient to ship, powerful in amplifying the training samples [152, 153], and — most importantly — precise [45, 163, 178, 209]. Conditional versions of these established generative networks then enable new analysis methods, like probabilistic unfolding [61–65, 68, 210–212], inference [56, 154], or anomaly detection [213–218].

We introduce a new MEM-ML-analysis framework in Sec. 5.1. It combines two generative network and one classifier network and pushes the precision beyond our conceptual study [56], towards an experimentally required level. For a fast and bi-directional evaluation we use the established cINNs with advanced coupling layers [45], updated to current precision requirements in Sec. 5.2. In Sec. 5.3, we add a learned acceptance network. In Sec. 5.4, we show how a generative diffusion network [1] improves the precision, albeit at the expense of speed. Finally, we employ a transformer architecture [1, 170, 219] to solve the jet combinatorics in Sec. 5.5. This series of improvements allows us to extract likelihood distributions from small and moderate-size event samples without a network bias and with close-to-optimal performance.

Reference process

The focus of this paper is entirely on our new ML-method to enable MEM analyses at the LHC. However, we use an established, challenging, and realistic physics process to illustrate our method. This reference process is introduced and discussed in Ref. [56]. We target the purely hadronic signature

$$pp \to tHj \to (bjj)\,(\gamma\gamma)\,j + \text{jets}\;, \tag{5.1}$$

with up to four additional jets from QCD radiation. The production process allows for a measurement of a CP-phase in the top Yukawa coupling at future LHC runs [220–228]. The challenge of having to work with small event numbers is motivated by choosing the rare decay channel $H \to \gamma\gamma$, which allows us to control continuum backgrounds efficiently. The total cross section is 43.6 fb, when we combine top and anti-top production.

To probe the symmetry structure of the Yukawa coupling, we introduce a mixed CP-even and CP-odd interaction [229],

$$\mathscr{L}_{t\bar{t}H} = -\frac{y_t}{\sqrt{2}}\Big[a\cos\alpha\,\bar{t}t + ib\sin\alpha\,\bar{t}\gamma_5 t\Big]H\;. \tag{5.2}$$

Choosing $a = 1$ and $b = 2/3$ [230] keeps the cross section for $gg \to H$ constant. The model parameter we target with the matrix element method is the CP-angle α. For more details on this reference process we refer to our conceptual study [56]. Obviously, all our findings can be generalized to other LHC processes.

5.1 ML-matrix element method

The matrix element method is a simulation-based inference method which uses the fact that for a given parameter of interest, α, the likelihood can be extracted from a simulation of the differential cross section. It describes the hard scattering process and factorizes into the total cross section and a normalized probability density,

$$\frac{d\sigma(\alpha)}{dx_\text{hard}} = \sigma(\alpha)\,p(x_\text{hard}|\alpha) \qquad \Leftrightarrow \qquad p(x_\text{hard}|\alpha) = \frac{1}{\sigma(\alpha)}\,\frac{d\sigma(\alpha)}{dx_\text{hard}}\;. \tag{5.3}$$

Given the hard process, we then simulate the parton shower, hadronization, detector effects, and the reconstruction of analysis objects, with a forward-transfer or response function r [231]. This function is assumed to be independent of the theory parameter α

$$\begin{tikzcd} x_\text{hard} \arrow[r, "r(x_\text{reco}|x_\text{hard})"] \arrow[dr, "p_\text{reject}(x_\text{hard})"'] & x_\text{reco} \\ & \text{rejected} \end{tikzcd} \quad . \tag{5.4}$$

The detector geometry and acceptance cuts will lead to, either, a valid reco-level event x_reco or a rejected event, introducing $p_\text{reject}(x_\text{hard})$ as the probability that a given hard event x_hard is rejected. The transfer function r is not normalized, and a proper normalization condition defines the efficiency or acceptance function,

$$\epsilon(x_\text{hard}) := \int dx_\text{reco}\; r(x_\text{reco}|x_\text{hard}) = 1 - p_\text{reject}(x_\text{hard}) \; . \tag{5.5}$$

Using the transfer function we can parametrize the forward evolution of the differential cross section following

$$\frac{d\sigma_\text{fid}(\alpha)}{dx_\text{reco}} = \int dx_\text{hard}\; r(x_\text{reco}|x_\text{hard})\, \frac{d\sigma(\alpha)}{dx_\text{hard}} \; , \tag{5.6}$$

where the subscript 'fid' indicates that the reco-level phase space is different from the parton level. In this relation we can use Eq.(5.5) to replace r with a normalized transfer probability $p(x_\text{reco}|x_\text{hard})$,

$$r(x_\text{reco}|x_\text{hard}) = \epsilon(x_\text{hard})\; p(x_\text{reco}|x_\text{hard}) \qquad \text{with} \qquad \int dx_\text{reco}\; p(x_\text{reco}|x_\text{hard}) = 1 \; . \tag{5.7}$$

Inserting Eq.(5.7) in Eq.(5.6) we obtain the final expression for the differential cross section

$$\frac{d\sigma_\text{fid}(\alpha)}{dx_\text{reco}} = \int dx_\text{hard}\; \epsilon(x_\text{hard})\; p(x_\text{reco}|x_\text{hard})\, \frac{d\sigma(\alpha)}{dx_\text{hard}} \; , \tag{5.8}$$

Equivalent to Eq.(5.3) we can now define the likelihood for reco-level events in terms of the fiducial cross section and the differential cross section

$$\frac{d\sigma_\text{fid}(\alpha)}{dx_\text{reco}} = \sigma_\text{fid}(\alpha)\; p(x_\text{reco}|\alpha) \qquad \Leftrightarrow \qquad p(x_\text{reco}|\alpha) = \frac{1}{\sigma_\text{fid}(\alpha)} \frac{d\sigma_\text{fid}(\alpha)}{dx_\text{reco}} \; . \tag{5.9}$$

To obtain the fiducial cross section $\sigma_\text{fid}(\alpha)$, we now need to integrate Eq.(5.8) over the reco-level phase space

$$\begin{aligned} \sigma_\text{fid}(\alpha) &= \int dx_\text{reco} \int dx_\text{hard}\; \epsilon(x_\text{hard})\; p(x_\text{reco}|x_\text{hard})\, \frac{d\sigma(\alpha)}{dx_\text{hard}} \\ &= \int dx_\text{hard}\; \epsilon(x_\text{hard})\, \frac{d\sigma(\alpha)}{dx_\text{hard}} \\ &= \sigma(\alpha) \int dx_\text{hard}\; \epsilon(x_\text{hard})\; p(x_\text{hard}|\alpha) \\ &= \sigma(\alpha)\; \big\langle \epsilon(x_\text{hard}) \big\rangle_{x \sim p(x_\text{hard}|\alpha)} \; , \end{aligned} \tag{5.10}$$

where we first use Eq.(5.7) to integrate out the reco-level phase space and then replace the differential cross section using Eq.(5.3). This allows us to express the integral in terms of the average acceptance $\langle \epsilon \rangle_\alpha$ which is used to evaluate the integral numerically.

Using Eq. (5.8) in Eq. (5.9) we obtain the final expression for the reco-level likelihood

$$p(x_\text{reco}|\alpha) = \frac{1}{\sigma_\text{fid}(\alpha)} \int dx_\text{hard} \, \frac{d\sigma(\alpha)}{dx_\text{hard}} \, \epsilon(x_\text{hard}) \, p(x_\text{reco}|x_\text{hard}) \; . \tag{5.11}$$

Note that in our training dataset, consisting of simulated event pairs $(x_\text{reco}, x_\text{hard})$, the hard-scattering momenta are not distributed according to Eq.(5.3), because it does not contain events x_hard that have been rejected. Consequently, the accepted x_hard are distributed as

$$p_\text{fid}(x_\text{hard}|\alpha) = \frac{1}{\sigma_\text{fid}(\alpha)} \, \frac{d\sigma(\alpha)}{dx_\text{hard}} \, \epsilon(x_\text{hard}) \; . \tag{5.12}$$

This means, we can directly relate the reco-level likelihood to a modified parton-level likelihood

$$p(x_\text{reco}|\alpha) = \int dx_\text{hard} \, p(x_\text{reco}|x_\text{hard}) \, p_\text{fid}(x_\text{hard}|\alpha) \; , \tag{5.13}$$

which connects the MEM with the completeness relation from statistics.

Acceptance classifier and transfer network

To compute the reco-level likelihood defined in Eq.(5.11) we rely on $\epsilon(x_\text{hard})$ and $p(x_\text{reco}|x_\text{hard})$, defined through a forward simulation. We encode both functions in neural networks trained on these forward simulations.

First, the acceptance $\epsilon(x_\text{hard})$ can be encoded as a standard classifier network

$$x_\text{hard} \xrightarrow{\text{Acceptance network}} \epsilon_\psi(x_\text{hard}) \; , \tag{5.14}$$

where ψ denotes the trainable network parameters. Given the input x_hard it learns the labels 1 for accepted events and 0 otherwise. Because the network is a classifier with a cross entropy loss, its output will be the acceptance probability for the given event.

The transfer probability introduced in Eq.(5.7) is encoded in a generative network with density estimation capability, like a normalizing flow or diffusion model, and is trained on event pairs $(x_\text{reco}, x_\text{hard})$. For this training dataset, we only include accepted events. The generative network defines a bijective mapping between Gaussian random numbers and reco-level phase space conditioned on parton-level events,

$$x_\text{reco} \sim p_\theta(x_\text{reco}|x_\text{hard}) \quad \xleftrightarrow{\text{Transfer network}} \quad r \sim p_\text{latent}(r) \; , \tag{5.15}$$

with trainable parameters θ. This mapping can than be used for density estimation in the forward direction and for conditional generation of reco-level events in the inverse direction.

Sampling-cINN

The integration in Eq.(5.13) is challenging, because the differential cross section spans several orders of magnitude, and the transfer probability typically forms a narrow peak.

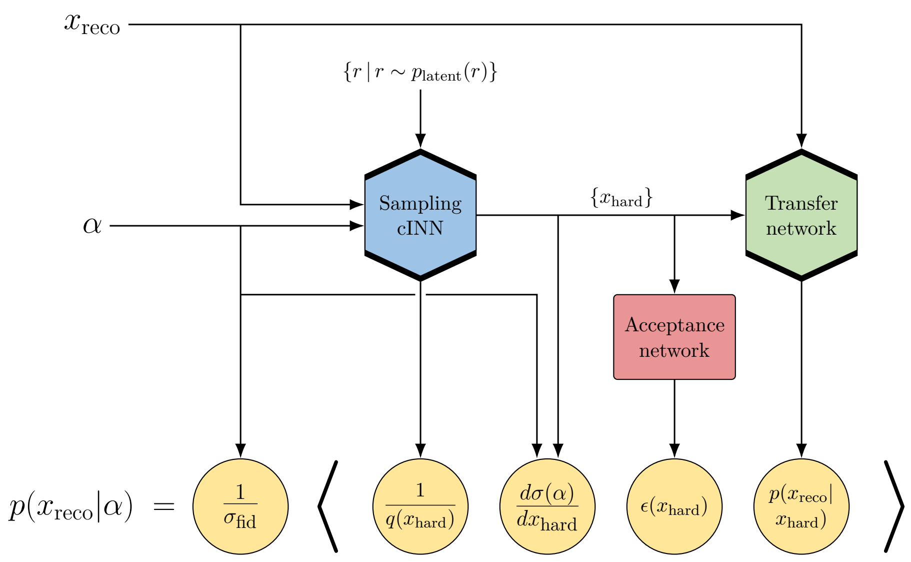

x_reco
$\{r\,|\,r \sim p_\text{latent}(r)\}$
α
Sampling cINN
$\{x_\text{hard}\}$
Transfer network
Acceptance network
$p(x_\text{reco}|\alpha)$ =
$\frac{1}{\sigma_\text{fid}}$
$\frac{1}{q(x_\text{hard})}$
$\frac{d\sigma(\alpha)}{dx_\text{hard}}$
$\epsilon(x_\text{hard})$
$p(x_\text{reco}|x_\text{hard})$

Figure 5.1: Three-network MEM integrator evaluating Eq.(5.23) 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.

We solve the integral using Monte Carlo integration sampling $x_\text{hard} \sim q(x_\text{hard}|x_\text{reco}, \alpha) \equiv q(x_\text{hard})$,

$$\begin{aligned} p(x_\text{reco}|\alpha) &= \int dx_\text{hard}\; p_\text{fid}(x_\text{hard}|\alpha)\; p_\theta(x_\text{reco}|x_\text{hard}) \\ &= \left\langle \frac{1}{q(x_\text{hard})}\; p_\text{fid}(x_\text{hard}|\alpha)\; p_\theta(x_\text{reco}|x_\text{hard}) \right\rangle_{x_\text{hard}\sim q(x_\text{hard})} , \end{aligned} \tag{5.16}$$

Ideally, this assumes

$$p_\theta(x_\text{reco}|x_\text{hard}) = p(x_\text{reco}|x_\text{hard}) \,, \tag{5.17}$$

in which case we can use Bayes' theorem to arrive at

$$\begin{aligned} p(x_\text{reco}|\alpha) &= \left\langle \frac{1}{q(x_\text{hard})}\; p_\text{fid}(x_\text{hard}|\alpha)\; p(x_\text{reco}|x_\text{hard}) \right\rangle_{x_\text{hard}\sim q(x_\text{hard})} \\ &= \left\langle \frac{1}{q(x_\text{hard})}\; p(x_\text{hard}|x_\text{reco}, \alpha) p(x_\text{reco}|\alpha) \right\rangle_{x_\text{hard}\sim q(x_\text{hard})} . \end{aligned} \tag{5.18}$$

For this integral the variance vanishes when

$$q(x_\text{hard}) \equiv q(x_\text{hard}|x_\text{reco}, \alpha) \propto p(x_\text{hard}|x_\text{reco}, \alpha) \,, \tag{5.19}$$

where $p(x_\text{hard}|x_\text{reco}, \alpha)$ corresponds to the generative unfolding probability from reco-level to parton-level [64]. However, in practice, we cannot expect the learned transfer probability to match its truth counterpart perfectly. In that case the condition in Eq.(5.19) becomes

$$q(x_\text{hard}|x_\text{reco},\alpha) \propto p_\text{fid}(x_\text{hard}|\alpha)\; p_\theta(x_\text{reco}|x_\text{hard}) \; . \tag{5.20}$$

In both cases, we train a second conditional normalizing flow with trainable parameters φ to encode this optimal transformation of the integration variables,

$$r \sim p_\text{latent}(r) \quad \xleftrightarrow{\text{Sampling-cINN}} \quad x_\text{hard}(r) \sim q_\varphi(x_\text{hard}|x_\text{reco},\alpha) \; , \tag{5.21}$$

which allows to parameterize the conditional sampling density as

$$\begin{aligned} q_\varphi(x_\text{hard}|x_\text{reco},\alpha) \equiv q_\varphi(x_\text{hard}(r)|x_\text{reco},\alpha) &= \frac{p_\text{latent}(r)}{J_\varphi(r)} \\ \text{with} \qquad J_\varphi(r) &= \left| \frac{\partial x_\text{hard}(r;x_\text{reco},\alpha;\varphi)}{\partial r} \right| \; . \end{aligned} \tag{5.22}$$

The MEM integral in Eq.(5.11) now reads

$$\begin{aligned} p(x_\text{reco}|\alpha) &= \frac{1}{\sigma_\text{fid}(\alpha)} \int dr \; J_\varphi(r) \; \left[\frac{d\sigma(\alpha)}{dx_\text{hard}} \epsilon_\psi(x_\text{hard})\, p_\theta(x_\text{reco}|x_\text{hard}) \right]_{x_\text{hard}(r;x_\text{reco},\alpha;\varphi)} \\ &= \frac{1}{\sigma_\text{fid}(\alpha)} \left\langle \frac{J_\varphi(r)}{p_\text{latent}(r)} \; \left[\frac{d\sigma(\alpha)}{dx_\text{hard}} \epsilon_\psi(x_\text{hard})\, p_\theta(x_\text{reco}|x_\text{hard}) \right]_{x_\text{hard}(r;x_\text{reco},\alpha;\varphi)} \right\rangle_{r\sim p(r)} \; . \end{aligned} \tag{5.23}$$

The architecture of our MEM integrator is illustrated in Fig. 5.1.

5.2 Two-network baseline

In the proof-of-concept implementation of Ref. [56] we used a series of ad-hoc fixes to stabilize the critical phase space integration in Eq.(5.11). Before we present more substantial improvements to our framework, we introduce a series of numerical improvements to our baseline two-cINN setup. For the two-network setup we assume that we can neglect the phase-space dependence of the acceptance in the MEM integration,

$$p(x_\text{reco}|\alpha) \approx \frac{1}{\sigma_\text{fid}(\alpha)} \int dx_\text{hard} \; \frac{d\sigma(\alpha)}{dx_\text{hard}} \; p_\theta(x_\text{reco}|x_\text{hard}) \; . \tag{5.24}$$

Single-pass integration over model parameters

Initially, we integrate over the phase space for each theory parameter value separately. This general approach does not make use of the fact that the detector response does not depend on α, and the mapping for the importance sampling only has a small α-dependence. The phase space samples $x_\text{hard} \sim q_\varphi(x_\text{hard}|x_\text{reco},\alpha)$ and the corresponding values of $p_\theta(x_\text{reco}|x_\text{hard})$ can be used to evaluate the differential cross section for multiple points in α. Moreover, parts of the cross section calculation only depend on the phase space point and not on α, like for example parton densities.

Consequently, we can understand the integrand for a given Monte Carlo sample as a smooth function of α, so the integral will also be a smooth function of α. This means we do not have to fit an explicit function to the likelihood values and instead extract a smooth log-likelihood as a function of α. The MEM integration for a given x_reco and a discrete set $\{\alpha\}$ can be performed as:

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.(5.24) for all α values at the same time

$$p(x_\text{reco}|\alpha) \approx \frac{1}{\sigma_\text{fid}(\alpha)} \frac{1}{N} \sum_{j=1}^{N} \frac{1}{q_\varphi(x_\text{hard}^{(j)}|x_\text{reco}, \alpha^{(j)})} \frac{d\sigma(x_\text{hard}^{(j)}|\alpha)}{dx_\text{hard}} p_\theta(x_\text{reco}|x_\text{hard}^{(j)}) \tag{5.25}$$

This integral converges quickly for some events, while more statistics are needed for others. One reason is that the peaks of the transfer probability and the importance sampling distribution are not perfectly aligned for some events, resulting in a higher variance. To reduce the integration time while guaranteeing a small integration error, we compute the integral iteratively. We specify the number of samples per iteration as well as a minimal and maximal number of iterations. Furthermore, we specify a threshold for the maximum relative uncertainty over the results for all values of α. The integration is repeated for new batches of samples until the combined uncertainty drops below the threshold. In practice, a batch size of 10000, at least two and at most 15 iterations meet a target uncertainty of 2%. The uncertainty on the normalized negative log-likelihood will be much smaller than these 2% because of the correlation between different α.

Integration uncertainties

Using this single-pass integration, the results for different α values become correlated, because the new algorithm ensures that the result is a smooth function of α. This means that the MC integration error cannot be easily estimated point-wise. The uncertainty on the likelihood ratio should be much smaller than the uncertainty of the absolute value of the likelihood before normalization. To account for the correlations, we use bootstrapping to resample the integrand multiple times and propagate the resulting replicas through the downstream tasks. For this bootstrapping we take our samples of the integrand $I^{(j)}(\alpha_i)$ and randomly draw M batches of N samples from $\{I^{(j)}(\alpha_i) \mid j \in \{1, \dots, N\}\}$ with replacement. We compute the mean over the N samples per batch, defining M replicas of the integral as a function of α. They can be used to estimate uncertainties on the following normalized negative log-likelihoods.

Next, we can quantify the uncertainty from the training of the transfer probability using a Bayesian network [155–161]. To estimate the training uncertainty we perform the phase space integration for different samples from the distribution over the trainable parameters. In Ref. [56] this is done by repeating the integration for different sampled networks. However, the idea of the single-pass integration also applies to the Bayesian transfer probabilities. The same importance sampling distribution should work well for different sampled networks, making the integration more efficient. The training uncertainty estimation can be combined with the bootstrapping procedure described above. For each replica, we do not only resample the integrand but also compute the transfer probability for a different sample from the distribution over the trainable parameters.

Factorization of differential cross section

For our example process, single-top plus Higgs production with an anomalous CP-phase, the Lagrangian given in Eq.(5.2) can be written as

$$\mathscr{L} = \mathscr{L}_1 + \sin\alpha\, \mathscr{L}_2 + \cos\alpha\, \mathscr{L}_3 \;, \tag{5.26}$$

and the squared matrix element has the corresponding form

$$\frac{d\sigma(x_\text{hard}|\alpha)}{dx_\text{hard}} = g_1 + \sin\alpha\, g_2 + \cos\alpha\, g_3 + \sin\alpha\cos\alpha\, g_4 + \sin^2\alpha\, g_5 \;, \tag{5.27}$$

with phase space dependent $g_i(x_\text{hard})$. This is an example where the matrix element factorizes into an x_hard-dependent and an α-dependent part. Similar factorization properties hold for SMEFT corrections where it is often referred to as operator morphing [232]. For

$$\frac{d\sigma(x_\text{hard}|\alpha)}{dx_\text{hard}} = \sum_i f_i(\alpha) g_i(x_\text{hard}) \tag{5.28}$$

the MEM integration in Eq.(5.24) becomes

$$p(x_\text{reco}|\alpha) = \frac{1}{\sigma_\text{fid}(\alpha)} \sum_i f_i(\alpha) \int dx_\text{hard}\, g_i(x_\text{hard})\, p_\theta(x_\text{reco}|x_\text{hard}) \;. \tag{5.29}$$

The same can be done for the Monte Carlo estimate of the integral,

$$p(x_\text{reco}|\alpha) \approx \frac{1}{\sigma_\text{fid}(\alpha)} \frac{1}{N} \sum_{j=1}^N \frac{1}{q_\varphi(x_\text{hard}^{(j)}|x_\text{reco}, \alpha^{(j)})} \frac{d\sigma(x_\text{hard}^{(j)}|\alpha)}{dx_\text{hard}} p_\theta(x_\text{reco}|x_\text{hard}^{(j)}) \tag{5.30}$$

$$= \frac{1}{\sigma_\text{fid}(\alpha)} \sum_i f_i(\alpha) \frac{1}{N} \sum_{j=1}^N \frac{1}{q_\varphi(x_\text{hard}^{(j)}|x_\text{reco}, \alpha^{(j)})}\, g_i(x_\text{hard}^{(j)})\, p_\theta(x_\text{reco}|x_\text{hard}^{(j)}) \;, \tag{5.31}$$

where $x_\text{hard}^{(j)} \sim q_\varphi(x_\text{hard}|x_\text{reco}, \alpha^{(j)})$. The exact functional form of the integral is only preserved if the same $x_\text{hard}^{(j)}$ are used for all values of α.

Importance sampling trained on transfer probability

The training of the Sampling-cINN assumes that the transfer network encodes $p(x_\text{reco}|x_\text{hard})$ perfectly. The Sampling-cINN is then used for importance sampling. From that perspective, it is less important to learn the truth distribution

$$q_\varphi(x_\text{hard}|x_\text{reco}, \alpha) \approx p(x_\text{hard}|x_\text{reco}, \alpha) \propto p(x_\text{reco}|x_\text{hard}) p_\text{fid}(x_\text{hard}|\alpha) \;. \tag{5.32}$$

than the modeled distribution

$$q_\varphi(x_\text{hard}|x_\text{reco}, \alpha) \approx p_\theta(x_\text{reco}|x_\text{hard}) p_\text{fid}(x_\text{hard}|\alpha) \;. \tag{5.33}$$

The training data, consisting of tuples $(\alpha, x_\text{hard}, x_\text{reco})$ should then be modified by replacing the reco-level momentum with the generated $\tilde{x}_\text{reco} \sim p_\theta(x_\text{reco}|x_\text{hard})$. To increase the training statistics we re-sample the reco-level momenta at the beginning of each epoch. Because of the sharply peaked form of the transfer probability, even small deviations from the truth that do not have a significant impact on the inference performance, can lead to a significant misalignment with the importance sampling distribution. Hence, training the importance sampling on the learned transfer probability leads to a significantly better variance of the integrations weights and a faster convergence of the integral.

Vegas latent space refinement

Even when the Sampling-cINN is trained on the learned transfer probability, some events lead to a large variance in the MEM integration. This can be solved by further adapting the proposal distribution during the integration. Specializing the importance sampling network for such an event is impracticable. An alternative is to refine the INN latent space using VEGAS. Instead of directly sampling random numbers and mapping them to phase space, we transform them with a VEGAS grid first. Note, that the grid is shared for all α because of the small α dependence of the importance sampling. After each iteration of the integration, this grid is adapted to reduce the variance of the integral. Because we need to pass the integrand value back to VEGAS, we choose a value in the middle of the relevant α-interval being evaluated. The results from the different iterations of the integrals are combined by weighting them by the inverse variance to reduce the overall variance and especially the effect of early iterations where the grid is not yet well adapted.

Figure 5.2 illustrates the effect of training the Sampling-cINN on the transfer probability and using VEGAS refinement for the MEM integration performance with 1000 SM events and networks with a similar architecture and hyperparameters as in Ref. [56]. For our baseline, we use single-pass integration including a factorized differential cross section. While this guarantees smooth likelihood curves as a function of α, we find that the integration uncertainty does not meet the target precision of 2% within 15 iteration for most events. Running the integration with VEGAS refinement improves the convergence, and the importance sampling trained on the transfer probability leads to a even larger improvements. The combination of both methods ensures that the target precision is reached within 15 iterations for most events. This shows that the Sampling-cINN, trained

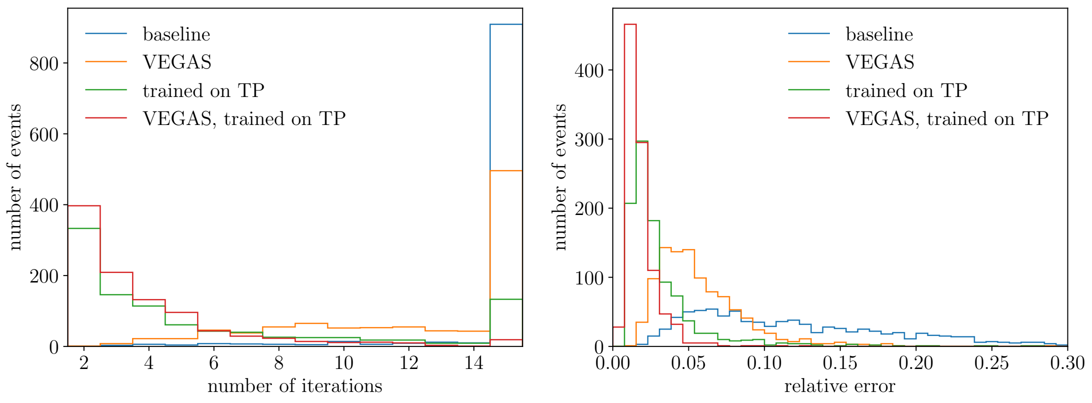


Figure 5.2: Integration performance with and without importance sampling trained on the transfer probability and VEGAS refinement. Left: number of iterations (10000 samples each) to reach the 2% target precision, with 2 to 15 iterations. Right: relative integration error after 10 iterations of 10000 samples each.

on the transfer function and with Vegas refinement, appears to be sufficiently precise to ensure fast convergence of the phase space integral.

Two-network cINN benchmark

The purple line in Fig. 5.3 shows the extracted log-likelihoods for our example process, using all improvements described in this section, and similar architecture and hyperparameters as in Ref. [56]. In the top two rows we show the extracted likelihoods from a small set of 400 events and from a large set of 10k events. In both cases, we compare the likelihood extracted from the reconstructed events to the hard-process truth. Note that we show the integration uncertainties as error bands in the plots, but due to our low error threshold and the single-pass integration these are barely visible. By repeating the integration with the same networks, we confirm that the result is perfectly stable and consistent with these uncertainties.

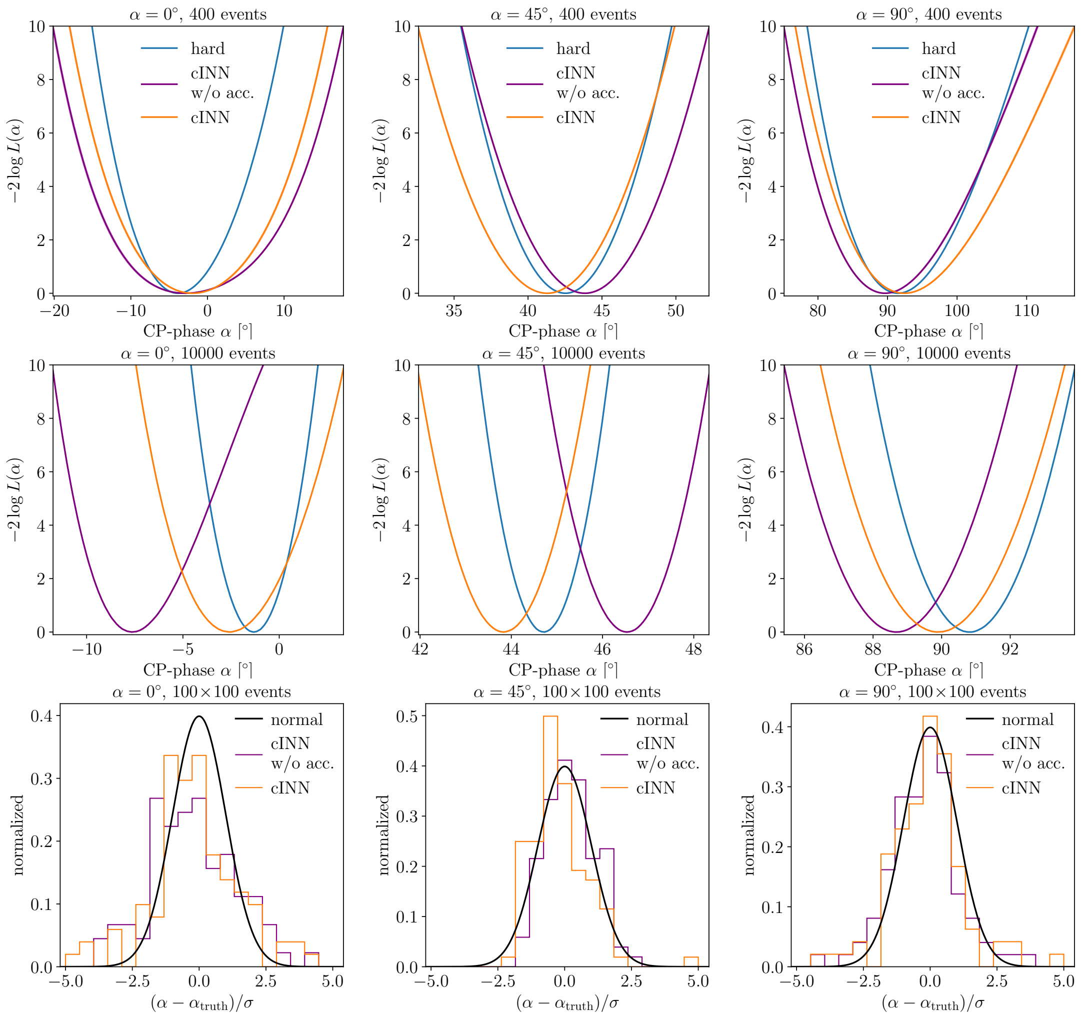


Figure 5.3: **cINN benchmark and learned acceptance:** likelihoods for different CP-angles. We use the same architecture as in Ref. [56], but with the improved integration. The purple curve shows the two-network cINN benchmark and the orange curve also includes the learned acceptance. From top to bottom: likelihoods for 400 events, 10000 events, and pulls.

Performance issues occur when we increase the number of events. The precision of the combined likelihood increases and leads to a systematic deviation between the hard-process and the reconstructed likelihoods. This is not caused by the integration, and we will target this shortcoming by improving the architecture and the training of the transfer probability.

5.3 Acceptance classifier

Moving from the two-network setup in Sec. 5.2 to the new, three-network setup introduced in Sec. 5.1, we are back to the more general form of the MEM-integral,

$$p(x_\text{reco}|\alpha) = \frac{1}{\sigma_\text{fid}(\alpha)} \int dx_\text{hard} \, \frac{d\sigma(\alpha)}{dx_\text{hard}} \, \epsilon_\psi(x_\text{hard}) \, p_\theta(x_\text{reco}|x_\text{hard}) \; . \tag{5.34}$$

The acceptance function will be encoded in a straightforward classifier network. It targets the scenario where the jet from the hard process escapes detection, i.e. $|\eta_j| > 2.4$, while the event is still accepted since a ISR jet is tagged instead.The two possible origins of the jets are taken into account by the transfer probability. An additional challenge is the significant drop in acceptance which is now remedied automatically by the introduction of the classifier to include the acceptance rate.

We train the classifier on a dataset of hard process configurations with the additional information of the acceptance label. Its output then provides $\epsilon_\psi(x_\text{hard})$ to solve Eq.(5.34). Its hyperparameters are given in Tab. A.4, and its training only takes a few minutes. The learned and true acceptances as a function of different kinematic observables are shown in Fig. 5.4. Indeed, we see a large jump in the acceptance at $|\eta_j| = 2.4$, by almost a factor three. Also for other observables, like the p_T of the top, the acceptance varies considerably over phase space.

We then evaluate the MEM integral, now including the learned acceptance. Comparing the new results (orange) with the two-network baseline (purple) in Fig. 5.3 we see a considerable improvement. For the small set of 400 events there is no bias left between the extracted likelihoods and the hard-process truth. Also for 10k events the large bias from Fig. 5.3 is reduced to a level where it is comparable to the statistical precision. Even for the challenging SM-case $\alpha = 0°$ the extracted likelihoods agrees well with the truth

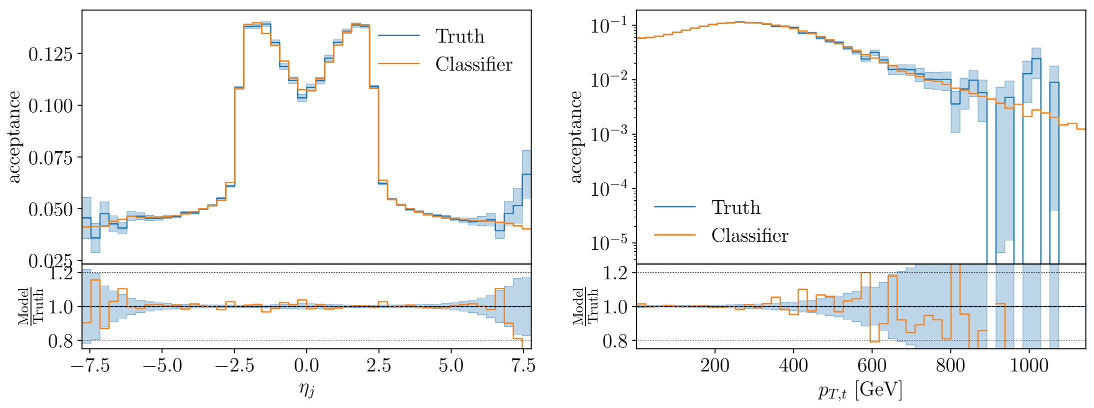


Figure 5.4: Truth (dashed line) and learned (solid line) acceptance as a function of different kinematic observables.

extracted from the hard process. The remaining question is how close we can bring the widths of the extracted likelihood-curves to the optimal outcome from the hard process, and if a remaining systematic bias can keep up with statistical improvements. From now on, we will keep the acceptance network within our MEM setup throughout the rest of our paper.

5.4 Transfer diffusion

Instead of a Transfer-cINN [56], as discussed in Sec. 5.1, we can also use other neural networks to encode the transfer probability. The great advantages of the INN are its stability, its controlled precision in estimating the density, and its speed in both directions. However, these advantages come at the prize of limited flexibility, and we can use diffusion networks to slightly shift this balance [1]. Conditional flow matching (CFM) networks [51, 164, 165] allow for more flexibility in encoding an underlying density, with the main disadvantage of a significant loss in speed in the likelihood evaluation. While this speed might become a relevant factor eventually, we compare the performance of the cINN with the CFM at face value. For a detailed introduction of conditional flow matching in the context of particle physics we refer to Ref. [1] and only repeat the key points here.

The Transfer-CFM replaces the Transfer-cINN in Eq.(5.15). The CFM models the transformation between a latent distribution $p_\text{latent}(r)$ and a conditional phase space distribution $p_\theta(x_\text{reco}|x_\text{hard})$ inspired by a a time-dependent process. The time evolution is described by an ordinary differential equation

$$\frac{dx(t)}{dt} = v(x(t),t) \; , \tag{5.35}$$

with the velocity field $v(x(t),t)$. The corresponding time-dependent probability density $p(x,t)$ obeys the continuity equation

$$\frac{\partial p(x,t)}{\partial t} + \nabla_x \left[p(x,t) v(x,t) \right] = 0 \; . \tag{5.36}$$

To obtain a generative model we need a velocity field that evolves the probability density in time such that

$$p(x,t) \to \begin{cases} p_\theta(x) \approx p_\text{data}(x) & t \to 0 \\ p_\text{latent}(x) = \mathcal{N}(x;0,1) & t \to 1 \; . \end{cases} \tag{5.37}$$

To construct this velocity field we start from a sample-conditional diffusion trajectory

$$x(t|x_0) = (1-t)x_0 + tr \to \begin{cases} x_0 & t \to 0 \\ r \sim \mathcal{N}(0,1) & t \to 1 \; , \end{cases} \tag{5.38}$$

that evolves the phase space sample x_0 towards a latent space sample. The associated sample-conditional velocity field directly follows from the ODE Eq.(5.35)

$$v(x(t|x_0), t|x_0) = \frac{d}{dt} \left[(1-t)x_0 + tr \right] = -x_0 + r \; . \tag{5.39}$$

The desired velocity field for the generative model is then given by [51]

$$v(x,t) = \int dx_0 \, \frac{v(x,t|x_0)p(x,t|x_0)p_{\text{data}}(x_0)}{p(x,t)} \,. \tag{5.40}$$

Learning the velocity field from data is a straightforward regression task and can again be reformulated in terms of the conditional velocity field [51]

$$\begin{aligned}
\mathcal{L}_{\text{FM}} &= \left\langle [v_\theta(x,t) - v(x,t)]^2 \right\rangle_{t,x\sim p(x,t)} \,, \\
&\Big\downarrow \text{ reparametrization + neglecting constants} \\
\mathcal{L}_{\text{CFM}} &= \left\langle [v_\theta(x(t|x_0),t) - v(x(t|x_0),t|x_0)]^2 \right\rangle_{t\sim U(0,1),x_0\sim p_{\text{data}},r\sim\mathcal{N}(0,1)}
\end{aligned} \tag{5.41}$$

Once the model is trained to encode the velocity it defines a bijective mapping between the latent and the phase space via numerically solving the ODE Eq.(5.35). Crucially for our application the Jacobian of this transformation is tractable through another ODE [233]

$$\frac{d \log p(x(t),t)}{dt} = -\nabla_x v(x(t),t). \tag{5.42}$$

To calculate the likelihood of a phase space sample x we map it to the latent space according to Eq.(5.35) and calculate the jacobian determinant of this transformation according to Eq.(5.42)

$$r(x) = x + \int_0^1 v_\theta(x,t)dt \quad \text{with} \quad \left|\frac{\partial r}{\partial x}\right| = \exp\left(\int_0^1 dt \nabla_x v_\theta(x(t),t)\right) \,. \tag{5.43}$$

$$\Rightarrow \qquad p(x) = p_{\text{latent}}(r(x)) \exp\left(\int_0^1 dt \nabla_x v_\theta(x(t),t)\right) \tag{5.44}$$

Solving the ODEs numerically with the required precision takes $\mathcal{O}(100)$ evaluations of the function. For the transformation ODE this is relatively fast as the function is just the velocity, i.e. the neural network. For the likelihood ODE however evaluating the function means calculating the gradients of all components of the velocity with respect to the inputs, making likelihood calculation significantly slower.

The hyperparameters of our CFM network are given in Tab. A.5. It is straightforward to replace the Transfer-cINN with a Transfer-CFM in our MEM architecture, so we can benchmark the performance gain through the increased expressivity, at the possible expense of speed.

The likelihoods extracted with the help of the Transfer-CFM are illustrated in Fig. 5.5 and can be compared to the same MEM setup, but with a Transfer-cINN in Fig. 5.3. For 400 events the difference between the Transfer-cINN and the Transfer-CFM is not visible, suggesting that both of them work extremely well given the statistical limitations and the phase space integration. There is no systematic bias, and the width of the extracted likelihoods are close to the optimal hard-process curves.

For the high-precision case with 10k events the Transfer-CFM leads to a significant improvement over the cINN architecture. Now, the picture is the same as for 400 events, where the extracted likelihoods do not show any significant bias, and the extracted likelihoods are extremely close to the optimal information.

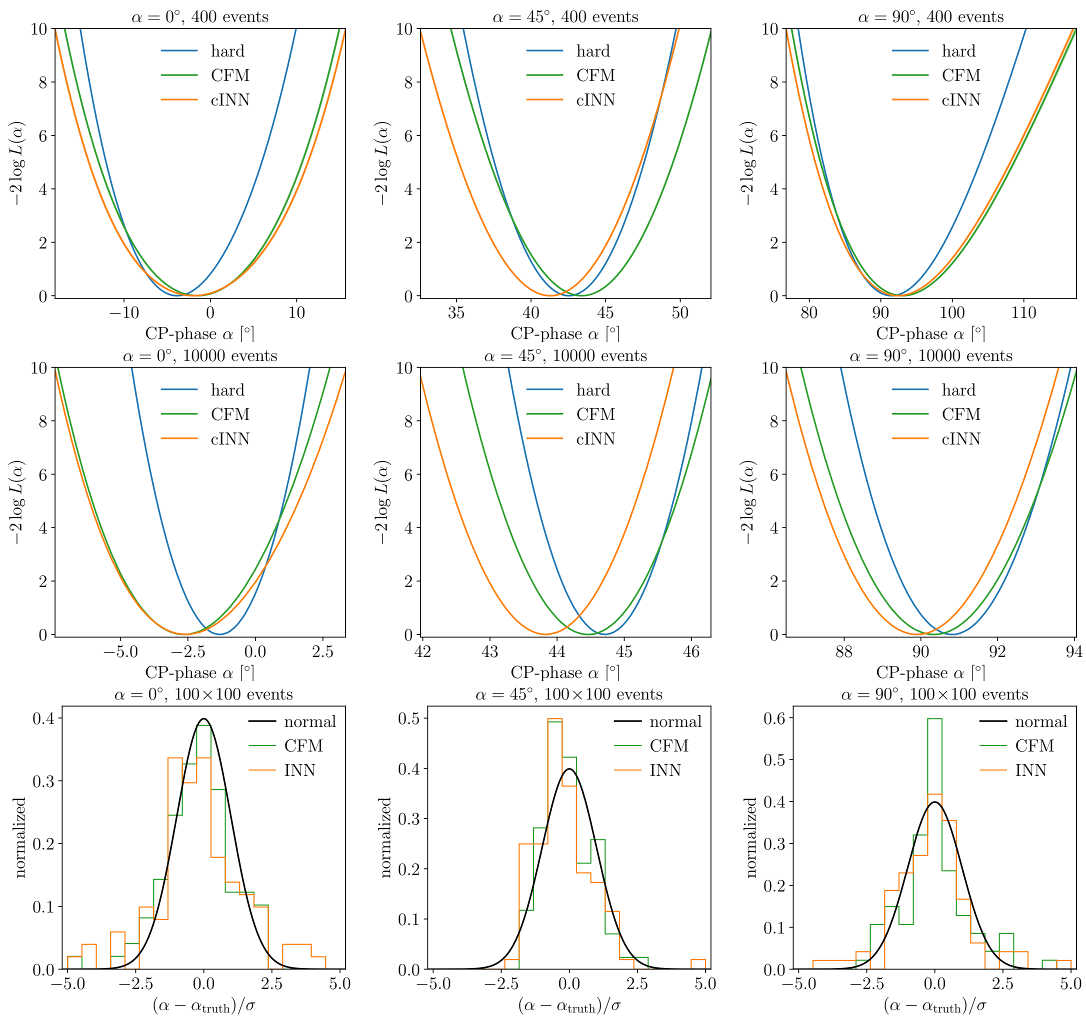


Figure 5.5: **Transfer-CFM:** likelihoods for different CP-angles. We compare the cINN baseline with a CFM diffusion network, both including the learned acceptance. From top to bottom: likelihoods for 400 events, 10000 events, and pulls.

5.5 Combinatorics transformer

In our last step, we introduce a transformer [1, 170, 219] to combine the stability and precision of the Transfer-cINN and Transfer-CFM with an appropriate treatment of jet combinatorics [234]. The structure follows the idea that the transfer probability turns a sequence of parton-level momenta into a sequence of reco-level momenta. The Transfer-Transformer, in short Transfermer, should be ideal to encode the correlations between the different particles, without relying on locality or any other physics-inspired requirement.

Transfermer

The challenge of using a transformer in our MEM setup is that it is not invertible and does not guarantee a tractable Jacobian. We can solve this problem by making the architecture autoregressive at the level of reco-level momenta and splitting it into two parts, as illustrated in the left panel of Fig. 5.6: (i) the transformer encodes the correlations between the parton-level and reco-level objects. Their cross-correlation describes the input-output combinatorics; (ii) a small and universal cINN encodes the correlations between the momentum components of a single particle, conditioned on the output of the transformer $c^{(i)}$.

To guarantee a tractable Jacobian of the full normalizing flow, we apply an autoregressive factorization of the transfer probability defined in Eq.(5.49),

$$p(x_\text{reco}|x_\text{hard}) = \prod_{i=1}^{n} p(x_\text{reco}^{(i)}|c(e_\text{reco}^{(0)},\dots,e_\text{reco}^{(i-1)},e_\text{hard})) \; . \tag{5.45}$$

The function c denotes the transformer encoding. We define a special starting token $e_\text{reco}^{(0)}$, shift the inputs by one and mask the self-attention matrix using a triangular mask to ensure that every momentum is only conditioned on the previous momenta. $e_\text{reco}^{(i)}$ and $e_\text{hard}^{(i)}$ denote the particle-wise embeddings of the momenta and their position. We define this embedding as the concatenation of the momenta and their one-hot-encoded position in the event, padded with zeros. Using a single linear layer instead of the zero-padding does not lead to any performance improvements. We then sample from the transfer probability iteratively, which requires n Transfermer evaluations,

$$p(x_\text{reco}^{(i)}|x_\text{hard}) \equiv p(x_\text{reco}^{(i)}|c(e_\text{reco}^{(0)},\dots,e_\text{reco}^{(i-1)},e_\text{hard})) \; . \tag{5.46}$$

Since all $c^{(i)}$ can be computed in a single step from the reco-level momenta, density estimation and training this model is very fast. This is also the way the Transfermer is used during the MEM integration.

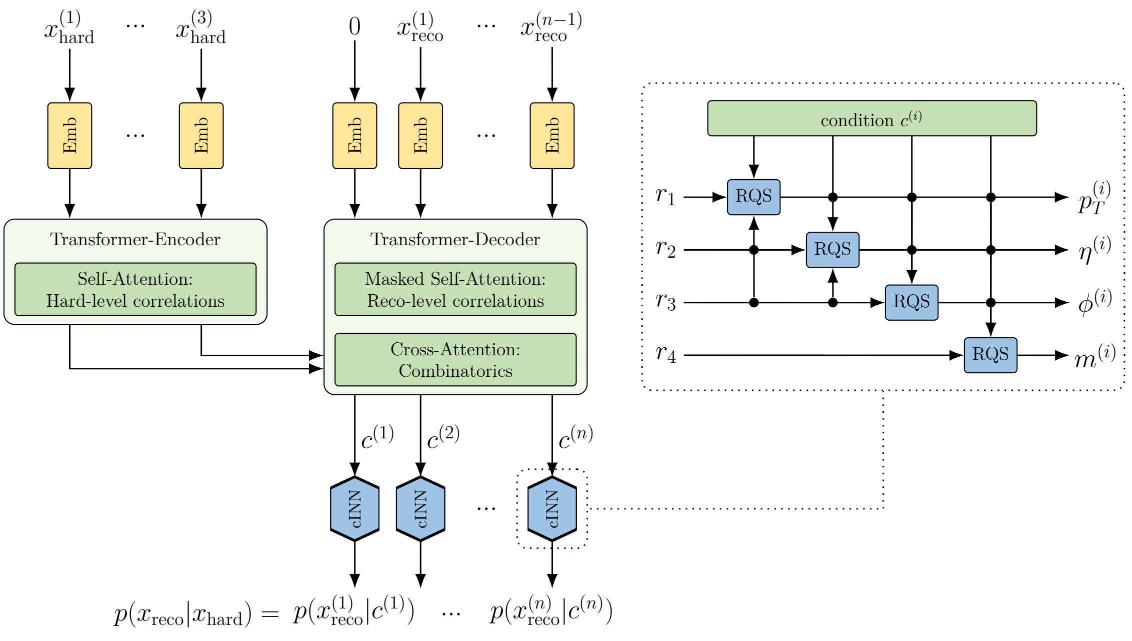

$x_\text{hard}^{(1)}$ ··· $x_\text{hard}^{(3)}$
0 $x_\text{reco}^{(1)}$ ··· $x_\text{reco}^{(n-1)}$
Emb
Transformer-Encoder
Self-Attention: Hard-level correlations
Transformer-Decoder
Masked Self-Attention: Reco-level correlations
Cross-Attention: Combinatorics
$c^{(1)}$ $c^{(2)}$ $c^{(n)}$
cINN
$p(x_\text{reco}|x_\text{hard}) = p(x_\text{reco}^{(1)}|c^{(1)}) \cdots p(x_\text{reco}^{(n)}|c^{(n)})$
condition $c^{(i)}$
r_1 r_2 r_3 r_4
RQS
$p_T^{(i)}$ $\eta^{(i)}$ $\phi^{(i)}$ $m^{(i)}$

Figure 5.6: 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.

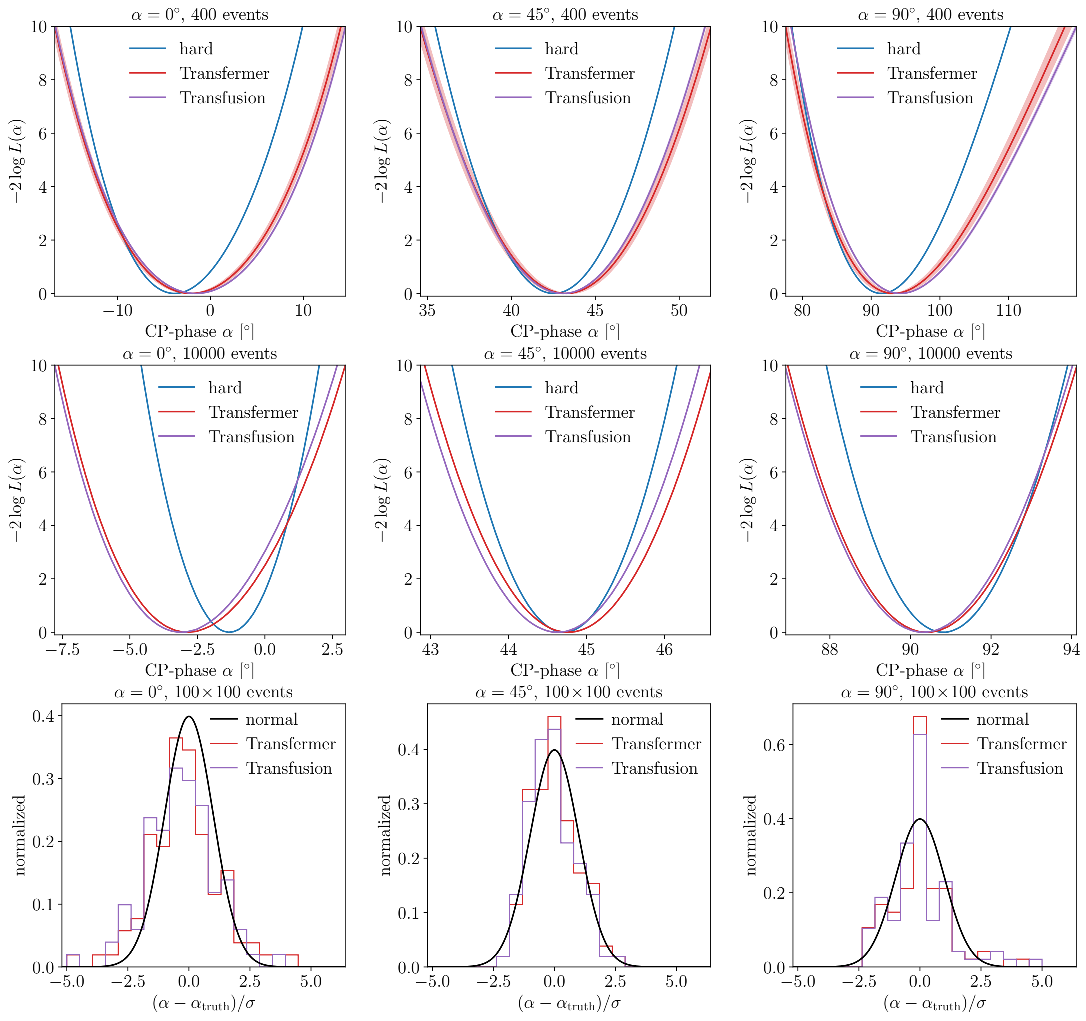


Figure 5.7: **Transfermer and Transfusion:** likelihoods for different CP-angles using a transformer for the transfer probability, combined with a cINN or a CFM network, respectively. From top to bottom: likelihoods for 400 events, 10000 events, and pulls. Only the Transfermer curve includes the training uncertainties estimated with the Bayesian network.

The transfer probability in Eq.(5.45) still has to be converted into a probability distribution for the 4-momentum components of the external particles. To encode massless and massive particles in the same cINN we factorize it into

$$p(x_\text{reco}^{(i)}|c^{(i)}) = p(p_T^{(i)}, \eta^{(i)}, \phi^{(i)}|c^{(i)}) \times p(m^{(i)}|p_T^{(i)}, \eta^{(i)}, \phi^{(i)}, c^{(i)}) \; , \tag{5.47}$$

such that the generation of the mass direction can be omitted without affecting the other three components. The corresponding cINN architecture is given in the right panel of Fig. 5.6. Rational quadratic spline coupling layers model the one-dimensional distributions. By transforming each momentum component once and conditioning it on the other components and the transformer output, using a feed-forward network, we build a minimal cINN that is able to model the correlations between the momentum components.

In practice, we use normalized versions of $\log p_T$ and $\log m$ as inputs for the network and map them to Gaussian latent spaces. Similarly, we map ϕ and η to uniform latent spaces, taking into account the detector-level η cuts. For ϕ we use periodic RQS splines [212]. The cINN for single momenta and the transformer are trained jointly by minimizing the negative log-likelihood loss $\mathcal{L} = -\log p_\theta(x_\text{reco}|x_\text{hard})$.

We implement the Transfermer with the standard PyTorch [173] transformer module and the cINN architecture described above. The hyperparameters are given in Tab. A.5. In Fig. 5.7, we show the likelihoods for the Transfermer architecture. This plot shows much larger error bands because they also include the systematic uncertainty from the Transfermer training, estimated with a Bayesian network. For the other architectures, we omit these due to runtime constraints. The likelihoods can be compared to the cINN results in Fig. 5.3, and we see that their bias and accuracy have improved. Even for 10k events, the likelihoods are largely unbiased, albeit not significantly better than for the Transfer-CFM from Fig. 5.5. The Transfermer architecture can be easily generalized to support variable numbers of reco-level jets. We show this extension in Appendix 5.7 but do not find any additional improvements for our reference process. Furthermore, we show how sensitive this architecture is to the choice of simulation tool in Appendix 5.7.

Transfusion

As a last transfer architecture we consider the CFM equivalent of the Transfermer, an autoregressive Transfusion. We keep the autoregressive structure and the masked self-attention from Fig. 5.6 and simply replace the small cINN with a small CFM network to generate the individual particle momenta. The CFM learning task is a simple regression of the velocity field. As long as we can track gradients through the network, we obtain a tractable Jacobian according to Eq.(5.42). The velocity of the i^th particle is then denoted in analogy to Eq.(5.45)

$$v^{(i)}(x_\text{reco}^{(i)}(t), t|c(e_\text{reco}^{(0)}, \dots, e_\text{reco}^{(i-1)}, e_\text{hard})) \, , \tag{5.48}$$

From the velocity field the likelihoods are again obtained by solving the ODEs Eq.(5.44), in the Transfusion setup now autoregressively for each particle. For on-shell and off-shell particles, we use two different small CFMs, one 3-dimensional and one 4-dimensional. This setup outperforms just using the same 4-dimensional network and discarding the generated masses for on-shell particles. The hyperparameters of the Transfusion network are given in Tab. A.6.

We show the MEM likelihoods obtained with the Transfusion in Fig. 5.7 and find that they are indistinguishable from the Transfermer results. This indicates that the difference between the cINN and CFM likelihoods can be attributed to cINN issues with the jet combinatorics. Outsourcing this task to the transformer significantly improves the performance. For the CFM the corresponding improvement is minimal.

5.6 Outlook

The matrix element method is an example of an LHC inference method, which is hugely attractive but only enabled by modern machine learning [56]. Specifically, it requires a fast and precise forward-transfer probability, an extremely efficient phase space mapping for the integration over the hard phase space, and a flexible encoding of the detector efficiency. We have shown, for a CP measurement in the associated production of a Higgs and a single top, how each of these tasks can be assigned to a neural network. This

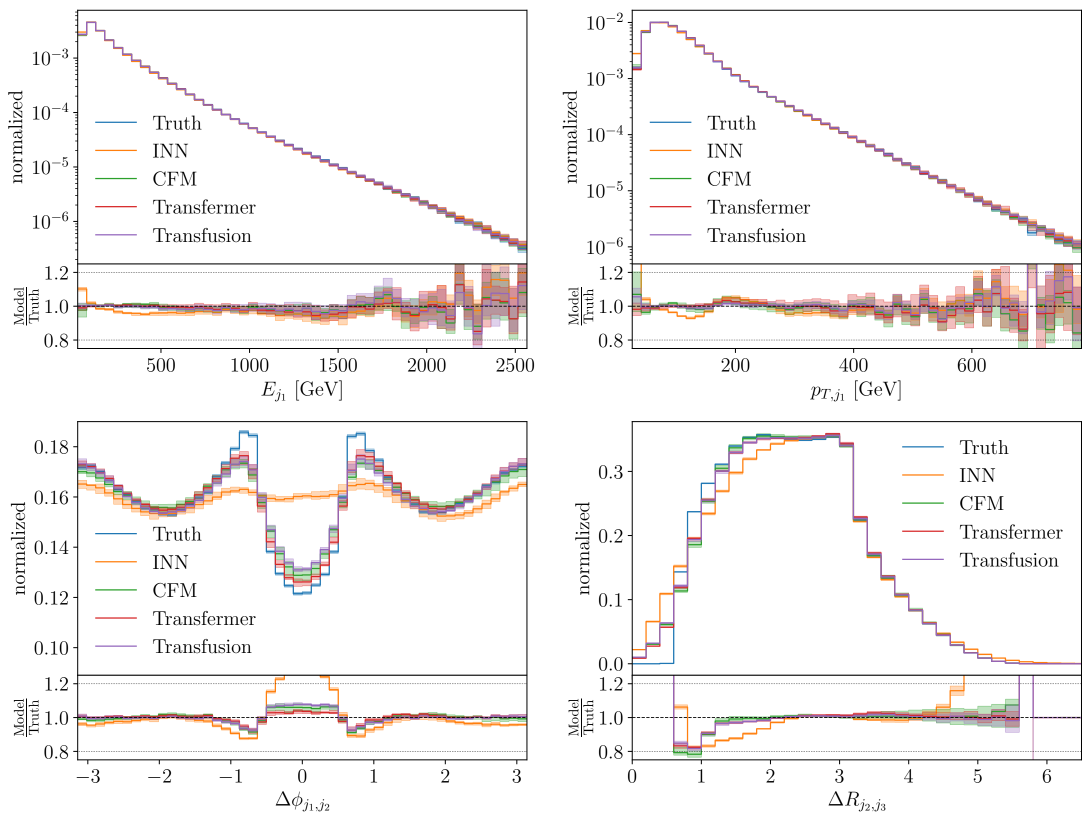


Figure 5.8: Reco-level distributions for different kinematic observables, obtained from the different generative transfer networks, conditioned on the hard-scattering momenta. Truth corresponds to the high-statistics training data.

combination of three networks with modern architectures provides the required precision and speed.

To illustrate the performance of the different network architectures and MEM frameworks, we show a set of kinematic observables from the generative transfer networks in Fig. 5.8. Before integrating the likelihood, we can show the distributions at the reco-level and compare them to the truth, or training data. We immediately see that the standard cINN is stable and extremely fast, but limited in its expressivity. The CFM diffusion network improves the performance significantly. The transformer architectures, i.e. the cINN-driven Transfermer and the CFM-driven Transfusion deliver a precision at least on par with the CFM diffusion network.

For the likelihood, we have compared the extracted likelihoods from the different architectures with the hard-process target. As a benchmark, we first improved a range of numerical aspects of our concept paper [56], with a focus on the integration with the Sampling-cINN. The improved precision of the integration raises two questions: a systematic bias in the minimum of the extracted likelihoods especially going from 400 to 10k events; and the optimality of the extracted likelihoods seen in the widths in the CP-angle α. These benchmark results are shown in Fig. 5.3.

We then upgraded our two-network setup to a three-network setup, with a learned acceptance as a function of phase space. In Fig. 5.3, we saw that this removes the leading source of systematic bias, including the challenging SM-case $\alpha = 0°$.

Next, we targeted the performance of the transfer network by replacing it with a more expressive, albeit slower CFM diffusion network. This did not improve the low-statistics results, but for the high-statistics case of 10k events the Transfer-CFM showed a clear advantage over the cINN, as can be seen in Fig. 5.5.

Finally, we solved the problem with the jet multiplicity of the cINN approach by applying a generative autoregressive Transfer-Transformer, i.e. combining a transformer with a cINN network (Transfermer) and a CFM-model (Transfusion). In Fig. 5.7, we saw that both transformer-based models outperformed the cINN, but showed similar performance as the Transfer-CFM. Notably, both transformer-based models can naturally be extended to describe a variable number of particles at both reco- and parton-level. This feature will eventually be needed for a proper description of the MEM at NLO.

In our LO example, all three models, CFM, Transfermer and Transfusion, parametrize the transfer probability flexibly and reliably. However, the Transfermer integration is approximately a factor 30 faster than the two diffusion-based models. This gap might eventually be closed using techniques like diffusion distillation [235–237]. Further improvements on the architecture, like the parallel Transfusion introduced in Appendix 5.7, might also improve the performance for more complex processes. Altogether, we conclude that a range of modern generative networks are available for the MEM, awaiting final judgment from an actual analysis.

5.7 Appendix

A.1 Variable jet number and permutation invariance

Transfermer with variable jet number

The Transfermer is easy to generalize to events with a variable number of jets at the reconstruction level. To this end, we split the inclusive transfer probability and evaluate it autoregressively,

$$\begin{aligned} p(x_\text{reco}, n|x_\text{hard}) &= p(n|x_\text{hard})\, p(x_\text{reco}|x_\text{hard}, n) \\ &= p(n|x_\text{hard})\, p(x_\text{reco}^{(1:n_\text{min})}|x_\text{hard}, n) \prod_{i=n_\text{min}+1}^{n} p(x_\text{reco}^{(i)}|x_\text{reco}^{(1:i-1)}, x_\text{hard}, n) \;, \end{aligned} \tag{5.49}$$

where n is the number of final-state particles, $x_\text{reco}^{(1:k)}$ denotes the first k reco-level momenta, $x_\text{reco}^{(k)}$ denotes the k-th reco-level momentum and n_min is the minimal number of momenta for an accepted event. The probability $p(n|x_\text{hard})$ can be extracted using a simple classifier network with a categorical cross-entropy loss and the number of additional jets as labels. The autoregressive factorization of $p(x_\text{reco}|x_\text{hard}, n)$ matches the way in which the Transfermer learns these probabilities. We pass the information about the number of additional jets to the Transfermer by appending it to the embedding of x_hard in one-hot encoded form. We can sample from the transfer probability by first sampling the multiplicity using the probabilities given by the classifier and then sampling the momenta as described in Eq.(5.46). Note that it is even possible to generalize the Transfermer to a variable number of hard-scattering momenta, because the transformer encoder accepts a

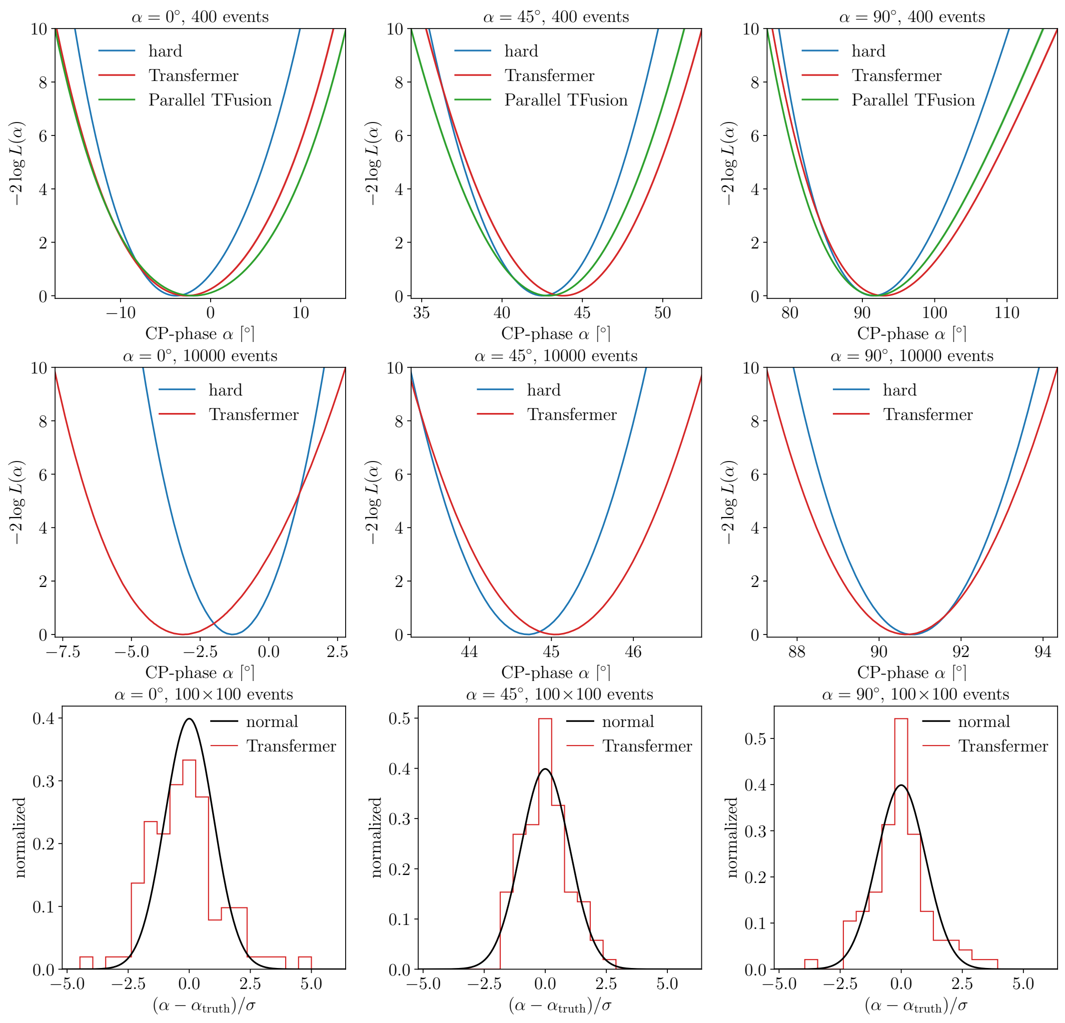


Figure 5.9: **Transfermer with variable jet numbers and parallel Transfusion:** likelihoods for different CP-angles using the Transfermer with variable jet multiplicity and the parallel Transfusion as the transfer probability. From top to bottom: likelihood for 400 events, 10000 events, and pulls.

variable number of inputs without any further changes to the architecture, making it a good candidate for a machine-learned MEM at NLO.

We train the jet multiplicity classifier with the hyperparameters given in Tab. A.4. We observe that they are mostly flat for the top and Higgs, but there is a stronger variation as a function of the forward jet momentum, especially η_j. Like for the acceptance function, this is explained by ISR jets being tagged instead of the forward jet, leading to a lower probability of extra jets for $|\eta| > 2.4$.

We then run the MEM integration to obtain the results shown in Fig. 5.9. They are mostly similar to the results with fixed multiplicity shown in Fig. 5.7. It shows that for our specific process, we do not gain constraining power by including the information from additional jets. However, that might be different for other processes and especially at NLO. So the ability to deal with a variable number of jets is still a valuable addition to our MEM toolbox.

Permutation-invariant Transfusion

The Transfusion can be generalized to events with a variable jet number in complete analogy to the Transfermer. However, as diffusion models do not require invertibility, they allow for an additional approach in combining the transformer with the CFM network where we drop the autoregressive setup and instead generate all particle 4-momenta in parallel.

Before, in the autoregressive setup the transformer calculates a condition based on the hard-level momenta and the already generated reco-level momenta, which is then fed to the CFM that predicts the time-dependent velocity field. Crucially, the transformer itself has no time dependence. In the alternative parallel setup, the transformer decoder no longer sees the first $i-1$ reco-level particles $x_\text{reco}^{(1,\dots,i-1)}$ to describe $c^{(i)}$. Instead, its inputs are the conditional and time-dependent diffusion states $x_\text{reco}^{(1,\dots,n)}(t|x_0)$, as defined in Eq.5.38, of all n reco-level particles, and the time t. The encoder, which acts only on the hard-level momente, is unchanged. Now, the transformer calculates time-dependent embeddings, one for each particle. These time-dependent embeddings are then again fed to a small CFM network predicting the velocity field. In this setup the velocity field of the i^th particle is calculated as

$$v^{(i)}(c(e_\text{reco}(t), e_\text{hard}, t), t) \; , \tag{5.50}$$

where the transformer c is now a time-dependent function of the embeddings e of all momenta. The overall setup is illustrated in Fig. 5.10. In practice, a single linear layer

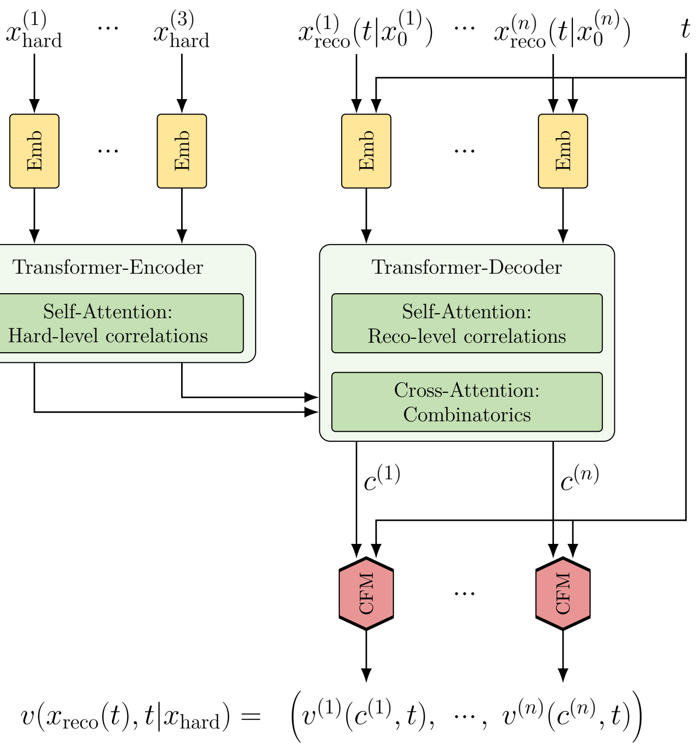

$x_\text{hard}^{(1)}$ ⋯ $x_\text{hard}^{(3)}$
$x_\text{reco}^{(1)}(t|x_0^{(1)})$ ⋯ $x_\text{reco}^{(n)}(t|x_0^{(n)})$
t
Emb
Transformer-Encoder
Self-Attention: Hard-level correlations
Transformer-Decoder
Self-Attention: Reco-level correlations
Cross-Attention: Combinatorics
$c^{(1)}$
$c^{(n)}$
CFM
$v(x_\text{reco}(t), t|x_\text{hard}) = \left(v^{(1)}(c^{(1)}, t), \; \cdots, \; v^{(n)}(c^{(n)}, t)\right)$

Figure 5.10: Parallel Transfusion architecture. Compared to the autoregressive setup we no longer use masked self-attention in the transformer decoder, but instead make it time-dependent.

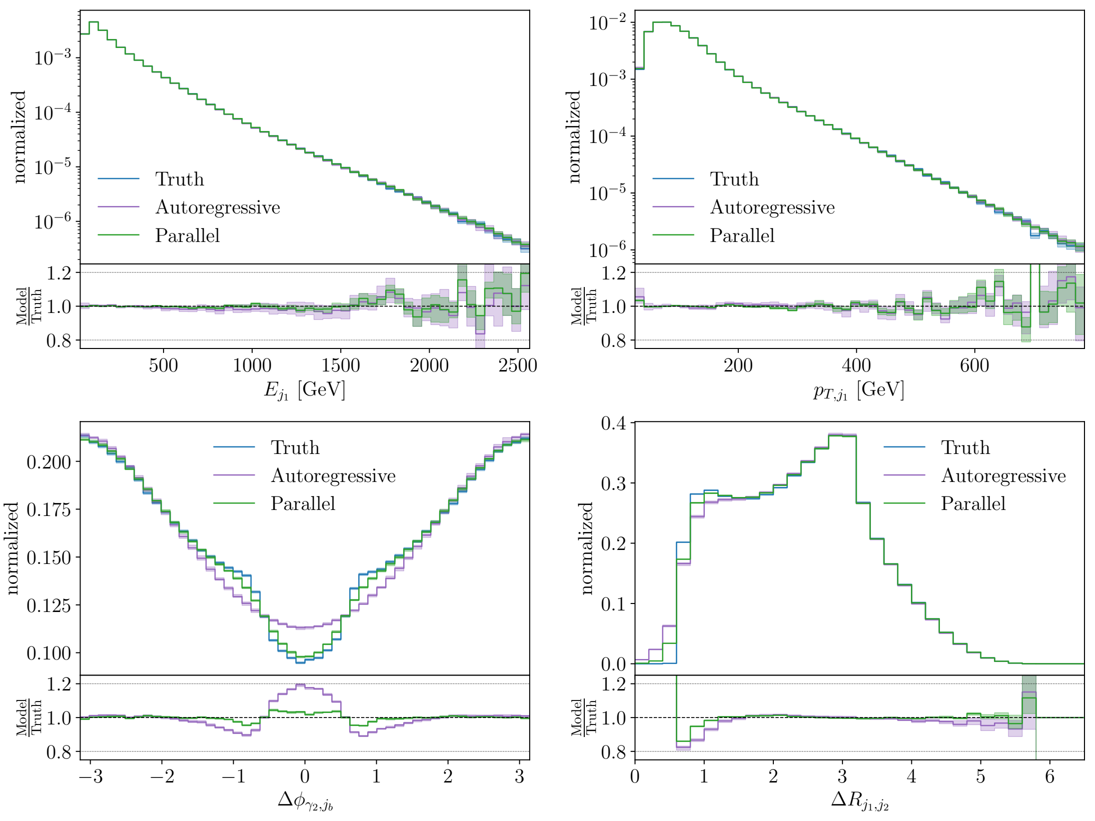


Figure 5.11: Reco-level distributions for different kinematic observables, obtained from the autoregressive and parallel Transfusion networks, conditioned on the hard-scattering momenta. Truth corresponds to the high-statistics training data.

is sufficient to map the transformer outputs to the velocity field components. Note, that during sampling the initial input to the transformer is the unconditional latent space vector r which is then mapped onto x_{reco} with the learned velocity field and the ODE solver. The parallel Transfusion setup naturally generalizes to varying particle multiplicities at both hard- and reco-level without requiring an arbitrary autoregressive order, as it is permutation-invariant at both levels.

Reco-level distributions for different kinematic observables are shown in Fig. 5.11. The marginal distributions show no difference between the parallel Transfusion and the other networks, but for the angular correlations we see the parallel Transfusion having a clear edge. Giving the transformer itself a time-dependence forces us to evaluate it repeatedly inside the ODE solver, making sampling and likelihood calculation in this setup even slower than for the pure CFM or the autoregressive Transfusion. We show integration results for 400 events using the parallel Transfusion in Fig. 5.9, finding that they are comparable to the results from the autoregressive Transfusion. Due to the slow likelihood calculation this setup did not scale up to 10000 events. The strong performance on the observable distribution level indicates that this architecture might proof useful in combination with speed-up techniques like diffusion distillation or for applications that do not require likelihood calculation.

A.2 Evaluating on Herwig

The critical backbone of our inference method is the learned transfer probability $p_\theta(x_\text{reco}|x_\text{hard})$. We have demonstrated that generative networks can learn this conditional density from simulated data to very high precision. However, even a perfect network will only encode the forward transfer of the simulation, which is close but not necessarily identical to nature. In this section, we investigate how this impacts the results of our method by using different simulation setups:

1. a baseline simulation with Pythia8 for network training ;
2. an alternative simulation based on Herwig [35] for inference, emulating the truth reco-level data of the experiment. The detector effects are still modeled with Delphes.

The results obtained with our method in this setup are shown in Fig. 5.12. For 400 events we find that the extracted reco-level likelihoods mostly agree with the hard-level likelihoods. Note that the hard-level likelihoods are not fixed but also affected by the underlying simulation assumption, most visible for $\alpha = 90°$. This is because the fiducial hard-level likelihood is only defined on hard-level events x_hard leading to accepted x_reco events, which critically depends on the efficiency $\epsilon(x_\text{hard})$ of the underlying normalized transfer function r, as defined in Eqs.(5.5) and (5.12). This effectively encodes a dependence on the assumed forward simulation

$$p_\text{fid}(x_\text{hard}|\alpha) \equiv p_\text{Pythia8}(x_\text{hard}|\alpha) \; . \tag{5.51}$$

Hence, evaluating the fiducial hard-level likelihoods on the Herwig simulation can generally lead to a bias in the likelihood distribution. In the high-statistics scenario with 10k events, we observe good agreement for $\alpha = 0,\ 45°$, comparable to the results when evaluating on Pythia8, which means we can assume

$$p_\text{Pythia8}(x_\text{hard}|\alpha) \approx p_\text{Herwig}(x_\text{hard}|\alpha) \; . \tag{5.52}$$

In these cases, we find that the reco-level likelihood obtained using our method still agrees well with the hard-scattering likelihood, and the results are still well-calibrated. However, for $\alpha = 90°$ the hard-level likelihoods are significantly off from the true value, indicating that Eq.(5.52) is no longer valid. Further, training the transfer function on events that do not follow the true distribution of the measured data may introduce α-dependent effects. Consequently, we also find a large deviation between the hard- and reco-level likelihoods.

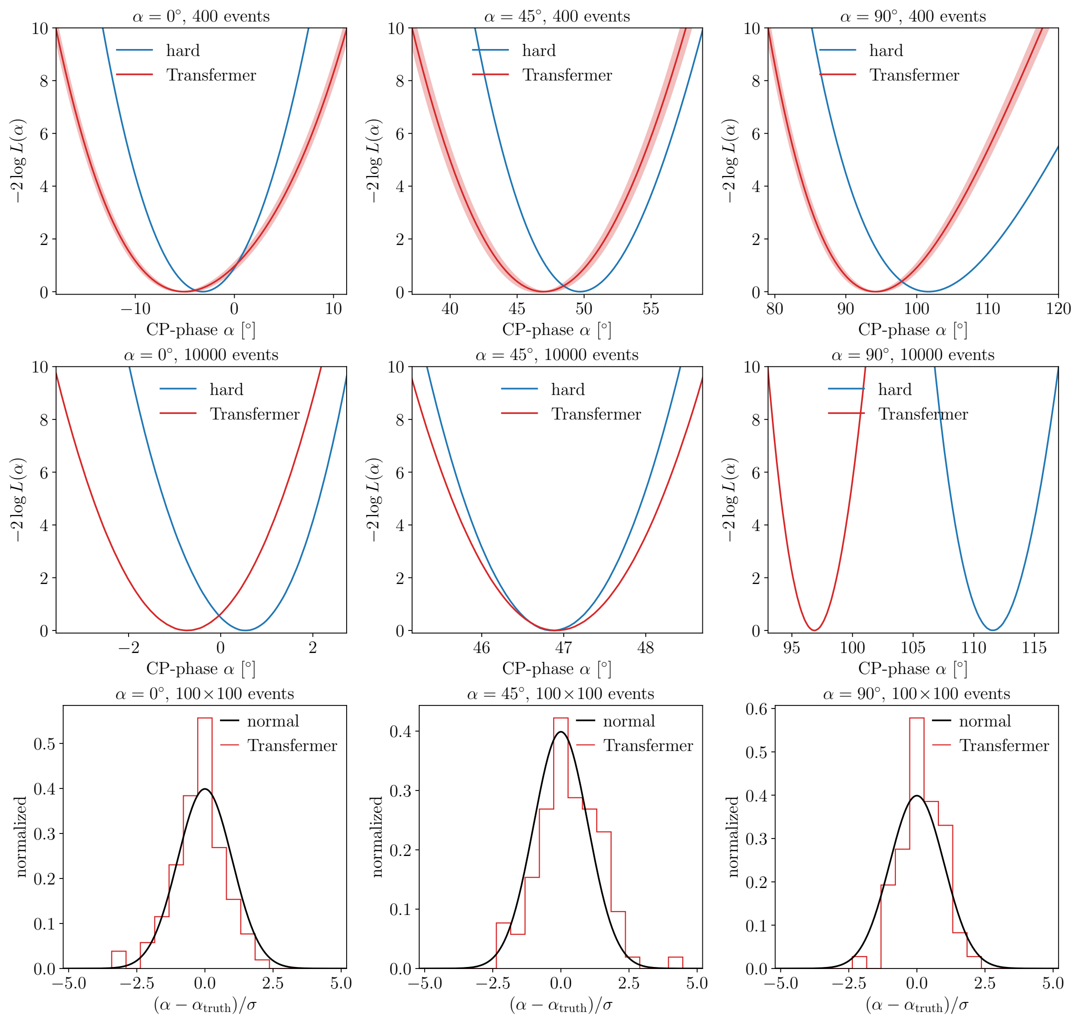


Figure 5.12: **Transfermer applied on Herwig simulation:** likelihoods for different CP-angles using the Transfermer trained on PYTHIA8 simulations but evaluated on HERWIG simulations. From top to bottom: likelihood for 400 events, 10000 events, and pulls.

Chapter 6
The Landscape of Unfolding with Machine Learning

The research presented in this chapter is based on work in collaboration with Javier Mariño Villadamigo, Alexander Shmakov, Sascha Diefenbacher, Vinicius Mikuni, Theo Heimel, Michael Fenton, Kevin Greif, Benjamin Nachman, Daniel Whiteson, Anja Butter and Tilman Plehn and has been previously published in Ref. [3]. The content is similar or identical to the content of this article.

Particle physics experiments seek to reveal clues about the fundamental properties of particles and their interactions. A key challenge is that predictions from quantum field theory are at the level of partons, while experiments observe the corresponding detector signatures. Precise and detailed simulations link these two levels [30]. They fold predictions for the hard process through QCD effects, hadronization, and the detector response to compare with data. This statistically powerful *forward inferences* approach has been widely used.

However, forward inference requires access to the data and accurate detector simulations. These conditions are rarely satisfied outside of a given experiment, severely limiting the ability of the broader community to study particle physics data. In addition, analysis of data from the high-luminosity LHC with forward inference will require precise simulations for every hypothesis, challenging available computing resources.

An alternative approach is *unfolding*. Rather than correcting predictions for the effects of the detector, the data are adjusted to provide an estimate of their pre-detector distributions. Since the effects described by our forward simulation are stochastic, this adjustment is performed on a statistical basis. Unfolding offers important advantages, such as making data analysis possible by a broader community and enabling an efficient combination of data from several experiments, such as in global analyses of the Standard Model Effective Theory [238, 239].

Traditional unfolding algorithms have been used extensively, successfully delivering a multitude of differential cross section measurements [57–60]. The most widely-used methods are Iterative Bayesian Unfolding [111, 240–242], Singular Value Decomposition [243], and TUnfold [110]. However, each of these methods can only be applied to binned datasets of small dimensionality, such that the unfolded observables and their binning have to be selected in advance.

Machine learning (ML) techniques have revolutionized unfolding by allowing for unbinned cross sections to be measured across many dimensions [26, 59]. Where sufficient information is unfolded, new observables can be calculated from unbinned data, long after the initial publication. The first ML-based unfolding method applied to data is OmniFold [63,112], which uses classifiers to reweight simulations. It has recently been applied to studies of hadronic final states at H1 [244–247], LHCb [248], CMS [249], and STAR [250]. Alternative ML-unfolding methods use generative networks, either for distribution mapping [61, 67–69] or for probabilistic, conditional generation [62, 64, 65, 210–212, 251].

The goal of this paper is to lay out and extend the landscape of ML methods. We benchmark a diverse set of approaches on the same datasets, to facilitate direct comparisons. Some methods have been studied with an iterative component to mitigate the sensitivity to starting particle-level simulations. To simplify the setup and reduce stochastic effects from iterating, we apply all methods with only a single step. The goal is to estimate the posterior with the starting simulation as the prior. Performing this step well is the essential component of a full unfolding approach.

We begin with a brief introduction of the different methods for ML-based unfolding in Sec. 6.1. In Sec. 6.2, we show how all approaches can accurately unfold from detector level to the particle level using a Z+jets benchmark dataset. For certain theory questions it is useful to further unfold to the parton level, treating QCD radiation as a distortion to be corrected like detector effects. As an example of this type of unfolding, we study top quark pair production in Sec. 6.3. In Sec. 6.4 we summarize the advantages of the different methods, to help the experimental collaborations pick the method(s) best-suited for a given task.

6.1 ML-Unfolding

We define our unfolding problem using four phase space densities, which are encoded in the corresponding samples, in the sense of unsupervised density estimation in ML-terms. We rely on simulated predictions at the particle/parton level, $p_\text{gen}(x_\text{part})$, and the detector or reconstruction (reco) level, $p_\text{sim}(x_\text{reco})$. Unfolding turns the measured p_data into p_unfold,

$$\begin{tikzcd}
p_\text{gen} \arrow[r, leftrightarrow, "\text{unfolding inference}"] \arrow[d, "\text{simulation}"'] & p_\text{unfold}(x_\text{part}) \\
p_\text{sim} \arrow[r, leftrightarrow, "\text{forward inference}"] & p_\text{data}(x_\text{reco}) \arrow[u, "\text{unfolding}"']
\end{tikzcd} \tag{6.1}$$

Our simulated samples come in pairs $(x_\text{part}, x_\text{reco})$, which can be used for unfolding. Data only exist on the x_reco level. The features of the unfolded data p_unfold should be determined by p_data, but will always include a data-independent bias from the assumed p_gen. The question how we can minimize the resulting model dependence will be part of a follow-up of this study.

Established ML-techniques for unfolding rely on one of two approaches. They either reweight simulated samples, or they generate unfolded samples from conditional probabilities. We will briefly introduce both original methods [62–64], as well as a more recent hybrid approach of mapping distributions using generative networks.

6.1.1 Reweighting: OmniFold and bOmnifold

The deep learning-based approach to unfolding via re-weighting is OmniFold [63, 112]. It is based on the Neyman–Pearson lemma [252], stating that an optimally trained, calibrated classifier C will learn the likelihood ratio of the two underlying phase space distributions. If we use a binary cross entropy (BCE) loss to distinguish between data and simulated reco-level events, then the following combination approximates the likelihood ratio:

$$w(x_\text{reco}) \equiv \frac{p_\text{data}(x_\text{reco})}{p_\text{sim}(x_\text{reco})} = \frac{C(x_\text{reco})}{1 - C(x_\text{reco})} \; . \tag{6.2}$$

OmniFold computes these classifier weights at the reco-level, and uses the paired simulated data to pull these weights from the reco-level events to the particle-level events. The re-weighted simulated events then define

$$p_\text{unfold}(x_\text{part}) = w(x_\text{reco}) \, p_\text{gen}(x_\text{part}) \; . \tag{6.3}$$

This weight-pushing is the first step in the two-step iterative OmniFold algorithm. Because we are leaving out the model dependence to a dedicated second study, we restrict ourselves to this first iteration, which in the scheme of Eq.(6.1) looks like

$$\begin{tikzcd} p_\text{gen} \arrow[r, "\text{apply reweighting}"] & p_\text{unfold}(x_\text{part}) \\ p_\text{sim} \arrow[u, "\text{pull weights}"] \arrow[r, leftrightarrow, "\text{train reweighting}"] & p_\text{data}(x_\text{reco}) \end{tikzcd} \tag{6.4}$$

Bayesian neural network (BNN) Bayesian versions can be derived for any deterministic neural network with a likelihood loss [23, 155–158]. The BNN training does not fix the network parameters, but allows them to learn distributions, such that sampling over the network parameters gives the probability distribution in model space, i.e. for the network output. Based on studies for regression [159, 166] and classification tasks [160], there is evidence that for a sufficiently deep network we can assign independent Gaussians to each network parameter [157]. This effectively doubles the size of the network which now learns a central prediction and the error bar simultaneously. Even though the weights are Gaussian distributed, the final network output is generally not a Gaussian. As we will see below, Bayesian networks can be generalized to generative tasks [1, 45, 161].

One benefit of Bayesian networks is that they automatically include a generalized dropout and a weight regularization [23, 253, 254], derived from Bayes' theorem together with the likelihood loss. This means that BNNs are automatically protected from overtraining and an attractive option for applications where the precision of the network is critical, like the classifier reweighting in OmniFold.

6.1.2 Mapping distributions: Schrödinger Bridge and Direct Diffusion

Instead of reweighting phase-space events, we can use generative neural networks to morph a base distribution to a target distribution. In our case, we train a network to map event distributions from x_reco to x_part based on the paired or unpaired simulated events and apply this mapping to $p_\text{data}(x_\text{reco})$ to generate $p_\text{unfold}(x_\text{part})$:

$$\begin{tikzcd}
p_\text{gen} & & p_\text{unfold}(x_\text{part}) \\
p_\text{sim} \arrow[u, "\text{training}"] \arrow[rr, leftrightarrow, "\text{correspondence}"] & & p_\text{data}(x_\text{reco}) \arrow[u, "\text{distribution mapping}"']
\end{tikzcd} \tag{6.5}$$

As mentioned above, the trained mapping assumes that p_sim and p_data describe the same features at the reco-level. Two ML-methods that we study for this task include Schrödinger Bridges [68] and Direct Diffusion [69], see also Ref. [61] for an early study.

Schrödinger Bridge

Schrödinger Bridges define the transformation between particle-level events $x_\text{part} \sim p_\text{gen}$ to reco-level events $x_\text{reco} \sim p_\text{sim}$ as a time-dependent process following a forward-time stochastic differential equation (SDE)

$$dx = f(x,t)\text{d}t + g(t)dw \; . \tag{6.6}$$

The drift term f controls the deterministic part of the time-evolution, g is the noise schedule, and dw a noise infinitesimal. For such an SDE, the reverse time evolution follows the SDE

$$dx = [f(x,t) - g(t)^2 \nabla \log p(x,t)]dt + g(t)dw \; , \tag{6.7}$$

with the corresponding score $s(x,t) = \nabla \log p(x,t)$. To construct an unfolding, we need to find f and g for our forward process from particle level to reco level, and then encode the score function in the unfolding network $s_\theta(x,t)$ [123].

Constructing a forward-time SDE that transforms an arbitrary distribution into another is much more challenging than mapping a distribution into a noise distribution with known probability density (e.g. a Gaussian), as is the case for standard SDE-based diffusion networks. A framework to construct a transport plan in the general case was proposed by Erwin Schrödinger [255]. It introduces two wave functions describing the time-dependent density as $p(x,t) = \widehat{\Psi}(x,t)\Psi(x,t)$. By setting the drift coefficient to $f = g(t)^2 \nabla \log \Psi(x,t)$ the forward and reverse SDEs in Eqs.(6.6) and (6.7) become

$$\begin{aligned}
dx &= g(t)^2 \nabla \log \Psi(x,t)dt + g(t)dw \\
dx &= -g(t)^2 \nabla \log \widehat{\Psi}(x,t)dt + g(t)dw \; .
\end{aligned} \tag{6.8}$$

If the two wave-functions fulfill the coupled partial differential equations

$$\begin{aligned}
\frac{\partial \Psi(x,t)}{\partial t} &= -\frac{1}{2} g(t)^2 \Delta \Psi(x,t) \\
\frac{\partial \widehat{\Psi}(x,t)}{\partial t} &= \frac{1}{2} g(t)^2 \Delta \widehat{\Psi}(x,t) \; ,
\end{aligned} \tag{6.9}$$

with the boundary conditions

$$\widehat{\Psi}(x,t)\Psi(x,t) = \begin{cases} p_\text{gen}(x) & t=0 \\ p_\text{sim}(x) & t=1\ , \end{cases} \tag{6.10}$$

then the SDEs in Eq.(6.8) transform particle-level events to reco-level events and vice versa.

Next, we need to find Ψ, $\widehat{\Psi}$ that fulfill the conditions. The authors of Ref. [256] observe that reverse generation following Eq.(6.8) does not require access to the wave functions, but only to the score function $\nabla \log \widehat{\Psi}$. For paired training data,

$$(x_0, x_1) \sim (p_\text{gen}, p_\text{sim}) \tag{6.11}$$

the posterior encoded in the SDEs in Eq.(6.8), conditioned on the respective initial and final states, has the analytic form

$$\begin{aligned} &q(x|x_0,x_1) = \mathcal{N}(x_t; \mu_t(x_0,x_1), \Sigma_t) \\ \text{with} \qquad &\mu_t(x_0,x_1) = \frac{\bar{\sigma}_t^2}{\bar{\sigma}_t^2+\sigma_t^2}x_0 + \frac{\sigma_t^2}{\bar{\sigma}_t^2+\sigma_t^2}x_1 \quad \text{and} \quad \Sigma_t = \frac{\sigma_t^2\bar{\sigma}_t^2}{\bar{\sigma}_t^2+\sigma_t^2}\ , \end{aligned} \tag{6.12}$$

denoting $\sigma_t^2 = \int_0^t g^2(\tau)d\tau$ and $\bar{\sigma}_t^2 = \int_t^1 g^2(\tau)d\tau$. This allows for the generation of samples from this stochastic process as $x_t(x_0,x_1) = \mu_t + \Sigma_t \epsilon$ with $\epsilon \sim \mathcal{N}(0,1)$ and (x_0,x_1), a pair of reco-level and particle-level events. Moreover, the score $\nabla \log \widehat{\Psi}$ can be learned by minimizing the loss

$$\mathcal{L}_\text{SB} = \left\langle \left[\epsilon_\theta(x_t(x_0,x_1),t) - \frac{x_t(x_0,x_1)-x_0}{\sigma_t} \right]^2 \right\rangle_{t\sim\mathcal{U}([0,1]),(x_0,x_1)\sim p(x_\text{part},x_\text{reco})}\ , \tag{6.13}$$

where x_t is sampled according to Eq.(6.12). After training, the network unfolds by numerically solving the reverse SDE Eq.(6.8) with the x_reco values as the initial conditions.

We follow a slight variation, where the dynamics are reduced to a deterministic process [256]. This can be achieved by replacing the posterior distribution Eq.(6.12) by its mean and training the network to encode not the score function, but the velocity field of the reverse process, which then takes the form of an ordinary differential equation:

$$dx_t = v_t(x_t|x_0)dt = \frac{\beta_t}{\sigma_t^2}(x_t - x_0)dt\ . \tag{6.14}$$

For the noise schedule, we follow Ref. [68] and use $g(t) = \sqrt{\beta(t)}$, with $\beta(t)$ the triangular function

$$\beta(t) = \begin{cases} \beta_0 + 2(\beta_1-\beta_0)t & 0 \le t < \frac{1}{2} \\ \beta_1 - 2(\beta_1-\beta_0)\left(t-\frac{1}{2}\right) & \frac{1}{2} \le t \le 1\ . \end{cases} \tag{6.15}$$

with $\beta_0 = 10^{-5}$ and $\beta_1 = 10^{-4}$.

Direct Diffusion

Like the Schrödinger Bridge, Direct Diffusion (DiDi) describes a time evolution between particle-level events at $t=0$ and reco-level events at $t=1$. Following the Conditional

Flow Matching (CFM) [51] framework, DiDi uses an ordinary differential equation (ODE)

$$\frac{dx(t)}{dt} = v_\theta(x(t), t) \; , \tag{6.16}$$

with a velocity field $v_\theta(x(t), t)$ encoded in a neural network. This time evolution of the individual events is related to the time evolution of the underlying density via the continuity equation

$$\frac{\partial p(x,t)}{\partial t} + \nabla_x \left[p(x,t) v_\theta(x,t)\right] = 0 \; . \tag{6.17}$$

The learning task is then to find a velocity field that transforms the density $p(x,t)$ such that

$$p(x,t) \to \begin{cases} p_\text{gen}(x) & t \to 0 \\ p_\text{sim}(x) & t \to 1 \; . \end{cases} \tag{6.18}$$

Such a velocity field can be constructed by building on event-conditional velocity fields. For a given particle-level event $x_0 \sim p_\text{gen}(x_\text{part})$, the algorithm samples a corresponding reco-level event $x_1 \sim p_\text{sim}(x_\text{reco}|x_\text{part} = x_0)$, and the two are connected with a linear trajectory

$$x(t|x_0) = (1-t)x_0 + t x_1 \to \begin{cases} x_0 & t \to 0 \\ x_1 \sim p(x_\text{reco}|x_\text{part} = x_0) & t \to 1 \; . \end{cases} \tag{6.19}$$

Differentiating this trajectory defines the conditional velocity field

$$v(x(t|x_0), t|x_0) = \frac{d}{dt}\left[(1-t)x_0 + t x_1\right] = -x_0 + x_1 \; . \tag{6.20}$$

This is not yet useful as an unfolding network, as it can only unfold to a pre-specified hard event. The desired unconditional velocity field can be obtained via

$$v(x,t) = \int dx_0 \; \frac{v(x,t|x_0) p(x,t|x_0) p_\text{gen}(x_0)}{p(x,t)} \; , \tag{6.21}$$

where $p(x,t|x_0)$ is the conditional density defined via sampling from equation (6.19) and $p(x,t)$ is obtained by integrating out the condition x_0. In practice, it is sufficient to train on fixed data pairs $(x_\text{part}, x_\text{reco})$ instead of resampling the posterior $p(x_\text{reco}|x_\text{part} = x_0)$ in each epoch. The velocity field can be learned from data as a simple regression task with the MSE loss

$$\mathcal{L}_\text{DiDi} = \left\langle \left[v_\theta((1-t)x_0 + t x_1, t) - (x_1 - x_0)\right]^2 \right\rangle_{t \sim \mathcal{U}([0,1]), (x_0, x_1) \sim p(x_\text{part}, x_\text{reco})} \; . \tag{6.22}$$

Once the network is trained, a reco-level event $x_1 \sim p(x_\text{reco})$ can be transferred by numerically solving the coresponding ODE in Eq.(6.16)

$$x_0 = x_1 - \int_0^1 v_\theta(x(t), t) dt \; . \tag{6.23}$$

Unpaired DiDi The starting premise of most unfolding methods is that the forward model $p(x_\text{reco}|x_\text{part})$ is known, within uncertainty. There may be cases where it is not known [67] and instead of pairs $(x_\text{part}, x_\text{reco})$, we only have access to the marginals $\{x_\text{part}\}$,

$\{x_\text{reco}\}$. There is no unique solution to this problem even if the detector response is deterministic; however, we can proceed by assuming that the function corresponds to the optimal transport map. We consider a variation of DiDi for this configuration by dropping the pairing information between training events [69]. This can be achieved by modifying the conditional trajectory so that x_1 is sampled independently of x_0, so Eq.(6.19) becomes

$$x(t|x_0) = (1-t)x_0 + tx_1 \to \begin{cases} x_0 & t \to 0 \\ x_1 \sim p(x_\text{reco}) & t \to 1 \; . \end{cases} \tag{6.24}$$

The loss function is

$$\mathcal{L}_\text{DiDi-U} = \left\langle \left[v_\theta((1-t)x_0 + tx_1, t) - (x_1 - x_0) \right]^2 \right\rangle_{t\sim\mathcal{U}([0,1]), x_0 \sim p(x_\text{part}), x_1 \sim p(x_\text{reco})} \; . \tag{6.25}$$

During training we now sample events independently of each other, and the learned map will be purely determined by the network and its training.

Bayesian network Because the distribution mapping loss function does not have a straightforward interpretation as a likelihood, it cannot be simply transformed into a Bayesian network from first principles. However, we can add the relevant features of a Bayesian network, as for the CFM [1, 69]. This includes Bayesian layers, Gaussian distributions of all or some network parameters, and a KL-term regularizing the network parameters towards a Gaussian prior,

$$\mathcal{L}_\text{B-CFM} = \Big\langle \mathcal{L}_\text{CFM} \Big\rangle_{\theta \sim q(\theta)} + c\, \text{KL}[q(\theta), p(\theta)] \; . \tag{6.26}$$

The factor c balances the deterministic loss with the Bayesian-inspired regularization. If the network loss follows from a likelihood, this factor is fixed by Bayes' theorem. In all other cases it is a hyperparameter. We have checked that the network performance as well as the extracted posteriors are stable when varying c over several orders of magnitudes, suggesting that the learned weight distribution corresponds to an inherent property of the setup.

6.1.3 Generative unfolding

Generative unfolding uses conditional generative networks to learn the conditional probability describing the inverse simulation $p_\theta(x_\text{part}|x_\text{reco})$,

$$\begin{tikzcd} p_\text{gen} \arrow[d, leftrightarrow, "\text{paired data}"'] & & p_\text{unfold}(x_\text{part}) \\ p_\text{sim} \arrow[rr, leftrightarrow, "\text{correspondence}"] & & p_\text{data}(x_\text{reco}) \arrow[u, "p_\theta(x_\text{part}|x_\text{reco})"'] \end{tikzcd} \tag{6.27}$$

Building a forward surrogate network for $p(x_\text{reco}|x_\text{part})$ uses the same data and has nearly the same setup as learning the inverse probability $p(x_\text{part}|x_\text{reco})$. The usual assumption of unfolding is that the detector response is universal, which breaks the symmetry of the forward and backwards networks via Bayes' theorem,

$$p(x_\text{part}|x_\text{reco}) = p(x_\text{reco}|x_\text{part})\, \frac{p(x_\text{part})}{p(x_\text{reco})} \; . \tag{6.28}$$

For the forward simulation, we assume that the condition on x_part does not induce a significant prior for the generated p_sim. For the inverse simulation, this prior dependence is relevant and it formally implies that there is no notion of unfolding single events, even though the generative unfolding tools provide the corresponding conditional probabilities.

Technically, we start from a simple latent distribution, where the generative network transforms the required phase space distribution,

$$z \sim p_\text{latent}(z) \quad \xrightarrow{G_\theta(z;x_\text{reco})} \quad x_\text{part} \sim p_\theta(x_\text{part}|x_\text{reco}) \; . \tag{6.29}$$

The phase space distribution of an unfolded dataset is then given as

$$p_\text{unfold}(x_\text{part}) = \int dx_\text{reco}\; p_\theta(x_\text{part}|x_\text{reco})\; p_\text{data}(x_\text{reco}) \; . \tag{6.30}$$

This approach is based on posterior distributions for individual events, which means that we can also take single measured events and run them through the model any number of times. In practice, Eq. 6.30 is achieved by sampling from $p_\theta(x_\text{part}|x_\text{reco})$ for data events that follow $p_\text{data}(x_\text{reco})$ and then ignoring the x_reco argument from the resulting dataset of pairs $(x_\text{part}, x_\text{reco})$. If we wanted to sample further from the result and/or iterate the procedure, we would need to do something like the second step of OmniFold, which is a local averaging as done for generative models in Ref. [65]. A key ingredient to unfolding with generative networks [62] is to either train this network with a likelihood loss [64], like for the cINN, or to guarantee the probabilistic interpretation through the mathematical setup, like in the CFM.

Conditional INN

The original generative network used for unfolding is a normalizing flow [37] in its conditional invertible neural network (cINN) variant [38, 64]. It defines the mapping between the latent and phase space as an invertible function, conditioned on the reco-level event,

$$z \sim p_\text{latent}(z) \quad \overset{G_\theta(z;x_\text{reco}) \rightarrow}{\underset{\leftarrow\, G_\theta^{-1}(x_\text{part};x_\text{reco})}{\longleftrightarrow}} \quad x_\text{part} \sim p_\theta(x_\text{part}|x_\text{reco}) \; . \tag{6.31}$$

The bijection form allows us to write down the learned density as

$$p_\theta(x_\text{part}|x_\text{reco}) = p_\text{latent}(G_\theta^{-1}(x_\text{part};x_\text{reco})) \left| \det \frac{\partial G_\theta^{-1}(x_\text{part};x_\text{reco})}{\partial x_\text{part}} \right| \; . \tag{6.32}$$

Having access to the network likelihood enables us to use it directly as loss function and train via maximum likelihood estimation

$$\mathcal{L}_\text{cINN} = -\big\langle \log p_\theta(x_\text{part}|x_\text{reco}) \big\rangle_{(x_0,x_1)\sim p(x_\text{part},x_\text{reco})} \; . \tag{6.33}$$

This approach requires a bijective map that is flexible enough to model complex transformations, while still allowing for efficient computation of the Jacobian determinant. We employ coupling blocks [64], but replace the affine coupling blocks with the more flexible rational quadratic spline blocks [124].

Transformer-cINN We also consider a transformer extension to the standard cINN [2]. The architecture translates a sequence of reco-level momenta into a sequence of particle-level momenta. A transformer network encodes the correlations between all event dimensions at particle level as well as their correlation with the reco-level event. A small 1D-cINN then generates the hard-level momenta conditioned on the transformer output. To guarantee invertibility and a tractable Jacobian, the likelihood and the generation process are factorized autoregressively

$$p_\theta(x_\text{part}|x_\text{reco}) = \prod_{i=1}^{n} p_\theta(x_\text{part}^{(i)}|c(x_\text{part}^{(0)}, \dots, x_\text{part}^{(i-1)}, x_\text{reco})) \; . \tag{6.34}$$

The product in Eq.(6.34) covers all dimensions at particle level. The function c is learned by the transformer to encode the information about the reco-level momenta as well as the already generated hard-level momenta. The one-dimensional conditional densities are encoded in the cINN. Note that in contrast to Ref. [2], this so-called transfermer is autoregressive in individual one-dimensional components, instead of in the four momenta grouped by particles.

Conditional Flow Matching

As an alternative generative network, we employ a diffusion approach called Conditional Flow Matching (CFM) [1, 51]. The mathematical structure is the same as for the DiDi network introduced in Sec. 6.1.2. The key difference here is that the CFM now samples from a Gaussian latent distribution, conditional on a reco-level event, Eq.(6.29). This means the time-evolving density is conditional and interpolates between the boundary conditions

$$p(x,t|x_\text{reco}) \to \begin{cases} p(x_\text{part}|x_\text{reco}) & t \to 0 \\ \mathcal{N}(x;0,1) & t \to 1 \; , \end{cases} \tag{6.35}$$

while the ODE now reads

$$\frac{dx(t)}{dt} \equiv v_\theta(x(t), t|x_\text{reco}) \; . \tag{6.36}$$

The information about the reco-level event to unfold is no longer encoded in the initial condition of the ODE, but in an additional input to the network that predicts the velocity field. Again we start with paired training data, $x_0 \sim p(x_\text{part})$ and $x_1 \sim p(x_\text{reco}|x_\text{part} = x_0)$, and define a simple conditional trajectory towards Gaussian noise,

$$x(t|x_0, x_\text{reco}) = (1-t)x_0 + t\epsilon \to \begin{cases} x_0 & t \to 0 \\ \epsilon \sim \mathcal{N}(0,1) & t \to 1 \; . \end{cases} \tag{6.37}$$

The conditional velocity field is defined via the derivative of the trajectory

$$v(x(t|x_0, x_\text{reco}), t|x_0, x_\text{reco}) = \frac{d}{dt}\left[(1-t)x_0 + t\epsilon\right] = -x_0 + \epsilon \; . \tag{6.38}$$

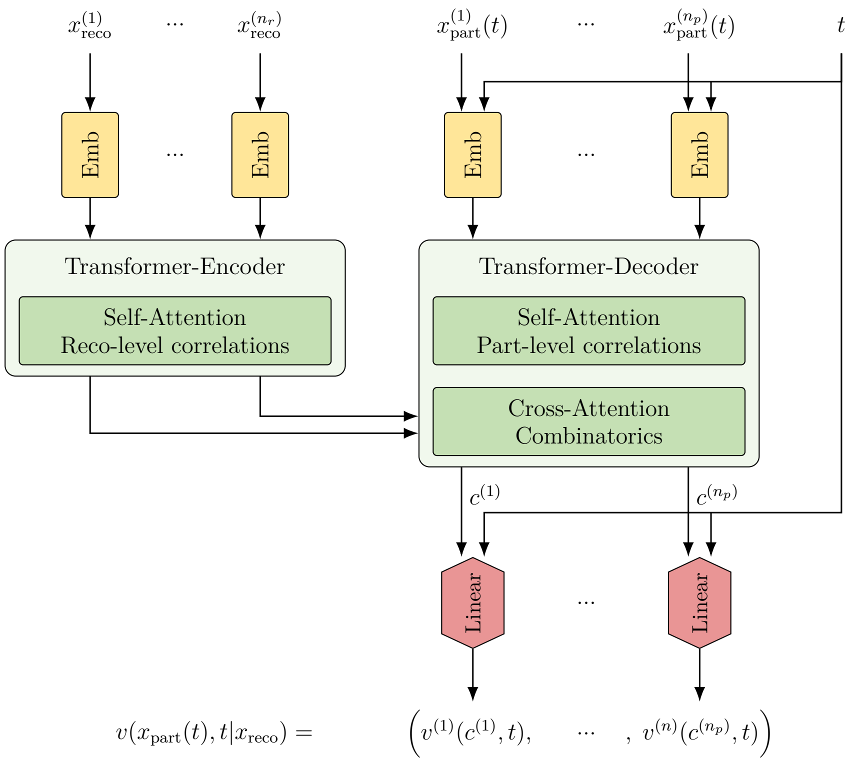


Figure 6.1: TraCFM architecture, combining the CFM generator with a Transformer encoder-decoder combination to improve combinatorics.

The rest of the derivation follows analogously to the DiDi derivation. The loss function is given by the MSE

$$\mathcal{L}_\text{CFM} = \left\langle [v_\theta((1-t)x_0 + t\epsilon, t, x_1) - (\epsilon - x_0)]^2 \right\rangle_{t\sim\mathcal{U}([0,1]),(x_0,x_1)\sim p(x_\text{part},x_\text{reco}),\epsilon\sim\mathcal{N}} . \quad (6.39)$$

After training, the CFM can unfold by sampling from the latent noise distribution and solving the ODE in Eq.(6.36) conditioned on the reco-level event we want to unfold. The crucial difference to DiDi is that this procedure allows us to unfold the same reco-level event repeatedly, each time from different noise as starting point, to sample the posterior distribution $p_\theta(x_\text{part}|x_\text{reco})$.

Transformer-CFM The velocity field can be encoded in any type of neural network, in that sense CFMs do not impose any architectural constraints. While linear layers already achieve high precision [1,69], we find that when dealing with complex correlations employing a Transformer network further improves results [2], similar to the INN vs Transfermer case.

Our TraCFM architecture encoding $v(x_\text{part}, t|x_\text{reco})$ is shown in Fig. 6.1. Its inputs are the reco-level event, the intermediate noisy diffusion state $x_\text{part}(t)$ and the time t. First, each of the reco-level and particle-level dimensions is individually mapped into a higher-dimensional embedding space. This is done by concatenating the kinematic variable with its one-hot-encoded position and filling with zeros up to the specified embedding dimension [2]. For the particle-level dimensions we also concatenate the time t to the vector before filling with zeros. We experimented with more sophisticated embedding strategies, but found no performance improvements. The reco-level embeddings are then fed to the transformer encoder, which encodes the correlations among them using a self-attention mechanism. The transformer decoder does the same for the particle-level dimensions. Finally, the updated embeddings are fed to a cross-attention block that learns to resolve the combinatorics between reco-level and particle-level objects and outputs a final condition $c^{(i)}$ for each particle-level dimension. A single linear layer, shared between all dimensions, maps this condition together with the time t to the individual velocity field components.

To unfold, we start with a sample from the latent distribution as $x_{\text{part}}(t = 1) = \epsilon \sim \mathcal{N}(0, 1)$ and solve the ODE Eq. (6.36) numerically. Notice that the transformer encoder has no time dependence, so we do not need to recalculate it at every function call.

Bayesian generative network The concept of Bayesian networks can be applied to generative networks by assigning an uncertainty to the learned underlying phase space density. This way, the network learns an underlying density to sample from, and an uncertainty on this density which it can report as an error of the unit-weight of each generated event [45, 161]. Because the loss of the normalizing flow is a maximized likelihood, the relation between the likelihood loss and the regularizing KL-divergence can be derived from Bayes' theorem. As an approximation to the full posterior, the error bars reported by Bayesian networks are a learned approximation to the true uncertainty on the phase space density.

Latent Variational Diffusion

To reduce the disparity between different parameterizations of the set of observables and to enable a more robust network, Latent Variational Diffusion [211] introduces a Variational Autoencoder to initially map observables from particle/parton phase space to a latent space. This particle encoder learns the mapping

$$x_{\text{part}} \;\to\; z = \textsc{Encoder}_{\text{part}}(x_{\text{part}}) \in \mathbb{R}^{D_{\text{Latent}}} \;. \tag{6.40}$$

It is implemented as a deep feed-forward network. This latent space can be fine-tuned for the diffusion step, allowing enhanced control over the generation process before mapping the result back to the observables.

To accommodate variable-length reco-level objects, an additional detector encoder maps them to a fixed-length latent vector

$$x_{\text{reco}} \;\to\; w = \textsc{Encoder}_{\text{reco}}(x_{\text{reco}}) \in \mathbb{R}^{D_{\text{Latent}}} \;. \tag{6.41}$$

It utilizes a deep feed-forward network for fixed-length inputs and a transformer encoder for variable-length inputs.

These latent observables provide the inputs for a conditional variational diffusion network [171]. VLD employs a continuous- time, variance-preserving, stochastic diffusion process with a noise-prediction parameterization for the score function. The governing stochastic differential equations are

$$\begin{aligned} dz &= f(z_t, t)\, dt + g(t)\, dw && \text{(VLD Forward SDE)} \\ dz &= \Big[f(z_t, t) - g^2(t) s_\theta(z, w, t)\Big]\, dt + d\bar{w}\;, && \text{(VLD Reverse SDE)} \end{aligned} \tag{6.42}$$

where f is the drift term of the SDE, g is the diffusion coefficient, and s is the score function, as in Eqs. (6.6) and (6.7). The drift and diffusion terms are parameterized through a learnable noise schedule $\gamma_\phi(t)$, which controls the diffusion rate. It is encoded in a monotonically increasing deep network as a function of t,

$$\begin{aligned} f(z,t) &= -\frac{1}{2}\left[\frac{d}{dt}\log\left(1+e^{\gamma_\phi(t)}\right)\right] z \\ g(t) &= \sqrt{\frac{d}{dt}\log\left(1+e^{\gamma_\phi(t)}\right)} \; . \end{aligned} \tag{6.43}$$

This simplifies the forward process to a time-dependent normal distribution, controlled by $\gamma_\phi(t)$, now interpreted as the logarithmic signal-to-noise ratio.

$$z_t \sim \mathcal{N}\left(\sigma(-\gamma_\phi(t))\, z, \sigma(\gamma_\phi(t))\, \mathbb{I}\right) \qquad \text{where} \quad \sigma(x) = \sqrt{\frac{1}{1+e^{-x}}} \; . \tag{6.44}$$

The diffusion score is parameterized via a noise-prediction network,

$$s_\theta(z,w,t) = \frac{\hat{\epsilon}_\theta(z_t,w,t)}{\sigma(-\gamma_\phi(t))} \; , \tag{6.45}$$

trained to predict the sampled noise used to generate the forward sample from the diffusion process [171, 211]. It is implemented as a deep feed-forward network, concatenating the three inputs before processing.

Finally, a decoder transforms the initial noisy latent particle representation back into phase space observables,

$$z_0 \;\rightarrow\; \hat{x}_{\text{part}} = \textsc{Decoder}(z_0) \; . \tag{6.46}$$

It is again implemented as a deep feed-forward network and outputs real-valued estimates of the observables.

All networks are trained in a end-to-end fashion using a unified loss which allows the encoders and decoders to fine-tune the latent space to the diffusion process, while accurately reconstructing the observables. We use a standard normal distribution as the prior over the final noisy latent vector, $p(z_1) \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$. The denoising network, $\gamma_\phi(t)$, is trained to minimize the variance of the following loss term, while all other networks are trained to minimize its expectation value:

$$\begin{aligned} \mathcal{L}_{\text{VLD}} &= \text{KL}[q(z_1|x), p(z_1)] && \text{(Prior Loss)} \\ &+ \left\langle \|\textsc{Decoder}(z_0) - x\|_2^2 \right\rangle_{q(z_0|x)} && \text{(Reconstruction Loss)} \\ &+ \left\langle \gamma'_\phi(t) \|\epsilon - \hat{\epsilon}_\theta(z,w,t)\|_2^2 \right\rangle_{\epsilon\sim\mathcal{N}(\mathbf{0},\mathbb{I}), t\sim\mathcal{U}(0,1)} && \text{(Denoising Loss)} \end{aligned} \tag{6.47}$$

6.2 Detector unfolding: Z+jets

6.2.1 Data and preprocessing

As a first test case for the various ML-Unfolding methods we use a new, bigger version of the public dataset from Ref. [63], now available at Ref. [257][3]. The events describe

$$pp \to Z + \text{jets} \tag{6.48}$$

production at $\sqrt{s} = 14$ TeV, simulated with Pythia 8.244 [33] with Tune 26. In contrast to the original dataset, detector effects are now simulated with the updated Delphes 3.5.0 [36], and the CMS tune, that uses particle flow reconstruction. The jets are clustered using all particle flow objects available at detector level and all stable non-neutrino truth particles at particle level. Jets are defined by the anti-k_T algorithm [109] with $R = 0.4$, as implemented in FastJet 3.3.2 [177]. The dataset contains around 24M simulated events, 20M for training and 4M for testing.

We focus on six observables describing the leading jet: mass m, width $\tau_1^{(\beta=1)}$, multiplicity N, soft-drop [259,260] mass $\rho = m_{\text{SD}}^2/p_T^2$ and momentum fraction z_g using $z_{\text{cut}} = 0.1$ and $\beta = 0$, and the N-subjettiness ratio $\tau_{21} = \tau_2^{(\beta=1)}/\tau_1^{(\beta=1)}$ [261]. For 0.8% of the events we map an undefined jet groomed mass $\log \rho$ or N-subjettiness ratio τ_{21} to zero.

The distributions are shown in Fig. 6.2. We apply a dedicated preprocessing to the jet multiplicity and the groomed momentum fraction. The jet multiplicity is an integer feature, which forces the network to interpolate, so we smooth the distribution by adding uniform noise $u \sim \mathcal{U}[-0.5, 0.5)$. This preprocessing can be inverted. The groomed momentum fraction features a discrete peak at $z_g = 0$ and sharp cuts at $z_g = 0.1$ and 0.5. We move the peak to $z_g = 0.097$ and add uniform noise $u \sim \mathcal{U}[0, 0.003)$. Next, we take the logarithm to make the distribution more uniform. We then shift and scale the

[3]For a comparison with classical unfolding methods, we refer to Refs. [63] and [258].

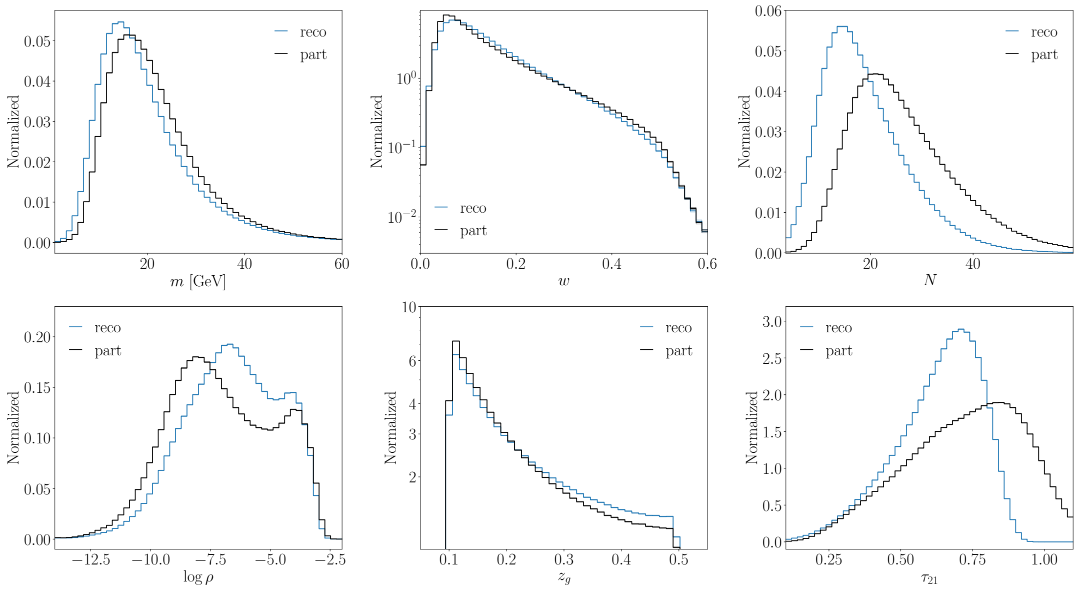


Figure 6.2: Subjet distributions for the Z+jets dataset, at the particle level, and at the reco level.

distribution to stretch from -1 to $+1$ and take the inverse error function to transform its shape to an approximate normal distribution. Finally, all six observables are standardized by subtracting the means and dividing by the standard deviations.

Also in Fig. 6.2, we show the effect of the detector simulation. They are most significant for the jet multiplicity, the groomed jet mass, and the N-subjettiness ratio. All these shifts are driven by the finite energy threshold of the detector.

6.2.2 Reweighting

As in Sec. 6.1, we start with the OmniFold reweighting on the $Z+\text{jets}$ dataset. We then introduce the Bayesian version (bOmniFold) and compare their performance. We train both networks for two different unfolding tasks. First, we evaluate their performance on the same dataset as the other generative networks, splitting it in two halves, but adding noise to one of them as described below. Then, we go back to the previous Pythia dataset and task the classifiers with learning the likelihood ratio between Pythia and Herwig. We use this ratio to reweight Herwig onto Pythia.

Training on Pythia with added noise

For this section, we employ the combination of Pythia with the updated Delphes version. We merge the training and test sets with 24.3M events, of which we use 10.9M for training, 1.2M for validation, and 12.2M for testing. In each of these splits, we label half of the events as Pythia 1 and the other half as Pythia 2. The classifier has to learn to reweight Pythia 1 onto Pythia 2.

If we train (b)Omnifold on this task, it will just learn a constant classifier value of 0.5, so we add Gaussian noise $\varepsilon \sim \mathcal{N}(0,1)$ to each of the raw features before preprocessing,

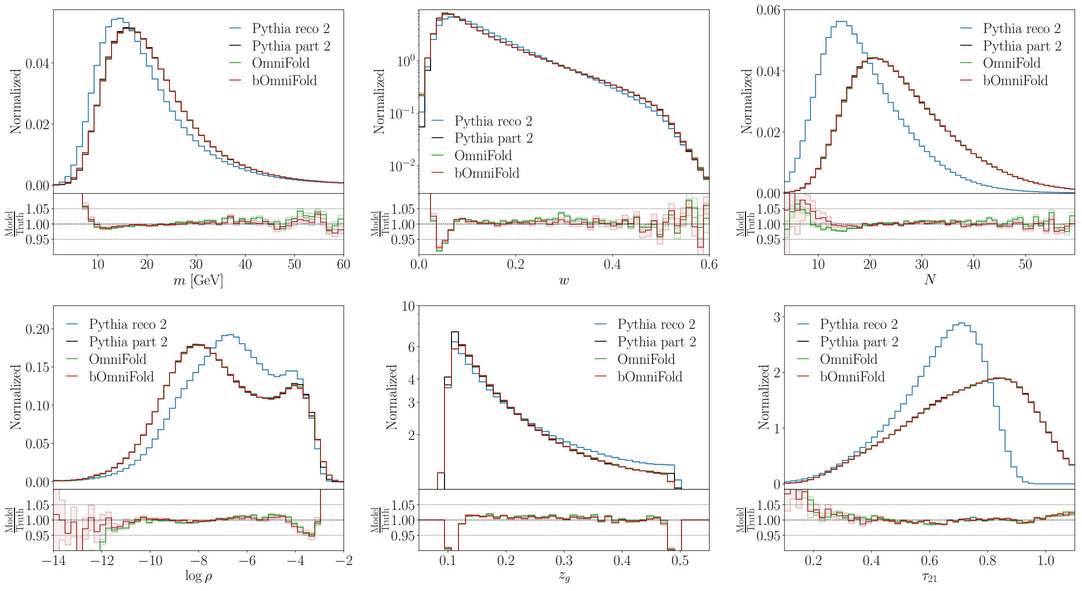


Figure 6.3: Unfolded distributions from event reweighting using OmniFold and bOmniFold. The bOmniFold error bar is based on drawing 20 Bayesian samples. For OmniFold the error bar represents the bin-wise statistical uncertainty.

	m [GeV]	w	N
OmniFold	0.59098 / **0.12493** / 13.72203	0.01001 / **1.62601** / 2.99618	0.67919 / 0.03034 / 18.47942
bOmniFold	**0.37180** / 0.14208 / **9.89718**	**0.00542** / 1.64286 / **2.24587**	**0.22693** / **0.02176** / **4.97982**
	$\log\rho$	z_g	τ_{21}
OmniFold	0.40320 / 0.72494 / 15.60005	0.01550 / 15.30356 / 4.81947	**0.00931** / 0.02746 / **1.40143**
bOmniFold	**0.12501** / **0.67605** / **5.59003**	**0.01109** / **15.27470** / **4.51572**	0.00956 / **0.02183** / 1.54405

Table 6.1: Metrics evaluating the performance of the different unfolding networks, for each of the one-dimensional kinematic distributions. We show the Wasserstein 1-distance ($\times 10$), the triangular distance ($\times 1000$), and the energy distance ($\times 1000$).

scaled by the standard deviation of the respective feature σ_x and an additional custom factor f to modify the relative importance of the noise,

$$x \to \tilde{x} = x + f \cdot \sigma_x \varepsilon \qquad \text{with} \qquad \varepsilon \sim \mathcal{N}(0,1) \; , \tag{6.49}$$

where we use $f = 0.1$.

We train OmniFold (13k parameters) and its Bayesian-network counter part bOmniFold (2×13k parameters) with identical settings for 30 epochs. The unfolded distributions are shown in Fig. 6.3. While this reweighting task might not be realistic, it defines an illustrative benchmark for the performance of an unfolding network. For each of the one-dimensional kinematic distributions, the agreement between the unfolded and true particle-level events is at the percent level over most of the phase space. The only exceptions are sparsely populated tails with too little training data, or sharp features with limited resolution. The differences between the OmniFold and bOmniFold results are even smaller. A selection of summary statistics are presented in Tab. 6.1, where we show the Wasserstein 1-distance, the triangular distance, and the energy distance for the six kinematic observables. The two methods were not separately optimized, we just started with a generic OmniFold setup and supplemented it with the Bayesian network features. Uncertainties on the statistics are not included in these illustrative metrics.

For the uncertainties, we see that it tends to cover the deviation of the unfolded distributions from the truth target towards increasingly sparse tails. Far in the tails, where there is too little training data altogether, the networks learn neither the density nor an error bar on it.

Reweighting Herwig onto Pythia

For a more realistic (b)OmniFold task, we go back to the original Pythia dataset, for which we also have a Herwig [262] version with the same version of Delphes, as introduced in Ref. [63]. We train (b)OmniFold for 500 epochs on 2M events, and test on 664k events [68].

First, we show the losses as a function of the training in Fig 6.4. This comparison shows the challenge of the classifier training, which rapidly overtrains after about 20 epochs. This behavior does not appear in the previous study with noisy Pythia events and is due to the smaller training data for the Herwig reweighting. For large numbers of training epochs, the loss on the validation dataset indicates a decreasing performance due to overtraining. For applications which require an LHC-level of precision, such overtraining may become a problem. It can be avoided, for instance, using regularization techniques, such as dropout. Both of these mechanisms are part of the Bayesian network architecture, in case of the regularization with a strength given by Bayes' theorem. In Fig. 6.4 we see that the bOmniFold training continues to improve even after a large number of epochs, with no overtraining. Interestingly, bOmniFold has larger epoch-to-epoch fluctuations and has a worse minimum validation loss than OmniFold, but does not show signs of overtraining. This illustrates a potential tradeoff between accuracy and stability.

We verified that the unfolded observables between OmniFold and bOmbiFold are in agreement. Because of the difference between the training data, or prior, and the data we then unfold, the true particle-level distributions are not exactly reproduced. The interesting feature of the bOmniFold training is that it has suppressed tails in the weight distribution with respect to OmniFold, as shown in Fig. 6.5, even though both networks learn the same reweighting map. Large and small weights lead to undesired statistical dilution of the dataset, and it will be interesting to explore in the future the interplay between statistical dilution and accuracy.

6.2.3 Mapping distributions

The same subjet unfolding can be tackled with distribution mapping, using the Schrödinger Bridge and Direct Diffusion, both introduced in Sec. 6.1.2. The implementation of the Schrödinger Bridge follows the original Pytorch [173] implementation [68]. The noise prediction network is implemented using a fully connected architecture with additional skip connections, specifically using six ResNet [263] blocks, with each residual layer connected to the output of a single MLP through a skip connection. The Bayesian version replaces the original MLPs. The training uses the Adam [114] optimizer. The total number of trainable parameters is around 2M split equally between the mean and standard deviation of the trainable weights.

During data generation, we sample using the MAP prediction, i.e. fix every network weight at the learned mean. Uncertainties are derived by sampling 50 times from the learned weight distributions. In Fig. 6.6, we quantify the agreement between the unfolded and truth one-dimensional kinematic distributions. The unfolding performance can be

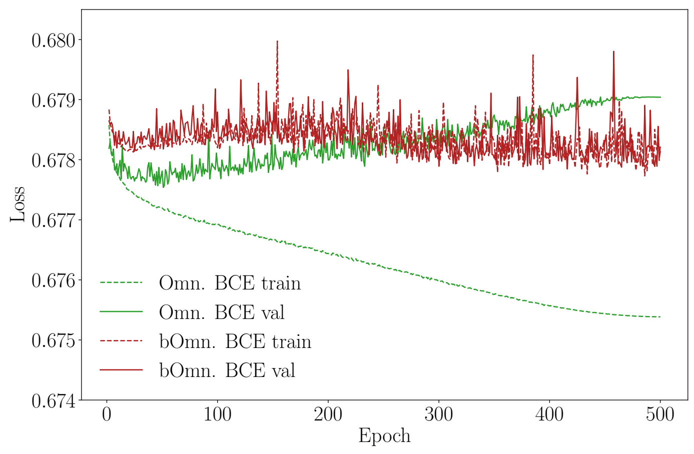


Figure 6.4: BCE losses during training for 500 epochs for Omnifold (green) and bOmnifold (red), for Herwig-to-Pythia reweighting.

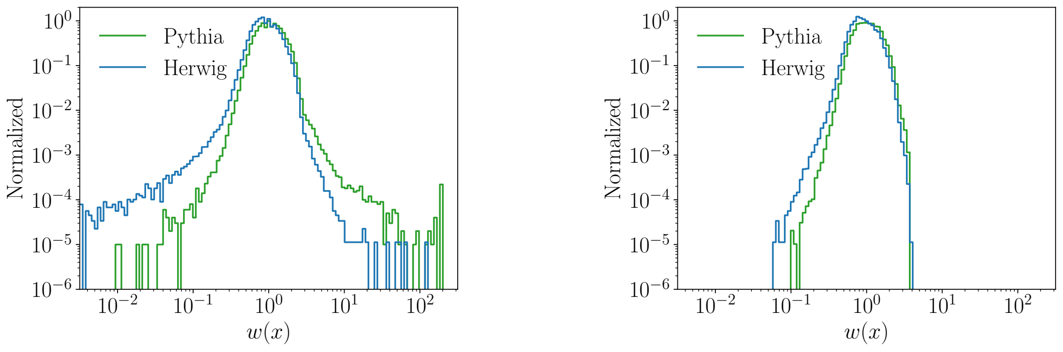


Figure 6.5: Weight distribution (clipped at 200) in the training set for Herwig-to-Pythia reweighting: OmniFold (left) vs bOmniFold (right). For each network we histogram the weights for the Herwig and Pythia data points.

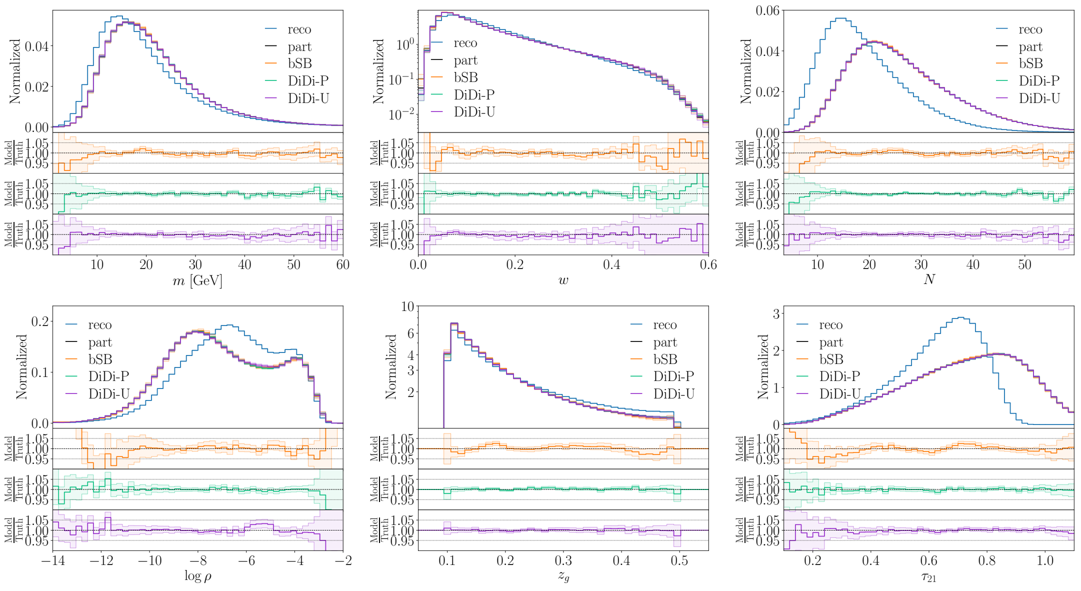


Figure 6.6: Unfolded distributions from distribution mapping, using the Schrödinger Bridge and DiDi. The Bayesian error bars are based on drawing 50 samples.

compared to the noisy reweighting benchmark in Fig. 6.3. The agreement between unfolded and truth-level observables is still precise to the percent level. Notably, the largest deviations from the truth distribution occur in the low-statistics edges, while the bulks of the distributions are well described by the generative mapping, and the deviations from the truth are well covered by the Bayesian uncertainty.

An alternative method for the same tasks is Direct Diffusion. We encode the velocity field in a standard Bayesian MLP, after not seeing better results with more advanced networks. Again, we implement the network in Pytorch, train it using the Adam optimizer, use the MAP prediction, and draw 50 Bayesian weight samples to estimate the uncertainties. We use the same setup for paired and unpaired DiDi. The only difference is the reshuffling of the reco-particle pairings at the beginning of each epoch in the unpaired setting. The network size comes to about 3M parameters, 1.5M weights each associated with the mean and the standard deviation.

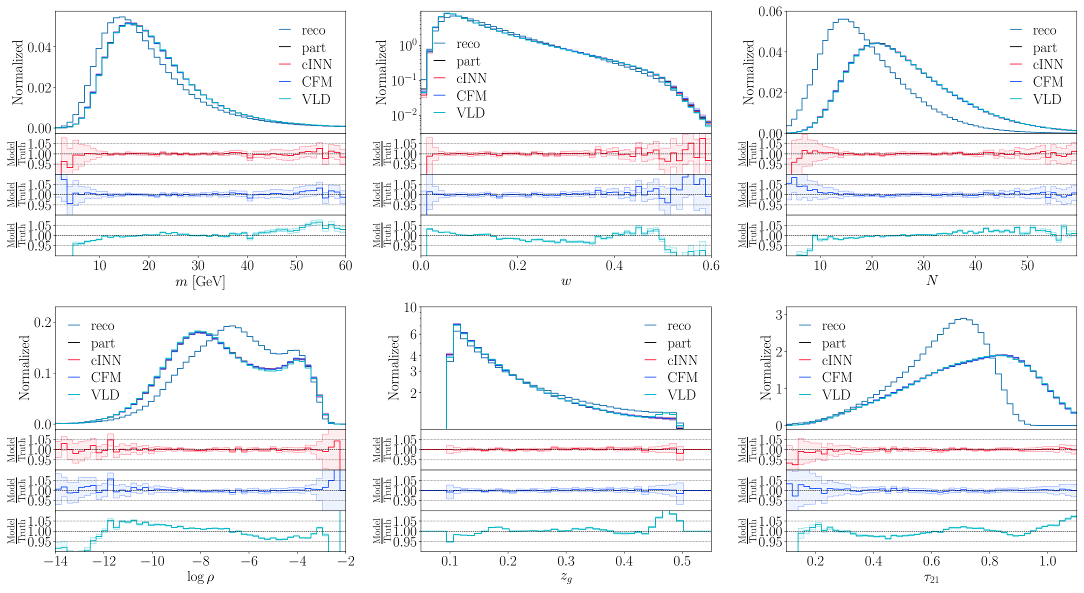


Figure 6.7: Unfolded distributions from conditional generation, using cINN, CFM and VLD. For cINN and CFM, the Bayesian errors are based on drawing 50 samples, and the MAP estimate is obtained by unfolding each event 30 times. For VLD, we show the bin-wise mean and standard deviation of 33 unfoldings.

The results are compared to the Schrödinger Bridge in Fig. 6.6. Both variants of Direct Diffusion learn the observables with percent precision over the entire phase space, and better than that in the well-sampled bulk. For the central prediction, the paired training data makes the unfolding slightly more precise and more stable.

A difference between paired and unpaired DiDi is that the latter might be slightly less stable and learns significantly larger Bayesian uncertainties. This is consistent over several trainings. At the level of kinematic distribution we do not observe any shortcoming for unfolding through distribution matching, and the difference between paired and unpaired training data is minor. We will come back to the conceptual difference in Sec. 6.2.5.

6.2.4 Generative unfolding

The third unfolding method we study is based on learned conditional probabilities, as defined in the statistical description of unfolding. It relies on paired training data. Differences appear when we vary the generative network architecture used. We skip the original GAN implementation [62], because more modern generative networks have been shown to learn phase space distributions more precisely [1, 45]. The cINN [64] is implemented in PyTorch [173] and uses the FrEIA library [264] with RQS coupling blocks. By default, we use its Bayesian version [161], which tracks the uncertainty in the learned phase space density as variations of unit event weights. Our CFM [1] encodes the velocity field in the same linear layer architecture as DiDi, including the Bayesian version. Predictions are obtained by unfolding each event 30 times with the MAP weights, and the uncertainties are obtained from drawing 50 sets of weights from the Bayesian network.

VLD is implemented and trained using the same JAX codebase released alongside Ref. [211]. Observables are first pre-processed so that each marginal distribution follows a standard normal via a quantile transform. Predictions are generated using the DPM++

multi-step solver [265] with 1000 inference steps and the learned schedule. Unlike the other models, VLD is not implemented as a Bayesian network. Uncertainties are estimated by sampling each unfolding 33 times using a different seed for generating the prior noise.

In Fig. 6.7 we show the results from the cINN, CFM, and VLD. As for the (b)OmniFold reweighting and the distribution matching, all kinematic distributions are reproduced at the percent level or better. While the performance of the cINN and the CFM are very similar, the VLD approach shows slightly larger deviations from the target distributions.

6.2.5 Learned event migration

For the generative network methods, it can be instructive to examine the learned map between reco events and truth events. In the top panels of Fig. 6.8 we start with the migration described by the paired events from the forward detector simulation. We show three of the kinematic distributions defined in Fig. 6.2. The results for the other distributions lead to the same conclusions. For the jet mass, the multiplicity, and τ_{21} we see that the optimal transport defined by the detector is quite noisy. While for the jet mass most events form a diagonal with little bias, the jet multiplicity shows a linear correlation with a non-trivial slope, and τ_{21} indicates a saturation effect in the forward direction $x_\text{part} \to x_\text{reco}$, such that $\tau_{21}(x_\text{reco})$ does not reach one.

In the next row we show the results from the Schrödinger Bridge, which is similar, but slightly noisier than DiDi trained on paired events, shown in the third row. Using paired events, these generative networks learn a very efficient diagonal transport map, with a spread that is more narrow than the actual detector. The main features of the detector truth are reproduced well.

Next, we see that the unpaired DiDi network again learns an efficient transport map, but with a significantly broader spread than the same network trained on paired events. The reason is that ignoring the event pairing leads to a noisier training, but again reproducing the main features of the detector. We emphasize that unpaired training seems to bring DiDi-like implementations closer to describing the actual detector, but this is an artifact in that the detector mapping is noisier than the optimal transports from distribution mapping, and training on unpaired samples is also noisier, but the two are not positively related. Finally, we show the transport learned by the conditional CFM networks. Not shown are the corresponding cINN and VLD results, which are visually identical to the CFM results. The conditional generative models indeed learn the correct detector transport from the paired events, indicating that conditional generative networks indeed encode the conditional probabilities from Eq.(6.27).

6.2.6 Classifier check

Finally, there is the question if the learned distributions have failure modes that cannot be seen from the marginal distributions. Following [162, 178, 266], we systematically search for mis-matched correlations using a trained binary classifier between the true training events and the same number of unfolded events. Using Eq.(6.2) this classifier can be turned into a re-weighting function $w(x)$, evaluated for each data point in phase space.
We train individual classifiers for each of the discussed methods, the results are presented in Fig. 6.9. The top left plot shows the phase space weights w for the three distribution

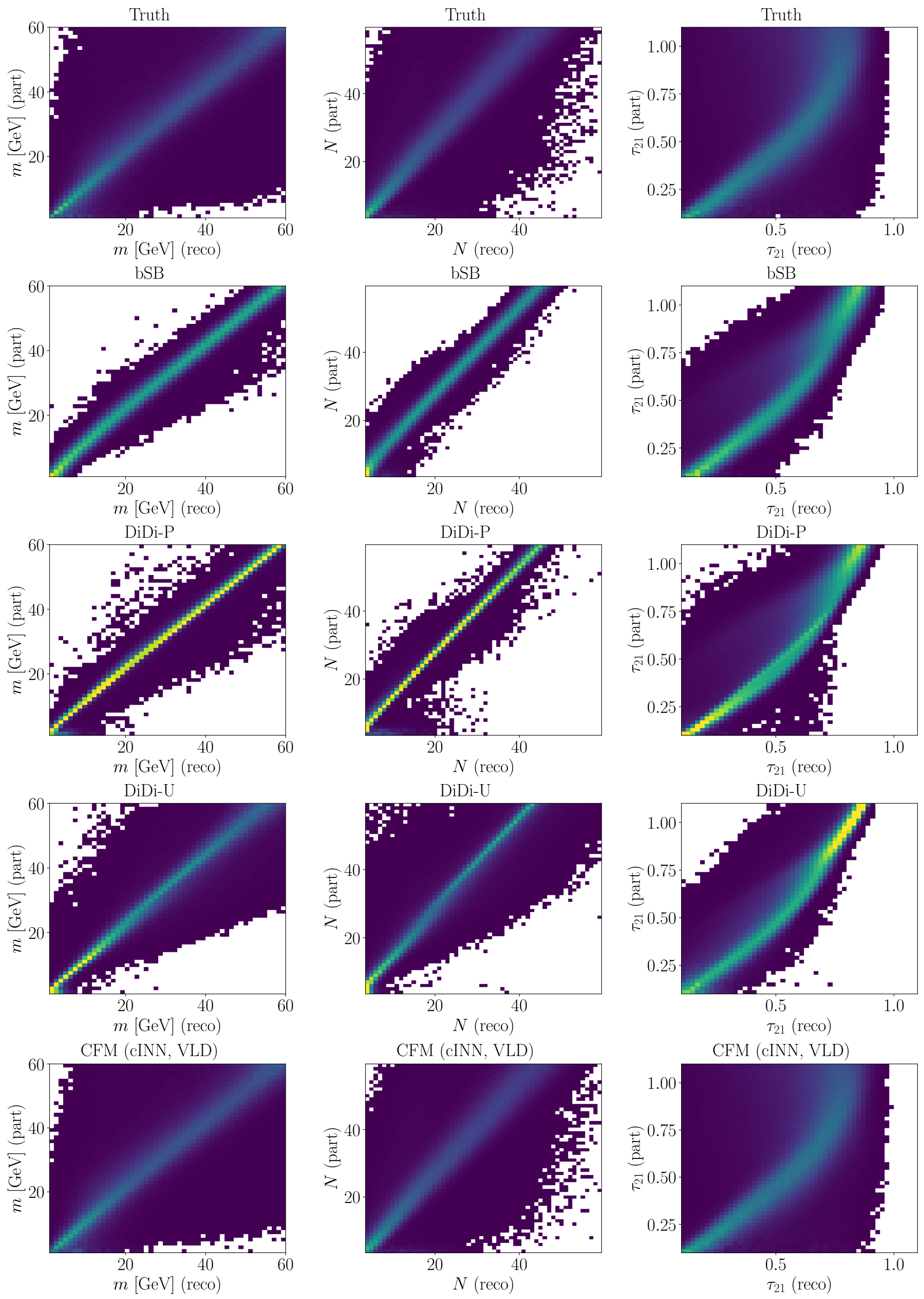


Figure 6.8: Migration maps for three representative distributions. From the top: forward detector simulation, Schrödinger Bridge, paired DiDi, unpaired DiDi, and CFM/cINN/VLD, which are all looking identical. The bin contents are normalized such that each row sums to one.

mapping methods bSB, DiDi-P and DiDi-U. For all three networks, the dominant feature is a sharp peak in $w \approx 1$, showing overall excellent agreement between the learned unfolding and the truth distribution. A tail towards higher weights indicates the existence of phase space regions that are underpopulated by the network, while lower weights indicate an overpopulated region. While for all three networks weak tails in both directions exist, they are suppressed by several orders of magnitude compared to the peak around unit weights. The right plot shows the same weight distribution zoomed into the area around $w(x) = 1$ to better visualize the peak. Comparing the three methods, we find a slightly sharper peak for the DiDi networks compared to the bSB implementation.

The bottom row of Fig. 6.9 shows the same two plots for the three generative networks, cINN, CFM, and VLD. Again, for all three methods almost all events fall into a sharp peak around weight one. Tails towards high and low weights exist, but are again strongly suppressed compared to the peak and only visible due to the logarithmic scale of the y-axis. Comparing the three networks, we find a slightly larger spread for the VLD. The right plot shows the zoomed-in version of the same distribution. Here we see that the density in the peak is about one order of magnitude larger for the CFM and cINN, compared to the VLD.

Finally, we can compare the distribution mapping methods in the top row to the generative networks in the bottom row. They show overall very similar performance, indicating that despite their different migration patterns shown in Fig. 6.8, both model classes can recover the unfolded distribution with high precision. In the high resolution plots on the right, we find a slightly sharper peak around unit weights for the CFM and the cINN, indicating the overall best agreement with the truth distribution, as measured by the trained classifier.

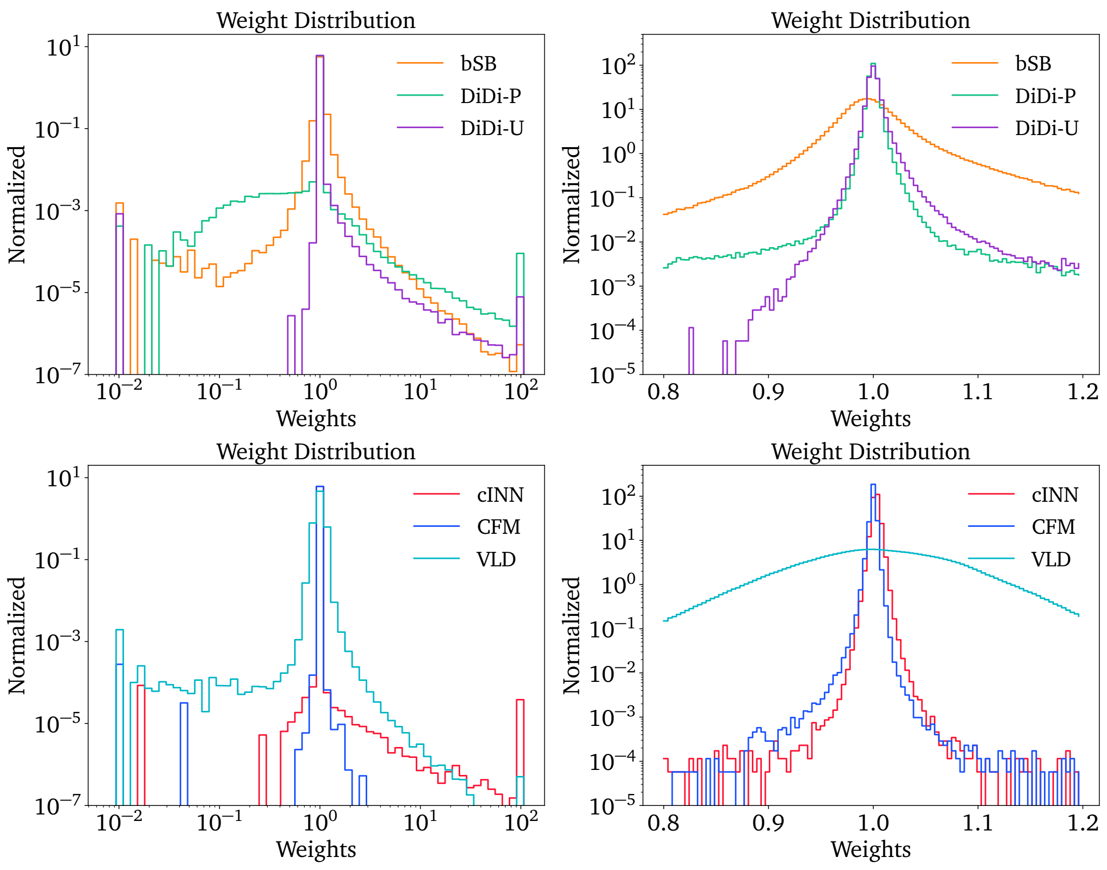


Figure 6.9: Classifier weight distribution on the Z+jets dataset. The top row shows the results for the distribution mapping methods, the bottom row for the generative methods. The left plots show the weight distribution over a large range, the right plots are zoomed in to resolve the area around $w = 1$.

6.3 Unfolding to parton level: top pairs

6.3.1 Data

As a second benchmark we apply ML-methods to top quark pair production, unfolding from reco-level to parton level, i.e. the level of the top quarks and their decay products from the hard scattering, before undergoing hadronization. While more physics assumptions/approximations are required for this type of unfolding, it is often performed by ATLAS and CMS [267–274]. Parton-level results are extremely useful, for instance, to combine measurements into a global analysis [238, 239], extract SM parameters [274, 275], or to compare new theory predictions without requiring these to be matched to parton showering programs [276].

The task is to map reco-level 4-momenta to parton-level 4-momenta defined by the $2 \to 2$ scattering and subsequent decays, in our case [211]

$$q\bar{q}/gg \to t\bar{t} \to (b\ell^+\nu_\ell)\ (\bar{b}qq) \qquad \text{with} \quad \ell = e, \mu \quad , \quad q = u, d, s, c\ , \tag{6.50}$$

plus the charge-conjugated process. The events are simulated with Madgraph5 3.4.2 [32] at $\sqrt{s} = 13$ TeV and with a top quark mass $m_t = 173$ GeV. One of the W bosons decays leptonically, the other hadronically. Showering and hadronization are simulated with Pythia 8.306 [33], and detector effects with Delphes 3.5.0 [36] with the standard CMS card. We again reconstruct jets using the anti-k_T algorithm [109], now with $R = 0.5$ and a p_T-dependent b-tagging efficiency. Leptons and jets are subject to the acceptance cuts $p_T > 25$ GeV and $|\eta| < 0.25$. We only keep events with exactly one lepton, at least 2 b-tagged jets, and at least two more jets.

This second benchmark process is technically more challenging than the Z+jets unfolding in terms of the six subjet observables, because of the higher phase space dimensionality and because we can no longer directly match reco-level and parton-level observables. To reconstruct the hard scattering, the network has to learn the non-trivial combinatorics as well as complex correlations reflected in the intermediate mass peaks. We focus on a comparison of the different generative unfolding methods, which reproduce the forward simulation in their event-wise migration, but are most challenging from an ML-perspective. As before, we postpone the important question of model dependence to a later paper.

6.3.2 Generative unfolding

As a first attempt, we employ a straightforward phase space parametrization for the six top decay products,

$$\begin{aligned}
&(p_{T,b_\ell}, \eta_{b_\ell}, \phi_{b_\ell},\ p_{T,\ell}, \eta_\ell, \phi_\ell,\ p_{T,\nu}, \eta_\nu, \phi_\nu) \\
&(p_{T,b_h}, \eta_{b_h}, \phi_{b_h}, m_{q_1},\ p_{T,q_1}, \eta_{q_1}, \phi_{q_1},\ p_{T,q_2}, \eta_{q_2}, \phi_{q_2})\ .
\end{aligned} \tag{6.51}$$

The lepton masses are common to all events, and we set them to zero at the level of our simulations. The bottom jets are generated with a common finite bottom mass. For the remaining jets, we have to keep track of the charm mass in the corresponding charm jets. This leads to a binary degree of freedom, in addition to the 18 standard phase space dimensions.

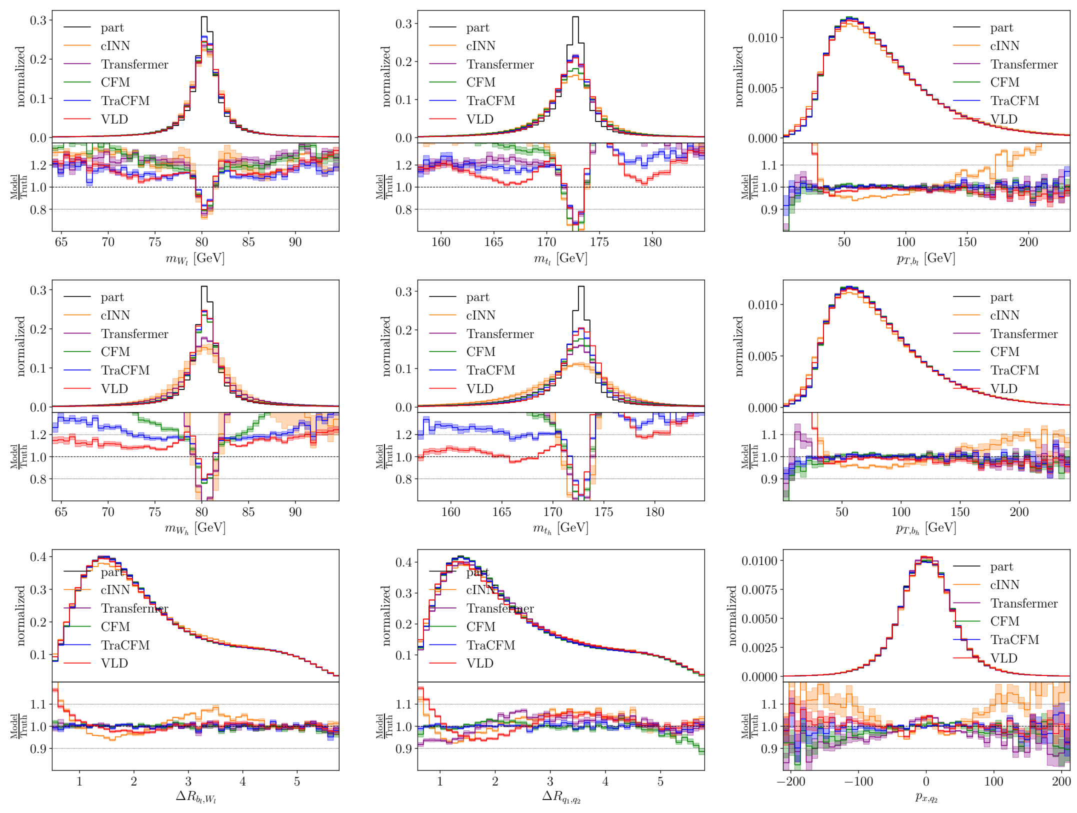


Figure 6.10: Unfolded top pair distributions from conditional generation using the naive phase space parametrization of Eq.(6.51). For the Bayesian cINN, Transfermer, CFM and TraCFM the error bars are based on drawing 50 samples. For the VLD the error bars are given by unfolding each event 128 times and showing the bin-wise mean and standard deviation.

While at parton level all events have the same number of particles, at reco level we see a variable number of jets. Jets are produced in top and W-decays, but also in initial-state and final-state radiation, multi-parton interactions, underlying event, or pileup. Their number also strongly depends on the acceptance cuts. Naively, these additional jets are not expected to carry information on the hard process. However, they can sometimes cause events to pass selections by replacing top decay jets which do not pass the acceptance cuts, or lead to challenging event reconstruction due to jet combinatorics [2]. This means we cannot just ignore them.

While the standard cINN and CFM require a fixed-dimensionality condition, their transformer variants can handle variable dimensionality. Alternatively, we could employ an embedding network to overcome this limitation. Testing the impact of additional jets on our specific unfolding task with the Transfermer and TraCFM networks, we find that they do not benefit from additional reco-level jets significantly. Consequently, we restrict ourselves to a fixed maximum number of particles at reco-level for these networks. The particles we include in an ordered vector are the lepton, the missing transverse momentum, the two leading b-jets, and the two leading light-flavor jets. VLD does naturally includes all jets in an un-ordered fashion. The masses and transverse momenta of the particles are log-scaled before feeding them to the network.

We again use an RQS-cINN implementation from the FrEIA library [264], in the last block we replace the linear layers with Bayesian layers to track the network uncertainties.

The CFM encodes the velocity in a standard MLP network. The time t is embedded to a higher dimension using a random Fourier feature encoding [277] before being concatenated to the other network inputs, as we found that this improves results in higher-dimensional tasks. Following [278] we only make the last network layer Bayesian, as too many Bayesian weights can make the training of large networks unstable. For the two Transformer-based networks we employ the standard PyTorch Transformer implementation. The attention blocks are then followed by a single Bayesian RQS block or a single Bayesian linear layer for the Transfermer and the TraCFM respectively. For the TraCFM we employ the same time embedding as for the CFM and concatenate the encoded time to the transformer output before feeding it to the final layer.

The results for the cINN, its Transfermer variant, the CFM, and its TraCFM variant along with VLD are shown in Fig. 6.10. In the plots we focus on the challenging distributions, mainly the intermediate mass resonances and the angular correlations. We have checked that the rest of the kinematics is reproduced mostly at the percent level by the generative networks. Generally, we find that the lepton and neutrino kinematics are learned slightly better than the quark kinematics. As shown in the top rows, the correlations describing the intermediate particles are not learned as well. For the resonances, all networks struggle. Because they only require to correlate two independent 4-momenta, the W-peaks are learned a little better than the top peaks. Also, the leptonic decay is learned better than the hadronic decay. Altogether, the Transformer-enhanced networks perform better than the CFM, which in turn beats the cINN.

6.3.3 Generative unfolding using physics

The choice of phase space parametrization can be crucial for the performance of generative networks [209]. To solve the problems with intermediate on-shell propagators, we employ the dedicated top-mass parametrization proposed in Ref. [212]. It directly predicts the top and W-kinematics and makes the simpler decay kinematics accessible via correlations. As the phase space basis we choose the top 4-momentum in the lab frame, three components of the W 4-momentum in the top rest frame, and two (three for the hadronic case) components of the first W-decay product in the W rest frame,

$$\begin{aligned}&(m_t, p_{T,t}^L, \eta_t^L, \phi_t^L,\ m_W, \eta_W^T, \phi_W^T,\ \eta_\ell^W, \phi_\ell^W)\\ &(m_t, p_{T,t}^L, \eta_t^L, \phi_t^L,\ m_W, \eta_W^T, \phi_W^T,\ m_{q_1}, \eta_{q_1}^W, \phi_{q_1}^W)\ .\end{aligned} \tag{6.52}$$

The superscripts L, T, W denote the rest frames. We then employ a Breit-Wigner mapping using the mass values in the event generator

$$\sqrt{2} * \text{erfinv}\left[\frac{2}{\pi}\arctan(m - m_\text{peak})\right]\ , \tag{6.53}$$

to turn the sharp mass peaks into a Gaussian-like shape.

The results with this paramerization are shown in Fig. 6.11. We drop the cINN and focus on the better CFM implementations. Now that the intermediate masses are directly predicted by the networks, we reproduce them within a few percent. The kinematics of the decay particles, now correlations between the directly predicted dimensions, are also faithfully modeled. Because the learning task has become easier, the difference between the CFM and the TraCFM is smaller. So physics helps, as it tends to.

Similar to Sec. 6.2.5, we again train a classifier to distinguish the generated events from the training data truth. In Fig. 6.12 we show the distribution of learned classifier weights for the three generative unfoldings. In this case, we see that while the one-dimensional kinematic distributions look similarly good for the three models in Fig. 6.11, there are significant differences in the precision with which the generative networks reproduce the multi-dimensional target distributions. The fact that all weight distributions are peaked around $w \approx 1$ and that the tails on the parton-level training and generated datasets are identical indicates that there is no definite failure mode [178]. On the other hand, the level of agreement is significantly improved, going from the VLD to the CFM, and then adding the transformer feature of the TraCFM to encode combinatorics [2]. For the latter, we again reach the percent-level precision we observed for the Z+jets detector unfolding in Sec. 6.2.

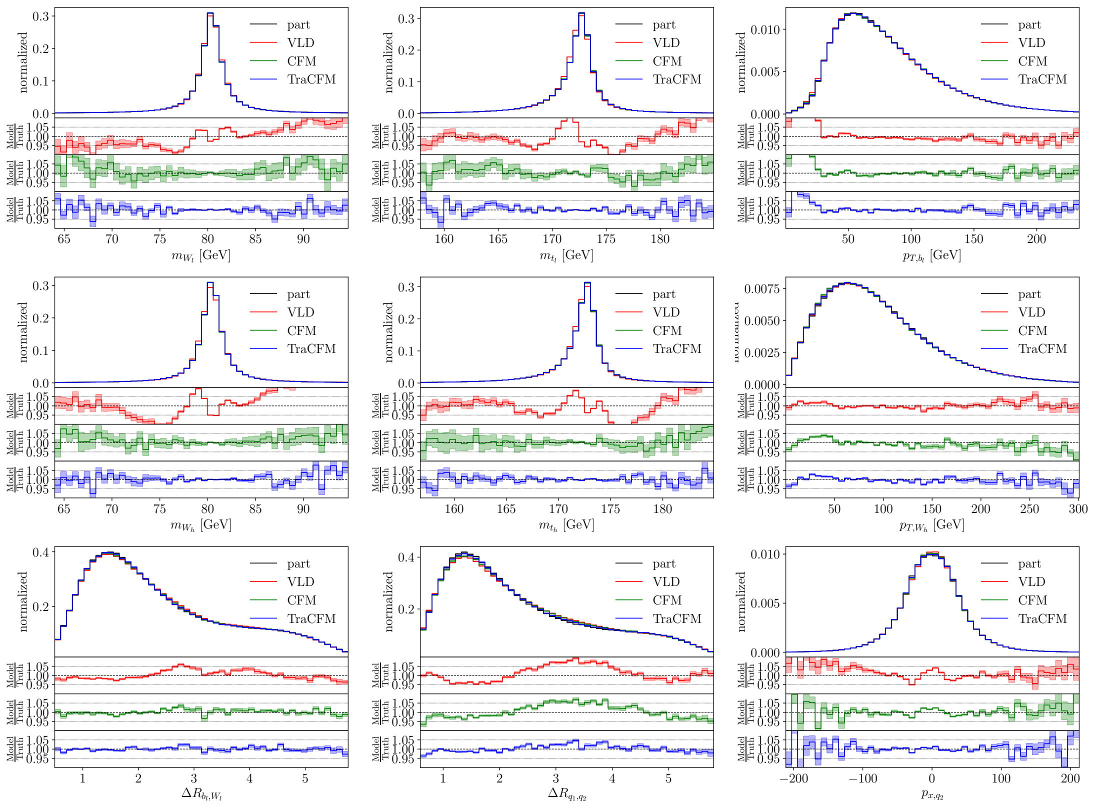


Figure 6.11: Unfolded top pair distributions from conditional generation using the dedicated phase space parametrization of Eq.(6.52). For the Bayesian cINN, Transfermer, CFM and TraCFM the error bars are based on drawing 50 amples. For the VLD the error bars are given by unfolding each event 32 times and showing the bin-wise mean and standard deviation.

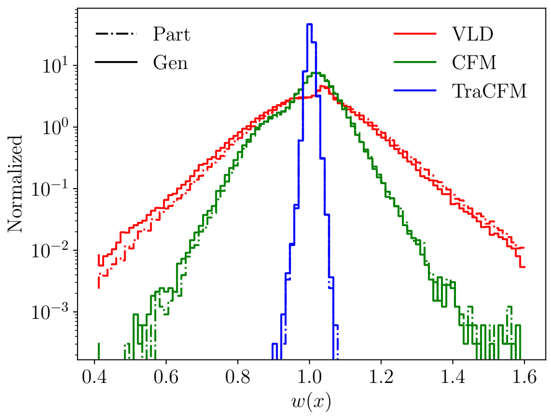


Figure 6.12: Weight distributions from a trained classifer between true and generated top pair events. The corresponding AUC values are 0.53 for the VLD, 0.51 for the CFM and 0.501 for the TraCFM.

6.4 Outlook

Machine learning is changing the face of LHC physics, and one of the most exciting developments is that it enables unbinned, high-dimensional, precise unfolding. This includes detector unfolding as well as inverting the first-principle simulations to the parton level. There exist three different ML-methods and tools for such an unfolding, (i) event reweighting or OmniFold, (ii) mapping distributions, and (iii) conditional generative unfolding. All these methods have been developed to a level, where they are ready to be further studied for use by the LHC experiments. In this paper, we give an overview of the different methods and corresponding tools, including an update to the most recent neural network architectures and a rough comparison of the strengths of the different methods.

Our first task is to unfold detector effects for a set of six subjet observables in Z+jets production. Here, reweighting-based unfolding, a supervised classification task, reproduces all true particle-level distributions and defines a precision benchmark shown in Fig. 6.3. A new Bayesian variant of OmniFold might provide complementary strengths to the existing method.

Alternatively, distribution mapping can be trained on matched events efficiently. We found that the (Bayesian) Schrödinger Bridge and Direct Diffusion implementations consistently provide high performance, shown in Fig. 6.6. Alternatively, distribution mapping can be trained on unmatched data, which limits its ability to reproduce the actual detector effects, but can be useful when one is missing matched training data.

Third, unfolding by learning and sampling conditional inverse probabilities is ideally suited to model complex detector effects, but also the most challenging network architecture. We have compared a series of tools, including invertible networks without and with a transformer encoding, as well as diffusion networks without and with a transformer, and with an enhanced latent representation. In Fig. 6.7 we have shown that the conditional generative tools match the precision of distribution mapping. In addition, we have shown to which level the different methods learn the event migration or optimal transport defined by the forward detector simulation, rather than an abstract mapping defined by the network architecture.

Finally, we have applied our unfolding methods and tools to invert $t\bar{t}$ events to the hard process of top pair production with subsequent decays. Here, correlations pose a serious challenge, specifically the intermediate mass peaks. We have found that they can be learned precisely once we represent the phase space in a physics-inspired kinematic basis, as can be seen in Fig. 6.11. In addition to the physics pre-processing, the combination of a diffusion model with a transformer guaranteed the best performance among the conditional generative unfolding networks.

Altogether, we have shown a multitude of different methods and tools for ML- unfolding, with dedicated individual strengths.[4] All of them are ready to be studied further in the context of LHC analyses. Their complementarity is a strength for building confidence in advanced tools for high-dimensional cross section measurements. Future work will focus on how the different approaches handle prior dependence, backgrounds, and acceptance effects, as well as a comprehensive treatment of the uncertainties associated with these steps.

[4]Many of the codes used in this paper will be made publicly available, together with a set of tutorials accompanying Ref. [23].

Chapter 7

Generative Unfolding with Distribution Mapping

The research presented in this chapter is based on work in collaboration Anja Butter, Sascha Diefenbacher, Vinicius Mikuni, Benjamin Nachman, Sofia Palacios Schweitzer and Tilman Plehn, and has been previously published in Ref. [4]. The content is similar or identical to the content of this article.

Differential cross sections are the central currency of exchange in particle and nuclear physics. These objects connect theory with experiment, allowing for most of the data analysis performed at particle colliders. Simulations of collisions from the smallest distance scales up to the macroscopic size of detectors enable comparisons of synthetic and real data [30]. Comparing fully simulated and real events directly has a number of advantages, but it also requires access to the experimental data and prespecifying the physics questions and calculations. An alternative approach that trades off precision with versatility/longevity is *unfolding*, where the data are corrected for detector effects. In this way, the resulting differential cross sections are independent of their detector and can be interrogated with a number of questions and calculations, even those that were not known at the time of the measurement.

Classical unfolding methods act on histograms to produce binned, differential cross section measurements [57, 58, 60, 279]. These tools have been used for a multitude of measurements over decades and have in turn been used for many downstream analyses. However, traditional approaches are fundamentally limited in scope. Recently, there have been a number of proposals to use modern machine learning (ML) to address these limitations [26, 59]. ML-unfolding is either based on reweighting [63, 112, 280, 281] or generative networks [3, 61, 62, 64, 65, 67–69, 211, 212, 251, 282], and it allows for unbinned measurements with a large number of input and output dimensions. This greatly increases the long-term utility of the measurements, as the cross section of new observables can be automatically extracted from the result. Experimental results with these tools are already being published, including measurements based on the OmniFold method [63, 112] from H1 [244–247], LHCb [248], CMS [249], STAR [250], and ATLAS [29]. These results are promising, but due to the ill-posed nature of the problem, it is essential to have alternative methods.

In this paper, we focus on generative unfolding, and in particular, on the key step of sampling from likely particle-level (gen-level) events given detector-level (reco-level) events[5]. One class of generative model-based approaches use distribution mapping, whereby the experimental events are morphed to match the corresponding gen-level events. By starting from the experimental events directly, the generative model only needs to move the events a little (assuming a precise detector), whereas other generative approaches need to map generic Gaussian random variables into the data distribution. Previously, two distribution mapping approaches were proposed, both based on conditional diffusion models [122]: one using Schrödinger Bridges (SBUnfold [68]) and one using Direct Diffusion (DiDi [3]). Previous work showed that these techniques showed excellent performance on the marginal distributions of the target cross sections, but they were not able to preserve the conditional distributions of the detector response. This could lead to a strong dependence on the gen-level simulation and is thus undesirable. The goal of this paper is to remedy this issue through conditioning [283]. Along the way, we introduce a new benchmark dataset, inspired by the recent ATLAS measurement [29], that can be used for distribution mapping as well as any unfolding method.

This paper is organized as follows. Section 7.1 explains distribution mapping in detail, including a remedy to the challenge with current methods. This section provides important and useful derivations, but it can be skipped without interrupting the flow of the paper. Section 7.2 highlights the fix to distribution mapping methods with a simple, illustrative example. Next, the new methods are tested on a benchmark dataset of single jet substructure 7.3 and then we create a a new dataset describing a 22-dimensional phase space in $Z + 2$-jets at the Large Hadron Collider 7.4. This latter dataset combines jet substructure and kinematic information. For all applications, we provide a detailed discussion of the conditional Schrödinger Bridge and Direct Diffusion performance and a comparison with the state of the art in generative unfolding, a diffusion model using transformer layers [3, 64]. The paper ends with conclusions and outlook in Sec. 7.5.

7.1 Distribution Mapping

Diffusion networks offer the possibility of sampling a phase space distribution from any given distribution, not just from Gaussians or other standard distributions [3, 68, 69, 208]. Before we use this for unfolding, we review in some detail about how this feature can be realized for stochastic conditional sampling. The derivations presented in this section are not new and not necessary to understand the results for the simple model in Sec. 7.2 and for the physics examples in Secs. 7.3 and 7.4. Nevertheless, we use this opportunity to carefully describe the methods in a physics context and illustrate how score matching (Schrödinger Bridge) and velocity matching (Direct Diffusion) approaches unify.

7.1.1 Distribution to noise

We want to design a process $p(x, t)$ that transforms a general data distribution (at reco level) into a general latent distribution (at gen level),

$$p(x,t) = \begin{cases} p_{\text{data}}(x) & t = 0 \\ p_{\text{latent}}(x) & t = 1 \; . \end{cases} \tag{7.1}$$

[5]A complete unfolding approach might include approaches to mitigate the dependence on the starting simulation [65] as well as acceptance effects [112].

This can be done using the stochastic differential equation (SDE)

$$dx = f(x,t)\;dt + g(t)\;dW\;. \tag{7.2}$$

Here $f(x,t)$ is the so-called drift coefficient, describing the deterministic part of the time evolution. The diffusion coefficient $g(t)$ describes the strength of the noising process, and W is a standard Wiener process, dW a noise infinitesimal. The connection between the evolving density in Eq.(7.1) and the trajectories in Eq.(7.2) is given by the Fokker-Planck equation (FPE)

$$\frac{\partial p(x,t)}{\partial t} = -\nabla_x[f(x,t)p(x,t)] + \frac{g(t)^2}{2}\nabla_x^2 p(x,t)\;. \tag{7.3}$$

For $g(t)=0$ the SDE reduces to an ordinary differential equation (ODE), and the FPE to the usual continuity equation. Unlike ODEs, SDEs are not time-symmetric. This is because adding noise to a system is fundamentally different from removing noise. It can be shown that the time-reversal of Eq.(7.2) is given by another diffusion SDE [284],

$$dx = \left[f(x,t) - g(t)^2\;\nabla_x \log p(x,t)\right]\;dt + g(t)\;d\bar{W}\;. \tag{7.4}$$

where $d\bar{W}$ is the noise infinitesimal corresponding to the reverse Wiener process. The new and unknown element is the score function

$$s(x,t) = \nabla_x \log p(x,t)\;, \tag{7.5}$$

where $p(x,t)$ is the solution to the forward and reverse SDEs. If we know the score function, we can use numerical SDE solvers to propagate samples backward in time.

Forward process

Diffusion generative networks usually define the latent space to be a standard Gaussian $\mathcal{N}(0,1)$. We can construct the forward process Eq.(7.2) by simplifying the drift to be linear in x, i.e.
$f(x,t) = xf(t)$. Now the SDE and the FPE read

$$\begin{aligned} dx &= x\;f(t)\;dt + g(t)\;dW\;. \\ \frac{\partial p(x,t)}{\partial t} &= -f(t)p(x,t) - xf(t)\nabla_x p(x,t) + \frac{g(t)^2}{2}\nabla_x^2 p(x,t)\;. \end{aligned} \tag{7.6}$$

In this case, we can analytically derive the solution of the FPE for given $f(t)$ and $g(t)$. We make the ansatz that the time evolution starting from an event $x_0 \sim p_{\text{data}}$ is a Gaussian

$$\begin{aligned} p(x,t|x_0) = \mathcal{N}(x|\mu(t),\sigma(t)) &= \frac{1}{\sqrt{2\pi}\sigma(t)}\exp\left(-\frac{(x-\mu(t))^2}{2\sigma(t)^2}\right) \\ \Leftrightarrow \qquad x(t|x_0) &= \mu(t) + \sigma(t)\epsilon \quad \text{with} \quad \epsilon \sim \mathcal{N}(0,1)\;, \end{aligned} \tag{7.7}$$

with time dependent mean $\mu(t)$ and standard deviation $\sigma(t)$. Using this ansatz in the FPE Eq.(7.6), we obtain

$$\frac{x-\mu}{\sigma^2}\frac{\partial\mu}{\partial t} + \frac{(x-\mu)^2}{\sigma^3}\frac{\partial\sigma}{\partial t} - \frac{1}{\sigma}\frac{\partial\sigma}{\partial t} = f(t)\left(x\frac{x-\mu}{\sigma^2} - 1\right) + \frac{g(t)^2}{2}\left(\frac{(x-\mu)^2}{\sigma^4} - \frac{1}{\sigma^2}\right)$$

$$= f(t)\frac{(x-\mu)^2+\mu(x-\mu)-\sigma^2}{\sigma^2}+g(t)^2\frac{(x-\mu)^2-\sigma^2}{2\sigma^4}\ . \tag{7.8}$$

Sorting this equation by powers of $(x-\mu)$ and comparing coefficients we find relations between $\mu(t), \sigma(t)$ and $f(t), g(t)$,

$$\begin{aligned}\frac{\partial\mu(t)}{\partial t} &= f(t)\mu(t)\\ \frac{\partial\sigma(t)}{\partial t} &= f(t)\sigma(t)+\frac{g(t)^2}{2\sigma(t)}\ .\end{aligned} \tag{7.9}$$

The solutions of those two differential equations with initial conditions $\sigma(0)=0$ and $\mu(0)=x_0$ are

$$\begin{aligned}\mu(t) &= x_0\alpha(t)\\ \sigma(t) &= \alpha(t)\left[\int_0^t dt'\frac{g(t')^2}{\alpha(t')^2}\right]^{1/2} \qquad \text{with} \qquad \alpha(t)=\exp\int_0^t dt' f(t')\ .\end{aligned} \tag{7.10}$$

If these equations are fulfilled, the Gaussian ansatz is the solution to the FPE. This gives us a solution for general $f(t), g(t)$. However, only if the boundary conditions $\alpha(1)=0$ and $\sigma(1)=1$ are fulfilled, the full unconditional density follows

$$\begin{aligned}p(x,0) &= \int dx_0\ p(x,0|x_0)\ p_{\text{data}}(x_0) = \int dx_0\ \delta(x-x_0)\ p_{\text{data}}(x_0) = p_{\text{data}}(x)\\ p(x,1) &= \int dx_0\ p(x,1|x_0)\ p_{\text{data}}(x_0) = \mathcal{N}(x;0,1)\int dx_0\ p_{\text{data}}(x_0) = \mathcal{N}(0,1)\ ,\end{aligned} \tag{7.11}$$

as specified in Eq.(7.1).

Relation to CFM

Equations (7.7) and (7.10) describe the mathematics behind all generative diffusion networks. Conditional flow matching (CFM) [51], a specific type of diffusion network that has seen a lot of success in particle physics [1,2,69,199,285,286], can be derived from this formalism. First, we use the fact that for any diffusion SDE there exists an associated ODE that encodes the same time-dependent density $p(x,t)$. It can be derived by rewriting the FPE (7.3) as

$$\begin{aligned}\frac{\partial p(x,t)}{\partial t} &= -\nabla_x\left[\left(f(x,t)-\frac{1}{2}g(t)^2\nabla_x\log p(x,t)\right)p(x,t)\right]\\ &= -\nabla_x(v(x,t)p(x,t))\\ \text{with}\quad v(x,t) &= f(x,t)-\frac{1}{2}g(t)^2\nabla_x\log p(x,t)\ .\end{aligned} \tag{7.12}$$

This continuity equation corresponds to the ODE

$$dx = v(x,t)\ dt\ . \tag{7.13}$$

The deterministic (ODE) and stochastic (SDE) processes are equivalent in the sense that they have the same density solution $p(x,t)$. The difference between the SDE drift $f(x,t)$ and the ODE velocity $v(x,t)$ is that the former can be hand-crafted such that the forward

SDE transports from the data to the latent distribution. To generate samples the score function $s(x,t)$ is also required. The velocity field combines the forward drift and the score function of the underlying process into one time-symmetric description. For known $f(x,t)$ and $g(t)$ the velocity and the score functions can be converted into each other.

CFMs work directly with the velocity field v of the ODE, the underlying SDE is not explicitly constructed. Instead, the trajectories Eq.(7.7) are used to define a forward process from data to noise. Following Ref. [1], we set $\alpha(t) = (1-t)$ and $\sigma(t) = t$, defining linear trajectories

$$x(t|x_0) = (1-t)x_0 + t\epsilon \; . \tag{7.14}$$

We then encode the ODE-velocity in a network using an MSE loss

$$\mathcal{L}_{\text{CFM}} = \left\langle \left[v_\theta(x,t) - v(x,t|x_0) \right]^2 \right\rangle_{t\sim\mathcal{U}(0,1), x_0\sim p_{\text{data}}(x_0), \epsilon\sim\mathcal{N}} \; . \tag{7.15}$$

For a detailed derivation of the formalism and the loss function see e.g. Refs. [1,3]. In practice, we can use a wealth of network architectures to encode the velocity, from a simple fully connected network [1], to transformers [2], vision transformers [286], and Lorentz-equivariant transformers [287,288].

7.1.2 Distribution to distribution

We now extend the SDE formalism to two arbitrary distributions. The generative direction starts from the initial $p(x_0)$ and samples the $p(x_1)$. The goal is to find a drift function $f(x,t)$ such that the SDE moves from $x_0 \sim p(x_0)$ at time $t=0$ to $x_1 \sim p(x_1)$ at time $t=1$.

Doob's h-transform

The key ingredient to this generalization is Doob's h-transform [289]. It conditions a given SDE, called the reference process, on a pre-defined final point. The reference process follows an SDE like Eq.(7.2),

$$dx_{\text{ref}} = f(x,t)\; dt + g(t)\; dW \; . \tag{7.16}$$

From $t=0$ to $t=1$ it encodes a time-evolving density $p_{\text{ref}}(x,t)$ for the entire stochastic process, as well as the conditional $p_{\text{ref}}(x,t|x_0)$ describing the stochastic trajectories starting from a specific x_0.

We modify this SDE to guarantee that the endpoint is a pre-defined x_1, by adding a term to the drift function,

$$\begin{aligned} dx &= \Big[f(x,t) + g(t)^2 h(x,t,x_1) \Big] \; dt + g(t) dW \\ \text{with} \quad & h(x,t,x_1) = \nabla_x \log p_{\text{ref}}(x_1, t=1|x) \; . \end{aligned} \tag{7.17}$$

The Doob's h-transform function depends on the current state of the SDE $x(t)$ at time t and on the specified final point x_1. The density $p_{\text{ref}}(x_1, t=1|x)$ is the transition probability that the reference process reaches x_1 at $t=1$ conditioned on the state $x(t)$ at time t. Including this term in the drift forces the trajectories to walk up the gradient of this density and pushes them towards intermediate states that are more compatible with the desired final state. This way, it corrects the initial SDE by forcing it towards the specified x_1.

We note that h depends on the reference process through $p_\text{ref}(x_1, t=1|x)$. This correction adapts to the chosen initial SDE. Different initial values $f(x,t)$ and $g(t)$ lead to a different correction from the Doob's h-transform, but eventually arrive in the specified x_1. This means we can simplify the reference process into a pure noising,

$$f(x,t) = 0 \qquad\qquad \Rightarrow \qquad dx^\text{ref} = g(t)\; dW \; . \tag{7.18}$$

For this choice we can use Eqs.(7.10) and (7.17) to calculate the h-transform

$$\begin{aligned} \alpha_\text{ref}(t) &= 1 \\ \sigma_\text{ref}(t)^2 &= \int_t^1 dt' g(t')^2 \\ p_\text{ref}(x_1, t=1|x) &= \mathcal{N}\Big(x_1; x, \sigma_\text{ref}(t)\Big) \propto \exp\left[-\frac{1}{2}\frac{(x_1 - x(t))^2}{\sigma_\text{ref}(t)^2}\right] \\ h(x,t,x_1) &= \frac{x_1 - x(t)}{\sigma_\text{ref}(t)^2} \; . \end{aligned} \tag{7.19}$$

We have explicitly constructed a forward SDE

$$dx = \frac{g(t)^2}{\sigma_\text{ref}(t)^2}(x(t) - x_1)\; dt + g(t)\; dW \; , \tag{7.20}$$

for which the solution trajectories are guaranteed to end in $x_1 \sim p_1$. According to Eq.(7.3) the underlying probability distribution fulfills

$$\frac{\partial p(x,t|x_1)}{\partial t} = -\nabla_x \frac{g(t)^2 (x(t) - x_1) p(x,t|x_1)}{\sigma_\text{ref}(t)^2} + \frac{g(t)^2}{2}\nabla_x^2 p(x,t|x_1) \; . \tag{7.21}$$

Using the same method, we can also describe a reverse process, for which the solution trajectories end in $x_0 \sim p_0$. The reference process

$$d\bar{x}_\text{ref} = g(t)\; d\bar{W} \; , \tag{7.22}$$

moves from $t=1$ to $t=0$. In complete analogy, it encodes the conditional probability $\bar{p}_\text{ref}(\bar{x}(t)|x_1)$ starting from a specific point x_1. Applying the h-transform leads to the SDE

$$\begin{aligned} d\bar{x} &= \Big[-g(t)^2 \bar{h}(\bar{x},t,x_0)\Big]\; dt + g(t) d\bar{W} \\ \text{with} \quad \bar{h}(\bar{x},t,x_0) &= \nabla_{\bar{x}} \log \bar{p}_\text{ref}(x_0, t=0|\bar{x}) \; , \end{aligned} \tag{7.23}$$

and modifies Eq.(7.19) to

$$\begin{aligned} \bar{\sigma}_\text{ref}(t)^2 &= \int_0^t dt' g(t')^2 \\ \bar{h}(\bar{x},t,x_0) &= \frac{x_0 - \bar{x}(t)}{\bar{\sigma}_\text{ref}(t)^2} \; . \end{aligned} \tag{7.24}$$

Again we constructed a generating SDE

$$d\bar{x} = \frac{g(t)^2}{\bar{\sigma}_\text{ref}(t)^2}(\bar{x}(t) - x_0)\; dt + g(t)\; d\bar{W} \; . \tag{7.25}$$

whose solutions end in x_0 and the underlying probability density follows

$$\frac{\partial \bar{p}(x,t|x_0)}{\partial t} = -\nabla_{\bar{x}} \frac{g(t)^2(\bar{x}(t)-x_0)\bar{p}(\bar{x},t|x_0)}{\bar{\sigma}_\text{ref}(t)^2} - \frac{g(t)^2}{2}\nabla^2_{\bar{x}}\bar{p}(\bar{x},t|x_0) \; . \tag{7.26}$$

So far, we have constructed the forward and the reverse processes independently. We now assume that they are the time-reversal of each other and that the forward and reverse FPEs (7.21) and (7.26) have a common Gaussian solution

$$p(x,t|x_0,x_1) = \bar{p}(\bar{x},t|x_0,x_1) = \mathcal{N}(x|\mu(t),\sigma(t)) \; . \tag{7.27}$$

Inserting this ansatz into the forward FPE (7.21), we obtain

$$\frac{x-\mu}{\sigma^2}\frac{\partial\mu}{\partial t} + \frac{(x-\mu)^2}{\sigma^3}\frac{\partial\sigma}{\partial t} - \frac{1}{\sigma}\frac{\partial\sigma}{\partial t} = \frac{g^2}{\sigma^2_\text{ref}}\left(\frac{(x_1-x)(x-\mu)}{\sigma^2}+1\right) + \frac{g^2}{2}\left(\frac{(x-\mu)^2}{\sigma^4}-\frac{1}{\sigma^2}\right) \; . \tag{7.28}$$

It is solved by

$$\begin{aligned} \frac{\partial\mu(t)}{\partial t} &= \frac{g(t)^2(x_1-\mu(t))}{\sigma_\text{ref}(t)^2} \\ \frac{\partial\sigma(t)}{\partial t} &= -\frac{g(t)^2\sigma(t)}{\sigma_\text{ref}(t)^2} + \frac{g(t)^2}{2\sigma(t)} \; . \end{aligned} \tag{7.29}$$

From the reverse FPE (7.26) we find the corresponding

$$\begin{aligned} \frac{\partial\mu(t)}{\partial t} &= \frac{g(t)^2(\mu(t)-x_0)}{\bar{\sigma}_\text{ref}(t)^2} \\ \frac{\partial\sigma(t)}{\partial t} &= \frac{g(t)^2\sigma(t)}{\bar{\sigma}_\text{ref}(t)^2} - \frac{g(t)^2}{2\sigma(t)} \; . \end{aligned} \tag{7.30}$$

Equating Eq.(7.29) and Eq.(7.30) yields

$$\begin{aligned} \mu(t) &= \frac{\bar{\sigma}_\text{ref}(t)^2 x_1 + \sigma_\text{ref}(t)^2 x_0}{\bar{\sigma}_\text{ref}(t)^2 + \sigma_\text{ref}(t)^2} \\ \sigma(t) &= \sqrt{\frac{\bar{\sigma}_\text{ref}(t)^2\sigma_\text{ref}(t)^2}{\bar{\sigma}_\text{ref}(t)^2+\sigma_\text{ref}(t)^2}} \; . \end{aligned} \tag{7.31}$$

This solves Eq.(7.29) and Eq.(7.30) with the boundary conditions $\sigma(0) = \sigma(1) = 0$, $\mu(0) = x_0$ and $\mu(1) = x_1$. Finally, the conditional probability of both processes is given by

$$\begin{aligned} p(x(t),t|x_0,x_1) &= \mathcal{N}\left(x\middle|\frac{\bar{\sigma}_\text{ref}(t)^2 x_1 + \sigma_\text{ref}(t)^2 x_0}{\bar{\sigma}_\text{ref}(t)^2 + \sigma_\text{ref}(t)^2}, \sqrt{\frac{\bar{\sigma}_\text{ref}(t)^2\sigma_\text{ref}(t)^2}{\bar{\sigma}_\text{ref}(t)^2+\sigma_\text{ref}(t)^2}}\right) \\ &\propto \mathcal{N}(x|x_0,\bar{\sigma}_\text{ref})\,\mathcal{N}(x|x_1,\sigma_\text{ref}) \; . \end{aligned} \tag{7.32}$$

Loss function

The full unconditional density encoded in the constructed stochastic process is obtained by marginalizing over the conditions.

$$\begin{aligned} p(x,t) &= \int dx_0\; dx_1\; p^{\text{train}}(x_0,x_1)\; p(x,t|x_0,x_1) \\ &= \int dx_0\; dx_1\; p^{\text{train}}(x_0,x_1)\; \mathcal{N}(x|\mu(t),\sigma(t)) = \begin{cases} p_0(x) & t=0 \\ p_1(x) & t=1\;. \end{cases} \end{aligned} \tag{7.33}$$

The joint distribution $p^{\text{train}}(x_0,x_1)$ is defined by the pairing in the training data, in the case of unpaired data it factorizes to $p^{\text{train}}(x_0,x_1) = p_0(x_0)p_1(x_1)$. Both limits of the stochastic process are fulfilled independent of the choice of joint distribution. For instance, for unfolding, we can use the pairing between reco-level and gen-level events from the forward simulation.

To construct a generative network, we need to remove the conditions on the two end points. This means we want to find an SDE that encodes the distribution from Eq.(7.33), but with a drift function that only depends on the current state of the SDE. We can derive this unconditional drift term similarly to the unconditional CFM-velocity [1, 51], starting with the FPE (7.3),

$$\begin{aligned} \frac{\partial p(x,t)}{\partial t} &= \int dx_0 dx_1 p^{\text{train}}(x_0,x_1) \frac{\partial p(x,t|x_0,x_1)}{\partial t} \\ &= \int dx_0 dx_1 p^{\text{train}}(x_0,x_1) \left[-\nabla_x [f(x,t|x_0,x_1) p(x,t|x_0,x_1)] + \frac{g^2}{2} \nabla_x^2 p(x,t|x_0,x_1) \right] \\ &= -\nabla_x \left[p(x,t) \int dx_0 dx_1 p^{\text{train}}(x_0,x_1) \frac{f(x,t|x_0,x_1) p(x,t|x_0,x_1)}{p(x,t)} \right] \\ &\quad + \frac{g^2}{2} \nabla_x^2 \int dx_0 dx_1 p^{\text{train}}(x_0,x_1) p(x,t|x_0,x_1) \\ &= -\nabla_x [p(x,t) f(x,t)] + \frac{g^2}{2} \nabla_x^2 p(x,t)\;, \end{aligned} \tag{7.34}$$

where we define

$$f(x,t) = \int dx_0 dx_1 p^{\text{train}}(x_0,x_1) \frac{f(x,t|x_0,x_1) p(x,t|x_0,x_1)}{p(x,t)}\;. \tag{7.35}$$

With this drift function and a diffusion term $g(t)$, the solution of the FPE is given by Eq.(7.33). This gives us an SDE which propagates samples between $x_1 \sim p_1$ and $x_0 \sim p_0$, only depending on the current state $x(t)$. Starting from one of the distributions and numerically solving the SDE generates samples from the other distribution.

The last problem with the drift in Eq.(7.35) is that we cannot evaluate it analytically, so we encode it into a network f_θ. For this regression problem the natural loss is the MSE, but this requires training samples $f(x,t)$. We re-write this objective in terms of the conditional drift $f(x,t|x_0,x_1)$ defined in the SDE Eq.(7.20) and the conditional trajectories $p(x,t|x_0,x_1)$ defined in Eq.(7.39), as these allow for efficient generation of training samples. Following all steps from Ref. [1] the distribution mapping loss becomes

$$\mathcal{L}_{\text{DM}} = \left\langle \Big(f_\theta(x,t) - f(x,t|x_0,x_1) \Big)^2 \right\rangle_{t,(x_0,x_1)\sim p^{\text{train}}(x_0,x_1), x\sim p(x,t|x_0,x_1)}$$

$$= \left\langle \left(f_\theta(x,t) - \frac{g(t)^2 (x - x_0)}{\bar{\sigma}(t)^2} \right)^2 \right\rangle_{t,(x_0,x_1)\sim p^{\text{train}}(x_0,x_1), x\sim p(x,t|x_0,x_1)} . \tag{7.36}$$

The learned drift function depends on the pairing information in the training data, encoded via $p^{\text{train}}(x_0, x_1)$. Different pairings lead to different SDEs encoding different trajectories, but they all result in a generative network with the correct boundary distributions in Eq.(7.33).

Noise schedules for SB and DiDi

Choosing $g(t) = \sqrt{\beta(t)}$, with $\beta(t)$ the triangular function

$$\beta(t) = \begin{cases} \beta_0 + 2(\beta_1 - \beta_0)t & 0 \le t < \dfrac{1}{2} \\ \beta_1 - 2(\beta_1 - \beta_0)\left(t - \dfrac{1}{2}\right) & \dfrac{1}{2} \le t \le 1 \end{cases} \tag{7.37}$$

and $\beta_0 = 10^{-5}$ and $\beta_1 = 10^{-4}$, we obtain the SB formulation [68, 256].

For constant $g(t) = g$, Eq.(7.19) simplifies to $\bar{\sigma}_{\text{ref}}(t) = g\sqrt{t}$ and $\sigma_{\text{ref}}(t) = g\sqrt{1-t}$. Consequently, Eq.(7.31) yields

$$\mu(t) = (1-t)x_0 + t x_1 \qquad \text{and} \qquad \sigma(t) = g\sqrt{t(1-t)} . \tag{7.38}$$

Our trajectory and probability distribution take the form

$$\begin{aligned} & x(t) = (1-t)x_0 + t x_1 + g\sqrt{t(1-t)}\, \epsilon \qquad \text{with } \epsilon \sim \mathcal{N}(0,1) \\ \Leftrightarrow \qquad & p(x(t), t|x_0, x_1) = \mathcal{N}\big(x(t)|(1-t)x_0 + t x_1, g\sqrt{t(1-t)}\big) . \end{aligned} \tag{7.39}$$

The noise term vanishes at the endpoints $t = 0, 1$ and takes its maximum at $t = 1/2$. This constructs an SDE that interpolates between two arbitrary distributions and reduces to DiDi [69] for $g \to 0$. To see this we start from Eq.(7.20) and insert the training trajectory from Eq.(7.39),

$$\begin{aligned} dx(t) &= \frac{x(t) - x_0}{t}\, dt + g\, dW \\ &= \frac{(1-t)x_0 + t x_1 + g\sqrt{t(1-t)}\, \epsilon - x_0}{t}\, dt + g\, dW \\ &= (x_1 - x_0)\, dt + g \left[\frac{\sqrt{t(1-t)}\, \epsilon}{t}\, dt + dW \right] . \end{aligned} \tag{7.40}$$

For $g \to 0$ the training SDE reduces to the training ODE from DiDi, with the linear velocity field $x_1 - x_0$ and the distribution mapping loss reduces to the flow matching loss.

7.1.3 Conditional distribution mapping

In the last section we have constructed an SDE-based mapping between two arbitrary distributions. We now describe the new aspect of adapting this DM-formalism to reproduces the correct conditional distributions [283]. Specifically, we look at the trajectories $p(x, t|x_1)$ obtained when solving the learned SDE repeatedly from the same starting point $x_1 \sim p_1$, and modify our formalism such that for $t \to 0$ it reproduces the correct data pairing $p^{\text{train}}(x_0|x_1)$.

First, we check how this density looks in our conditional training trajectories $p(x(t), t|x_0, x_1)$ by marginalizing over x_0. Similar to Eq.(7.33) we can write

$$\begin{aligned} p(x, t|x_1) &= \int dx_0 \, p^{\text{train}}(x_0|x_1) \, p(x(t), t|x_0, x_1) \\ &= \int dx_0 \, p^{\text{train}}(x_0|x_1) \, \mathcal{N}(x|\mu(t), \sigma(t)) = \begin{cases} p^{\text{train}}(x|x_1) & t = 0 \\ \delta(x_1 - x) & t = 1 \, . \end{cases} \end{aligned} \tag{7.41}$$

This conditional density has the correct boundary behavior. When conditioned on a latent-space event $x_1 \sim p_{\text{latent}}$, the density converges to a delta peak around this event at time $t = 1$ and converges to the training pairing conditional distribution $p^{\text{train}}(x|x_1)$ at time $t = 0$.

However, there is no guarantee that the generative SDE defined via the drift from Eq.(7.35) shares these conditional densities. We have derived this drift f by showing that it solves the same unconditional FPE in Eq.(7.34) as our training process, so we are only sure that they share the unconditional density $p(x, t)$.

We need a drift function leading to an SDE that shares the conditional density $p(x, t|x_1)$ of the training trajectories. This can be achieved by going through the same derivation as we did for the drift initially, but this time with the FPE for the conditional density

$$\begin{aligned} \frac{\partial p(x, t|x_1)}{\partial t} &= \int dx_0 p^{\text{train}}(x_0|x_1) \frac{\partial p(x, t|x_0, x_1)}{\partial t} \\ &= \int dx_0 p^{\text{train}}(x_0|x_1) \left[-\nabla_x [f(x, t|x_0, x_1) p(x, t|x_0, x_1)] + \frac{g^2}{2} \nabla_x^2 p(x, t|x_0, x_1) \right] \\ &= -\nabla_x \left[p(x, t|x_1) \int dx_0 p^{\text{train}}(x_0|x_1) \frac{f(x, t|x_0, x_1) p(x, t|x_0, x_1)}{p(x, t|x_1)} \right] \\ &\quad + \frac{g^2}{2} \nabla_x^2 \int dx_0 p^{\text{train}}(x_0|x_1) p(x, t|x_0, x_1) \\ &= -\nabla_x [p(x, t|x_1) f(x, t|x_1)] + \frac{g^2}{2} \nabla_x^2 p(x, t|x_1) \, . \end{aligned} \tag{7.42}$$

Here, we identify the conditional drift term to be

$$f(x, t|x_1) = \int dx_0 \, p^{\text{train}}(x_0|x_1) \frac{f(x, t|x_0, x_1) p(x, t|x_0, x_1)}{p(x, t|x_1)} \, . \tag{7.43}$$

Sampling a latent $x_1 \sim p_{\text{latent}}$ and solving the SDE with this drift function samples from the conditional density $p^{\text{train}}(x|x_1)$. This new conditional drift is not only a function of the current state of the SDE, but also of the initial state x_1. This additional input acts as the condition under which we want to generate a sample from the data distribution, similar to the way the CFM velocity is a function of the current state of the ODE and of the condition.

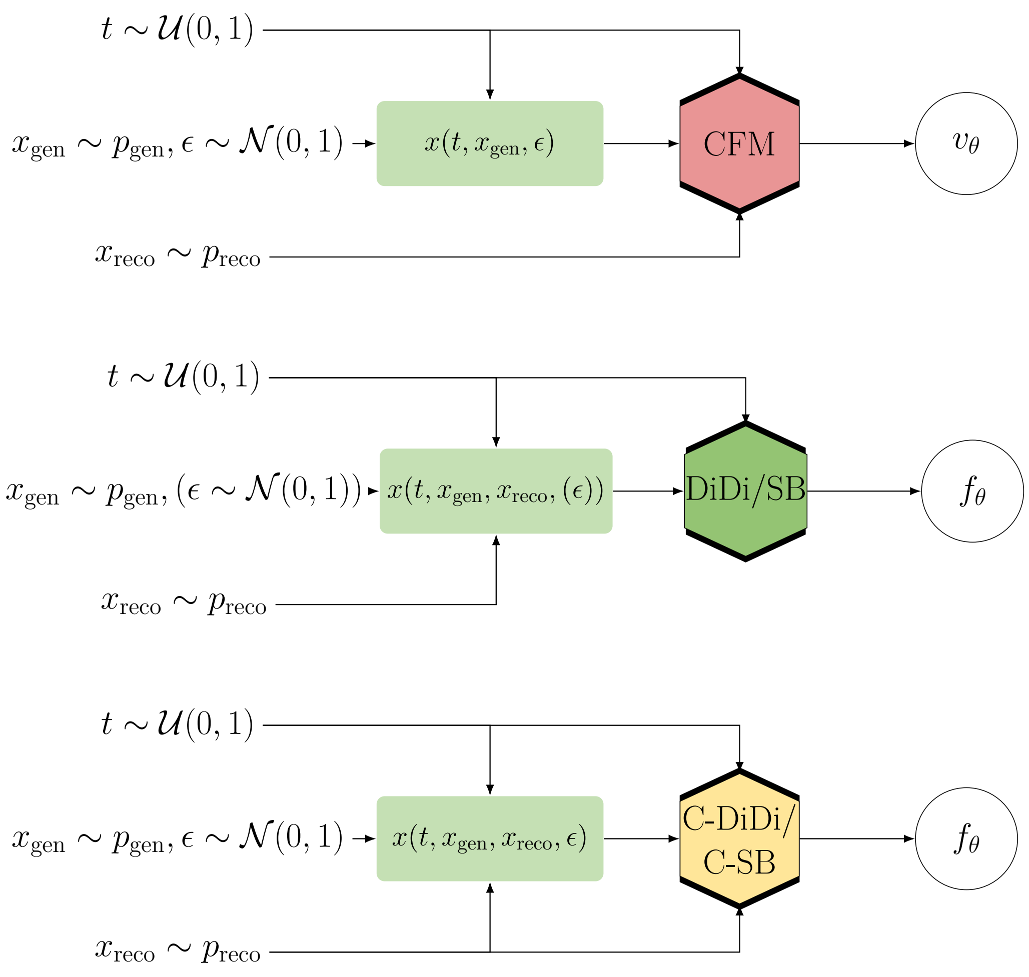

$t \sim \mathcal{U}(0,1)$
$x_\text{gen} \sim p_\text{gen}, \epsilon \sim \mathcal{N}(0,1)$
$x(t, x_\text{gen}, \epsilon)$
CFM
v_θ
$x_\text{reco} \sim p_\text{reco}$
$t \sim \mathcal{U}(0,1)$
$x_\text{gen} \sim p_\text{gen}, (\epsilon \sim \mathcal{N}(0,1))$
$x(t, x_\text{gen}, x_\text{reco}, (\epsilon))$
DiDi/SB
f_θ
$x_\text{reco} \sim p_\text{reco}$
$t \sim \mathcal{U}(0,1)$
$x_\text{gen} \sim p_\text{gen}, \epsilon \sim \mathcal{N}(0,1)$
$x(t, x_\text{gen}, x_\text{reco}, \epsilon)$
C-DiDi/ C-SB
f_θ
$x_\text{reco} \sim p_\text{reco}$

Figure 7.1: Schematic illustration of the training procedure of a CFM-based (top), a distribution mapping-based (middle) and a conditional distribution mapping-based (bottom) generative unfolding pipeline.

Loss function

This conditional drift is once again encoded in a network. Repeating the derivation of Ref. [1], we obtain the conditional distribution mapping (CDM) loss

$$\begin{aligned}\mathcal{L}_\text{CDM} &= \left\langle \Big(f_\theta(x,t,x_1) - f(x,t|x_0,x_1) \Big)^2 \right\rangle_{t,(x_0,x_1)\sim p(x_0,x_1), x\sim p(x,t|x_0,x_1)} \\ &= \left\langle \left(f_\theta(x,t,x_1) - \frac{g(t)^2(x-x_0)}{\bar\sigma(t)} \right)^2 \right\rangle_{t,(x_0,x_1)\sim p^\text{train}(x_0,x_1), x\sim p(x,t|x_0,x_1)} . \end{aligned} \tag{7.44}$$

It is identical to the standard DM-loss, except for the network also using the initial condition x_1 as a third input. This is the only change necessary to allow the network to learn the proper conditional densities.

7.1.4 Unfolding

For unfolding, we want to transform a measured reco-level distribution p_reco to the corresponding gen-level distribution p_gen. Recent implementations [3] of generative unfolding use a CFM network to generate samples from the posterior distribution $p(x_{\text{gen}}|x_{\text{reco}})$. It learns the velocity $v(x, t, x_{\text{reco}})$, linked to the posterior distribution via Eq.(7.12) and flowing between a point of a Gaussian latent space and a point in the gen-level phase space conditioned on a given rec-level event. A schematic illustration of the training procedure is shown in the top part of Fig. 7.1.

Alternatively, we can also map reco-level events directly to their gen-level counterpart either using the SB or DiDi. The key difference to standard generative unfolding is that we use the reco-level information to define the trajectories instead of treating it as an additional input to the network. This is visualized in the center of Fig. 7.1. We learn the drift term of the probability $p(x, t)$ as described in Eq.(7.36). Optionally, we can add Gaussian noise to the trajectories to make the networks stochastic rather than deterministic. The impact of the noise is governed by the choice of the diffusion term $g(t)$.

Finally, we can combine the conditional generative approach with the DM by giving the reco-level information to the network directly. The training objective is to learn a drift term linked to the conditional probability $p(x(t), t|x_{\text{reco}})$ as in Eq.(7.44). In this scenario adding noise is not optional. The exact training procedure is illustrated in the lower part of Fig. 7.1.

7.2 Gaussian Example

To illustrate the motivation for using conditional DM for unfolding, we turn to a simple example: a Gaussian mixture model ('double Gaussian') with equally weighted components, unit variances, and means at $\mu_1 = -4$ and $\mu_2 = 4$. We run this distribution through an extreme hypothetical detector which inverts all observations

$$f(x) = -x \; . \tag{7.45}$$

Since our initial distribution is symmetric around $x = 0$, this detector leaves the marginal distribution unchanged, but allows us to see how the mapping is learned by the network.

We compare the mapping constructed by the standard and conditional Schrödinger Bridge (SB). The results are shown in Fig. 7.2. Each of the displayed plots is divided into three panels. The left panels show the marginal distributions for the staring double Gaussian (blue, solid) and the starting point of the SB network (red, dotted). The right panels show the marginals of the double Gaussian after the inversion function is applied (blue, solid), as well as of the final step of the SB (red, dotted). The central panels show the SB trajectories which transform a subset of points. Each path is color coded to indicate whether the sign of the transported point changes; a blue line corresponds to a correct mapping $f(x) = -x$, while a red line indicates an incorrect mapping $f(x) \sim x$.

In the upper part of Fig. 7.2, we see that for the unconditional SB a large number of mappings are incorrect. Notably, the mappings can be seen to converge around $x = 0$ after half the number of total steps, and diverge after. However, the majority of paths do not cross over from x to $-x$, as required, but return to the original peak. This is because at $x = 0$, both paths intersect. Since the network moves points one step at a time, it no longer has any information where to map a point once it reaches $x = 0$. As discussed in Sec. 7.1.3, we can break this degeneracy by providing the original starting point as input.

The lower part of Fig. 7.3 shows the result of a conditional SB network. We see that all paths are blue, indicating a correct mapping.

So far, the marginal distribution is not impacted by the mis-modeled mapping. The network still reproduces the overall target distribution available during training. However, the mapping can present an issue to the quality of the final marginal distribution if the data the network is unfolding differs in composition to the training data. We illustrate an extreme case in Fig. 7.3. In the top part we apply the unconditional bridge, trained on the full double Gaussian, only to the positive peak. Now, the unconditional SB produces a wrong marginal distribution, where only a small fraction of points is mapped to the correct negative peak, and the majority remains in the positive peak. In contrast, the lower part of Fig. 7.3 shows that the conditional SM learns the correct mapping and therefore easily produces the correct marginal distribution.

7.3 Unfolding Jet Substructure Observables

As a first physics example, we consider the updated version [257] of the OmniFold dataset [63] which has become a standard benchmark for unfolding methods [3, 68, 281]. It consists of events describing

$$pp \to Z + \text{jets} \tag{7.46}$$

production at $\sqrt{s} = 14$ TeV. The events are generated and decayed with Pythia 8.244 [33] with Tune 26, the detector response is simulated with Delphes 3.5.0 [36] with the CMS card, that uses particle flow reconstruction. At both pre-detector (gen level) and post-detector (reco level) jets are clustered with the anti-k_T algorithm [109] with $R = 0.4$, as implemented in FastJet 3.3.2 [177].

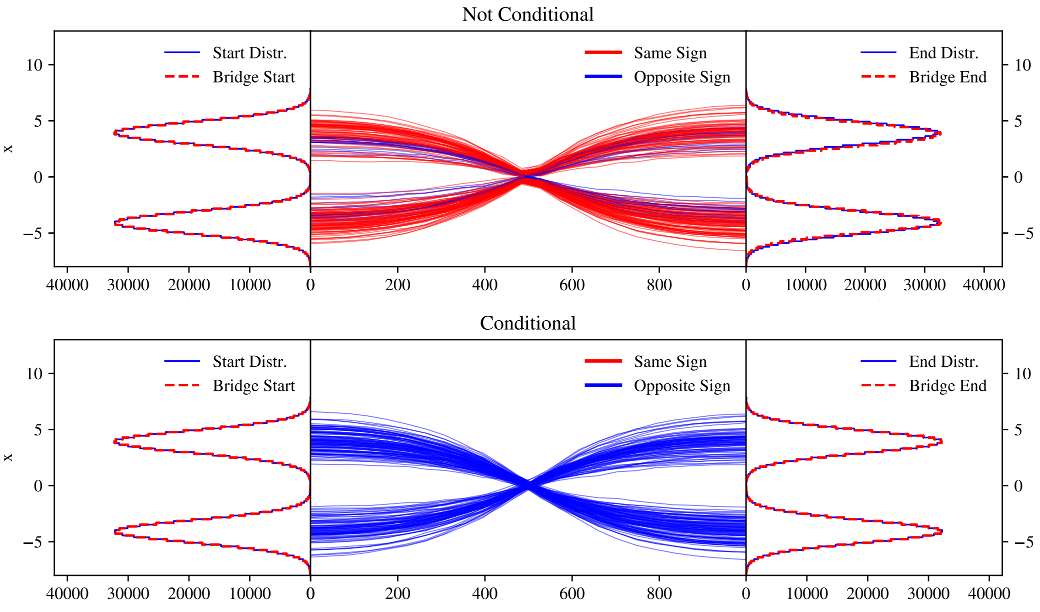


Figure 7.2: Non-conditional (top) vs conditional (bottom) distribution mapping when learning a $x \to -x$ mapping of a double-Gaussian.

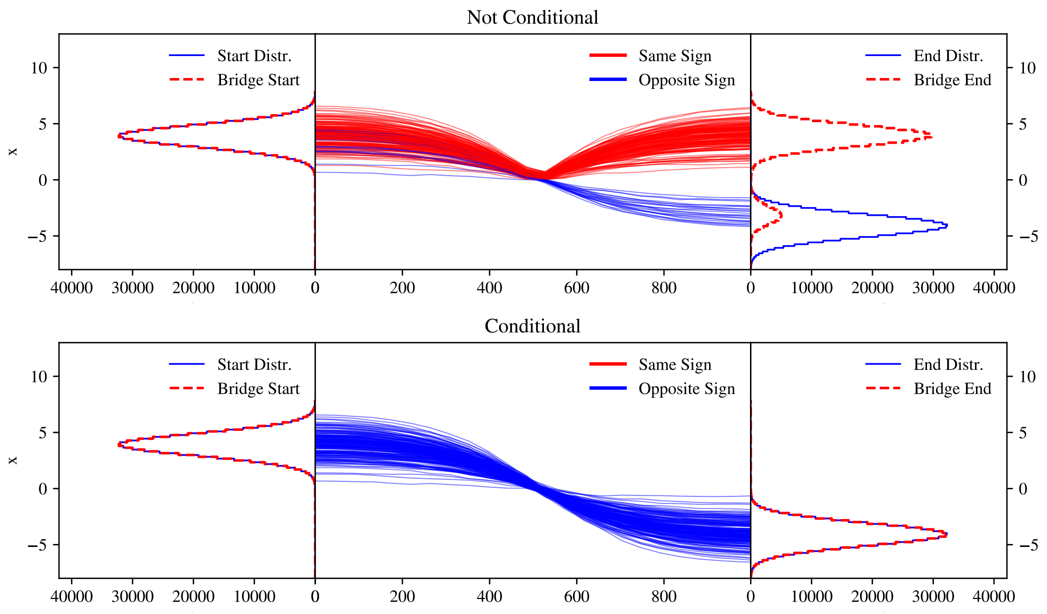


Figure 7.3: Non-conditional (top) vs conditional (bottom) distribution mapping when learning a $x \to -x$ mapping of a double-Gaussian, applied only to the positive peak

We unfold six jet substructure observables of the leading jet: mass m, width $\tau_1^{(\beta=1)}$, multiplicity N, soft drop mass [259, 260] $\rho = m_{\text{SD}}^2/p_T^2$, momentum fraction z_g using $z_{\text{cut}} = 0.1$ and $\beta = 0$, and the N-subjettiness ratio $\tau_{21} = \tau_2^{(\beta=1)}/\tau_1^{(\beta=1)}$ [261]. The dataset contains about 24M simulated events, 20M for training and 4M for testing.

7.3.1 Unfolded distributions

We unfold the 6-dimensional phase space using all five network implementations of the three methods

- conditional generative (Conditional Flow Matching, CFM);
- unconditional distribution mapping DiDi and SB; and
- conditional distribution mapping C-DiDi and C-SB.

The respective velocity fields and drift terms are encoded in standard MLPs, the hyperparameters are given in Tab. A.12. All networks are implemented in PyTorch [173] and trained with the Adam [114] optimizer. We follow the preprocessing from Ref. [3].

Due to the varying numerical requirements of different networks, we choose suitable numerical solvers for their evaluation. For the CFM, an ODE-based network, we unfold reco-level samples using a numerical ODE-solver [50]. The SDE-based networks C-DiDi and C-SB use the DDPM SDE-solver [50]. For the unconditional DiDi and SB we have the choice between an ODE-based formulation with noise scale $g = 0$ and an SDE-based formulation with $g > 0$. We observe no significant difference in performance between them, the shown results use the SDE formulation.

In Fig. 7.4 we show all unfolded distribution together with the true gen-level and reco-level distributions. For CFM, DiDi and SB, we reproduce the results shown in Ref. [3]. Both

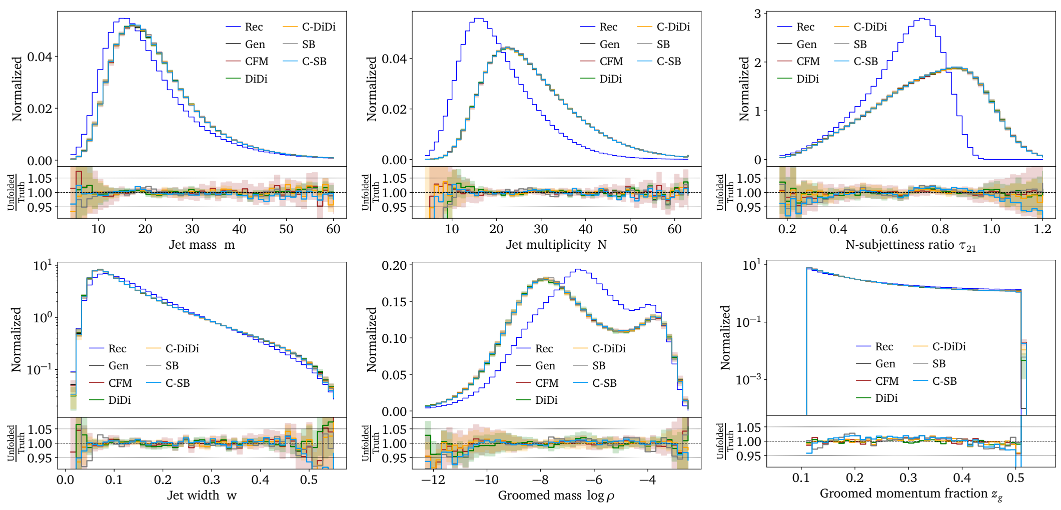


Figure 7.4: Unfolded distributions of the 6d jet-substructure dataset using CFM, DiDi, C-DiDi, SB and C-SB. All unfolded distributions reproduce the truth at percent level. The remaining differences are well covered by the BNN uncertainties.

methods can reliably solve this unfolding task to sub-percent precision. The new C-DiDi and C-SB give a precision on par with the established CFM method. For all networks, we only observe significant deviations from the truth far into the tails or at hard edges, for instance, in the groomed momentum fraction z_g.

The uncertainties reported in the figures are produced from posterior sampling. This is possible by making all of the models Bayesian neural networks, approximated with independent Gaussians for every network parameter, doubling the number of model parameters [23, 155–157]. Previous studies in the context of the LHC have shown that the posterior is a reasonable estimate of the variation introduced by the limited size of the training dataset [159, 160, 166]. Even though the weights are Gaussian distributed, the final network output is generally not a Gaussian. The concept of BNNs can also be applied to the density estimation in generative networks [1, 45, 161], including diffusion generators [1, 69].

The uncertainties shown in Fig. 7.4 are obtained by evaluating the respective networks 20 times, each time with a new set of network weights sampled from the learned distribution. The deviations from the true gen-level distribution as well as the differences between the five networks are generally covered by these uncertainties. As expected, the uncertainty increases in regions of low training statistics e.g. the tail of the jet mass distribution m.

7.3.2 Learned Mapping

Next, we check if the learned mapping between the reco-level and gen-level distributions agrees with the physical forward simulation. We show migration matrices for some of the observables in Fig. 7.5: the first row shows the truth encoded in the training data; the following columns show the learned event-wise mapping from the the CFM, unconditional DiDi/SB and C-DiDi/C-SB. While the 6-dimensional unfolded distributions are nearly identical for all methods, the migration plots show a significant difference between the unconditional and conditional networks.

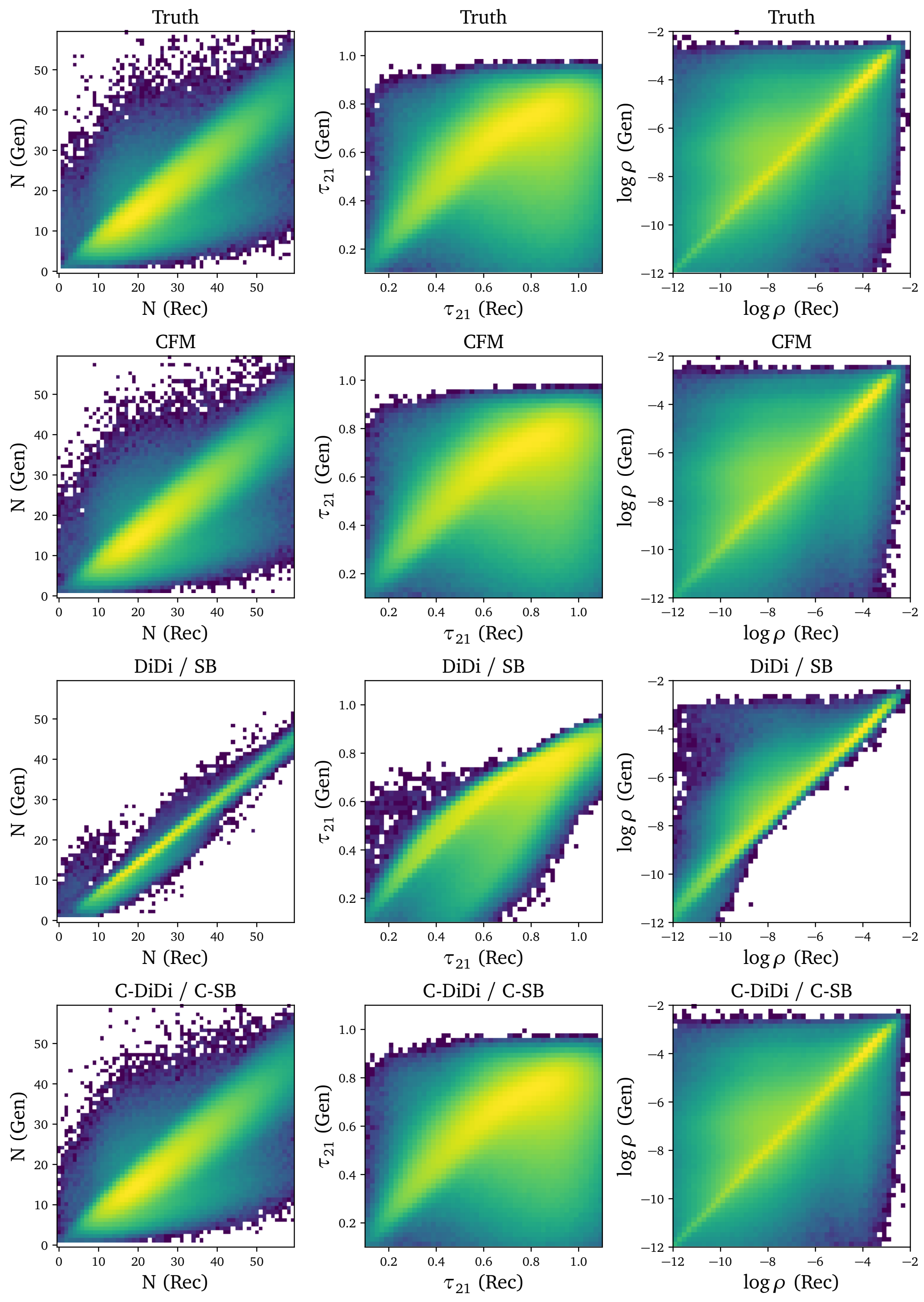


Figure 7.5: Migration maps for the 6D OmniFold dataset, truth compared to three different methods. We only show DiDi and C-DiDi, after verifying that the results are indistinguishable from SB and C-SB.

The conditional CFM generator is, by design, trained to reproduce the conditional distribution $p(x_\text{gen}|x_\text{reco})$. In contrast, the unconditional DM learns to map $p(x_\text{reco}) \to p(x_\text{gen})$, but with an unphysically diagonal optimal transport prescription, as showcased in the third row. Finally, we see that the conditional C-DiDi and C-SB encode the conditional probabilities just like the CFM does.

Finally, we take a closer look at the learned posterior distributions $p(x_\text{gen}|x_\text{reco})$. While single-event-unfolding is an ill-defined analysis task, we can use the per-event posterior to illustrate the performance of the different unfolding generators [2, 56]. All our methods are inherently non-deterministic when inputting the same reco-level event repeatedly, which allows us to generate non-trivial learned posteriors. For the CFM, we sample the latent Gaussian distribution, for any one given latent space point the ODE trajectory is deterministic. In contrast, the DM-methods always start from the same latent space point, the reco event, but evolve it using a non-deterministic SDE.

In Fig. 7.6, we show some single-event posterior distributions from the three methods, obtained by unfolding the same reco-event 10000 times. For reference, we include the unfolded event at reco-level, the gen-level truth, and the full unconditional gen-level distribution. Again, we observe a different behavior between the unconditional DM on the one hand and the CFM and conditional DM on the other. As expected from the derivation and from the migration plots, the unconditional DiDi network does not learn a physics-defined posterior, and our sampled distribution shows simply Gaussian smearing. The width of the Gaussian is related to the noise scale g of the SDE, which we verified by varying the noise scale over four orders of magnitude. The two conditional methods learn physically meaningful posteriors. Their shapes vary widely for the shown events and observables, but they agree between the different methods. We checked that the C-DiDi posteriors are invariant when varying the SDE noise scale g, so the learned single-event unfoldings illustrate how the CFM and the conditional DM-methods learn the same non-trivial conditional posteriors.

7.3.3 Classifier test

One approach to quantitatively test the performance of unfolding across the entire measured phase space is to use a post-hoc classifier, assuming that supervised classifier training is more effective than unsupervised density estimation[6] as part of the generative networks [3, 178]. A well-trained and calibrated classifier C comparing training and generated events will approximate the likelihood ratio

$$w(x_\text{gen}) = \frac{p_\text{true}(x_\text{gen})}{p_\theta(x_\text{gen})} = \frac{C(x_\text{gen})}{1 - C(x_\text{gen})} \; . \tag{7.47}$$

With a slight modification, we can employ this technique to evaluate the quality of our learned posterior distributions. Instead of training the classifier only on gen-level, we train on the joint reco-level and gen-level data. This gives us access to the likelihood ratio of the joint distributions. Making use of the fact that the reco-level distribution is the same for generated and true, we can write

$$\frac{C(x_\text{gen}, x_\text{rec})}{1 - C(x_\text{gen}, x_\text{rec})} = \frac{p_\text{true}(x_\text{gen}, x_\text{rec})}{p_\theta(x_\text{gen}, x_\text{rec})}$$

[6]In practice, if this is the case, the post-hoc classifier could be used to improve the quality of the generative model [162].

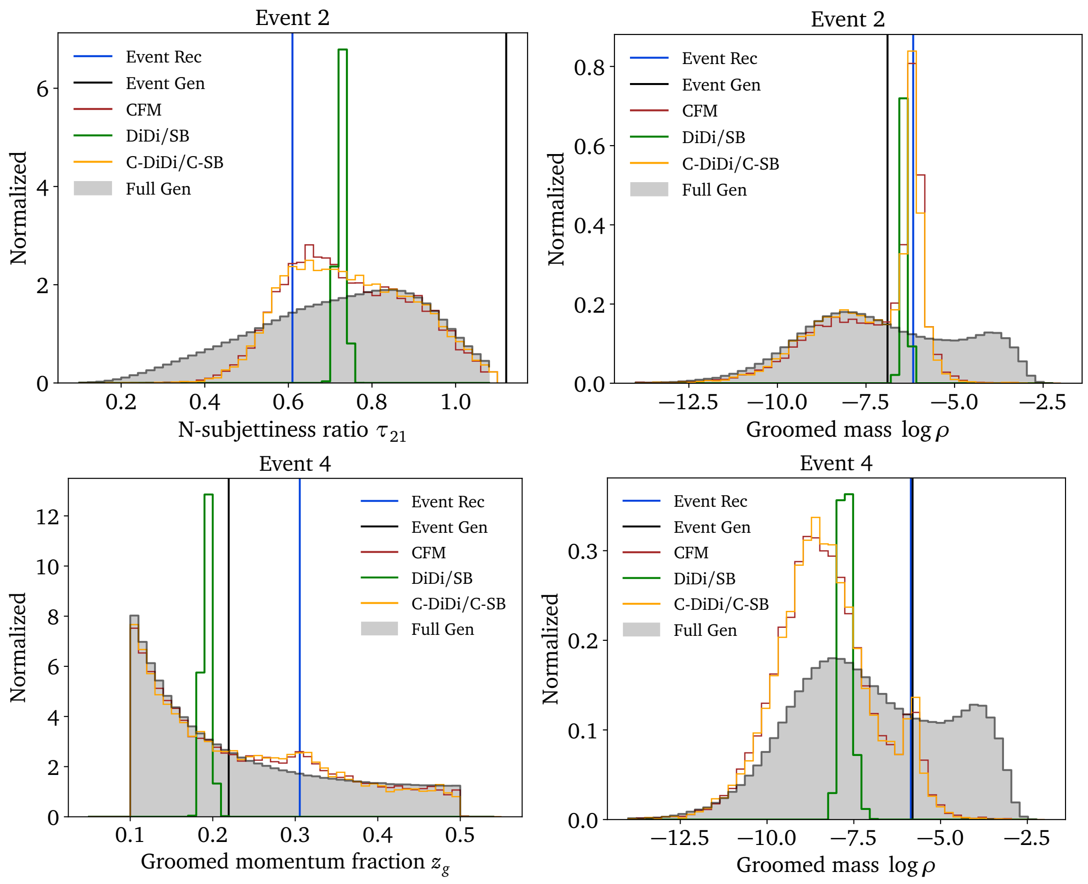


Figure 7.6: Posterior distributions obtained from unfolding single events with CFM, DiDi and C-DiDi on the 6D OmniFold dataset. Each of the three events is unfolded 10000 times. For reference, we also show the full gen-level distribution.

$$
\begin{aligned}
&= \frac{p_\text{true}(x_\text{rec}) p_\text{true}(x_\text{gen}|x_\text{rec})}{p_\theta(x_\text{rec}) p_\theta(x_\text{gen}|x_\text{rec})} \\
&= \frac{p_\text{true}(x_\text{gen}|x_\text{rec})}{p_\theta(x_\text{gen}|x_\text{rec})} \equiv w(x_\text{gen}|x_\text{rec}) \; . \qquad (7.48)
\end{aligned}
$$

Therefore, a classifier trained on the joint distributions gives us access to the likelihood ratio between the true and learned posterior distributions.

For each of our five networks we train a classifier, using the hyperparameters listed in Tab. A.12. First we only look at the gen-level unnfolded distributions and discriminate them from the true gen-level distribution. We show the corresponding weight distributions, evaluated on the generated events, in the left panel of Fig. 7.7. For all networks we see a dominant peak in one, indicating that for the overwhelming majority of events, the classifier cannot tell truth from generated. The tail towards lower weights indicates events which should not be there and which the classifier weight tries to remove, i.e. phase space regions that the network overpopulates. The right tail marks events and phase space regions underpopulated by the generative unfolding network.

Comparing the five networks and the three underlying methods, the CFM shows the smallest tails in both directions. The conditional DiDi network is almost on par with the CFM, the difference is covered by the classifier training fluctuations. Unconditional DiDi leads to a larger tail towards large weights, but hardly any events with small

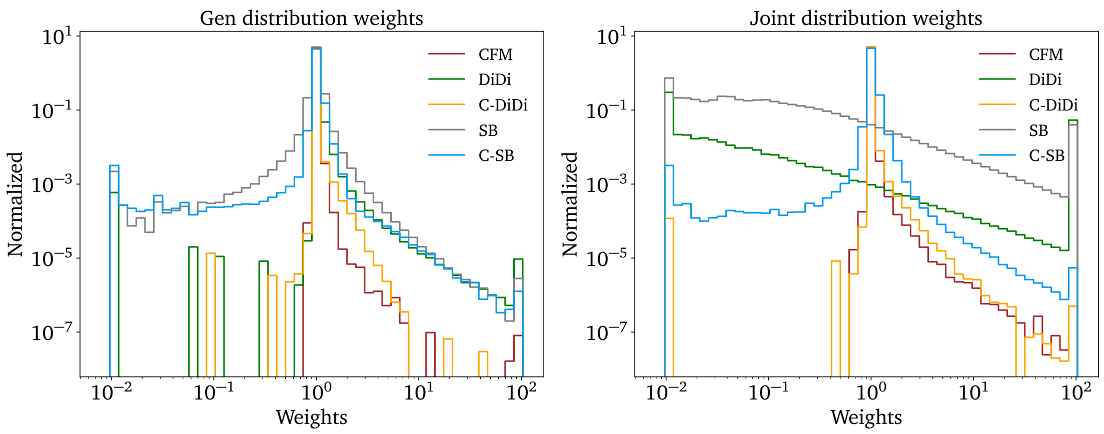


Figure 7.7: Classifier weight distributions for each network applied to the 6D OmniFold dataset. The left panel shows the gen-level weights according to Eq.(7.47), the right panel the joint distribution weights defined in Eq.(7.48).

weights, indicating a bias in learning features or their correlations. The SB networks show a slightly larger spread in the weight distribution, and the conditional SB is again significantly narrower than the unconditional version.

The right panel of Fig. 7.7 shows the weight distributions for the conditional phase space distributions. While for the conditional CFM and the conditional DM-networks the difference to the left panel is marginal, we now see that the unconditional DM-networks show little structure and large overflow bins, indicating that the learned joint distributions do not reproduce the training data.

7.4 Unfolding Substructure and Kinematic Properties

In order to stress-test the methods with a mixture of jet substructure and kinematic information, we simulated a dataset similar to the one used by a recent ATLAS analysis [29]. The resulting dataset has 22 instead of 6 dimensions, as in the previous dataset[7].

7.4.1 New $Z + 2$ jets dataset

We now consider the process

$$pp \to Z_{\mu\mu} + 2 \text{ jets} . \tag{7.49}$$

We generate the events with Madgraph 5 [32], showering and hadronization are simulated with Pythia 8.311 [290], and detector effects are included via Delphes 3.5.0 [36] using the default CMS card. Jets are clustered at gen-level and reco-level using an anti-k_T algorithm with $R = 0.4$ implemented in FastJet 3.3.4 [177].

We apply a set of selections resembling the ATLAS analysis: events are required to have exactly two muons with opposite charge and

$$p_{T,\mu} > 25 \text{ GeV} \qquad \text{and} \qquad m_{\mu\mu} \in [81, 101] \text{ GeV} . \tag{7.50}$$

[7]The OmniFold dataset does have the full set of jet constituents, but this is a high-dimensional variable-length set, which is beyond the scope of this paper.

Furthermore, we require at least two jets with

$$p_{T,j} > 10 \text{ GeV} \qquad \text{and} \qquad \Delta R_{\mu j} > 0.4 \; . \tag{7.51}$$

All events must pass all selections on gen and reco level (acceptance effects are small and ignored). The training set consists of 1.5M events, and the test set consists of 400k events.

Instead of restricting ourselves to jet substructure observables of the leading jet, we now unfold both kinematic information of the muons and leading jets as well as the substructure of the leading two jets. At the subjet level, we include the number of jet constituents N and the subjettiness variables τ_1, τ_2 and τ_3 [261] for each jets. In total this defines a 22-dimensional phase space to unfold into:

$$\Big((p_T, \eta, \phi)_{\mu_1,\mu_2} \, , \; (p_T, \eta, \phi, m, N, \tau_1, \tau_2, \tau_3)_{j_1,j_2} \Big) \; . \tag{7.52}$$

The challenge is to reproduce all correlations, most notably the di-muon kinematics and the angular separation R_{jj}. In contrast to classifier-based unfolding, generative unfolding does not easily allow to over-constrain the physical phase space with redundant degrees of freedom. Instead, we choose a phase space parametrization that makes key correlations directly accessible to the network [3, 212]. To this end, we replace $p_{T,\mu_1} \to m_{\mu\mu}$ in the event representation and extract it later via

$$p_{T,\mu 1} = \frac{m_{\mu\mu}^2}{2 p_{T,\mu_2} (\cosh \Delta\eta_{\mu\mu} - \cos \Delta\phi_{\mu\mu})} \; . \tag{7.53}$$

We standardize our training data in each dimension. For $m_{\mu\mu}$ we apply a Breit-Wigner mapping [3].

To accommodate the more challenging phase space, we replace the MLPs in all our generators with transformers [2, 3, 287]. We follow the transformer architecture proposed in Ref. [3], the network and training hyperparameters are listed in Tab. A.13 in the Appendix.

7.4.2 Unfolded distributions

The unfolding results obtained with all five networks representing all three methods are shown in Fig. 7.8. All of them unfold the bulk of the phase space distributions with a precision that is at the per-cent level. Small deviations from the true gen-level distributions as well as differences between different methods are only visible in the tails of the distributions or at hard edges. For $\eta_{\mu 2}$ and ϕ_{j2}, we expect the unfolding to be close to an identity mapping, with hard boundaries. While the DM-networks should be well-suited to those observables, we also find no performance loss for the classic CFM unfolding. Instead, we observe minor deviations of the (C-)SB networks at the hard edges, likely indicating a lack of expressivity in our specific implementation.

Since we turned it into a network input, the di-muon mass $m_{\mu\mu}$ is learned well by all networks. Moving on to complex correlations like the transverse momentum of the first muon, computed from Eq.(7.53), we still observe excellent agreement. With the exception of (C-)SB networks in the low-p_T region, the precision is at the level of a few per-cent except for fluctuating tails. A known challenge to all generative networks is the ΔR_{j1j2} distribution. This observable defines a non-trivial derived feature, combining collinear

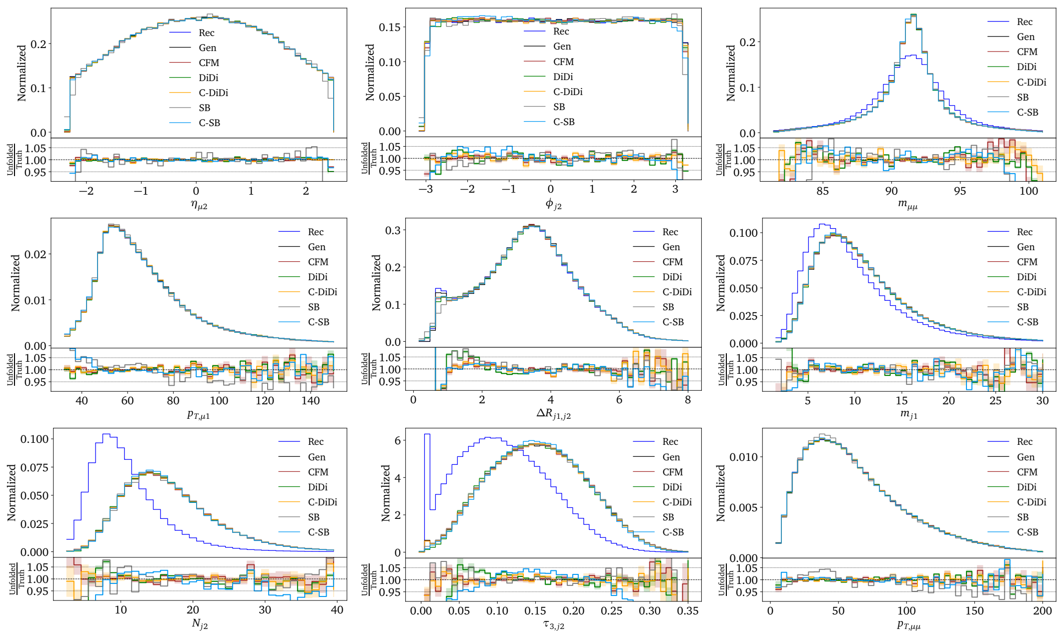


Figure 7.8: Unfolded distributions of the 22-dimensional Z+2 jets dataset using CFM, DiDi, C-DiDi, SB and C-SB.

enhancement with a hard phase cut. All five networks struggle with this feature, and all conditional networks show superior performance.

At the subjet level, the number of jet constituents, N_{j2}, tends to be larger at gen-level than at reco-level, since not all particle are eventually detected. This explains the strange peak at zero for τ_{3,j_2} at reco-level. Here, the jet algorithm clusters less than two particles within one jet, indeed giving $\tau_{3,j_2} = 0$. At gen-level this effect is highly suppressed. The CFM and C-DiDi manage to reliably unfold even this small excess of zeros, while the other networks fit through it. DiDi and the SB deviate above 10% from the truth when getting closer to the tails of the distributions. To compensate the SB is overpopulating the peak region.

7.5 Conclusion

In this paper, we have extended two distribution-mapping, ML-based unfolding methods, SBUnfold and DiDi and benchmarked them with an updated conditional generative unfolding method (CFM). The two distribution-mapping methods have been shown to accurately reproduce the marginal distributions of the target distributions, but were not able to model the correct detector response. By augmenting them with conditioning, C-SB and C-DiDi faithfully reproduce the detector response as well as the marginal target distributions. Like other ML-based unfolding methods, C-SB and C-DiDi are unbinned and readily extend to unfolding in many dimensions simultaneously.

We have started with a detailed discussion of the theoretical foundations of distribution mapping, conditional distribution mapping, and the relation of the SB and DiDi approaches. While this discussion is not needed to understand our results, it allows us to systematically understand the similarities and the benefits of the different generative unfolding architectures. In essence, C-SB and C-DiDi are two realization of the same

SDE description, and the main difference between the two implementations is the noise schedule chosen. In addition, within our (C)-SB implementation we sample discrete time steps during training, whereas all other models samples t continuously over a uniform distribution. The impact of conditioning was illustrated simply with a Gaussian example following the mathematical exposition.

Our first benchmark dataset is the well-studied OmniFold/MultiFold dataset composed of six substructure observables. All five networks representing the three methods, classic conditional generation, unconditional DM, and conditional DM, accurately reproduce the distributions in the target unfolding phase space. The two DM-methods are each implemented in Schrödinger Bridge and Direct Diffusion versions. Between our five networks and relative to the truth, the six marginal distributions vary at the per-cent level, with sizable deviations appearing only in the tails of distributions that are consistent with the uncertainty estimate from the Bayesian neural network implementations. The migration analysis for both conditional methods reproduce the physics encoded in the forward simulation. A classifier analysis of the generated unfolded events shows high precision and no clear failure mode for any of the three conditional networks.

Second, we apply generative unfolding to a new, high-dimensional $Z + 2$ jets dataset with 22 phase space dimensions, combining event-level and subjet phase spaces. Again, we find that all networks reproduce the true phase space distribution before the detector with high fidelity. For many of the networks, the one-dimensional marginal distributions agree with the truth at the per-cent level or better, and sizable deviations again appear only in kinematic tails.

With the release of the updated methods, codified in publicly available software, we have added a new ML-based unfolding method to the collider physics toolkit. Just as with classical unfolding, it is critical to have multiple, comparably accurate/precise techniques that have different methodological assumptions. Distribution mapping is a third type of method that can now be used to compare with standard conditional-generation and likelihood-ratio methods. Further work is required to fully integrate all aspects of a cross section measurement into generative unfolding (e.g. acceptance effects), but the core component (inverting the forward model) is now highly advanced. We look forward to the application of these methods to experimental data in the near future.

Chapter 8

Summary and Outlook

As particle physics enters the high-luminosity era of the LHC (HL-LHC), the enormous increase in data volume presents both immense opportunities and significant methodological challenges. LHC physics is in the privileged position of having access to a chain of very precise, mostly first-principles based simulation tools. This simulation chain is the backbone of the entire LHC physics program, however it is currently projected to become a major computational bottleneck if major upgrades are not implemented. Besides the simulation software, progress is also required in the inference methodology. Most traditional statistical tools are designed for specific hypotheses and become inadequate when searching for subtle, unknown signals in high-dimensional data.

Generative machine learning models offer a powerful new approach for learning complex, high-dimensional probability densities with remarkable accuracy. Their flexibility and scalability open up a broad range of applications in LHC physics, many of which are still being actively explored. This thesis addressed a subset of these opportunities, focusing on simulation and inference tasks where generative models can be used to mitigate existing computational and methodological limitations. For a broader overview of recent developments in this rapidly evolving field, see Ref. [291].

Chapter 4 systematically examined the capability of three modern generative architectures, DDPM diffusion models, CFM diffusion models and autoregressive transformers, for simulating LHC event data. We demonstrated that all three of these architectures can learn complex physical phase spaces to percent level precision, outperforming the previous state-of-the-art normalizing flows. We developed Bayesian network implementations that provided uncertainty estimates of the learned densities. Using toy models we analyzed the uncertainty patterns to extract insights into how the models learn the underlying densities. Diffusion models balanced expressivity and precision very well, most notably seen in the challenging Z-boson mass resonance. The autoregressive transformer excelled in learning angular jet correlations, however at the cost of a slightly mismodelled Z-resonance. Our results provided a strong motivation for the future use of these models. In particular Conditional Flow Matching (CFM), which we proposed for the first time in the context of particle physics, has since been established as the leading generative architecture in the field.

In Chapter 5, we addressed limitations of the classical matrix element method through generative machine learning. We developed a novel three-network framework that accounts for the transfer probability, efficiency effects and the computationally intensive phase space integration. Compared to prior work, we were able to significantly improve accuracy and computational efficiency of the method. We replaced the normalizing flows with advanced model architectures including CFMs and transformers, demonstrating how generative networks can efficiently handle the complex jet combinatorics and encode the transfer probability to percent-level precision. Using a measurement of the CP-violating top Yukawa coupling as a benchmark example, we showcased that our setup can accurately measure the underlying theory parameter, even with very limited event numbers.

Chapter 6 presented a comprehensive benchmark study of ML-based unfolding methods, systematically comparing reweighting (OmniFold), distribution mapping, and conditional generative unfolding. All methods demonstrated excellent performance, suggesting that experimental collaborations should consider adopting ML-based unfolding techniques. In particular the proposed transformer-enhanced diffusion model enabled unprecedented precision in unfolding the challenging multi-resonant $t\bar{t}$ phase space. We further showed how theory-inspired parametrizations can significantly enhance the models' capability to capture complex correlations. Using classifiers, we rigorously verified that no correlations were mismodeled and confirmed that the models truly learned the full high-dimensional distributions. Overall, this chapter highlighted the readiness of generative unfolding tools for LHC analyses and established clear benchmarks that future progress can be compared against.

Chapter 7 expanded the methodological toolbox further by introducing conditional versions of distribution mapping approaches, specifically Schrödinger Bridge and Direct Diffusion methods. These conditional approaches overcame previous limitations by accurately modeling both marginal distributions and detector responses. Through a detailed theoretical discussion, we clarified the connection between these two methods as well as the connection to diffusion models. Using a new 22-dimensional $Z + 2$ jets dataset, we demonstrated that they can achieve precision comparable to state-of-the-art CFM-based generative unfolding. The availability of complementary methods, each based on different theoretical frameworks, enhances the robustness and reliability of the unfolding toolkit.

Looking into the future, it is clear that generative machine learning techniques hold great promise for fundamentally reshaping statistical inference in collider physics. They not only address critical computational challenges of the HL-LHC era but also enable entirely new analytical methodologies. However, bridging the gap between phenomenological studies and real analyses requires further work. Future research should therefore prioritize the robust integration of these methods into actual LHC analyses, ensuring they reliably work under realistic experimental conditions. This includes, but is not limited to, comprehensive investigations of uncertainties and improved techniques to deal with simulation-data differences. Once these questions are answered, physicists have new possibilities to navigate the enormous datasets ahead, maximizing discovery potential and bringing us closer to answering the fundamental open questions that extend beyond the current Standard Model of particle physics.

Chapter A

Hyperparameters

A.1 Networks from Chapter 4

hyperparameter	toy models	LHC events
Timesteps	1000	1000
Time Embedding Dimension	-	64
# Blocks	1	2
Layers per Block	8	5
Intermediate Dimensions	40	64
# Model Parameters	20k	75k
LR Scheduling	one-cycle	one-cycle
Starter LR	10^{-4}	10^{-4}
Maximum LR	10^{-3}	10^{-3}
Epochs	1000	1000, 3000, 10000
Batch Size	8192	8192, 8192, 4096
# Training Events	600k	3.2M, 850k, 190k
# Generated Events	1M	1M, 1M, 1M

Table A.1: Training setup and hyperparameters for the Bayesian DDPM generator.

hyperparameter	toy models	LHC events
Embedding Dimension	-	32
# Blocks	1	2
Layers per Block	8	5
Intermediate Dimensions	40	128, 64, 64
# Model Parameters	20k	265k, 85k, 85k
LR Scheduling	cosine annealing	cosine annealing
Starter LR	10^{-2}	10^{-3}
Epochs	1000	1000, 5000, 10000
Batch Size	8192	16384
# Training Events	600k	3.2M, 850k, 190k
# Generated Events	1M	1M, 1M, 1M

Table A.2: Training setup and hyperparameters for the Bayesian CFM generator.

hyperparameter	toy models	LHC events
# Gaussians m	21	43
# Bins m	64	-
# TransformerDecoder N	4	4
# Self-attention Heads	4	4
Latent Space Size d	64	128
# Model Parameters	220k	900k
LR Scheduling	one-cycle	one-cycle
Starter LR	3×10^{-4}	10^{-4}
Maximum LR	3×10^{-3}	10^{-3}
Epochs	200	2000
Batch Size	1024	1024
RAdam ϵ	10^{-8}	10^{-4}
# Training Events	600k	2.4M, 670k, 190k
# Generated Events	600k	1M, 1M, 1M

Table A.3: Training setup and hyperparameters for the Bayesian autoregressive transformer.

A.2 Networks from Chapter 5

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.4: Hyperparameters of the classifiers learning the acceptance $\epsilon(x_\text{hard})$ (left) and the jet multiplicity used in Appendix 5.7 (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

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.5: Hyperparameters of the CFM (left) and the Transfermer (right).

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

Parameter	Value
Optimizer	Adam
Learning rate	0.001
LR schedule	Cosine-annealing
Batch size	8192
Epochs	600
Number of heads	4
Number of encoder layers	6
Number of decoder layers	6
Embedding dimension	128
Transf. feed-forward dim	512
Training samples	3.4M
Validation samples	340k
Testing samples	3.1M
Trainable parameters	2.7M
ODE solver method	Runge-Kutta, order 4
Solver step-size	0.05

Table A.6: Hyperparameters of the autoregressive Transfusion (left) and the parallel Transfusion (right)

A.3 Networks from Chapter 6

Parameter	(b)OmniFold Py-to-Py	(b)OmniFold He-to-Py
Optimizer	Adam	
Learning rate	0.001	
LR schedule	Cosine annealing	
Batch size	128	
Epochs	30	500
Network	MLP	
Number of layers	4	
Hidden nodes	80	
Bayesian regularization	1	1

Table A.7: Network and training hyperparameters for the OmniFold and bOmniFold networks in Figs. 6.3, 6.4, and 6.5.

Parameter	DiDi	CFM Z+jets	CFM $t\bar{t}$	TraCFM
Optimizer	Adam			
Learning rate	0.001			
LR schedule	Cosine annealing			
Batch size	16384			
Epochs	500	400	1000	500
Network	MLP	MLP	MLP	Transformer
Number of layers	8	8	8	-
Hidden nodes	512	512	1024	-
Transformer blocks	-	-	-	6
Transformer heads	-	-	-	4
Embedding dim	-	-	-	128
Bayesian regularization	1			

Table A.8: Network and training hyperparameters for the Direct Diffusion and CFM networks in Figs. 6.6, 6.7, 6.10, and 6.11

Parameter	cINN Z+jets	cINN $t\bar{t}$	Transfermer
Optimizer	Adam		
Max Learning rate	0.0003		
LR schedule	One cycle		
Batch size	1024		
Epochs	75	130	250
Network	RQS-INN	RQS-INN	Transformer+RQS
INN blocks	10	20	1
RQS bins	24	30	30
Subnet layers	5	5	5
Subnet dim	200	256	256
Transformer blocks	-	-	6
Transformer heads	-	-	4
Embedding dim	-	-	128

Table A.9: Network and training hyperparameters for the cINN and Transfermer in Figs. 6.7 and 6.10.

Parameter	SB
Optimizer	Adam
Learning rate	0.001
Batch size	128
Network Updates	250000
Network	Fully connected ResNet
Blocks	6
MLP size	256

Table A.10: Network and training hyperparameters for the Schrödinger Bridge in Fig. 6.6.

Parameter	VLD Z+jets	VLD $t\bar{t}$
Optimizer	Adam	Adam
Initial Learning rate	5×10^{-4}	5×10^{-4}
Fine-tune Learning rate	1×10^{-4}	1×10^{-4}
Batch size	1024	1024
Updates	1 Million	1 Million
Hidden Dimensions	64	64
Denoising Layers	8	8
Detector Encoder Layers	6	6
Part* Encoder Layers	6	6
Part* Decoder Layers	6	6

Table A.11: Network and training hyperparameters for the VLD networks in Figs 6.7, 6.10, and 6.11.

A.4 Networks from Chapter 7

Parameter	CFM	DiDi	C-DiDi	SB	C-SB	Classifier
Optimizer		Adam		Adam		Adam
Learning rate		0.001		0.001		0.001
LR schedule		Cosine annealing		Exponential decay		Cosine annealing
Batch size		16384		128		128
Epochs		300		20		50
Network		MLP		MLP		MLP
Number of layers		5		6		5
Hidden nodes		128		256		256
Dropout		-		-		0.1
Noise scale	-	0.1	0.1	0.1	0.1	-
BNN regularization		1		-		-

Table A.12: Network and training hyperparameters for all networks trained on the 6d jet substructure dataset. Results are shown in Figs. 7.4, 7.5, and 7.6

Parameter	CFM	DiDi	C-DiDi	SB	C-SB
Optimizer		Adam		Adam	
Learning rate		0.001		0.001	
LR schedule		Cosine annealing		Exponential decay	
Batch size		16384		128	
Epochs	500	500	2000	200	200
Network		Transformer		Transformer	
Embedding dim		64		64	
Transformer blocks		6		6	
Attention heads		4		4	
Feedforward dim		256		256	
Noise scale	-	0.001	0.1	0.1	0.1
BNN regularization		1		-	

Table A.13: Network and training hyperparameters for all networks trained on the full-dimensional Z+2j dataset. Results shown in Figs. 7.8

Acknowledgment

First, I would like to thank my advisors Anja Butter and Tilman Plehn for giving me the opportunity to do my PhD in the Heidelberg Pheno group despite my questionable loyalty to particle physics. I am grateful for all the opportunities to research, to travel, and most importantly to learn. After these three years, I am convinced that the physicist way is the right way to do machine learning.

I also want to thank Anja Butter and Ullrich Köthe for refereeing this thesis, and Tilman Plehn and Ulrich Uwer for completing my committee.

Next, I would like to thank Sofia Palacios Schweitzer for all the support, culminating in proof-reading this thesis on Easter Sunday. In this context I want to also thank ChatGPT for proof-reading this thesis.

The research presented in this thesis would not have been possible without my collaborators. I am very grateful to all of them, in particular to Sofia Palacios Schweitzer, Jonas Spinner, Theo Heimel, Ramon Winterhalder, Javier Mariño, Sascha Diefenbacher, Vinicius Mikuni, and Benjamin Nachman.

A special thanks to all the current and former members of the Heidelberg Pheno and Cosmo groups for creating a fun and inspiring environment.

Finally, I would like to thank my family and friends for their unwavering support.

Bibliography

[1] A. Butter, N. Huetsch, S. Palacios Schweitzer, T. Plehn, P. Sorrenson and J. Spinner, *Jet Diffusion versus JetGPT – Modern Networks for the LHC*, SciPost Phys. Core **8**, 026 (2025), doi:10.21468/SciPostPhysCore.8.1.026, arXiv:2305.10475.

[2] T. Heimel, N. Huetsch, R. Winterhalder, T. Plehn and A. Butter, *Precision-machine learning for the matrix element method*, SciPost Phys. **17**, 129 (2024), doi:10.21468/SciPostPhys.17.5.129, arXiv:2310.07752.

[3] N. Huetsch *et al.*, *The landscape of unfolding with machine learning*, SciPost Phys. **18**, 070 (2025), doi:10.21468/SciPostPhys.18.2.070, arXiv:2404.18807.

[4] A. Butter, S. Diefenbacher, N. Huetsch, V. Mikuni, B. Nachman, S. Palacios Schweitzer and T. Plehn, *Generative Unfolding with Distribution Mapping* (2024), arXiv:2411.02495.

[5] T. Heimel, N. Huetsch, F. Maltoni, O. Mattelaer, T. Plehn and R. Winterhalder, *The MadNIS reloaded*, SciPost Phys. **17**, 023 (2024), doi:10.21468/SciPostPhys.17.1.023, arXiv:2311.01548.

[6] S. L. Glashow, *Partial Symmetries of Weak Interactions*, Nucl. Phys. **22**, 579 (1961), doi:10.1016/0029-5582(61)90469-2.

[7] S. Weinberg, *A Model of Leptons*, Phys. Rev. Lett. **19**, 1264 (1967), doi:10.1103/PhysRevLett.19.1264.

[8] A. Salam, *Weak and Electromagnetic Interactions*, Conf. Proc. C **680519**, 367 (1968), doi:10.1142/9789812795915_0034.

[9] G. 't Hooft and M. J. G. Veltman, *Regularization and Renormalization of Gauge Fields*, Nucl. Phys. B **44**, 189 (1972), doi:10.1016/0550-3213(72)90279-9.

[10] H. Fritzsch, M. Gell-Mann and H. Leutwyler, *Advantages of the Color Octet Gluon Picture*, Phys. Lett. B **47**, 365 (1973), doi:10.1016/0370-2693(73)90625-4.

[11] D. J. Gross and F. Wilczek, *Ultraviolet Behavior of Non-Abelian Gauge Theories*, Phys. Rev. Lett. **30**, 1343 (1973), doi:10.1103/PhysRevLett.30.1343.

[12] H. D. Politzer, *Reliable Perturbative Results for Strong Interactions?*, Phys. Rev. Lett. **30**, 1346 (1973), doi:10.1103/PhysRevLett.30.1346.

[13] P. A. R. Ade *et al.*, *Planck 2015 results. XIII. Cosmological parameters*, Astron. Astrophys. **594**, A13 (2016), doi:10.1051/0004-6361/201525830, arXiv:1502.01589.

[14] R. Adam *et al.*, *Planck 2015 results. I. Overview of products and scientific results*, Astron. Astrophys. **594**, A1 (2016), doi:10.1051/0004-6361/201527101, arXiv:1502.01582.

[15] Y. Fukuda *et al.*, *Evidence for oscillation of atmospheric neutrinos*, Phys. Rev. Lett. **81**, 1562 (1998), doi:10.1103/PhysRevLett.81.1562, arXiv:hep-ex/9807003.

[16] Q. R. Ahmad *et al.*, *Direct evidence for neutrino flavor transformation from neutral current interactions in the Sudbury Neutrino Observatory*, Phys. Rev. Lett. **89**, 011301 (2002), doi:10.1103/PhysRevLett.89.011301, arXiv:nucl-ex/0204008.

[17] E. W. Kolb and M. S. Turner, *The Early Universe*, vol. 69, Taylor and Francis, ISBN 978-0-429-49286-0, 978-0-201-62674-2, doi:10.1201/9780429492860 (2019).

[18] G. Steigman, *Observational tests of antimatter cosmologies*, Ann. Rev. Astron. Astrophys. **14**, 339 (1976), doi:10.1146/annurev.aa.14.090176.002011.

[19] P. W. Higgs, *Broken symmetries, massless particles and gauge fields*, Phys. Lett. **12**, 132 (1964), doi:10.1016/0031-9163(64)91136-9.

[20] 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.

[21] 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.

[22] *ATLAS Software and Computing HL-LHC Roadmap*, Tech. rep., CERN, Geneva (2022).

[23] T. Plehn, A. Butter, B. Dillon, T. Heimel, C. Krause and R. Winterhalder, *Modern Machine Learning for LHC Physicists* (2022), arXiv:2211.01421.

[24] 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.

[25] A. Butter and T. Plehn, *Generative Networks for LHC events* (2020), arXiv:2008.08558.

[26] 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.

[27] G. Aad *et al.*, *An implementation of neural simulation-based inference for parameter estimation in ATLAS* (2024), arXiv:2412.01600.

[28] V. Chekhovsky *et al.*, *Model-agnostic search for dijet resonances with anomalous jet substructure in proton-proton collisions at $\sqrt{s} = 13$ TeV* (2024), arXiv:2412.03747.

[29] G. Aad *et al.*, *Simultaneous Unbinned Differential Cross-Section Measurement of Twenty-Four Z+jets Kinematic Observables with the ATLAS Detector*, Phys. Rev. Lett. **133**, 261803 (2024), doi:10.1103/PhysRevLett.133.261803, arXiv:2405.20041.

[30] J. M. Campbell *et al.*, *Event generators for high-energy physics experiments*, SciPost Phys. **16**, 130 (2024), doi:10.21468/SciPostPhys.16.5.130, arXiv:2203.11110.

[31] 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.

[32] 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.

[33] 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.

[34] S. Agostinelli *et al.*, *GEANT4 - A Simulation Toolkit*, Nucl. Instrum. Meth. A **506**, 250 (2003), doi:10.1016/S0168-9002(03)01368-8.

[35] J. Bellm *et al.*, *Herwig 7.1 Release Note* (2017), arXiv:1705.06919.

[36] 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.

[37] D. Rezende and S. Mohamed, *Variational inference with normalizing flows*, In F. Bach and D. Blei, eds., *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538. PMLR, Lille, France (2015), arXiv:1505.05770.

[38] L. Ardizzone, C. Lüth, J. Kruse, C. Rother and U. Köthe, *Guided image generation with conditional invertible neural networks* (2019), arXiv:1907.02392.

[39] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative adversarial networks* (2014), arXiv:1406.2661.

[40] 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.

[41] B. Hashemi, N. Amin, K. Datta, D. Olivito and M. Pierini, *LHC analysis-specific datasets with Generative Adversarial Networks* (2019), arXiv:1901.05282.

[42] 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.

[43] 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.

[44] Y. Alanazi *et al.*, *Simulation of electron-proton scattering events by a Feature-Augmented and Transformed Generative Adversarial Network (FAT-GAN)* (2020), doi:10.24963/ijcai.2021/293, arXiv:2001.11103.

[45] 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.

[46] I. Kobyzev, S. Prince and M. Brubaker, *Normalizing flows: An introduction and review of current methods*, IEEE Transactions on Pattern Analysis and Machine Intelligence p. 1–1 (2020), doi:10.1109/tpami.2020.2992934, `http://dx.doi.org/10.1109/TPAMI.2020.2992934`.

[47] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed and B. Lakshminarayanan, *Normalizing flows for probabilistic modeling and inference* (2019), arXiv:1912.02762.

[48] I. Kobyzev, S. Prince and M. A. Brubaker, *Normalizing flows: An introduction and review of current methods* (2019), arXiv:1908.09257.

[49] T. Müller, B. McWilliams, F. Rousselle, M. Gross and J. Novák, *Neural importance sampling* (2018), arXiv:1808.03856.

[50] J. Ho, A. Jain and P. Abbeel, *Denoising diffusion probabilistic models* (2020), `https://arxiv.org/abs/2006.11239`, arXiv:2006.11239.

[51] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel and M. Le, *Flow matching for generative modeling* (2023), arXiv:2210.02747.

[52] 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), arXiv:1706.03762.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] G. Cowan, *A survey of unfolding methods for particle physics*, Conf. Proc. C **0203181**, 248 (2002).

[58] F. Spano, *Unfolding in particle physics: a window on solving inverse problems*, EPJ Web Conf. **55**, 03002 (2013), doi:10.1051/epjconf/20135503002.

[59] M. Arratia *et al.*, *Publishing unbinned differential cross section results*, JINST **17**, P01024 (2022), doi:10.1088/1748-0221/17/01/P01024, arXiv:2109.13243.

[60] L. Brenner, P. Verschuuren, R. Balasubramanian, C. Burgard, V. Croft, G. Cowan and W. Verkerke, *Comparison of unfolding methods using RooFitUnfold* (2020), `https://arxiv.org/abs/1910.14654`, arXiv:1910.14654.

[61] K. Datta, D. Kar and D. Roy, *Unfolding with Generative Adversarial Networks* (2018), arXiv:1806.00433.

[62] 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.

[63] 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.

[64] 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.

[65] M. Backes, A. Butter, M. Dunford and B. Malaescu, *An unfolding method based on conditional invertible neural networks (cINN) using iterative training*, SciPost Phys. Core **7**, 007 (2024), doi:10.21468/scipostphyscore.7.1.007, arXiv:2212.08674.

[66] L. Favaro, R. Kogler, A. Paasch, S. Palacios Schweitzer, T. Plehn and D. Schwarz, *How to Unfold Top Decays* (2025), arXiv:2501.12363.

[67] 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.

[68] S. Diefenbacher, G.-H. Liu, V. Mikuni, B. Nachman and W. Nie, *Improving generative model-based unfolding with Schrödinger bridges*, Phys. Rev. D **109**, 076011 (2024), doi:10.1103/PhysRevD.109.076011, arXiv:2308.12351.

[69] A. Butter, T. Jezo, M. Klasen, M. Kuschick, S. Palacios Schweitzer and T. Plehn, *Kicking it off(-shell) with direct diffusion*, SciPost Phys. Core **7**, 064 (2024), doi:10.21468/SciPostPhysCore.7.3.064, arXiv:2311.17175.

[70] T. Plehn, *Lectures on LHC Physics*, Lect. Notes Phys. **844**, 1 (2012), doi:10.1007/978-3-642-24040-9, arXiv:0910.4182.

[71] M. D. Schwartz, *Quantum Field Theory and the Standard Model*, Cambridge University Press, ISBN 978-1-107-03473-0, 978-1-107-03473-0 (2014).

[72] M. E. Peskin and D. V. Schroeder, *An Introduction to quantum field theory*, Addison-Wesley, Reading, USA, ISBN 978-0-201-50397-5, 978-0-429-50355-9, 978-0-429-49417-8, doi:10.1201/9780429503559 (1995).

[73] G. Altarelli and G. Parisi, *Asymptotic Freedom in Parton Language*, Nucl. Phys. B **126**, 298 (1977), doi:10.1016/0550-3213(77)90384-4.

[74] 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).

[75] V. N. Gribov and L. N. Lipatov, *Deep inelastic e p scattering in perturbation theory*, Sov. J. Nucl. Phys. **15**, 438 (1972).

[76] 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.

[77] S. Weinzierl, *Introduction to monte carlo methods* (2000), arXiv:hep-ph/0006269.

[78] G. P. Lepage, *A New Algorithm for Adaptive Multidimensional Integration*, J. Comput. Phys. **27**, 192 (1978), doi:10.1016/0021-9991(78)90004-9.

[79] G. P. Lepage, *VEGAS: AN ADAPTIVE MULTIDIMENSIONAL INTEGRATION PROGRAM* (1980).

[80] G. P. Lepage, *Adaptive multidimensional integration: VEGAS enhanced*, J. Comput. Phys. **439**, 110386 (2021), doi:10.1016/j.jcp.2021.110386, arXiv:2009.05112.

[81] 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.

[82] 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.

[83] T. Heimel, O. Mattelaer, T. Plehn and R. Winterhalder, *Differentiable MadNIS-Lite*, SciPost Phys. **18**, 017 (2025), doi:10.21468/SciPostPhys.18.1.017, arXiv:2408.01486.

[84] K. G. Wilson, *Confinement of Quarks*, Phys. Rev. D **10**, 2445 (1974), doi:10.1103/PhysRevD.10.2445.

[85] 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.

[86] B. Andersson, G. Gustafson and B. Soderberg, *A General Model for Jet Fragmentation*, Z. Phys. C **20**, 317 (1983), doi:10.1007/BF01407824.

[87] T. Sjostrand, *Jet Fragmentation of Nearby Partons*, Nucl. Phys. B **248**, 469 (1984), doi:10.1016/0550-3213(84)90607-2.

[88] 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.

[89] S. Chatrchyan *et al.*, *The CMS Experiment at the CERN LHC*, JINST **3**, S08004 (2008), doi:10.1088/1748-0221/3/08/S08004.

[90] 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.

[91] 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.

[92] P. Musella and F. Pandolfi, *Fast and Accurate Simulation of Particle Detectors Using Generative Adversarial Networks*, Comput. Softw. Big Sci. **2**, 8 (2018), doi:10.1007/s41781-018-0015-y, arXiv:1805.00850.

[93] 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.

[94] 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.

[95] 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.

[96] 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.

[97] 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.

[98] 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.

[99] 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.

[100] 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: Workshop on Machine Learning and the Physical Sciences* (2022), arXiv:2211.15380.

[101] 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.

[102] A. Xu, S. Han, X. Ju and H. Wang, *Generative machine learning for detector response modeling with a conditional normalizing flow*, JINST **19**, P02003 (2024), doi:10.1088/1748-0221/19/02/P02003, arXiv:2303.10148.

[103] 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.

[104] 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.

[105] 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.

[106] B. Hashemi, N. Hartmann, S. Sharifzadeh, J. Kahn and T. Kuhr, *Ultra-high-granularity detector simulation with intra-event aware generative adversarial network and self-supervised relational reasoning*, Nature Commun. **15**, 4916 (2024),

doi:10.1038/s41467-024-49104-4, [Erratum: Nature Commun. 115, 5825 (2024)], arXiv:2303.08046.

[107] 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.

[108] 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.

[109] 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.

[110] S. Schmitt, *TUnfold: an algorithm for correcting migration effects in high energy physics*, JINST **7**, T10003 (2012), doi:10.1088/1748-0221/7/10/T10003, arXiv:1205.6201.

[111] G. D'Agostini, *A Multidimensional unfolding method based on Bayes' theorem*, Nucl. Instrum. Meth. A **362**, 487 (1995), doi:10.1016/0168-9002(95)00274-X.

[112] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman, A. Suresh and J. Thaler, *Scaffolding Simulations with Deep Learning for High-dimensional Deconvolution*, In *9th International Conference on Learning Representations* (2021), arXiv:2105.04448.

[113] Y. LeCun, Y. Bengio and G. Hinton, *Deep learning*, Nature **521**, 436 (2015), doi:10.1038/nature14539.

[114] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization* (2017), `https://arxiv.org/abs/1412.6980`, arXiv:1412.6980.

[115] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors* (2012), arXiv:1207.0580.

[116] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift* (2015), arXiv:1502.03167.

[117] A. Krizhevsky, I. Sutskever and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, In F. Pereira, C. Burges, L. Bottou and K. Weinberger, eds., *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc. (2012).

[118] R. M. Schmidt, *Recurrent neural networks (rnns): A gentle introduction and overview* (2019), arXiv:1912.05911.

[119] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother and U. Köthe, *Analyzing inverse problems with invertible neural networks* (2018), arXiv:1808.04730.

[120] L. Dinh, J. Sohl-Dickstein and S. Bengio, *Density estimation using real nvp* (2016), arXiv:1605.08803.

[121] D. P. Kingma and P. Dhariwal, *Glow: Generative flow with invertible 1x1 convolutions* (2018), arXiv:1807.03039.

[122] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan and S. Ganguli, *Deep unsupervised learning using nonequilibrium thermodynamics*, CoRR **abs/1503.03585** (2015), `http://arxiv.org/abs/1503.03585`, 1503.03585.

[123] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon and B. Poole, *Score-based generative modeling through stochastic differential equations* (2021), arXiv:2011.13456.

[124] C. Durkan, A. Bekasov, I. Murray and G. Papamakarios, *Cubic-spline flows* (2019), arXiv:1906.02145.

[125] 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.

[126] J. Hermans, V. Begy and G. Louppe, *Likelihood-free mcmc with amortized approximate ratio estimators* (2020), arXiv:1903.04057.

[127] G. Papamakarios, D. C. Sterratt and I. Murray, *Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows* (2019), arXiv:1805.07226.

[128] M. Glöckler, M. Deistler and J. H. Macke, *Variational methods for simulation-based inference* (2022), arXiv:2203.04176.

[129] G. Papamakarios and I. Murray, *Fast ϵ-free inference of simulation models with bayesian conditional density estimation* (2018), arXiv:1605.06376.

[130] J.-M. Lueckmann, P. J. Goncalves, G. Bassetto, K. Öcal, M. Nonnenmacher and J. H. Macke, *Flexible statistical inference for mechanistic models of neural dynamics* (2017), arXiv:1711.01861.

[131] S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone and U. Köthe, *Bayesflow: Learning complex stochastic models with invertible neural networks* (2020), arXiv:2003.06281.

[132] M. Dax, J. Wildberger, S. Buchholz, S. R. Green, J. H. Macke and B. Schölkopf, *Flow matching for scalable simulation-based inference* (2023), arXiv:2305.17161.

[133] M. D. Klimek and M. Perelstein, *Neural Network-Based Approach to Phase Space Integration* (2018), arXiv:1810.11509.

[134] 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.

[135] 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.

[136] 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.

[137] 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.

[138] 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.

[139] 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.

[140] 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.

[141] A. Butter, T. Plehn and R. Winterhalder, *How to GAN Event Subtraction*, SciPost Phys. Core **3**, 009 (2020), doi:10.21468/SciPostPhysCore.3.2.009, arXiv:1912.08824.

[142] B. Stienen and R. Verheyen, *Phase space sampling and inference from weighted events with autoregressive flows*, SciPost Phys. **10**, 038 (2021), doi:10.21468/SciPostPhys.10.2.038, arXiv:2011.13445.

[143] M. Backes, A. Butter, T. Plehn and R. Winterhalder, *How to GAN Event Unweighting*, SciPost Phys. **10**, 089 (2021), doi:10.21468/SciPostPhys.10.4.089, arXiv:2012.07873.

[144] F. A. Di Bello, S. Ganguly, E. Gross, M. Kado, M. Pitt, L. Santi and J. Shlomi, *Towards a Computer Vision Particle Flow*, Eur. Phys. J. C **81**, 107 (2021), doi:10.1140/epjc/s10052-021-08897-0, arXiv:2003.08863.

[145] P. Baldi, L. Blecher, A. Butter, J. Collado, J. N. Howard, F. Keilbach, T. Plehn, G. Kasieczka and D. Whiteson, *How to GAN Higher Jet Resolution*, SciPost Phys. **13**, 064 (2022), doi:10.21468/SciPostPhys.13.3.064, arXiv:2012.11944.

[146] 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.

[147] 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.

[148] A. Andreassen, I. Feige, C. Frye and M. D. Schwartz, *JUNIPR: a Framework for Unsupervised Machine Learning in Particle Physics*, Eur. Phys. J. **C79**, 102 (2019), doi:10.1140/epjc/s10052-019-6607-9, arXiv:1804.09720.

[149] K. Dohi, *Variational Autoencoders for Jet Simulation* (2020), arXiv:2009.04842.

[150] 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.

[151] 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*, SciPost Phys. **16**, 018 (2024), doi:10.21468/SciPostPhys.16.1.018, arXiv:2303.05376.

[152] 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.

[153] 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.

[154] 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.

[155] 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`.

[156] R. M. Neal, *Bayesian learning for neural networks*, Ph.D. thesis, Toronto (1995).

[157] Y. Gal, *Uncertainty in Deep Learning*, Ph.D. thesis, Cambridge (2016).

[158] A. Kendall and Y. Gal, *What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?*, Proc. NIPS (2017), arXiv:1703.04977.

[159] 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.

[160] 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.

[161] 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.

[162] 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.

[163] R. Winterhalder, M. Bellagente and B. Nachman, *Latent Space Refinement for Deep Generative Models* (2021), arXiv:2106.00792.

[164] M. S. Albergo and E. Vanden-Eijnden, *Building normalizing flows with stochastic interpolants* (2023), arXiv:2209.15571.

[165] X. Liu, C. Gong and Q. Liu, *Flow straight and fast: Learning to generate and transfer data with rectified flow* (2022), arXiv:2209.03003.

[166] 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.

[167] D. Maître and H. Truong, *A factorisation-aware Matrix element emulator*, JHEP **11**, 066 (2021), doi:10.1007/JHEP11(2021)066, arXiv:2107.06625.

[168] G. Louppe, K. Cho, C. Becot and K. Cranmer, *QCD-Aware Recursive Neural Networks for Jet Physics*, JHEP **01**, 057 (2019), doi:10.1007/JHEP01(2019)057, arXiv:1702.00748.

[169] J. Liu, A. Ghosh, D. Smith, P. Baldi and D. Whiteson, *Geometry-aware Autoregressive Models for Calorimeter Shower Simulations*, In *36th Conference on Neural Information Processing Systems: Workshop on Machine Learning and the Physical Sciences* (2022), arXiv:2212.08233.

[170] 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.

[171] D. P. Kingma, T. Salimans, B. Poole and J. Ho, *Variational diffusion models* (2023), `https://arxiv.org/abs/2107.00630`, arXiv:2107.00630.

[172] D. M. Blei, A. Kucukelbir and J. D. McAuliffe, *Variational inference: A review for statisticians*, Journal of the American statistical Association **112**, 859 (2017).

[173] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf *et al.*, *Pytorch: An imperative style, high-performance deep learning library* (2019), `https://arxiv.org/abs/1912.01703`, arXiv:1912.01703.

[174] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, *Language models are unsupervised multitask learners*, OpenAI blog **1**, 9 (2019).

[175] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao and J. Han, *On the variance of the adaptive learning rate and beyond* (2021), `https://arxiv.org/abs/1908.03265`, arXiv:1908.03265.

[176] 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.

[177] 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.

[178] 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, arXiv:2305.16774.

[179] K. Cranmer and T. Plehn, *Maximum significance at the LHC and Higgs decays to muons*, Eur. Phys. J. C **51**, 415 (2007), doi:10.1140/epjc/s10052-007-0309-4, arXiv:hep-ph/0605268.

[180] B. Abbott *et al.*, *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.

[181] V. M. Abazov *et al.*, *A precision measurement of the mass of the top quark*, Nature **429**, 638 (2004), doi:10.1038/nature02589, arXiv:hep-ex/0406031.

[182] A. Abulencia *et al.*, *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.

[183] 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.

[184] 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.

[185] 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.

[186] 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.

[187] J. R. Andersen, C. Englert and M. Spannowsky, *Extracting precise Higgs couplings by using the matrix element method*, Phys. Rev. D **87**, 015019 (2013), doi:10.1103/PhysRevD.87.015019, arXiv:1211.3011.

[188] 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.

[189] 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.

[190] D. E. Ferreira de Lima, O. Mattelaer and M. Spannowsky, *Searching for processes with invisible particles using a matrix element-based method*, Phys. Lett. B **787**, 100 (2018), doi:10.1016/j.physletb.2018.10.044, arXiv:1712.03266.

[191] 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.

[192] V. Khachatryan *et al.*, *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.

[193] G. Aad *et al.*, *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.

[194] V. Khachatryan *et al.*, *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.

[195] 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. D **94**, 055023 (2016), doi:10.1103/PhysRevD.94.055023, arXiv:1606.03107.

[196] 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.

[197] 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.

[198] 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.

[199] 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.

[200] 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.

[201] 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.

[202] C. Krause and D. Shih, *CaloFlow II: Even Faster and Still Accurate Generation of Calorimeter Showers with Normalizing Flows* (2021), arXiv:2110.11377.

[203] 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.

[204] ATLAS Collaboration, *Deep generative models for fast photon shower simulation in ATLAS* (2022), arXiv:2210.06204.

[205] C. Krause, I. Pang and D. Shih, *CaloFlow for CaloChallenge Dataset 1* (2022), arXiv:2210.14245.

[206] 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* (2023), arXiv:2305.04847.

[207] M. R. Buckley, C. Krause, I. Pang and D. Shih, *Inductive CaloFlow* (2023), arXiv:2305.11934.

[208] S. Diefenbacher, V. Mikuni and B. Nachman, *Refining Fast Calorimeter Simulations with a Schrödinger Bridge* (2023), arXiv:2308.12339.

[209] 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.

[210] 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.

[211] 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.

[212] J. Ackerschott, R. K. Barman, D. Gonçalves, T. Heimel and T. Plehn, *Returning CP-observables to the frames they belong*, SciPost Phys. **17**, 001 (2024), doi:10.21468/SciPostPhys.17.1.001, arXiv:2308.00027.

[213] 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.

[214] 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.

[215] 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.

[216] 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.

[217] 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.

[218] D. Sengupta, S. Klein, J. A. Raine and T. Golling, *CURTAINs flows for flows: Constructing unobserved regions with maximum likelihood estimation*, SciPost Phys. **17**, 046 (2024), doi:10.21468/SciPostPhys.17.2.046, arXiv:2305.04646.

[219] 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.

[220] 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.

[221] 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.

[222] 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.

[223] 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.

[224] 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.

[225] 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.

[226] 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.

[227] 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.

[228] M. Kraus, T. Martini, S. Peitzsch and P. Uwer, *Exploring BSM Higgs couplings in single top-quark production* (2019), arXiv:1908.09100.

[229] P. Artoisenet *et al.*, *A framework for Higgs characterisation*, JHEP **11**, 043 (2013), doi:10.1007/JHEP11(2013)043, arXiv:1306.6464.

[230] F. Demartin, F. Maltoni, K. Mawatari and M. Zaro, *Higgs production in association with a single top quark at the LHC*, Eur. Phys. J. C **75**, 267 (2015), doi:10.1140/epjc/s10052-015-3475-9, arXiv:1504.00611.

[231] G. Cowan, *Statistical data analysis*, ISBN 978-0-19-850156-5 (1998).

[232] 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.

[233] R. T. Q. Chen, Y. Rubanova, J. Bettencourt and D. Duvenaud, *Neural ordinary differential equations* (2019), arXiv:1806.07366.

[234] 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.

[235] T. Salimans and J. Ho, *Progressive distillation for fast sampling of diffusion models* (2022), arXiv:2202.00512.

[236] Y. Song, P. Dhariwal, M. Chen and I. Sutskever, *Consistency models* (2023), arXiv:2303.01469.

[237] 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*, JINST **19**, P04020 (2024), doi:10.1088/1748-0221/19/04/P04020, arXiv:2309.05704.

[238] I. Brivio, S. Bruggisser, F. Maltoni, R. Moutafis, T. Plehn, E. Vryonidou, S. Westhoff and C. Zhang, *O new physics, where art thou? A global search in the top sector*, JHEP **02**, 131 (2020), doi:10.1007/JHEP02(2020)131, arXiv:1910.03606.

[239] N. Elmer, M. Madigan, T. Plehn and N. Schmal, *Staying on Top of SMEFT-Likelihood Analyses* (2023), arXiv:2312.12502.

[240] L. B. Lucy, *An iterative technique for the rectification of observed distributions*, Astron. J. **79**, 745 (1974), doi:10.1086/111605.

[241] W. H. Richardson, *Bayesian-based iterative method of image restoration*, J. Opt. Soc. Am. **62**, 55 (1972), doi:10.1364/JOSA.62.000055, `http://www.osapublishing.org/abstract.cfm?URI=josa-62-1-55`.

[242] L. B. Lucy, *An iterative technique for the rectification of observed distributions*, Astron. J. **79**, 745 (1974), doi:10.1086/111605.

[243] A. Hocker and V. Kartvelishvili, *SVD approach to data unfolding*, Nucl. Instrum. Meth. A **372**, 469 (1996), doi:10.1016/0168-9002(95)01478-0, arXiv:hep-ph/9509307.

[244] V. Andreev *et al.*, *Measurement of Lepton-Jet Correlation in Deep-Inelastic Scattering with the H1 Detector Using Machine Learning for Unfolding*, Phys. Rev. Lett. **128**, 132002 (2022), doi:10.1103/PhysRevLett.128.132002, arXiv:2108.12376.

[245] H1 Collaboration, *Machine learning-assisted measurement of multi-differential lepton-jet correlations in deep-inelastic scattering with the H1 detector*, H1prelim-22-031 (2022), `https://www-h1.desy.de/h1/www/publications/htmlsplit/H1prelim-22-031.long.html`.

[246] V. Andreev *et al.*, *Unbinned deep learning jet substructure measurement in high Q2ep collisions at HERA*, Phys. Lett. B **844**, 138101 (2023), doi:10.1016/j.physletb.2023.138101, arXiv:2303.13620.

[247] H1 Collaboration, *Machine learning-assisted measurement of azimuthal angular asymmetries in deep-inelastic scattering with the H1 detector*, H1prelim-23-031 (2023), `https://www-h1.desy.de/h1/www/publications/htmlsplit/H1prelim-23-031.long.html`.

[248] R. Aaij *et al.*, *Multidifferential study of identified charged hadron distributions in Z-tagged jets in proton-proton collisions at $\sqrt{s}$ =13 TeV*, Phys. Rev. D **108**, L031103 (2023), doi:10.1103/PhysRevD.108.L031103, arXiv:2208.11691.

[249] P. T. Komiske, S. Kryhin and J. Thaler, *Disentangling quarks and gluons in CMS open data*, Phys. Rev. D **106**, 094021 (2022), doi:10.1103/PhysRevD.106.094021, arXiv:2205.04459.

[250] Y. Song, *Measurement of CollinearDrop jet mass and its correlation with SoftDrop groomed jet substructure observables in $\sqrt{s} = 200$ GeV pp collisions by STAR* (2023), arXiv:2307.07718.

[251] M. Vandegar, M. Kagan, A. Wehenkel and G. Louppe, *Neural Empirical Bayes: Source Distribution Estimation and its Applications to Simulation-Based Inference* (2020), arXiv:2011.05836.

[252] J. Neyman and E. S. Pearson, *On the problem of the most efficient tests of statistical hypotheses*, Phil. Trans. R. Soc. Lond. A **231**, 289 (1933).

[253] D. Molchanov, A. Ashukha and D. Vetrov, *Variational dropout sparsifies deep neural networks* (2017), arXiv:1701.05369.

[254] V. Fortuin, *Priors in bayesian deep learning: A review* (2022), arXiv:2105.06868.

[255] E. Schrödinger, *“über die Umkehrung der Naturgesetze,*, Sitzungsberichte der Preuss Akad. Wissen. Phys. Math. Klasse Sonderausgabe **9**, 144 (1931), doi:https://doi.org/10.1002/ange.19310443014.

[256] G.-H. Liu, A. Vahdat, D.-A. Huang, E. A. Theodorou, W. Nie and A. Anandkumar, *I^2sb: Image-to-image schrödinger bridge* (2023), arXiv:2302.05872.

[257] B. Nachman and V. Mikuni, *Large version of the omnifold dataset*, Zenodo, doi:10.5281/zenodo.10668638 (2024).

[258] M. Backes, A. Butter, M. Dunford and B. Malaescu, *Event-by-event comparison between machine-learning- and transfer-matrix-based unfolding methods*, Eur. Phys. J. C **84**, 770 (2024), doi:10.1140/epjc/s10052-024-13136-3, arXiv:2310.17037.

[259] A. J. Larkoski, S. Marzani, G. Soyez and J. Thaler, *Soft Drop*, JHEP **05**, 146 (2014), doi:10.1007/JHEP05(2014)146, arXiv:1402.2657.

[260] M. Dasgupta, A. Fregoso, S. Marzani and G. P. Salam, *Towards an understanding of jet substructure*, JHEP **09**, 029 (2013), doi:10.1007/JHEP09(2013)029, arXiv:1307.0007.

[261] J. Thaler and K. Van Tilburg, *Identifying Boosted Objects with N-subjettiness*, JHEP **03**, 015 (2011), doi:10.1007/JHEP03(2011)015, arXiv:1011.2268.

[262] G. Bewick *et al.*, *Herwig 7.3 release note*, Eur. Phys. J. C **84**, 1053 (2024), doi:10.1140/epjc/s10052-024-13211-9, arXiv:2312.05175.

[263] K. He, X. Zhang, S. Ren and J. Sun, *Deep residual learning for image recognition*, In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (2016).

[264] L. Ardizzone, T. Bungert, F. Draxler, U. Köthe, J. Kruse, R. Schmier and P. Sorrenson, *Framework for Easily Invertible Architectures (FrEIA)* (2018-2022).

[265] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li and J. Zhu, *DPM-solver++: Fast solver for guided sampling of diffusion probabilistic models* (2023), `https://openreview.net/forum?id=4vGwQqviud5`.

[266] O. Amram *et al.*, *CaloChallenge 2022: A Community Challenge for Fast Calorimeter Simulation* (2024), arXiv:2410.21611.

[267] G. Aad *et al.*, *Differential $t\bar{t}$ cross-section measurements using boosted top quarks in the all-hadronic final state with 139 fb^{-1} of ATLAS data*, JHEP **04**, 080 (2023), doi:10.1007/JHEP04(2023)080, arXiv:2205.02817.

[268] G. Aad *et al.*, *Measurements of top-quark pair single- and double-differential cross-sections in the all-hadronic channel in pp collisions at $\sqrt{s} = 13$ TeV using the ATLAS detector*, JHEP **01**, 033 (2021), doi:10.1007/JHEP01(2021)033, arXiv:2006.09274.

[269] G. Aad *et al.*, *Measurements of top-quark pair differential and double-differential cross-sections in the ℓ+jets channel with pp collisions at $\sqrt{s} = 13$ TeV using the ATLAS detector*, Eur. Phys. J. C **79**, 1028 (2019), doi:10.1140/epjc/s10052-019-7525-6, [Erratum: Eur.Phys.J.C 80, 1092 (2020)], arXiv:1908.07305.

[270] A. Tumasyan *et al.*, *Differential cross section measurements for the production of top quark pairs and of additional jets using dilepton events from pp collisions at $\sqrt{s}$ = 13 TeV*, JHEP **02**, 064 (2025), doi:10.1007/JHEP02(2025)064, arXiv:2402.08486.

[271] A. Tumasyan *et al.*, *Measurement of differential $t\bar{t}$ production cross sections in the full kinematic range using lepton+jets events from proton-proton collisions at $\sqrt{s}$ = 13 TeV*, Phys. Rev. D **104**, 092013 (2021), doi:10.1103/PhysRevD.104.092013, arXiv:2108.02803.

[272] A. M. Sirunyan *et al.*, *Measurement of differential* $\mathrm{t\bar{t}}$ *production cross sections using top quarks at large transverse momenta in pp collisions at $\sqrt{s}$ = 13 TeV*, Phys. Rev. D **103**, 052008 (2021), doi:10.1103/PhysRevD.103.052008, arXiv:2008.07860.

[273] A. M. Sirunyan *et al.*, *Measurement of the top quark polarization and* $\mathrm{t\bar{t}}$ *spin correlations using dilepton final states in proton-proton collisions at* $\sqrt{s} = 13$ *TeV*, Phys. Rev. D **100**, 072002 (2019), doi:10.1103/PhysRevD.100.072002, arXiv:1907.03729.

[274] A. M. Sirunyan *et al.*, *Measurement of* $\mathrm{t\bar{t}}$ *normalised multi-differential cross sections in pp collisions at* $\sqrt{s} = 13$ *TeV, and simultaneous determination of the strong coupling strength, top quark pole mass, and parton distribution functions*, Eur. Phys. J. C **80**, 658 (2020), doi:10.1140/epjc/s10052-020-7917-7, arXiv:1904.05237.

[275] M. V. Garzelli, J. Mazzitelli, S. O. Moch and O. Zenaiev, *Top-quark pole mass extraction at NNLO accuracy, from total, single- and double-differential cross sections for* $t\bar{t} + X$ *production at the LHC*, JHEP **05**, 321 (2024), doi:10.1007/JHEP05(2024)321, arXiv:2311.05509.

[276] M. Czakon, A. Mitov and R. Poncelet, *NNLO QCD corrections to leptonic observables in top-quark pair production and decay*, JHEP **05**, 212 (2021), doi:10.1007/JHEP05(2021)212, arXiv:2008.11133.

[277] P. von Platen, S. Patil, A. Lozhkov, P. Cuenca, N. Lambert, K. Rasul, M. Davaadorj, D. Nair, S. Paul, W. Berman, Y. Xu, S. Liu *et al.*, *Diffusers: State-of-the-art diffusion models*, `https://github.com/huggingface/diffusers` (2022).

[278] F. Ernst, L. Favaro, C. Krause, T. Plehn and D. Shih, *Normalizing Flows for High-Dimensional Detector Simulations*, SciPost Phys. **18**, 081 (2025), doi:10.21468/SciPostPhys.18.3.081, arXiv:2312.09290.

[279] T. Adye, *Unfolding algorithms and tests using RooUnfold*, In *PHYSTAT 2011*, pp. 313–318. CERN, Geneva, doi:10.5170/CERN-2011-006.313 (2011), arXiv:1105.1160.

[280] C.-C. Pan, X. Dong, Y.-C. Sun, A.-Y. Cheng, A.-B. Wang, Y.-X. Hu and H. Cai, *SwdFold:A Reweighting and Unfolding method based on Optimal Transport Theory* (2024), arXiv:2406.01635.

[281] K. Desai, B. Nachman and J. Thaler, *Moment extraction using an unfolding protocol without binning*, Phys. Rev. D **110**, 116013 (2024), doi:10.1103/PhysRevD.110.116013, arXiv:2407.11284.

[282] C. Pazos, S. Aeron, P.-H. Beauchemin, V. Croft, Z. Huan, M. Klassen and T. Wongjirad, *Towards Universal Unfolding of Detector Effects in High-Energy Physics using Denoising Diffusion Probabilistic Models* (2024), arXiv:2406.01507.

[283] V. D. Bortoli, G.-H. Liu, T. Chen, E. A. Theodorou and W. Nie, *Augmented bridge matching* (2023), arXiv:2311.06978.

[284] B. D. Anderson, *Reverse-time diffusion equation models*, Stochastic Processes and their Applications **12**, 313 (1982).

[285] J. Birk, E. Buhmann, C. Ewen, G. Kasieczka and D. Shih, *Flow matching beyond kinematics: Generating jets with particle identification and trajectory displacement information*, Phys. Rev. D **111**, 052008 (2025), doi:10.1103/PhysRevD.111.052008, arXiv:2312.00123.

[286] L. Favaro, A. Ore, S. P. Schweitzer and T. Plehn, *CaloDREAM – Detector Response Emulation via Attentive flow Matching*, SciPost Phys. **18**, 088 (2025), doi:10.21468/SciPostPhys.18.3.088, arXiv:2405.09629.

[287] J. Spinner, V. Bresó, P. de Haan, T. Plehn, J. Thaler and J. Brehmer, *Lorentz-Equivariant Geometric Algebra Transformers for High-Energy Physics* (2024), arXiv:2405.14806.

[288] J. Brehmer, V. Bresó, P. de Haan, T. Plehn, H. Qu, J. Spinner and J. Thaler, *A Lorentz-Equivariant Transformer for All of the LHC* (2024), arXiv:2411.00446.

[289] J. Doob, *Conditional brownian motion and the boundary limits of harmonic functions*, Bulletin de la Société Mathématique de France **85**, 431 (1957).

[290] 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.

[291] HEP ML Community, *A Living Review of Machine Learning for Particle Physics.*