Department of Physics and Astronomy
Heidelberg University

Bachelor Thesis in Physics
submitted by

Andreas Michael Hölzl

born in Hausham (Germany)

2024

Consistency Models for Fast and Scalable LHC Event Generation

This Bachelor Thesis has been carried out by Andreas Hölzl at the
Institute for Theoretical Physics Heidelberg University
under the supervision of
Prof. Tilman Plehn

Abstract

Score-based generative models, such as diffusion models and flow matching are considered state-of-the-art in generative machine learning. However, their reliance on iterative sampling results in slow generation. To overcome this limitation, we implement Consistency Models (CM), a new class of generative models. Designed as one-step generators that map directly from noise to data. CM also support multi-step sampling, allowing for a flexible trade-off between sample quality and speed. The CM implementation utilizes a Transformer architecture and is trained by distilling a pre-trained Conditional Flow Matching model. To compare the performance of these generative models, we use a classifier-based metric. Through experiments on a benchmark process involving Z boson plus jets, we demonstrate that CMs not only surpass the conditional flow matching models in terms of sampling speed but also achieve competitive sample quality.

Zusammenfassung

Score-basierte generative Modelle wie Diffusionsmodelle und Flow-Matching zählen derzeit zu den führenden Ansätzen im Bereich des generativen maschinellen Lernens. Diese Modelle generieren jedoch Daten durch das wiederholte Aufrufen eines neuronalen Netzwerks, was zu einer langsamen Generierung führt. Um diese Einschränkung zu überwinden, implementieren wir eine neue Klasse generativer Netzwerke, die sogenannten Consistency Models (CM). Consitency Models sind darauf ausgelegt, direkt von Rauschen zu Daten abzubilden, unterstützen jedoch auch einen mehrstufigen Generierungsprozess, der einen flexiblen Kompromiss zwischen Qualität und Geschwindigkeit bietet. Die Implementierung von CM nutzt eine Transformer-Architektur, die durch Distillation eines vortrainierten Conditional Flow Matching Modells trainiert wird. Um die Leistung zu bewerten, verwenden wir ein weiteres neuronales Netzwerk, das ausschließlich darauf trainiert ist, generierte Daten von echten Daten zu unterscheiden. Experimente mit einem Datensatz von Z-Bosonen und Jets zeigen, dass Consistency Models nicht nur die Conditional Flow Matching Modelle in Bezug auf die Geschwindigkeit übertreffen, sondern auch eine vergleichbare Qualität erreichen.

Contents

1 Introduction **1**

2 Particle Physics Fundamentals **3**
2.1 Standard model . . . 3
2.2 LHC Framework . . . 3
2.3 Kinematic Observables . . . 4
2.4 Drell-Yan Process at the LHC . . . 4

3 Machine Learning Fundamentals **6**
3.1 Deep learning . . . 6
3.1.1 Neural Networks . . . 6
3.1.2 Activation Functions . . . 7
3.1.3 Training Neural Networks . . . 8
3.1.4 Transformer and Self-Attention . . . 10

4 Novel Generative Networks **11**
4.1 Conditional Flow Matching . . . 11
4.2 Consistency Models . . . 14
4.3 Limitations of Generative Networks . . . 16
4.3.1 1D Metrics . . . 17
4.3.2 Classifier metric . . . 17
4.4 Related literature . . . 18

5 Experiments **20**
5.1 Dataset . . . 20
5.2 Training Conditional Flow Matching . . . 21
5.3 Training Consistency Models . . . 23
5.4 Wasserstein and Energy Metric . . . 25
5.5 Classifier Metric . . . 26
5.6 Performance and Comparison . . . 29

6 Conclusion and Outlook **30**

7 Appendix **34**
7.1 Hyperparameters . . . 34
7.2 Mathematics . . . 35
7.3 Kinematics . . . 35

8 Acknowledgements

9 Declaration

1 Introduction

The Large Hadron Collider (LHC) at the European Organization for Nuclear Research (CERN) in Geneva, Switzerland, is a proton-proton (pp) collider designed for center-of-mass energies up to $\sqrt{s} = 14$ TeV with a luminosity of $\mathcal{L} = 10^{34}$ cm^{-2} s^{-1} [18]. High-energy collisions at the LHC, observed by the multi-purpose detector ATLAS, led to the discovery of the Higgs boson in 2012, marking a significant milestone in the completion of the Standard Model (SM) of particle physics [1].

However, several fundamental physics questions, including the baryon asymmetry of the universe, cosmological inflation, dark matter, and neutrino oscillations and their masses, remain unexplained, indicating the need for theories beyond the Standard Model [6]. As particle physics research is moving away from testing predefined models, Data-driven approaches might address the unanswered questions.

With the ongoing LHC Run 3 and the High Luminosity Run scheduled for 2029, the collider is expected to produce significantly more data than in previous runs [3]. This data increase will enable more precise measurements and may also lead to the discovery of physics beyond the Standard Model. Thus, a systematic understanding of all recorded data is crucial for the future of LHC physics research.

To interpret LHC measurements, Monte Carlo simulations are used to predict observed LHC events [8]. These simulations, which include event generation based on the Lagrangian, Quantum Field Theory, and detector simulations that describe particle interactions with the detector, are essential for verifying our understanding of the underlying physics, as illustrated in Fig. 1.1. However, the computational expense of current simulations is a significant challenge, especially given the anticipated data output from the high-luminosity LHC [29].

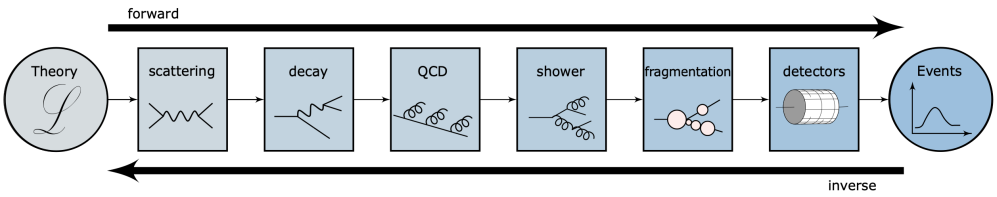


Figure 1.1: Illustration of the LHC simulation chain. Figure taken from [10].

Recent advancements in computing power, software architectures like Transformers [44], and the increasing volume of data suggest that novel machine learning (ML) tools have the potential to revolutionize research at the LHC [10]. Machine learning can be applied at every stage of data analysis, from anomaly detection to rapid event identification [10].

The primary advantages of machine learning and neural networks are their speed once trained, flexibility in being trained on both simulated and real data, and capability to solve inverse problems, such as unfolding, which involves correcting distorted events caused by the limited detector resolution to recover their underlying true distribution [4].

Among ML approaches, generative models offer the most potential for enhancing high-energy physics simulations. In Event generation Deep learning methods can improve both the accuracy and speed of simulations [12, 11, 7] .

Generative models in high-energy physics must meet criteria of (i) precision, (ii) full uncertainty control, and (iii) speed. Classical generative network architectures, such as variational autoencoders (VAEs) [21] and generative adversarial networks (GANs) [25] are fast, but have not yet achieved the precision required by LHC experiments. Recent advances point to novel generative models like Denoising Diffusion Probabilistic

Models (DDPMs) and Conditional Flow Matching (CFM) as the most promising candidate for event generation [10, 31, 27].

Diffusion models are state-of-the-art in generation tasks but depend on an iterative sampling process that slows down generation. This limitation is particularly problematic for real-time analysis applications and when large sample volumes are required. To overcome this bottleneck, this thesis explores the use of Consistency Models, a new class of generative models capable of producing high-quality samples in a single step [43, 42]. Consitency Models are designed for one-step generation but can also perform multi-step sampling, allowing a flexible trade-off between sample quality and speed [43]. Our implementation of Consistency Models are trained through a distillation task, meaning they use another model as a "teacher" and attempt to achieve similar quality. We benchmark this model by generating Z+Jets events at the LHC, a particularly challenging task for particle physics simulations.

Finally, we evaluate both trained models using 1-dimensional metrics and a classifier metric to quantify performance and diagnose potential failure modes of generative models. We demonstrate the advantages of the Consistency Model over Conditional Flow Matching, showing that the Consistency Model can produce competitive results in terms of both sample quality and generation speed.

The remainder of this thesis is organized as follows. In Chapter 2, we introduce the fundamentals of particle physics. Chapter 3 covers the basics of machine learning. In Chapter 4, we introcude the theoretical background of two generative models: Conditional Flow Matching and the Consistency Model. The experimental results on the Z+Jets Dataset are presented in Chapter 5. Finally, in Chapter 6, we summarize the results and discuss future work for Consitency Modes for LHC physics.

2 Particle Physics Fundamentals

2.1 Standard model

The Standard model (SM) of particle physics describes three of the four fundamental forces: electromagnetic, weak and strong interaction and classfies all known elementary particles. The standard model is formulated as quantum field theory and the Lagrangian allows to make predictions for particle collsions in search for new particles. The SM is symmetric under the Poincaré Group and the gauge group

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

The particles are divided into two classes: fermions (half-integer spin particles) and bosons (integer-spin). The fermions are further divided into quarks and leptons that make up matter. The bosons are the force carriers of the fundamental forces. Photons mediate the electromagnetic interaction i.e force between electrically charged particles, gluons mediate the strong interaction and charged W and Z-bosons mediate the weak interaction [36].

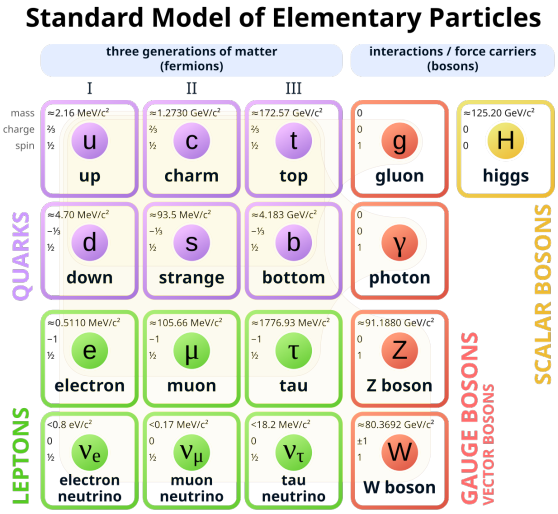


Figure 2.1: Standard model of Particle physics with all elementary particles taken from [19]

2.2 LHC Framework

Particle colliders, such as the Large Hadron Collider (LHC), can be understood as powerful microscopes observing particles and their properties. However, it's important to note that the LHC does not directly observe the fundamental processes themselves. Instead, it detects the byproducts of these processes. The collisions generate a variety of particles, and the multipurpose detectors such as ATLAS, LHCb, and ALICE record the final-state particles and their properties, providing indirect evidence of the underlying interactions [17].

These particles are then identified by the LHC's various subdetectors, each specifically designed to measure different types of particles and their respective properties:

- **Tracking Detectors**: Meassures the trajectories of charged particles by tracks the curvature of the particles in a magnetic field.
- **Electromagnetic Calorimeters**: Measures path and energy of particle that interact electromagnetically producing electromagnetic showers, such as electrons and photons.
- **Hadronic Calorimeters**: Measures the energy of particles that interact strongly like protons, neutrons and pions.
- **Muon Detectors**: Specialized detectors are dedicated to identifying and measuring muons, which are highly penetrating particles capable of passing through first layers of the detector without showering.

With the energy and momentum of the particles measured, physicists can reconstruct the properties of the initial collision products and infer the underlying processes. This reconstruction process is crucial for interpreting the data and extracting meaningful information about the fundamental interactions occurring at the LHC.

2.3 Kinematic Observables

We use the symmetry of particle collisions and the geometry of the detector to define useful kinematic observables. The four-momentum of a particle is represented as:

$$p = (E, p_x, p_y, p_z), \tag{2.2}$$

where E is the energy and p_x, p_y, p_z are the momentum components in Cartesian coordinates. The particle cross sections are invariant under rotation around the collision beam axis. Thus, we parameterize the transverse momentum of a particle as:

$$p_T = \sqrt{p_x^2 + p_y^2}, \tag{2.3}$$

and define the azimuthal angle ϕ around the beam axis.

The invariant mass m, which is Lorentz invariant, is given by:

$$m = \sqrt{E^2 - |p|^2}, \tag{2.4}$$

where $|p|$ is the magnitude of the three-momentum. Since collisions take place in the center-of-mass frame of the proton-proton (pp) system, but not necessarily in the center-of-mass frame of the colliding partons, the final states of processes like $pp \rightarrow 2\,\text{jets} + X$ are often Lorentz-boosted along the beam direction. To describe this, we express the jet angle in terms of rapidity:

$$y = \frac{1}{2} \ln\left(\frac{E + p_z}{E - p_z}\right), \tag{2.5}$$

The differences of rapidity are Lorentz invariant under Lorentz boosts along the beam axis.

Typically, the jet mass resulting from hadronization is small compared to the jet energy. In such cases, we use the pseudorapidity η, defined as:

$$\eta = -\ln\left(\tan\left(\frac{\theta}{2}\right)\right), \tag{2.6}$$

where θ is the polar angle relative to the beam axis.

Using these definitions, we obtain a set of kinematic observables that are invariant under Lorentz transformations along the beam axis and under rotation:

$$p = (m, p_T, \eta, \phi). \tag{2.7}$$

We can reconstruct the components of the four-momentum using the following equations:

$$p_x = p_T \sin(\phi), \quad p_y = p_T \cos(\phi), \quad p_z = p_T \sinh(\eta), \quad E = \sqrt{m^2 + |p|^2}. \tag{2.8}$$

2.4 Drell-Yan Process at the LHC

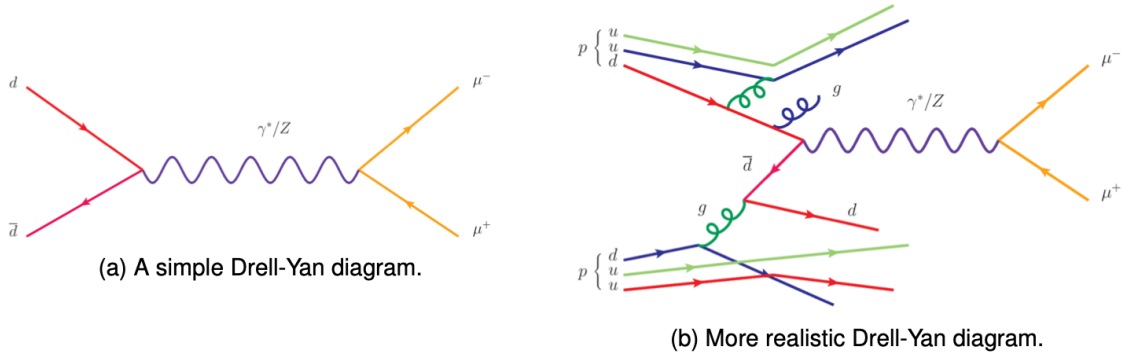
The Drell-Yan process is a fundamental mechanism in high-energy physics that describes the production of a lepton pair in hadron-hadron collisions. First proposed by Sidney Drell and Tung-Mow Yan in 1970 [47], this process serves as a tool to probe the internal structure of hadrons through the annihilation of quark-antiquark pairs [23]. It is a dominant process in particle collisions, and achieving a high-precision understanding of it is crucial for many applications because its often a backgroudprocess. Specifically, we use a dataset of Z + Jets events for our machine learning tasks.

The Drell-Yan process for Z-boson production and subsequent decay into an electron-positron pair is:

$$pp \to Z \to l^+ l^-, \tag{2.9}$$

where the proton-proton collisions occur at a center-of-mass energy of 13 TeV. The mass and decay width of the Z-boson are:

$$M_Z = 91.19\,\text{GeV}, \qquad \Gamma_Z = 2.49\,\text{GeV}. \tag{2.10}$$



(a) A simple Drell-Yan diagram.

(b) More realistic Drell-Yan diagram.

Figure 2.2: Leading-order Feynman diagram for the Drell-Yan process(Left) and (Right) more realistic Drell-Yan process involing proton-proton collision. Taken from [5]

The leading-order (LO) contribution to the Drell-Yan process involves the annihilation of a quark and an antiquark into a virtual photon (γ^*) or a Z-boson, which decays into a lepton pair, as shown in Fig. 2.2. This process is of significant interest not only at LO but also at higher-order corrections. In particular, next-to-leading order (NLO) quantum chromodynamics (QCD) corrections, including gluon radiation and quark-gluon interactions, play a critical role in making precise predictions for cross sections and other observables [2]. We use a dataset of $Z \to \mu^+\mu^- +$ Jets events to train and evaluate the generative models in this thesis.

3 Machine Learning Fundamentals

Machine learning develops statistical algorithms capable of learning from data. Over the decades, physicists have utilized and advanced machine learning frameworks in various applications, such as classification, regression, clustering, and generative models. Recent advancements in the field, driven by improvements in hardware computing power, software architectures like Transformers, and the increasing amount of data, have further accelerated the development and application of machine learning.

This brief introduction to deep learning largely follows the lecture notes on modern machine learning for physics, given by Tilman Plehn at Heidelberg University [38].

3.1 Deep learning

Deep learning, a subset of machine learning, utilizes artificial neural networks to perform various tasks. Unlike traditional methods that require predefined features and patterns, neural networks can extract core patterns through a training process. Neural networks can be understood as a fit function with a huge number of model parameters trying to approximate an underlying unknown function.

$$f_\vartheta(x) \approx f(x) \tag{3.1}$$

3.1.1 Neural Networks

The key is to think of building blocks that can be combined to solve complex problems. Here, the fundamental building block of neural networks is the artificial neuron.

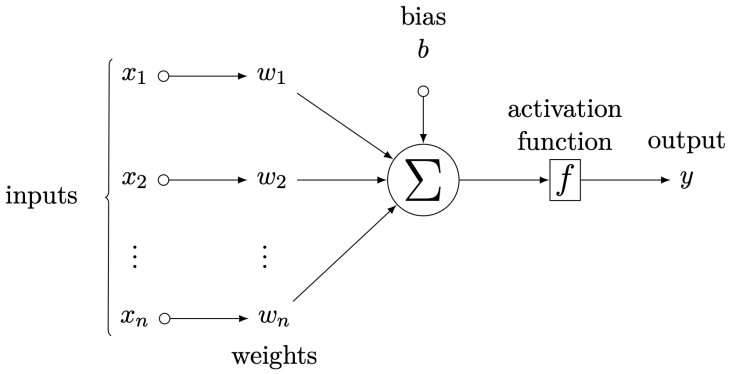


Figure 3.1: Schematic of an artificial neuron. A neuron takes N inputs, denoted as $(x^1, \ldots, x^m)$. Each input is multiplied by a corresponding scalar called a weight. The output is the sum of these weighted inputs, plus an additional bias term b, passed through an activation function f. Taken from [16]

$$y = f\left(\sum_{i=1}^{m} w_i x_i + b\right) \tag{3.2}$$

The purpose of activation functions is to introduce non-linearity and will be discussed in more detail in Chapter 3.1.2. We can generalize the artificial neural to a multilayer perceptron unit (MLP) that has multiple outputs and multiple layers, each with different biases and weights.

Def. 1 ***Multilayer Perceptron Unit (MLP)***
Let W be a matrix of weights, b the vector of biases, and f the activation function. x^n is the output of the n-th layer and x^{n-1} is the input:

$$x^{n-1} \rightarrow x^n = f[W^n x^{n-1} + b^n] \tag{3.3}$$

Neural networks consist of many neurons that are arranged in layers $l = 0, \ldots, L$. We define the input layer as the beginning layer, the intermediate layers as *hidden layers*, and the last layer, which is the result of the network, as the output layer.

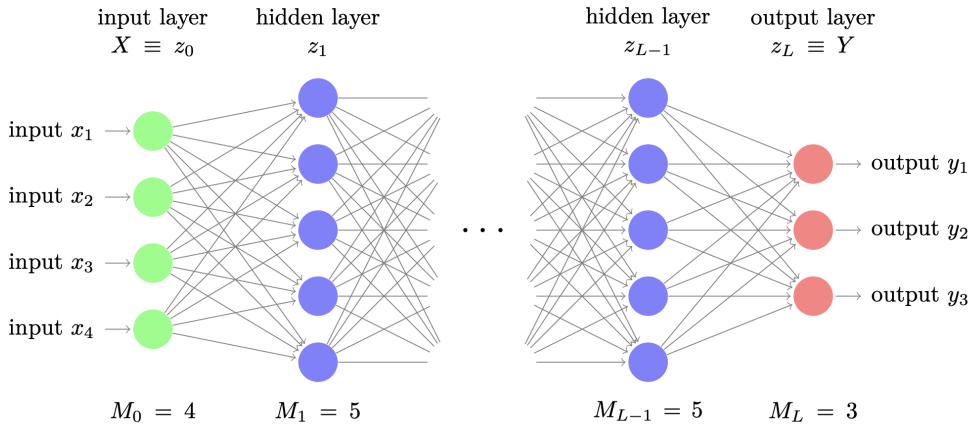


Figure 3.2: Schematic of a feed-forward neural network. Taken from [46]

Neurons in a given layer work in parallel, and neurons in subsequent layers work in series. We refer to such networks as feed-forward architectures. In a complete network process, the inputs x are taken, and then all layers are computed sequentially, resulting in an output layer. The weights w_i and bias b resulting from all artificial neurons are the core of every neural network. They represent the tunable parameters that are adjusted by the learning algorithm. Each connection is weighted and thus influences the output of the neuron.

Parameters that are not adapted during the training algorithm, such as the number of hidden layers (depth of the network), and the number of neurons per layer (width of each layer), are called *hyperparameters*.

The idea behind this structure is that the last layers correspond to patterns that the network has learned, while the previous layers still capture fine sub-patterns. Instead of enforcing features and patterns, the network extracts the core pattern through a training process and an associated loss/scoring function.

3.1.2 Activation Functions

To model non-linear data, we need a non-linearity function in the neural network. Specifically, the equation 3.3 without an activation function is an affine transformation that forms a group. This means that chaining affine transformations together would only result in another affine transformation. To approximate non-linear data, the activation function is introduced and should fulfill these properties:

1. Non-linearity
2. Differentiable on intervals for backpropagation algorithms and gradient descent (training)

3. Computationally inexpensive

The activation function must be differentiable so that the weights can be adjusted during the learning process. One of the most common activation functions is the Rectified Linear Unit (ReLU) function.

Def. 2 ***Rectified Linear Unit (ReLU)***

$$ReLU(x) = \max(0, x) = \begin{cases} 0, & if\, x \leq 0 \\ x, & if\, x > 0 \end{cases} \tag{3.4}$$

(Additionally, we also define the derivative of ReLU at 0 to be 0.) While ReLU is computationally efficient, it has a significant drawback: for inputs less than or equal to zero, the output is zero and its derivative therefor also zero, causing the neuron to become permanently inactive. To overcome this disadvantage, we can use the Leaky ReLU function, which has a small slope for negative inputs.

Def. 3 ***Leaky ReLU***

$$LeakyReLU(x) = \max(\alpha x, x) = \begin{cases} \alpha x, & if\, x \leq 0 \\ x, & if\, x > 0 \end{cases} \quad with\, 0 < \alpha \ll 1 \tag{3.5}$$

For binary classification tasks, we can use the sigmoid function that normalizes the output of the last layter to a value between 0 and 1 and can be interpreted as a probability.

Def. 4 ***Sigmoid***

$$sigmoid(x) = \frac{1}{1 + \exp(-x)} \tag{3.6}$$

To generalize the sigmoid function to multi-class classification tasks, we can use the softmax function that normalizeses multiple outputs

Def. 5 ***Softmax***

$$softmax(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \tag{3.7}$$

3.1.3 Training Neural Networks

We need a loss function in neural networks to quantify prediction errors, enabling the model to adjust its parameters and improve accuracy during training. It is a function that maps data and predicted values to real numbers, providing feedback on the model's performance. Its primary purpose is to guide the model's learning process by minimizing this error, leading to more accurate predictions. The Mean Squared Error (MSE) loss function is typically used in regression problems, as it calculates the average of the squared differences between predicted and actual values in the dataset.

Def. 6 ***Mean Squared Error (MSE) loss***
Let n be the number of samples in the dataset, y_i the target value of the i-th sample and $\hat{y}_i$ the predicted value

$$\mathcal{L}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.8}$$

For binary classification tasks, we can use the Binary Cross Entropy Cross (BCE) loss. It calculates the cross-entropy loss between the predicted and actual labels of two classes in the dataset. This lossfunction can be derived from the Likelihood-ratio and KL-divergence.

Def. 7 ***Binary Cross Entropy (BCE) loss***
Let n be the number of samples in the dataset, y_i target label of the i-th sample and $\hat{y}_i$ the predicted label

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \tag{3.9}$$

Backpropagation

Backpropagation is an algorithm used to train neural networks. It calculates the gradient of the loss function with respect to the network's weights by applying the chain rule in reverse, starting from the output layer and moving backward through the hidden layers. This process yields the gradient $\nabla_\theta L$ with respect to the weights, allowing the network to iteratively update them to minimize the loss function [39].

One way of using the gradient of the loss with respect to the weights is to update the weights in the direction of the negative gradient after every iteration:

$$\theta_j^{t+1} = \theta_j^t - \eta \frac{\partial L}{\partial \theta_j} \tag{3.10}$$

Where η is the learning rate. However, this approach is computationally expensive and leads to overall worse learning performance. An alternative is to accumulate the gradients for a given amout of data called "batches" and update the weights once after each batch. This approach is called *mini-batch gradient descent.*

$$\theta_j^{t+1} = \theta_j^t - \eta \left\langle \frac{\partial L}{\partial \theta_j} \right\rangle_{batch} \tag{3.11}$$

During the training process, the same dataset can be used multiple times to update the model's weights. Each complete pass through the entire dataset is referred to as an epoch.

After initializing the network with random weights $\theta \sim \mathcal{O}(1)$, a complete training protocol consists of calculating the forward pass of the network, computing the loss, backpropagating the gradients, and updating the weights. This process is repeated until the loss converges.

Since the loss function is not a priori convex, it is possible that the network only converges to a local minimum. We can overcome this problem by using mini-batch gradient descent and an advanced optimization algorithm such as Adam that includes momentum and adaptive learning rates. Futhermore, we face multiple challenges such as overfitting, vanishing gradients and biased data that can be overcome by using regularization techniques that will be discussed in chapter 5.5.

3.1.4 Transformer and Self-Attention

The Transformer is an architecture originally designed for various language processing tasks, such as translating sentences from one language to another. Since its introduction, it has been successfully applied to a wide range of applications, including image recognition, speech recognition, and other domains. The core innovation of the Transformer is the self-attention mechanism, which enables the model to learn contextual relationships, such as those between words in a sentence, more effectively than previous approaches [44].

Attention Mechanism

The attention mechanism [44] operates on three learnable matrices, W^k, W^q, and W^v, which are applied to input vectors x_i (e.g., words in a sentence). These matrices transform the input vectors into keys, queries, and values, respectively:

$$\text{keys: } k_i = W^k x_i, \quad \text{queries: } q_i = W^q x_i, \quad \text{values: } v_i = W^v x_i \tag{3.12}$$

The attention matrix A_{ij} is computed by taking the dot product of the query and key vectors, scaled by the inverse square root of the dimension of the key and query vectors for numerical stability. The result is then normalized using the softmax function:

$$A_{ij} = \text{Softmax}\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right) \tag{3.13}$$

This Attention matrix is applied to the value vectors, yielding a set of output vectors:

$$y_i = \sum_j A_{ij} v_j \tag{3.14}$$

The attention matrix can be interpreted as encoding the contextual relationships between elements in a sequence; for example, in a sentence, the meaning of a word depends on its context within the sentence. The dot product measures the similarity between the query and key vectors, helping to identify the most relevant context for each word. Given that there are often multiple different ways to construct these relationships, the simple attention mechanism is extended to a multi-head attention mechanism. We define a number of heads h and compute an independent attention matrix for each head. The resulting output vectors y_i from each head are then concatenated:

$$y_i = \text{Concat}(y_i^1, \ldots, y_i^h) \tag{3.15}$$

After the attention mechanism, the output is passed through a fully connected feed-forward network, which further transforms the data. This architecture is powerful because it allows for parallel computation, significantly improving efficiency over previous sequential models like RNNs.

Transformer Architecture

At its core, a Transformer consists of an encoder and a decoder. The encoder processes the input sequence and generates a representation that captures the relevant information, while the decoder takes this representation and generates the output sequence. Both the encoder and decoder are composed of multiple layers of self-attention mechanisms and feed-forward networks. The decoder also includes an additional attention mechanism that focuses on the output of the encoder, allowing it to condition the generated output on the entire input sequence. The Transformer's ability to model complex dependencies between sequence elements and its efficiency in training make it a foundational model in modern machine learning [44]. Note that in our application, we do not have an encoder because we are doing unconditional generation

4 Novel Generative Networks

Generative models are a class of deep learning algorithms designed to estimate and sample from an unknown data distribution. A generative model learns the underlying patterns and features of the data in order to generate new similar data. In general, we use a known prior density p_{latent}, such as a Gaussian distribution, from which we can sample and reshape it to a more complicated distribution p_{data}:

$$r \sim p_{latent}(r) \rightarrow f_\theta(r) \sim p_{data}(x) \tag{4.1}$$

Since we have no labels or truth information about the training data, generative network training is considered unsupervised. The most common generative models are Variational Autoencoders (VAE), Generative Adversarial Networks (GAN), Invertible Neural Networks (INN), and Diffusion models. In this chapter, we will focus on Conditional Flow Matching and Consistency Models.

4.1 Conditional Flow Matching

The notion of Conditional Flow Matching (CFM) is to use continuous time evolution from noise, such as a Gaussian distribution, to a data distribution via an ordinary differential equation (ODE) . Its dynamics are analogous to fluid dynamics (hence the term "velocity field"), describing the process of flowing from an initial state to a target distribution. We construct a vector field that represents the flow from latent space, e.g., pure noise, to data space through an ordinary differential equation [31, 13].

Def. 8 ***Velocity Field***
Let $x = (x^1, ..., x^d) \in \mathbb{R}^d$ be the datapoints in the d dimensional dataspace. Let $v : [0,1] \times R^d \rightarrow R^d$ be a time dependent vector field with the mapping $x(t) : [0,1] \times R^d \rightarrow R^d$

$$\frac{dx(t)}{dt} = v(x(t), t) \tag{4.2}$$

$$x(t=0) = x_{data} \tag{4.3}$$

Later we will see that $v(x(t), t)$ is exactly our Neural Network predicting the velocity field and allowing the generation of new data by solving the ODE from $t = 1$ to $t = 0$. The velocity field obeying eq 4.3 is equivalent to a probability density $p(x, t)$ solving eq 4.4 by using the continuum equation:

$$\frac{\partial p(x,t)}{\partial t} + \nabla_x [p(x,t)\, v(x,t)] = 0 \tag{4.4}$$

Proof in Appendix A
Again, in analogy to fluid dynamics, this means that instead of looking at the dynamics of individual data points in the data space, it's equivalent to solving the dynamics of the probability density. This is important because it allows us to use multiple samples to estimate the velocity field that generates the probability density.

We choose a Gaussian distribution to sample as the inital point ($t = 1$) and define the endpoint ($t = 0$) as datapoint. The CFM model therefore has following boundary conditions:

$$p(x,t) \rightarrow \begin{cases} p_{\text{data}}(x) & \text{with } t \rightarrow 0 \\ p_{\text{latent}}(x) = \mathcal{N}(x; 0, 1) & \text{with } t \rightarrow 1 \end{cases} \tag{4.5}$$

Since we have chosen a Gaussian distribution, we can define a path from data to a Gaussian distribution using a linear trajectory. We define a time evolution from the phase space $x(t = 0)$ (Data) to $x(t = 1)$ (Noise)

as:

$$x(t|x_0) = (1-t)x_0 + t\epsilon \rightarrow \begin{cases} x_0 & t \rightarrow 0 \\ \epsilon \sim \mathcal{N} & t \rightarrow 1 \end{cases} \tag{4.6}$$

As a consequence we can generate $x(t|x_0)$ by sampling

$$p(x,t|x_0) = \mathcal{N}(x;(1-t)x_0,t) \tag{4.7}$$

Equation 4.7 fullfills by construction the boadry condition of eq 4.5

$$p(x,0) = \int dx_0\, p(x,0|x_0)\, p_{\text{data}}(x_0) = \int dx_0\, \delta(x-x_0)\, p_{\text{data}}(x_0) = p_{\text{data}}(x), \tag{4.8}$$

$$p(x,1) = \int dx_0\, p(x,1|x_0)\, p_{\text{data}}(x_0) = \mathcal{N}(x;0,1) \int dx_0\, p_{\text{data}}(x_0) = \mathcal{N}(x;0,1). \tag{4.9}$$

We can obtain the velocity field by combining eq 4.6 and eq 4.3

$$v(x(t|x_0),t|x_0) = \frac{d}{dt}[(1-t)x_0 + t\epsilon] = -x + \epsilon \tag{4.10}$$

We know that this velocity field solves the continuum equation for $p(x,t|x_0)$ by construction. From the continuum equation we can derive uncondtional vectorfield:

$$\begin{aligned} \frac{\partial p(x,t)}{\partial t} &= \int dx_0 \frac{\partial p(x,t|x_0)}{\partial t} p_{\text{data}}(x_0) \\ &= -\int dx_0\, \nabla_x [\mathbf{v}(x,t|x_0)\, p(x,t|x_0)]\, p_{\text{data}}(x_0) \\ &= -\nabla_x \left[p(x,t) \int dx_0 \frac{\mathbf{v}(x,t|x_0)\, p(x,t|x_0)\, p_{\text{data}}(x_0)}{p(x,t)} \right] \\ &= -\nabla_x \left[p(x,t)\mathbf{v}(x,t) \right] \end{aligned} \tag{4.11}$$

by defining the velocity field as

$$\mathbf{v}(x,t) = \int dx_0 \frac{\mathbf{v}(x,t|x_0)\, p(x,t|x_0)\, p_{\text{data}}(x_0)}{p(x,t)} \tag{4.12}$$

The key point is that the conditional velocity field in eq. 4.10 is only defined by a pre-determined data point x_0. Therefore, its not possible to start from noise $t = 1$ and generate new data points. This is different to 4.12, where the velocity field depends only on the current (x,t) and not on the endpoint of the trajectory. Most importantly, we showed that it is possible to estimate the uncondtional velocity field that generates the probability density by using multiple conditional samples from the data distribution [31, 13].

Training

Encoding the velocity field in a Neural Network (NN) is a regession task $v_\vartheta = v$. Therefore, we use the Mean squared error (MSE) as loss function:

$$\mathcal{L}_{FM} = \mathbb{E}[(v_\vartheta(x,t) - v(x,t))^2] \tag{4.13}$$

However, the unconditional vector field is not tractable. Lipman et al. [31] derive that the loss of the conditional velocity field and the unconditional velocity field differs only by a constant. (Hence, $\nabla\mathcal{L}_{FM} = \nabla\mathcal{L}_{CFM}$.) We can therefore rewrite the flow-matching loss in terms of the conditional flow-matching loss.

$$\mathcal{L}_{CFM} = \mathbb{E}[(v_\vartheta(x(t|x_0),t) - \frac{d}{dt}x(t|x_0))^2] = \mathbb{E}[(v_\vartheta((1-t)x_0 + t\epsilon, t) - (\epsilon - x_0))^2] \tag{4.14}$$

A full training protocol consists of first drawing a sample $t \sim U(0,1)$ from a uniform distribution, a random data point $x \sim D$, and a random noise vector $\epsilon \sim \mathcal{N}$. We then calculate a point on the linear trajectory $x(t)$. Next, we compute the vector field by taking the derivative of the trajectory and compare it to the vector field obtained from the neural network. The quadratic deviation between these two is our loss function.We repeat this process until the loss converges. This algorithm is illustrated in figure 4.1 and more details can be found in the algorithm 1.

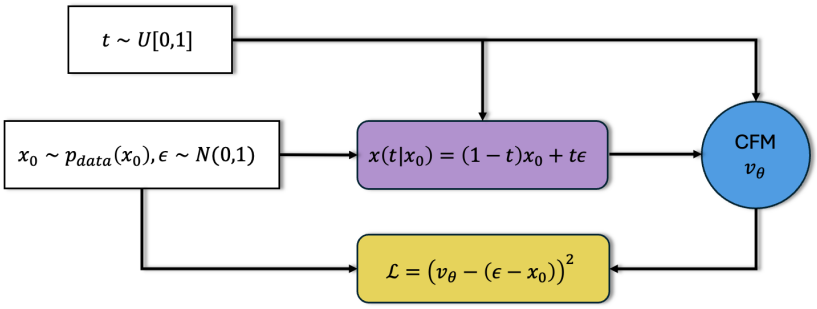


Figure 4.1: Training Conditional Flow Matching. Inspired from [12]

Sampling

A well trained model can generate new samples by solving the ODE from $t = 1$ to $t = 0$. We start by drawing a sample from the latent distribution $x_1 \sim p_{latent} = \mathcal{N}$ and calculate its time evolution by numerically solving the ODE backwards in time from $t = 1$ to $t = 0$.

$$x_0 = x_1 - \int_0^1 v_\vartheta(x(t), t)dt \tag{4.15}$$

This is an incredibly powerful method because it allows us to use advanced ODE solvers to generate new samples, surpasing most generative models in terms of sample quality and speed. For example, the Euler method aprroximates the ODE solution by updating the state of a system at discrete time steps taking only the current state into account:

$$x_{n+1} = x_n + \Delta t\, v_\vartheta(x_n, t_n) \tag{4.16}$$

Where Δt is a small time step (e.g $\Delta t = 0.01$). For the Conditional flow matching model we use the Runge-Kutta method [22]. What's important here is that the network is repeatedly used to obtain a final solution.

Algorithm 1 CFM Training

Input: dataset D, initial model parameter ϑ, learning rate η, Neural Network NN
repeat
  Sample $x \sim D$, $t \sim U(0,1)$ and $\epsilon \sim \mathcal{N}$
  $x(t) = (1-t)x + t\epsilon$
  $v(t) = -x + \epsilon$
  $v_\vartheta = \text{NN}(x_t, t)$
  $L = \text{MSE}(v_\vartheta, v_t)$
  Update $\vartheta \leftarrow \vartheta - \eta \nabla_\vartheta L$
until Convergence

Algorithm 2 CFM Sampling

Input: CFM Model $v_\vartheta(\cdot,\cdot)$
Sample $x(t) \sim \mathcal{N}$, $t = 1$
$x(t=0) = \text{solveODE}(v_\vartheta, x(t), t(1 \rightarrow 0))$
Output: $x(t=0)$

4.2 Consistency Models

Overview

Diffusion models are state-of-the-art in generation tasks but depend on an iterative sampling process that causes slow generation. This is particularly limiting for real-time applications and when generating a large number of samples. To solve this problem, we aim to use Consistency Models, a new family of generative models capable of producing high-quality samples in a single step. They are, by design, one-step generators but can perform multi-step sampling, trading off sample quality for speed. Consistency Models are trained through a distillation task, meaning they use another model as a "teacher" and attempt to achieve similar quality [43, 42].

Architecture

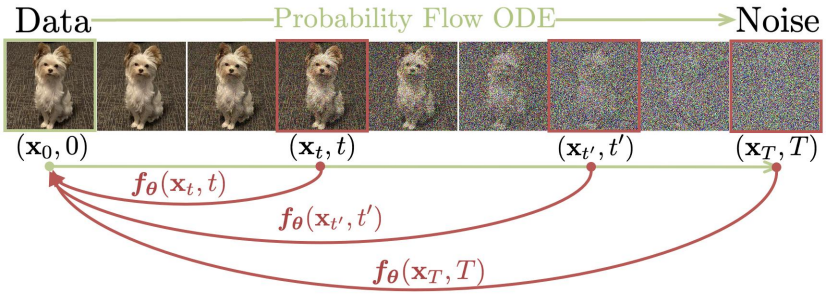


Figure 4.2: Schematic of Consistency Models. They learn to map any point x_t on the ODE trajectory to its origin (e.g. $x0$ for generative modeling. Taken from [43]

Def. 9 ***Consistency Function***
Given a solution trajectory $x(t)$ of an ODE where $x(0) = x_{data}$ and $x(1) = x_{latent}$, the consistency function f is defined as:

$$f : (x_t, t) \rightarrow x_{data} \tag{4.17}$$
$$f(x(t), t) = f(x(t'), t') \quad \forall t, t' \in [0, 1] \tag{4.18}$$

A consistency function has the property of *self-consistency*, meaning that for any pair of (x_t, t) and $(x_{t'}, t')$ on the same ODE solution of trajectory , the function returns the same consistent output: $f(x(t), t) = f(x(t'), t')$. Additionally, the boundary condition $f(x_0, 0) = x_0$ must be satisfied. The idea is illstruated in fig 4.2 for any solution trajectory. The goal is to train a model f_ϑ that approximates this consistency function by enforcing the self-consistency property:

$$f_\vartheta \approx f \tag{4.19}$$

The boundary condition is crucial for training because it allows us to approximate the trajectory solution from $t = 0$ to $t = 1$. To address potential trivial solutions $f = 0$, we use a parameterization:

$$f_\vartheta(x(t), t) = \begin{cases} x & \text{for } t = 0 \\ F_\vartheta(x(t), t) & \text{for } t \in (0, T) \end{cases} \tag{4.20}$$

For numerical stability, we use a small ϵ instead of 0 (e.g., $\epsilon = 10^{-3}$). Alternatively, we use a skip connection:

$$f_\vartheta = c_{\text{skip}}(t)x + c_{\text{out}}(t)F_\vartheta(x(t), t) \tag{4.21}$$

where c_{skip} and c_{out} are differentiable functions such that $c_{\text{skip}}(0) = 1$ and $c_{\text{out}}(0) = 0$, fulfilling the boundary conditions. We adopt the second parameterization in our experiments identical to [43, 42].

For the model architecture, we are not constrained, and we can utilize existing neural network architectures such as ResNet ,U-Net, Fully connected layers or Transformers.

Training Consistency Models via Distillation

The distillation method for training consistency models is based on training with a pre-trained diffusion model. In practice, we use a Conditional Flow Matching model $v_\theta(x_t, t)$ as the "teacher" model. In contrast to Conditional Flow Matching, where we train a time-dependent velocity field that solves the ODE (4.3), the consistency models try to directly learn the solution trajectories as a function that maps $f(x_t, t) \rightarrow x_0$ in a single step. This means that the overall training goal is to approximate the consistency function by enforcing the self-consistency property $f(x(t), t) = f(x(t'), t')$ by first obtaining two points on the trajectory and then using a regression task. Similar to Conditional Flow Matching, we use a linear trajectory from noise to data:

$$x(t|x_0) = (1-t)x_0 + t\epsilon \rightarrow \begin{cases} x_0, & t \rightarrow 0 \\ \epsilon \sim \mathcal{N}(0,1), & t \rightarrow 1 \end{cases} \tag{4.22}$$

$x(t)$ is directly obtained by sampling from

$$p(x, t|x_0) = \mathcal{N}(x; (1-t)x_0, t) \tag{4.23}$$

In addition to that, we can obtain an accurate estimate of the next point on the trajectory $x(t+\Delta t)$ by running one discretization step of a numerical ODE solver:

$$x(t+\Delta t) = x(t) + h\phi(x(t), t, v_\theta) \tag{4.24}$$

where h is the small discretization step size and v_θ the pre-trained model. For example, when using an Euler solver, we use the formula:

$$x(t+\Delta t) = x(t) + \Delta t\, v_\theta(x_t, t) \tag{4.25}$$

We train the consistency model by minimizing its output differences on the self consitency pairs $(x(t), t)$ and $(x(t+\Delta t), t+\Delta t)$. This motivates our following consistency distillation loss for training consistency models.

Def. 10 ***Consistency Distillation Loss***

$$\mathcal{L}_{CD} := \mathbb{E}\left[\left(\mathbf{f}_\theta(\mathbf{x}_{t_{n+1}}, t_{n+1}) - \mathbf{f}_\theta(\mathbf{x}_{t_n}, t_n)\right)^2\right] \tag{4.26}$$

A full training protocol is illustrated in Figure 4.3. It consists of first drawing a sample $t \sim U(0,1)$ from a uniform distribution, a data sample $x \sim D$, and a random noise vector $\epsilon \sim \mathcal{N}(0,1)$. We then calculate a point along the linear trajectory $x(t)$. Next, we compute an estimate of the next point on the trajectory $x(t+\Delta t)$ using the pre-trained model by running one discretization step of an ODE solver. We then compare the output of the neural network on the pair $(x(t), t)$ with the output on the pair $(x(t+\Delta t), t+\Delta t)$ and calculate the loss function. We repeat this process until the loss converges. More details can be found in the algorithm 4 taken from [43, 42].

Sampling

A well-trained Consitency model can generate new samples by drawing from the latent distribution $r \sim p_{latent}$ and passing it through the network. The sampling quality can be improved by using a multi-step

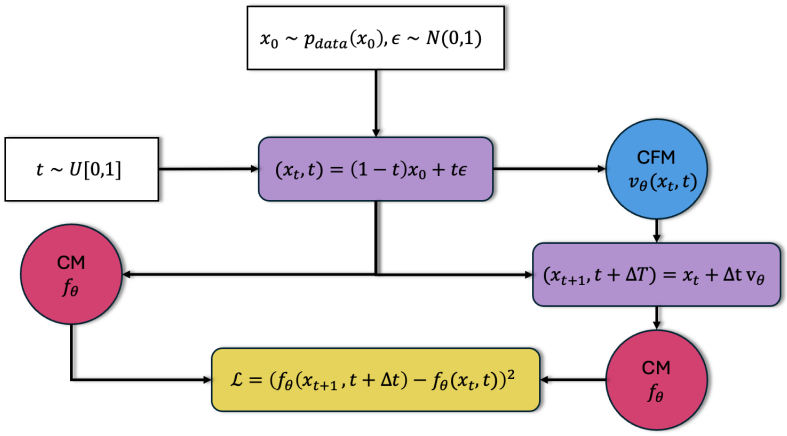


Figure 4.3: Consistency model training algorithm

sampling method: The multiple step sampling starts by sampling a point $x \sim \mathcal{N}(0,1)$ and passing it through the network like the single step iterations. Next, we inject noise, depending on the number of iterations, into this output via $x(t) = (1-t)x + \epsilon t$ and pass it again through the neural network. With each iteration, we inject progressively less noise. We repeat this process until we reach the desired number of iterations. The advantages is we have the flexibility to trade off computation time for sample quality after the model is trained. Details of this algorithm can be found in the algorithm 4.

Algorithm 3 Consistency Distillation (CD)

Input dataset $\mathcal{D}$, initial model parameter θ, learning rate η, ODE solver $\Phi(\cdot,\cdot;\phi)$, $d(\cdot,\cdot)$, $\lambda(\cdot)$, and μ
$\theta^- \leftarrow \theta$
repeat
  Sample $\mathbf{x} \sim \mathcal{D}$ and $n \sim \mathcal{U}[[1, N-1]]$
  Sample $\mathbf{x}_{t_{n+1}} \sim \mathcal{N}(\mathbf{x}; t_{n+1}^2 I)$
  $\hat{\mathbf{x}}^{\Phi}_{t_n} \leftarrow \mathbf{x}_{t_{n+1}} + (t_n - t_{n+1})\Phi(\mathbf{x}_{t_{n+1}}, t_{n+1}; \phi)$
  $\mathcal{L}(\theta, \theta^-; \phi) \leftarrow$
$\lambda(t_n) d(f_\theta(\mathbf{x}_{t_{n+1}}, t_{n+1}), f_{\theta^-}(\hat{\mathbf{x}}^{\Phi}_{t_n}, t_n))$
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta, \theta^-; \phi)$
  $\theta^- \leftarrow \text{stopgrad}(\mu\theta^- + (1-\mu)\theta)$
until Convergence

Algorithm 4 Multistep Consistency Sampling

Input: Consistency Model, sequence of time-points $t_1, t_2, \ldots t_n$,
Sample $x(t_n) \sim \mathcal{N}, t = 0$
for $n = 1$ to $N-1$ **do**
  sample $\epsilon \sim \mathcal{N}(0, I)$
  t = t + 1/N
  $x(t_{n+1}) = (1-t)x + x\epsilon$
  $x = f_\vartheta(x(t), t)$
end for
Output: x

4.3 Limitations of Generative Networks

A trained generative network can exhibit multiple failure modes:

1. Incorrectly learned phase space boundaries
2. Under- or overpopulated distribution tails
3. Washed-out features

Generative networks often struggle with specific challenges, such as accurately replicating hard edges in data such as a data cutoff or sharp peaks (e.g resonance curves of particle physics events).

To evaluate the generative quality we use two tools:

1. A set of 1 dimenionals metrics
2. A classifier trained to distinguish generated data from real data

4.3.1 1D Metrics

A straight forward benchmark to compare distributions is to measure the difference between the true and generated data with a metric. We can use the Wasserstein distance and the energy distance:

Def. 11 ***Wasserstein distance***
Given two 1D probability mass functions, u and v, the Wasserstein distance between the distributions is:

$$l_1(u,v) = \inf_{\pi \in \Gamma(u,v)} \int_{\mathbb{R}\times\mathbb{R}} |x-y| d\pi(x,y) \tag{4.27}$$

where $\Gamma(u,v)$ is the set of (probability) distributions on $\mathbb{R}\times\mathbb{R}$ whose marginals are u and v on the first and second factors respectively. For a given value x, $u(x)$ gives the probability of u at position x and the same for $v(x)$. If U and V are the respective CDFs of u and v, this distance also equals to:

$$l_1(u,v) = \int_{-\infty}^{+\infty} |U-V| \tag{4.28}$$

Def. 12 ***Energy distance***
The energy distance between two one dimensinonal distributions u and v equals to

$$D(u,v) = \sqrt{2} l_2(u,v) = \left(2\int_{\infty}^{\infty} (u-v)^2\right)^{\frac{1}{2}} \tag{4.29}$$

4.3.2 Classifier metric

While metrics provide valuable insights, they may not capture complex correlations or subtle issues that are invisible in 1-dimensions. The classifier serves as both a performance metric and a diagnostic tool, offering a more comprehensive evaluation of the generated data. Neural network classifiers are usually trained on the BCE Loss:

$$\mathcal{L} = \left\langle -\log D(x) \right\rangle_{x\sim p_1} + \left\langle -\log(1-D(x)) \right\rangle_{x\sim p_2} \tag{4.30}$$

where p_1 and p_2 are the two distributions to be classified and $D(x) \in [0,1]$ is the neural network prediction ($D(x) = 0 \rightarrow p_1$ and $D(x) = 1 \rightarrow p_2$). A perfectly trained model would minimise this loss. We can derive an expression for the output D(x) of a perfectly trained network depending on p_1, p_2 by varying the loss function with respect to D(x) and setting it to zero: $\frac{\delta \mathcal{L}}{\delta D} = 0$

$$D(x) = \frac{p_1(x)}{p_1(x) + p_2(x)} \tag{4.31}$$

We can rewrite this solution as an estimate of the likelihood ratio

$$t_{LR} = \frac{p_1}{p_2} = \frac{D(x)}{1-D(x)} \tag{4.32}$$

This is a powerful result because it shows that a perfectly trained classifier can be used to estimate the likelihood ratio between two distributions, meaning we are able to reweight one distribution p_1 into the other p_2 by using the likelihood ratio.

$$p_1 = p_2 \frac{D(x)}{1 - D(x)} \tag{4.33}$$

This property can be used as a diagnostic tool to see if the classifier is able to distinguish between the two distributions and if it actually learns the likelihood ratio. However, in reality, the classifier is not able to learn the likelihood ratio perfectly and the reweighted distribution is just an approximation of the target distribution [20].

Receiver Operating Characteristic (ROC) Curve

The Receiver Operating Characteristic (ROC) curve is commonly used to evaluate the performance of binary classifiers. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR), where these rates are defined as:

$$TPR = \frac{TP}{TP + FN} \quad FPR = \frac{FP}{FP + TN} \tag{4.34}$$

- **True Positive Rate (TPR)**: The proportion of correctly identified positive instances out of all actual positives.
- **False Positive Rate (FPR)**: The proportion of incorrectly identified positive instances out of all actual negatives.

Where (TP) is True Positives, FP (False Positives), (TN)True Negatives and (FN) means False Negatives.

To illustrate the idea: a classifier that perfectly distinguishes all inputs would have a TPR of 1 and an FPR of 0, resulting in a constant line that runs from (0,0) to (1,1). In contrast, a random guessing (50/50) classifier would produce a diagonal line from (0,0) to (1,1), as the TPR would be directly proportional to the FPR.

The ROC curve is especially useful for comparing the performance of generative models. In the case of a well-trained generative model, the generated data closely mimics real data. As a result, when a classifier is trained on distinguishing between real and generated data, a perfect generative model would cause the classifier to perform no better than random guessing, leading to an ROC curve near the diagonal, corresponding to an Area Under the Curve (AUC) of 0.5. On the other hand, if the generative model produces data that is easily distinguishable from real data, the classifier will perform better, resulting in a ROC curve farther from the diagonal, with an AUC closer to 1.

Therefore the performance of a generative model can be quantified using the AUC of the ROC curve, which ranges from 0 to 1. An AUC of 0.5 indicates no discriminative ability (random guessing) of the classifier, while an AUC of 1 indicates perfect discrimination. Meaning a AUC close to 0.5 suggests a good gernerative model performance [20].

However, reducing performance to a single value like the AUC can oversimplify the model's behavior. Therefore, in addition to the AUC, we also examine the ROC curve itself as a diagnostic tool, which can highlight potential issues such as outliers or specific regions where the model underperforms [20].

4.4 Related literature

There are other multiple destillation techniques that aim to simplify sampling from a trained model by fine-tuning or training a new model to produce samples with fewer function evaluations.

In *Bespoke Solvers for generative flow models* a framework is introduced to create custom ODE solvers that are optimized for a specific trained model . In short, these solvers are designed to be consistent and parameter-efficient, meaning they require fewer ODE steps to achieve high-quality sampling. The gain in speed is still limited by the fact that the model requires multiple Network evaluations to generate samples [41].

In *Knowledge Distillation in Iterative Generative Models for Improved Sampling Speed* they directly regressed the trained models samples via the following loss function

$$\mathcal{L} = \left\langle D_{KL}(p_{teacher}(x_0|x_T), p_{student}(x_0|x_T)) \right\rangle_{x_T} \tag{4.35}$$

where $p_{teacher}$ and $p_{student}$ refer to the teacher and student models, respectively. This loss does not incorporate the self-consistency property. Furthermore, the teacher models must be deterministic, and the models are unable to perform multi-step sampling, trading compute for efficiency [32].

In Diffusion Probabilistic Model Made Slim they are reducing the computational complexity of Diffusion Probabilistic Models (DPMs) with fewer model parameters while preserving their high image generation quality [26].

We are particularly interested in Consistency models because they offer several advantages:

1. One-step generation by design: This enables faster generation, which addresses a significant bottleneck in many AI applications.
2. Speed-performance trade-off: Consistency models allow for adjustments between speed and performance, providing more flexibility.
3. Unconstrained Network architecture: Freedom in model design and implementation.

Existing literature already demonstrates good performance in simulation-based inference and detector simulations [9, 40].

5 Experiments

This chapter focuses on the results obtained from training a Consistency Model on LHC events. We first introduce the benchmark model, CFM, using the Z boson as the main dataset. Then, we explain our specific Consistency Model architecture and the obtained results. Additionally, in Sec. 5.5, we demonstrate how classifiers can be utilized to evaluate the performance of generative models. We summarize our results in Sec. 5.6 and discuss the implications of our findings.

5.1 Dataset

The Conditional Flow Matching and the Consistency Models are trained on unweighted events at the hadronization level. Detector effects (e.g., reconstruction errors) are excluded because they soften sharp phase space features, meaning that our method will perform even better on reconstructed objects. We choose a dataset involving the production of leptonically decaying Z bosons, with a variable number up to three QCD jets, because its a challenging benchmark process for LHC events. The neural networks must learn to model an extremely sharp Z resonance peak, while the presence of jets, which involve complex QCD effects, multiplies the difficulty of reconstructing proper kinematics. The dataset is defined as:

$$pp \to Z_{\mu\mu} + \{1,2,3\} \text{ Jets} \tag{5.1}$$

We use the identical dataset from previous research involving INN, CFM, DDPM and Autoregressive Transformer [12, 13]. This data is simulated with SHERPA 2.2.10[8] , consisting of 5.4M events (4.0M + 1.1M + 300k) at 13 TeV. We use CKKW [15] merging to generate events with up to three jets, including ISR, parton shower, and hadronization, but no pile-up. The final state of the training sample is defined using the FastJet 3.3.4 [14] with the anti-k_T algorithm [14]. We use a typical cutoff:

$$p_{T,j} > 20 \text{ GeV} \quad \text{and} \quad \Delta R_{jj} > 0.4 \tag{5.2}$$

where $p_{T,j}$ is the transverse momentum of the jet, and ΔR_{jj} is the separation between two jets in the detector, defined as:

$$\Delta R_{jj} = \sqrt{\Delta\Phi_{jj}^2 + \Delta\eta_{jj}^2} \tag{5.3}$$

Here, $\Delta\Phi$ and $\Delta\eta$ are the differences in azimuthal angle and pseudorapidity between two jets, respectively.

The jets and muons are ordered by transverse momentum p_T. Since jets have a finite invariant mass, our final state dimensionality is three for each muon, plus four degrees of freedom per jet, i.e., 10 dimensions for $Z+1$ Jet, 14 dimensions for $Z+2$ Jets, and 18 dimensions for $Z+3$ Jets.

$$\text{Leptons:} \{p_T, \eta, \phi\} \tag{5.4}$$

$$\text{Jets:} \{p_T, \eta, \phi, m\} \tag{5.5}$$

We examine multiple datasets with increasing numbers of jets because the task of reconstructing the Z resonance becomes increasingly difficult, and we are interested in later assessing the scaling capability of the Consistency Models with increasing complexity.

Momentum conservation is not guaranteed, as some final-state particles might escape detection (e.g., due to the jet algorithm). Since the global azimuthal angle is a symmetry of LHC events, we train on azimuthal angles relative to the muon, thus reducing the dimensionality to 9, 13, and 17 dimensions.

Preprocessing

Preprocessing is a crucial step in the training of neural networks. This accelerates the training process and improves the performance of the model. Preprocessing is reversible and later on the output is transformed

back into the original form. We already reduced the dimensionality of the data due to the global rotational symmetry of the detector. Note that each lepton or reconstructed jet is represented by a 4-vector:

$$\{p_T, \eta, \phi, m\} \tag{5.6}$$

Since we can extract a global threshold in jet p_T, we represent the variables in terms of:

$$p_T = \log(p_T - p_{T,\min}) \tag{5.7}$$

to approximate a Gaussian shape, matching the Gaussian latent space distribution of the generative model. Since the global azimuthal angle is a symmetry of LHC events, we train on azimuthal angles relative to the muon with the largest transverse momentum, within the range of $\Delta\phi \in [-\pi, \pi]$. A transformation into $\Delta\phi = \text{artanh}(\Delta\phi/\pi)$ leads to an approximately Gaussian distribution. For all phase space variables, we apply a centralization and normalization step:

$$\tilde{q}_i = \frac{q_i - \bar{q}_i}{\sigma(q_i)} \tag{5.8}$$

since neural networks tend to work better with normalized values $\sim \mathcal{O}(1)$.

5.2 Training Conditional Flow Matching

The networks are implemented using PyTorch [37] with the ADAM [30] optimizer and a cosine annealing learning rate schedule. The setup for the CFM model is described in Chapter 3 and is consistent with previous literature [28]. The hyperparameters of the networks are described in table 7.1. The Transformer network was also implemented using PyTorch [35, 37].

The CFM is trained on the $Z+1$Jet dataset and we immediately conclude that Transformer perform better than MLPs. The MLPs version need way more parameters, longer training time and the features are more washed out. We adjust the network with an increasing number of jets due to the rising complexity and dimensionality of the data. We choose a linear trajectory from noise to data 4.6.

Results

In Fig 5.1 We show the kinematic distributions for our training data ("Train"), generated data of the CFM model with Transformer architecture ("TraCFM") and the overall dataset ("True"). We train exclusively on the Z boson dataset with a specific jet number $Z + \{1,2,3\}$. Additionally, one can see the deviation from the training data using $\frac{p_{model}}{p_{train}}$ and expressed as a percentage:

$$\delta\% = 100 \cdot \frac{|Model - Truth|}{Truth} \tag{5.9}$$

Note that $Z + Jets$ is 9,14,18 dimensional but we only plot the one example plot for the 1D histrogams and look at reconstructed objetcts like M_{ll} and ΔR_{ij}. We checked that all one dimensional distributions are learned to percent-level precision. The full kinematic distributions can be found in the appendix.

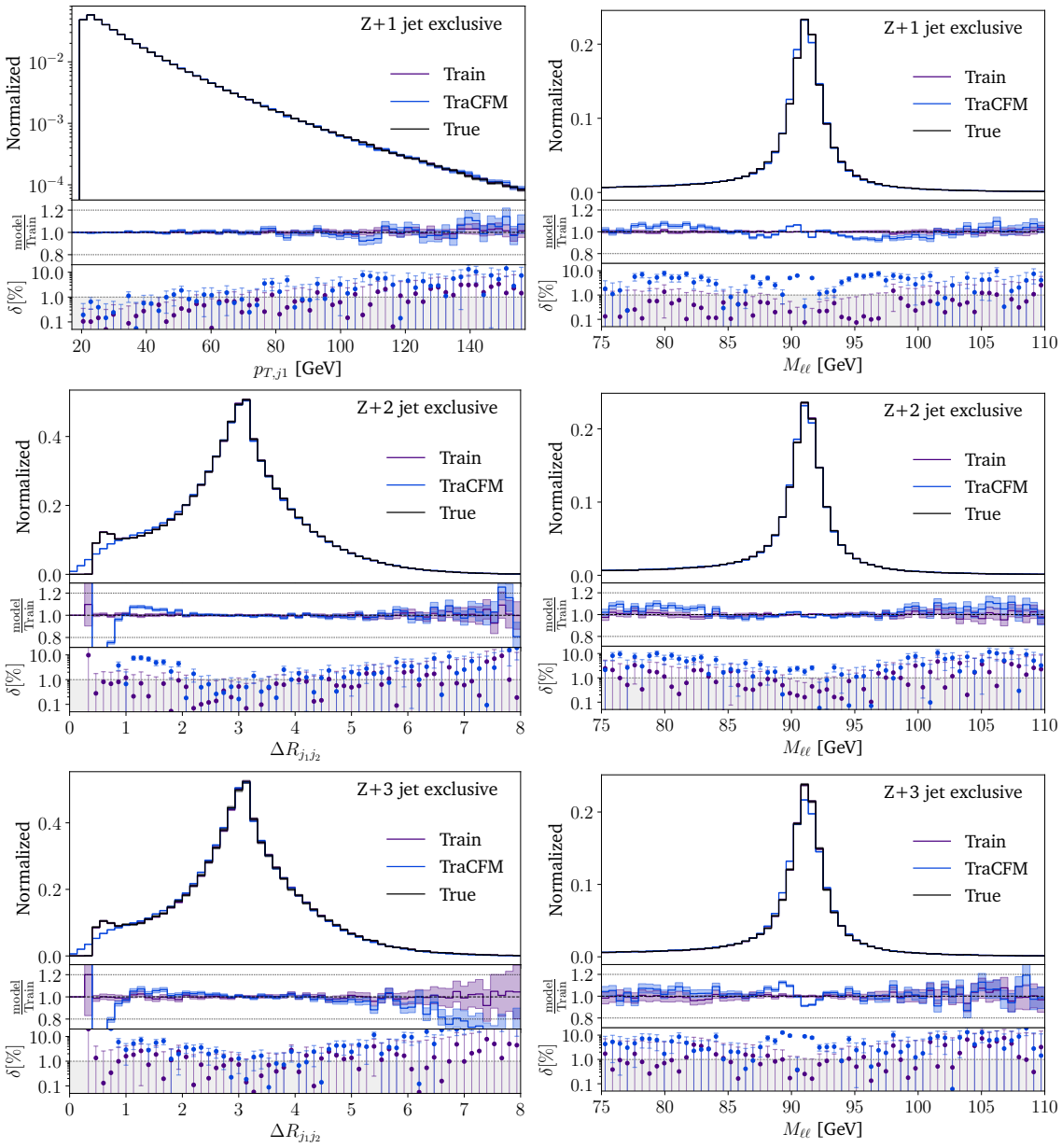


Figure 5.1: CFM generated data for Z+1Jets (top), Z+2Jets (middle) and Z+3Jets (bottom) on 1Mil. events

For the $Z+1$ jets process, the transverse momentum is learned with high precision. However, the tail regions show deviations from the true values. The Z-peak resonance is also accurately captured. Our analysis indicates that Conditional Flow Matching (CFM) effectively models the full complexity of $Z+1$ jet. We further examine correlations, such as the relationship between the pseudorapidities η_{l1} and η_{l2}, and the difference in azimuthal angles $\Delta\phi$, both of which are learned with high precision.

CFM can be trained on more challenging datasets, including events with two and three jets. In events involving multiple jets, Quantum Chromodynamic (QCD) effects cause an enhancement in the rate when the jets are close to each other. This leads to a sharp cutoff in the angular separation R_{ij}, imposed by the jet

algorithm. However, we observe that the CFM does not accurately capture the ΔR_{ij} distribution and fails to represent the sharp cutoff, a common limitation in generative models.

It is important to note that for this failure mode, there is an existing workaround known as the *magic transformation*, which can be applied to the data, but it was not implemented in our approach [12]. Additionally, in the $Z+3$ jets scenario, the CFM struggles to reconstruct the Z-peak, producing a slightly broader resonance curve. this is mostly due to the small amount of training data (300K) compaired to 1.1M and 4.0M from one and two jets

As mentioned earlier, we use the CFM model as both a benchmark and a teacher model for training the Consistency Model.

5.3 Training Consistency Models

The network is also implemented using PyTorch [37] and ADAM [30] with a one-cycle learning rate schedule.The hyperparameters can be seen in the table 7.2. We adjust the networks with an increasing number of jets due to the rising complexity and dimensionality of the data. For the network parameterization

$$f_\vartheta = c_{skip}(t)x + c_{out}(t)F_\vartheta(x(t),t) \tag{5.10}$$

we choose

$$c_{\text{skip}}(t) = \frac{\sigma_{data}^2}{(t-\epsilon)^2+\sigma_{data}^2},\ c_{\text{out}}(t) = \frac{\sigma_{data}(t-\epsilon)}{\sqrt{\sigma_{data}^2+t^2}} \tag{5.11}$$

which clearly satifies $c_{skip}(\epsilon)=1$ and $c_{out}(\epsilon)=0$ with $\sigma_{data}=0.5$ and $\epsilon = 1e-4$ [43]. For training to obtain x_{t+1} we use the Euler method as an ODE solver with the CFM model. For the training process we chose a time step of $\Delta t = 0.01$.

$$x_{t+1} = x_t + 0.01\, v_\vartheta(x(t),t) \tag{5.12}$$

Additionally, we used a linear trajectory similar to the one in CFM from noise to data.

$$x(t) = (1-t)x_o + \epsilon t \tag{5.13}$$

Training details

The CM must be trained for a significantly longer time period than the CFM model to achieve competitive results. Due to the high noise loss, we chose a large batch size of 16384. The size Model of the CM is approximately the same as the CFM model and all hyperparameters can be seen in table 7.2. We discovered that Transformer outperform Multi-Layer Perceptrons (MLPs), so we focused our discussion on the Transformer architecture, which aligns with the CFM results.

For multi-step sampling, we employed a linear noise injection method. This approach uses the same linear trajectory as described in 5.13 and select a number of steps (n). This process involves taking equal-distance steps from noise to data.

Additionally, we implemented various setups using different Ordinary Differential Equation (ODE) solvers, such as HEUN, and explored different noise trajectories. However, we found that the framework using the EULER solver with linear trajectories provides a stable solution to the problem. We did not observe any significant improvements with the other setups.

After training the Consistency models, we are flexible with the trade-off between sample quality and speed. We conclude in Chapter 5.7 that 7 steps is optimal for CM multistep sampling and plot our results with

this configuration. We show the kinematic distributions (Fig 5.2) for our benchmark model CFM 'TraCFM,' the generated data of the CM with additional information about the number of multistep sampling steps ('CM+X,' where X is the number of steps), and the overall dataset ('True'). We generated data consisting of (1.0M, 1.0M, 200K) events for one, two, and three jets, respectively. Single Step generated data can be found in the appendix 7.1.

Results

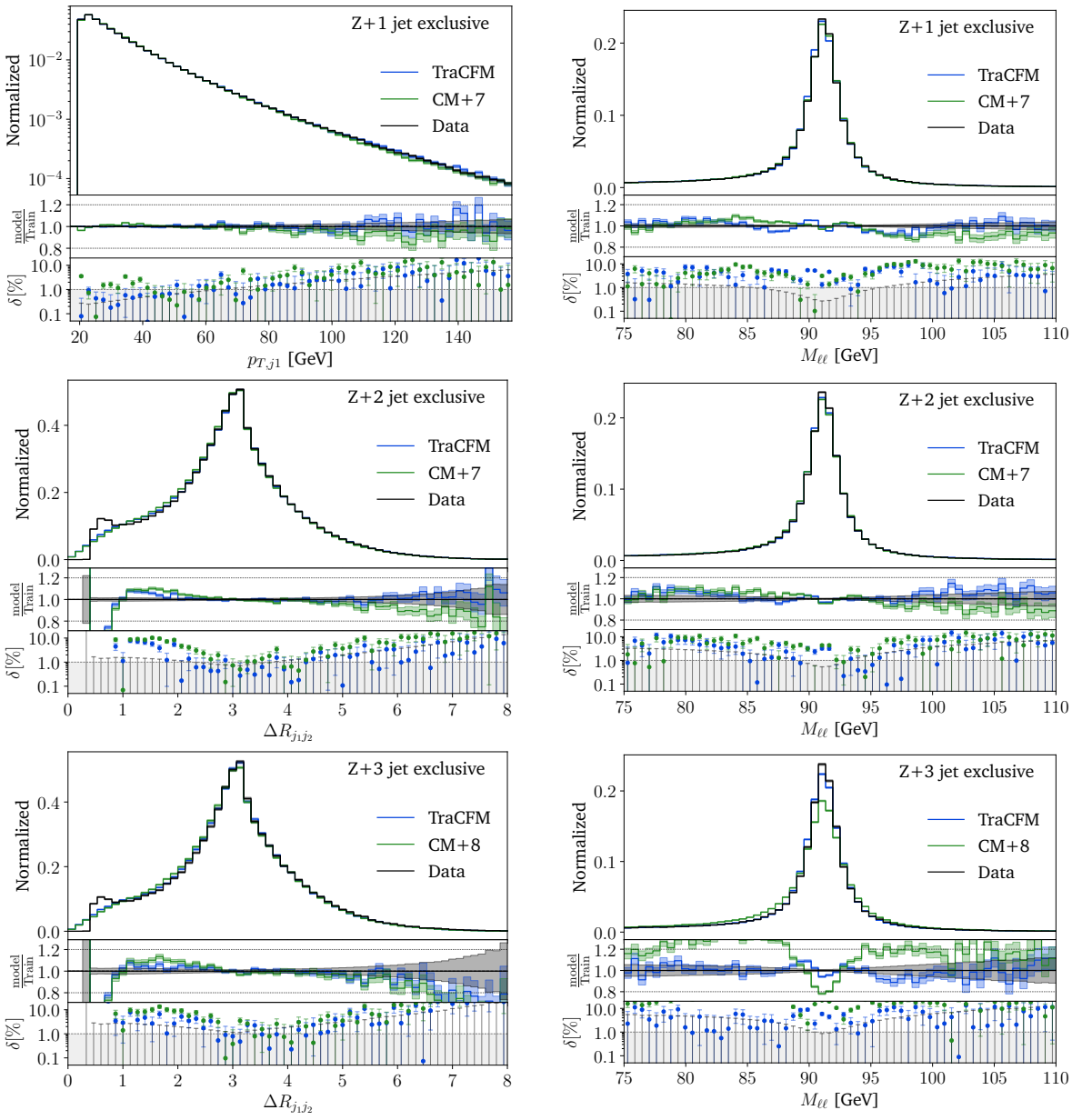


Figure 5.2: CM generated data for Z+1Jet (top), Z+2Jets (middle) and Z+3Jets (bottom) on 1Mil. Events

Z+1Jets In Figure 5.2, the Z resonance peak in the top-right plot is slightly wider compared to the CFM model and the training data. Additionally, the transverse momentum P_T (top left in Figure 5.2) is generated with percent-level precision and similar deviation to the CFM model in the tails. The CFM model performs slightly better with fewer deviations in the tails of the Z resonance peak in the top-right figure.

Z+2Jets For the more challenging dataset with Z boson and two jets, as shown in Figure 5.2 (middle), the Consistency Model is also able to generate accurate kinematics to a high degree of precision in the one-dimensional outputs, which can be found in the appendix. For the most challenging reconstructed objects, we observe that in the Z+2 jets scenario, the $\Delta R_{j_1 j_2}$ interpolates around the hard cutoff at $\Delta R_{j_1 j_2} = 0.4$, indicating that the model struggles to learn this sharp feature. This error is expected, given that the CM is trained on the CFM and uses the same preprocessing.

Z+3Jets For the most challenging dataset with a Z boson and three jets, which can be seen at the bottom of Figure 5.2, the sharp feature in $\Delta R_{j_1 j_2}$ and other jet configurations exhibits similar failure modes to those observed in the two-jet scenario. The Z resonance peak is now wider and significantly deviates from the CFM samples and the data. This issue is partly due to the limited number of training events (300K). Since the CFM model already performs suboptimally on this dataset, the cumulative errors result in a wider Z resonance peak of the CM model.

Figure 5.3 shows an example of the difference in single step and multistep with seven iterations. The single step generation is noiser and the features are washed out, resulting in a wider Z peak and larger tails. The multistep sampling is able to improve the sample quality. We can see that the Z peak is sharper and the tails are reduced.

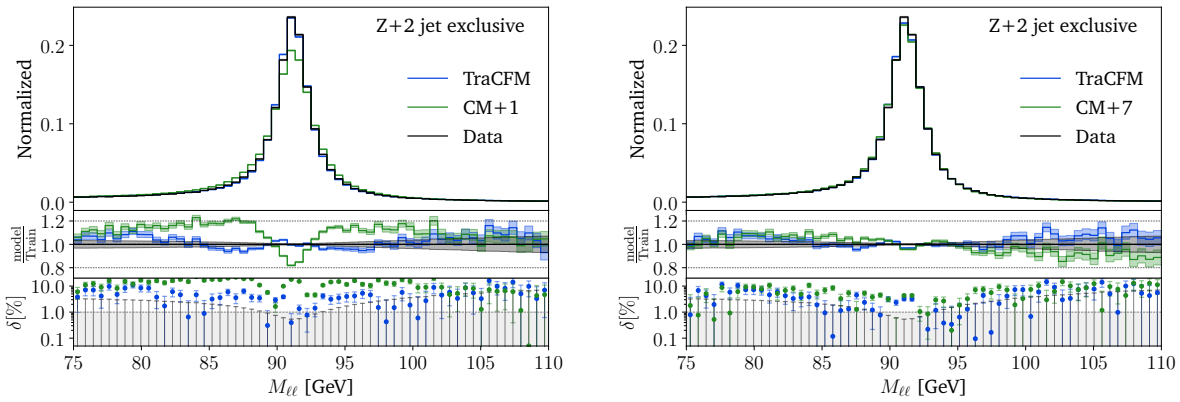


Figure 5.3: Difference in Mutlistep sampling vs Single step sampling for Consistency Model Z+2Jets

Our interest lies in the trade-off between sample quality and speed. Additionally, we are eager to understand the behavior of these improvements and to quantify the sample quality. Therefore, in the next chapter, we are going to evaluate the sample quality with different metrics.

5.4 Wasserstein and Energy Metric

We calculate the Wasserstein distance and energy distance for the Conditional Flow Matching (CFM) and Consistency Model (CM) using 1.000.000 generated events. The metrics quantify the difference between the dataset and generated samples on the reconstructed Z resonance peak. To obtain statistical uncertainties, the process is repeated 10 times. The results are shown in Figure 5.4 and Table 5.1. We evaluate the CM with different multistep iterations: $(1, 2, 4, 8, 16, 32)$. The figure shows the deviation from the CFM model as $\frac{CM}{CFM}$. Our results indicate that the CM generally improves sample quality with increasing iterations.

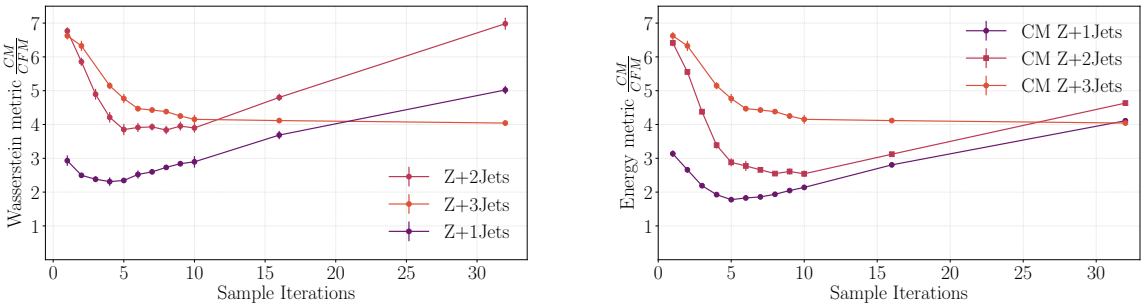


Figure 5.4: Waterstein and Energy distance of CM compaired to CFM

Table 5.1: 1D metrics of CFM, CM single step and CM multistep, Where W is the Wasserstein distance and E is the Energy distance

1D Metrics		Z+1Jets Dataset ($\times 10^{-1}$)		Z+2Jets Dataset ($\times 10^{-1}$)		Z+3Jets Dataset ($\times 10^{-1}$)	
Model	Network eval.	W_{M_Z} (↓)	E_{M_z}(↓)	W_{M_Z} (↓)	E_{M_z}(↓)	W_{M_Z} (↓)	E_{M_z}(↓)
CFM	68	0.95 ± 0.07	0.20 ± 0.01	1.15 ± 0.08	0.20 ± 0.01	2.86 ± 0.25	0.41 ±0.09
CM Single Step	1	3.37 ± 0.18	0.62 ± 0.02	6.45 ± 0.10	1.27 ± 0.02	17.26 ± 0.31	3.49 ± 0.04
CM Multistep	2	2.87 ± 0.06	0.53 ± 0.02	5.58 ± 0.11	1.10 ± 0.02	16.48 ± 0.43	3.35 ± 0.05
CM Multistep	4	2.66 ± 0.15	0.38 ± 0.01	4.02 ± 0.15	0.67 ± 0.02	13.41 ± 0.29	2.74 ± 0.04
CM Multistep	8	3.14 ± 0.06	0.38 ± 0.01	3.66 ± 0.12	0.50 ± 0.01	11.41 ± 0.19	2.23 ± 0.04
CM Multistep	16	4.24 ± 0.14	0.56 ± 0.01	4.58 ± 0.10	0.62 ± 0.01	10.72 ± 0.20	1.87 ± 0.03
CM Multistep	32	5.78 ± 0.14	0.82 ± 0.01	6.66 ± 0.17	0.92 ± 0.02	10.53 ± 0.22	1.60 ± 0.03

However, this improvement is only observed up to approximately 5-7 steps. Beyond this point, artifacts begin to accumulate with additional steps in mutlistep sampling, resulting in worse metric scores. This observation is consistent with findings in the literature [40]. Given that our primary interest lies in the speed and efficiency of the model, this limitation does not pose a significant problem for our objectives.

5.5 Classifier Metric

1D metrics can be a good initial indicator of performance, but they are not able to capture errors hidden in the 1D histograms, such as false correlations. Therefore, we train a classifier on samples from the generative network (CFM or CM) with label 0 and on samples from the data distribution (label 1) to use it as a better metric and diagnostic tool for identifying the failure modes of generative networks.

The classifier model is a simple feedforward neural network with 5 hidden layers and 256 neurons each. We use the binary cross entropy loss and the ADAM optimizer. Overfitting is prevented by using dropout and early stopping. Overall hyperparameters can be found in table 7.3. The classifer is trained on all datasets $Z+\{1,2,3\}$Jets with the configurations: $\{Data, CFM\}$ $\{Data, CM\}$ and $\{CFM, CM\}$. The last one is particularly interesting because it tests whether the CFM and CM have different failure modes.

We train the classifier on the same observables as the generator. Additionally, we feed the classifier reconstructed objects like ΔR_{ij} and M_{ll}:

$$\{p_{T,i}, \Delta\Phi_{i,i-1}, M_i\} \cup \{\Delta R_{ij}\} \cup \{M_{ll}\} \tag{5.14}$$

A trained classifer can be used to evaluate the performance of the generative network with the AUC score and ROC curved described in Chapter 4. To test if the classifier is able to extract the log likelihood ratio, we

reweight the samples from the generative network and compair the reweighted distribution with the target distribution.

$$w(x) = \frac{p_{data}}{p_{model}} = \frac{D(x)}{1 - D(x)} \quad \text{with } D(x) = \frac{p_{data}}{p_{data} + p_{model}} \tag{5.15}$$

with the assumption that the classifier D(x) learns the density ratio.

We examine in detail at the classifier trained on the CM model with Z+3Jets, which is the most challenging dataset and exhibits the most interesting failure modes. The other datasets are not shown here but results are summarized in the table 5.2 and in the figure 5.7.

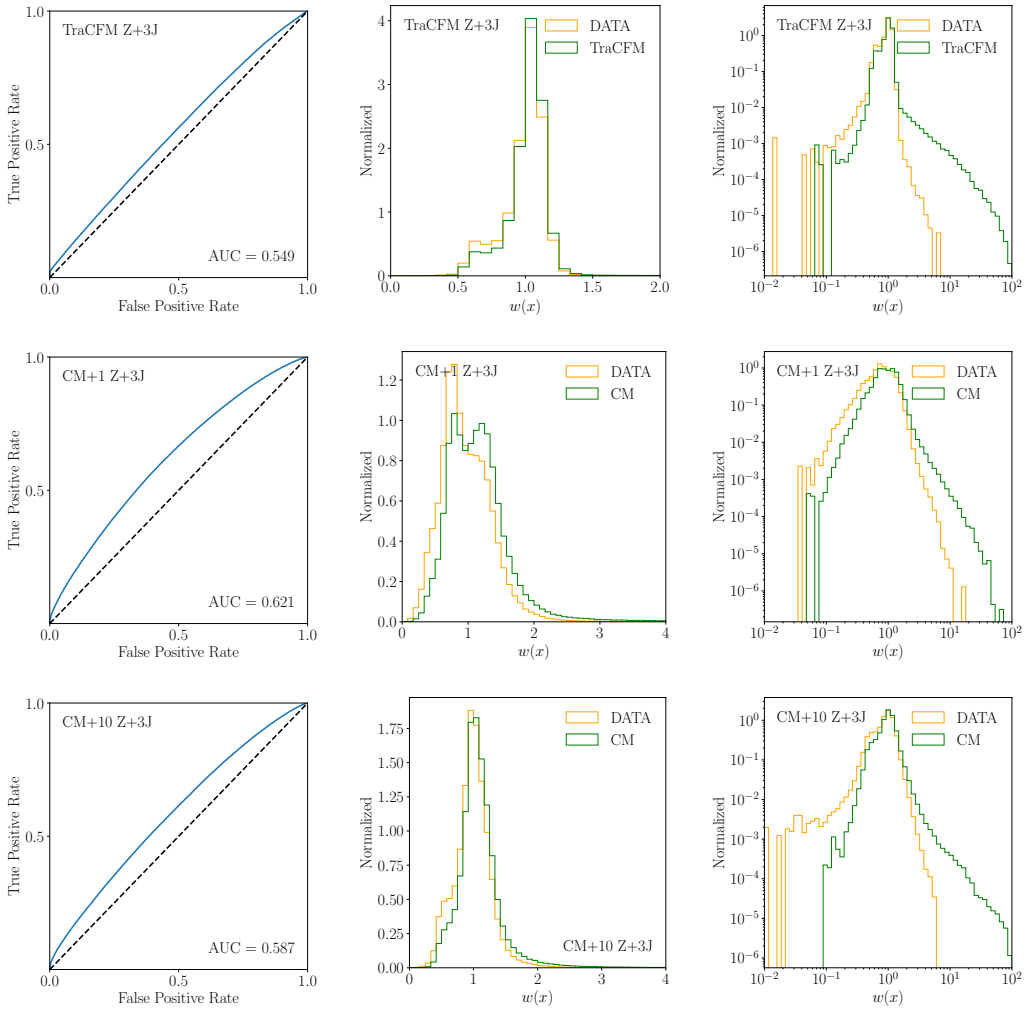


Figure 5.5: ROC and AUC score (left), classifier weights (Middle), classifier weights with log scale of Z+3Jet Models

The CFM model generates Z+3Jets with high precision. However, the Z resonance peak is slightly wider, and there is still false interpolation in the ΔR_{ij} distribution. Despite this, other distributions are accurately captured, resulting in an AUC score of 0.549, which is close to the diagonal line and indicates good performance. The weight distribution (figure 5.5 top) is peaked around one with small tails, which reflects the known issues with ΔR_{ij} and the wider Z resonance peak.

In contrast, the classifier trained on the CM model for Z+3Jets, using one-step generation, shows different behavior. In figure 5.5 middle)The ROC curve is notably farther from the diagonal line, with an AUC score of 0.621. This suggests that the generative model struggles to capture the data distribution effectively. The weight distribution now shows two peaks around one and significant tails, indicating deviations from the data distribution and poorer sample quality. This is mainly due to the washed-out features in the kinematic distributions, such as ΔR_{ij} and M_{ll}.

Performing multiple-step sampling with 10 multistep iterations improves the sample quality. The weights are now more peaked around one with reduced tails, and the AUC score improves to 0.587, with the ROC curve moving closer to the diagonal line. This indicates that the sample quality has improved.

Reweighting the samples generated by the CM model with the classifier weights brings the samples into better alignment with the true data distribution. This is evident from a sharper Z peak and reduced tails in the distribution. Additionally, the ΔR_{ij} distribution is now more accurately captured, further enhancing the precision of the generator's output.

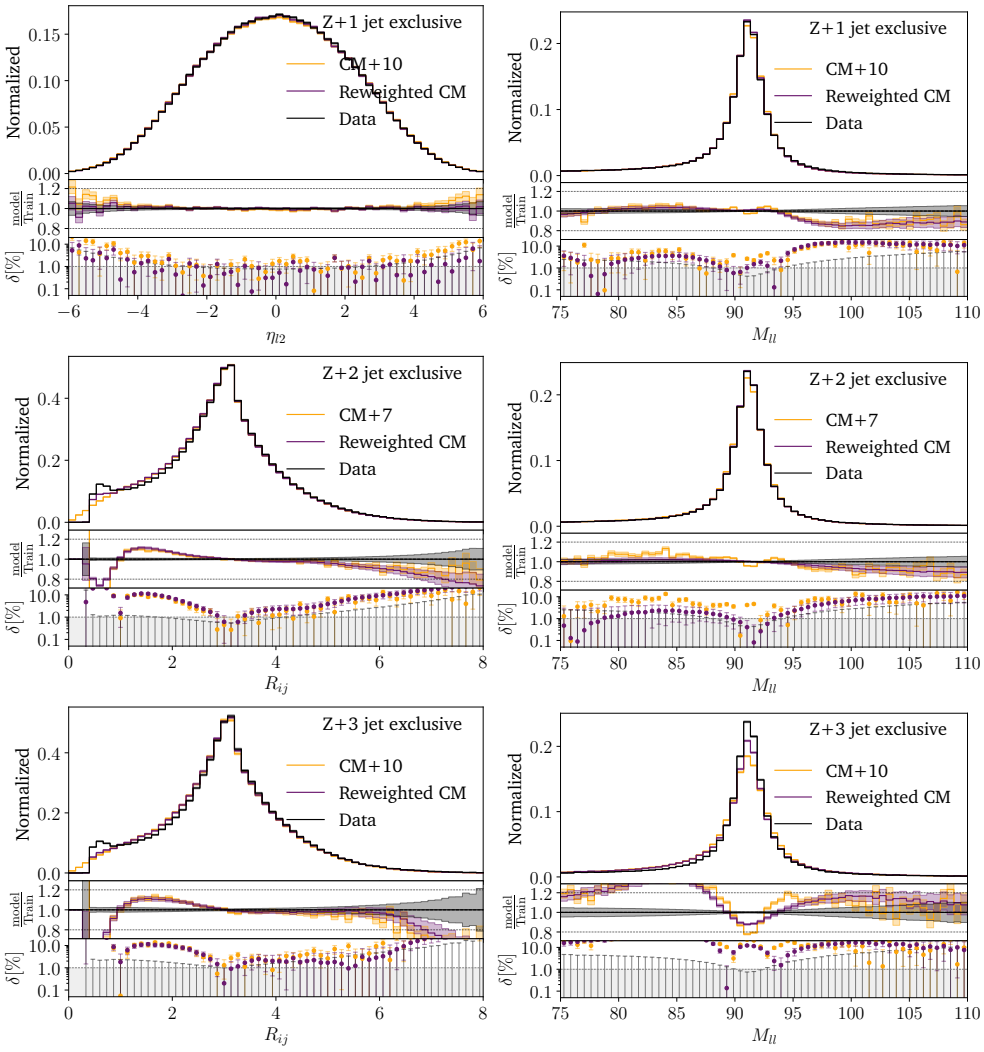


Figure 5.6: CM reweighted Samples. Z+1 Jets(Top), Z+2Jets (Middle) and Z+3Jets (Bottom)

5.6 Performance and Comparison

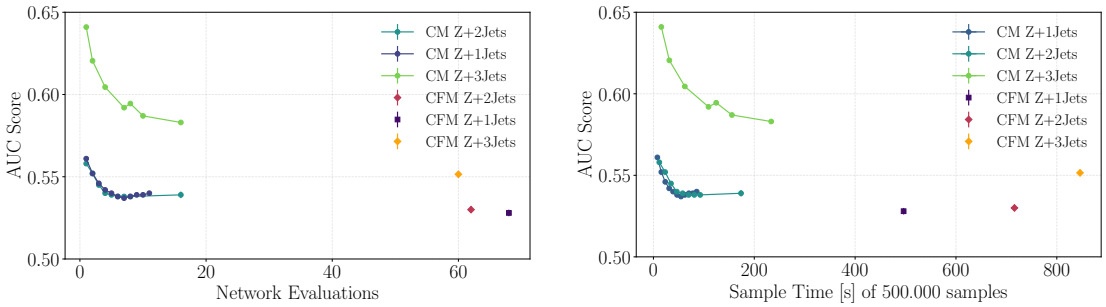


Figure 5.7: AUC Score of CFM, CM against the Sample Time (Right) and the number of Network Iterations (Left)

Table 5.2: AUC Score and Generation time of CFM and CM for Z Boson+Jets Dataset

Class. Metrics		Dataset Z+1J		Dataset Z+2J		Dataset Z+3J	
	Network eval.	AUC	GEN. Time [s]	AUC	GEN Time. [s]	AUC	GEN. Time[s]
CFM	(68, 62, 60)	0.528	496.19 ± 0.8	0.530	716 ± 0.9	0.551	846 ± 0.9
CM	1	0.561	7.8 ± 0.1	0.558	11.6 ± 0.1	0.641	15 ± 0.1
CM	2	0.552	15.6 ± 0.1	0.552	23.1 ± 0.2	0.620	32 ± 0.2
CM	4	0.542	31.2 ± 0.2	0.540	46.3 ± 0.2	0.592	62 ± 0.3
CM	7	0.537	54.6 ± 0.3	0.538	81.1 ± 0.3	0.591	109 ± 0.4
CM	8	0.538	62.4 ± 0.3	0.538	92.2 ± 0.4	0.592	124 ± 0.5
CM	16	-	-	0.539	184.5 ± 0.1	0.583	248 ± 0.1

In Figure 5.7 and table 5.2, we examine the overall AUC performance of the Consistency and Conditional Flow Matching Models, along with their generation speed. We provide not only an estimate of the sample time for 500,000 samples but also a comparison of network evaluations. For the Consistency Models, the number of multistep iterations is flexible, allowing us to generate samples with varying network evaluations. We train three classifiers for all generated samples and obtain an AUC uncertainty of 0.001. In table 5.2, we only show the average AUC score.

For the Conditional Flow Matching model, we measured the number of times the ODE solver called the CFM model and compared this to the Consistency Model with different generation steps. Depending on the dataset, the CFM maps from noise to data on average in 68 steps for Z+1j, 62 steps for Z+2j, and 60 steps for Z+3j. It is important to note that using faster ODE settings does not improve CFM generation speed while preserving sample quality. We tested this with faster ODE solver settings and observed that for Z+1 Jets, the CFM achieved an AUC of 0.65 with 32 network iterations, indicating that speedup via the ODE solver alone is not feasible.

We observe that the speedup factor for the Consistency Model scales linearly with the number of iterations, ranging from 5 to 60 times. We conclude that a Consistency Model with 7 iterations is optimal for matching CFM performance, providing speedup of 10 times while maintaining the best trade-off between speed and quality.

6 Conclusion and Outlook

As the Large Hadron Collider (LHC) enters a new phase of precision measurements, the demand for fast and accurate simulations becomes critical to process the immense volume of data produced. Machine learning techniques present a promising approach to enhancing data analysis at every stage of the LHC's processing pipeline.

In this thesis, we implemented an event generation model for LHC events using both a Conditional Flow Matching Model and Consistency Models, based on a Transformer architecture. The objective was to accelerate event generation while maintaining high-quality samples. Our results demonstrate that Consistency Models improve sampling speed by a factor of 5 to 60, depending on the number of generation steps. The optimal configuration achieved a seven-step iteration, resulting in a sampling speed improvement of approximately 10-fold compared to CFM, while maintaining comparable performance in terms of AUC.

We conclude that Consistency Models are powerful tools for accelerating simulations in score-based diffusion models at the LHC. Given the rapid advancements in machine learning applications within particle physics, it is likely that ML will become an essential tool for future high-luminosity LHC experiments.

Notably, Consistency Models perform well with the same hyperparameters as CFM, offering a straightforward distillation process. However, during training, we observed that Consistency Models required significantly longer training times. This limitation, widely recognized in the literature, has been recently addressed by an optimized training algorithm [24, 45]. Future research could investigate these techniques to further improve training efficiency.

A more advanced setup, particularly moving beyond the linear trajectory and Euler method, could improve sample quality. Exploring alternative numerical methods and trajectories is a promising direction for future research.

Finally, event generation is only the first step. Future work could focus on the development of conditional generative models. Additionally, adapting the network architecture to allow Consistency Models to solve inverse problems is an interesting task. Furthermore, the Consistency Models literature already suggests a training approach that eliminates the need for CFM as a teacher model. For a final implementation, we would use a Bayesian network to achieve better uncertainty control. For high-dimensional problems, Latent Consistency Models offer potential, as Latent Diffusion models have demonstrated superior performance in video, audio, and image generation [33, 34]. In conclusion, these advancements would contribute to more efficient and accurate simulations.

References

[1] GEORGES AAD et al. „Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC“. In: *Physics Letters B* 716.1 (2012), pp. 1–29.

[2] GUIDO ALTARELLI, R KEITH ELLIS, and G MARTINELLI. „Large perturbative corrections to the Drell-Yan process in QCD“. In: *Nuclear Physics B* 157.3 (1979), pp. 461–497.

[3] GIORGIO APOLLINARI et al. *High-Luminosity Large Hadron Collider (HL-LHC). Technical Design Report V. 0.1*. Tech. rep. Fermi National Accelerator Lab.(FNAL), Batavia, IL (United States), 2017.

[4] MATHIAS BACKES et al. „An unfolding method based on conditional Invertible Neural Networks (cINN) using iterative training“. In: *SciPost Physics Core* 7.1 (2024), p. 007.

[5] FALK BARTELS et al. *Studying the Z Boson with the ATLAS Detector at the LHC*. Version 1.2, December 11, 2020. 2020. URL: https://www.physi.uni-heidelberg.de/Einrichtungen/FP/anleitungen/F91.pdf.

[6] JAMES BEACHAM et al. „Physics beyond colliders at CERN: beyond the standard model working group report“. In: *Journal of Physics G: Nuclear and Particle Physics* 47.1 (2019), p. 010501.

[7] SEBASTIAN BIERINGER et al. „Calomplification—the power of generative calorimeter models“. In: *Journal of Instrumentation* 17.09 (2022), P09028.

[8] ENRICO BOTHMANN et al. „Event generation with Sherpa 2.2“. In: *SciPost Physics* 7.3 (2019), p. 034.

[9] ERIK BUHMANN et al. „CaloClouds II: ultra-fast geometry-independent highly-granular calorimeter simulation“. In: *Journal of Instrumentation* 19.04 (2024), P04020.

[10] ANJA BUTTER and TILMAN PLEHN. „Generative Networks for LHC events“. In: *Artificial intelligence for high energy physics*. World Scientific, 2022, pp. 191–240.

[11] ANJA BUTTER et al. „GANplifying event samples“. In: *SciPost Physics* 10.6 (2021), p. 139.

[12] ANJA BUTTER et al. „Generative networks for precision enthusiasts“. In: *SciPost Physics* 14.4 (2023), p. 078.

[13] ANJA BUTTER et al. „Jet Diffusion versus JetGPT–Modern Networks for the LHC“. In: *arXiv preprint arXiv:2305.10475* (2023).

[14] MATTEO CACCIARI, GAVIN P SALAM, and GREGORY SOYEZ. „FastJet user manual: (for version 3.0. 2)“. In: *The European Physical Journal C* 72 (2012), pp. 1–54.

[15] STEFANO CATANI et al. „QCD matrix elements+ parton showers“. In: *Journal of High Energy Physics* 2001.11 (2002), p. 063.

[16] CHRISLB. *ArtificialNeuronModel*. Wikimedia Commons. 2005.

[17] ATLAS COLLABORATION. „Technical Design Report for the ATLAS Inner Detector: Inner Tracker“. In: *CERN Technical Design Report* (1997). CERN-LHCC-97-16. URL: https://cds.cern.ch/record/527822/files/p3.pdf.

[18] ATLAS COLLABORATION et al. „The ATLAS experiment at the CERN Large Hadron Collider: a description of the detector configuration for Run 3“. In: *arXiv preprint arXiv:2305.16623* (2023).

[19] CUSH. *Standard Model of Elementary Particles*. Wikimedia Commons. Accessed: 2024-09-08. 2014. URL: https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg.

[20] RANIT DAS et al. „How to understand limitations of generative networks“. In: *SciPost Physics* 16.1 (2024), p. 031.

[21] CARL DOERSCH. „Tutorial on variational autoencoders“. In: *arXiv preprint arXiv:1606.05908* (2016).

[22] JOHN R DORMAND and PETER J PRINCE. „A family of embedded Runge-Kutta formulae“. In: *Journal of computational and applied mathematics* 6.1 (1980), pp. 19–26.

[23] SIDNEY D DRELL and TUNG-MOW YAN. „Massive lepton-pair production in hadron-hadron collisions at high energies“. In: *Physical Review Letters* 25.5 (1970), p. 316.

[24] ZHENGYANG GENG et al. „Consistency Models Made Easy“. In: *arXiv preprint arXiv:2406.14548* (2024).

[25] IAN GOODFELLOW et al. „Generative adversarial networks“. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[26] YANJUN HE et al. „The Annotated Transformer: A Data-Driven Study of Transformer Family Variants“. In: *arXiv preprint arXiv:2211.17106* (2022).

[27] JONATHAN HO, AJAY JAIN, and PIETER ABBEEL. „Denoising diffusion probabilistic models“. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[28] NATHAN HUETSCH et al. „The Landscape of Unfolding with Machine Learning“. In: *arXiv preprint arXiv:2404.18807* (2024).

[29] MARTINA JAVURKOVA. „The Fast Simulation Chain in the ATLAS experiment“. In: *EPJ Web of Conferences*. Vol. 251. EDP Sciences. 2021, p. 03012.

[30] DIEDERIK P KINGMA. „Adam: A method for stochastic optimization“. In: *arXiv preprint arXiv:1412.6980* (2014).

[31] YARON LIPMAN et al. „Flow matching for generative modeling“. In: *arXiv preprint arXiv:2210.02747* (2022).

[32] ERIC LUHMAN and TROY LUHMAN. „Knowledge Distillation in Iterative Generative Models for Improved Sampling Speed“. In: *arXiv preprint arXiv:2101.02388* (2021).

[33] SIMIAN LUO et al. *Latent Consistency Models: Synthesizing High-Resolution Images with Few-Step Inference*. 2023. arXiv: 2310.04378 [cs.CV].

[34] SIMIAN LUO et al. „LCM-LoRA: A Universal Stable-Diffusion Acceleration Module“. In: *arXiv preprint arXiv:2311.05556* (2023).

[35] PIERRE M NUGUES. „Self-Attention and Transformers“. In: *Python for Natural Language Processing: Programming with NumPy, scikit-learn, Keras, and PyTorch*. Springer, 2024, pp. 425–447.

[36] PARTICLE DATA GROUP. „Review of Particle Physics“. In: *Progress of Theoretical and Experimental Physics* 2022.8 (2022), p. 083C01. DOI: 10.1093/ptep/ptac097. URL: https://pdg.lbl.gov/.

[37] ADAM PASZKE et al. „Pytorch: An imperative style, high-performance deep learning library“. In: *Advances in neural information processing systems* 32 (2019).

[38] TILMAN PLEHN et al. „Modern machine learning for LHC physicists“. In: *arXiv preprint arXiv:2211.01421* (2022).

[39] DAVID E RUMELHART, GEOFFREY E HINTON, and RONALD J WILLIAMS. „Learning representations by back-propagating errors“. In: *nature* 323.6088 (1986), pp. 533–536.

[40] MARVIN SCHMITT et al. „Consistency Models for Scalable and Fast Simulation-Based Inference“. In: *arXiv preprint arXiv:2312.05440* (2023).

[41] NIV SHAUL et al. „Bespoke Solvers for Generative Flow Models“. In: *arXiv preprint arXiv:2310.19075* (2023).

[42] YANG SONG and PRAFULLA DHARIWAL. „Improved techniques for training consistency models“. In: *arXiv preprint arXiv:2310.14189* (2023).

[43] YANG SONG et al. „Consistency models“. In: *arXiv preprint arXiv:2303.01469* (2023).

[44] ASHISH VASWANI. „Attention is all you need“. In: *arXiv preprint arXiv:1706.03762* (2017).

[45] FU-YUN WANG et al. „Phased Consistency Model“. In: *arXiv preprint arXiv:2405.18407* (2024).

[46] RAMON PETER WINTERHALDER. „How to GAN, Novel simulation methods for the LHC“. PhD thesis. U. Heidelberg, 2024. URL: https://www.thphys.uni-heidelberg.de/~plehn/includes/theses/winterhalder_d.pdf.

[47] TUNG-MOW YAN. „Drell-Yan Mechanism“. In: *UNIVERSE-TAIPEI* 3.3 (2015), pp. 45–50.

7 Appendix

7.1 Hyperparameters

Table 7.1: CFM Hyperparameters of CFM on Z+Jets Dataset

CFM Hyperparameters	Z+1Jet Dataset	Z+2Jets Dataset	Z+3Jets Dataset
Input dim	10	15	19
Output dim	9	14	18
Network	Transformer	Transformer	Transformer
Embedding dim	64	64	128
Feedforward dim	256	256	512
Decode layers	6	8	6
Encoder layers	1	1	1
Self attention Heads	4	4	4
Model parameters	450K	585K	1.7M
Loss	MSE	MSE	MSE
Batch size	16384	16384	8192
LR Scheduling	Cosine annealing	Cosine annealing	Cosine annealing
Learning rate	3e-4	3e-4	1e-4
Epochs	300	500	10.000
Training Events	3.5M	1M	250K
Generated Events	1M	1M	300K
ODE Solver	`scipy.solve_ivp`	`scipy.solve_ivp`	`scipy.solve_ivp`
ODE setting atol:	1.0e-06	1.0e-06	1.0e-06
EULER	1.0e-3	1.0e-3	1.0e-3

Table 7.2: CM Hyperparameters of CFM on Z+Jets Dataset

CM Hyperparameters	Z+1Jet Dataset	Z+2Jets Dataset	Z+3Jets Dataset
Input dim	10	15	19
Output dim	9	14	13
Network	Transformer	Transformer	Transformer
Embedding dim	64	64	128
Feedforward dim	512	256	512
Decode layers	6	8	6
Encoder layers	1	1	1
Self attention Heads	4	4	4
Model parameters	690K	580K	1.7M
Loss	MSE	MSE	MSE
Batch size	16384	16384	4096
LR Scheduling	Cosine annealing	Cosine annealing	Cosine annealing
Learning rate	3e-4	3e-4	1e-4
Epochs	2.000	3.000	10.000
Training Events	3.5M	1M	250K
Generated Events	1M	1.0M	300K
EULER	1.0e-3	1.0e-3	1.0e-3

Table 7.3: Hyperparamters of Classification model for conditional flow matching and consitency model gen. data

Hyperparameters	CFM,CM and Training Dataset
	$Z + \{1,2,3\text{jets}\}$
Network	MLP
Input dim	10, 17, 23
Layers	5
Hidden dim	256
Learning rate	1e-4
Batch size	1.024
Activation function	ReLU
Loss	MSE
Training Data	(3M Data, 3M Gen. Samples)
regularisation	Early stopping, Dropout 10%
epochs	500

7.2 Mathematics

Invariant Z Mass Calculation

To calculate the invariant mass of the two leading leptons in the event, we use:

$$M_{ll}^2 = E_{ll}^2 - \vec{p}_{ll}^2 \tag{7.1}$$

Since the leptons are treated as massless, we use $E_l = |p_l|$. Given the geometric relationship for the transverse momentum: $p_T = p\sin(\theta)$, where θ is the polar angle, we can write the expression for the invariant mass as:

$$M_{ll}^2 = 2P_{t,1}P_{t,2}\left(\frac{1}{\sin(\vartheta_1)\sin(\vartheta_2)} - \cos(\varphi_1-\varphi_2) - \frac{1}{\tan(\vartheta_1)\tan(\vartheta_2)}\right) \tag{7.2}$$

Using the relationship between η and ϑ:

$$\sinh(\eta) = \frac{1}{\tan(\vartheta)} \quad \cosh(\eta) = \frac{1}{\sin(\vartheta)} \tag{7.3}$$

and hyperbolic trigonometric identities we can rewrite eq 7.2, we obtain:

$$M_{ll}^2 = 2P_{t,1}P_{t,2}\big(\cosh(\eta_1-\eta_0) - \cos(\varphi_1-\varphi_2)\big) \tag{7.4}$$

CFM Theorem 1

give a vectorfield $v_t(x|x_1)$ that generates the coditional probablity path $p_t(x|x_1)$, for any $q(x1)$ the marginal vector field v_t generated the marginal probability path p_t. Meaning v_t p_t satisfy the continuity equation

$$\frac{d}{dt}p_t(x) = \int\left(\frac{d}{dt}p_t(x|x_1)\right)q(x_1)dx_1 = -\int \text{div}\big(v_t(x|x_1)p_t(x|x_1)\big)\,q(x_1)dx_1 \tag{7.5}$$

$$= -\text{div}\left(\int v_t(x|x_1)p_t(x|x_1)q(x_1)dx_1\right) = -\text{div}\big(v_t(x)p_t(x)\big) \qquad \square \tag{7.6}$$

7.3 Kinematics

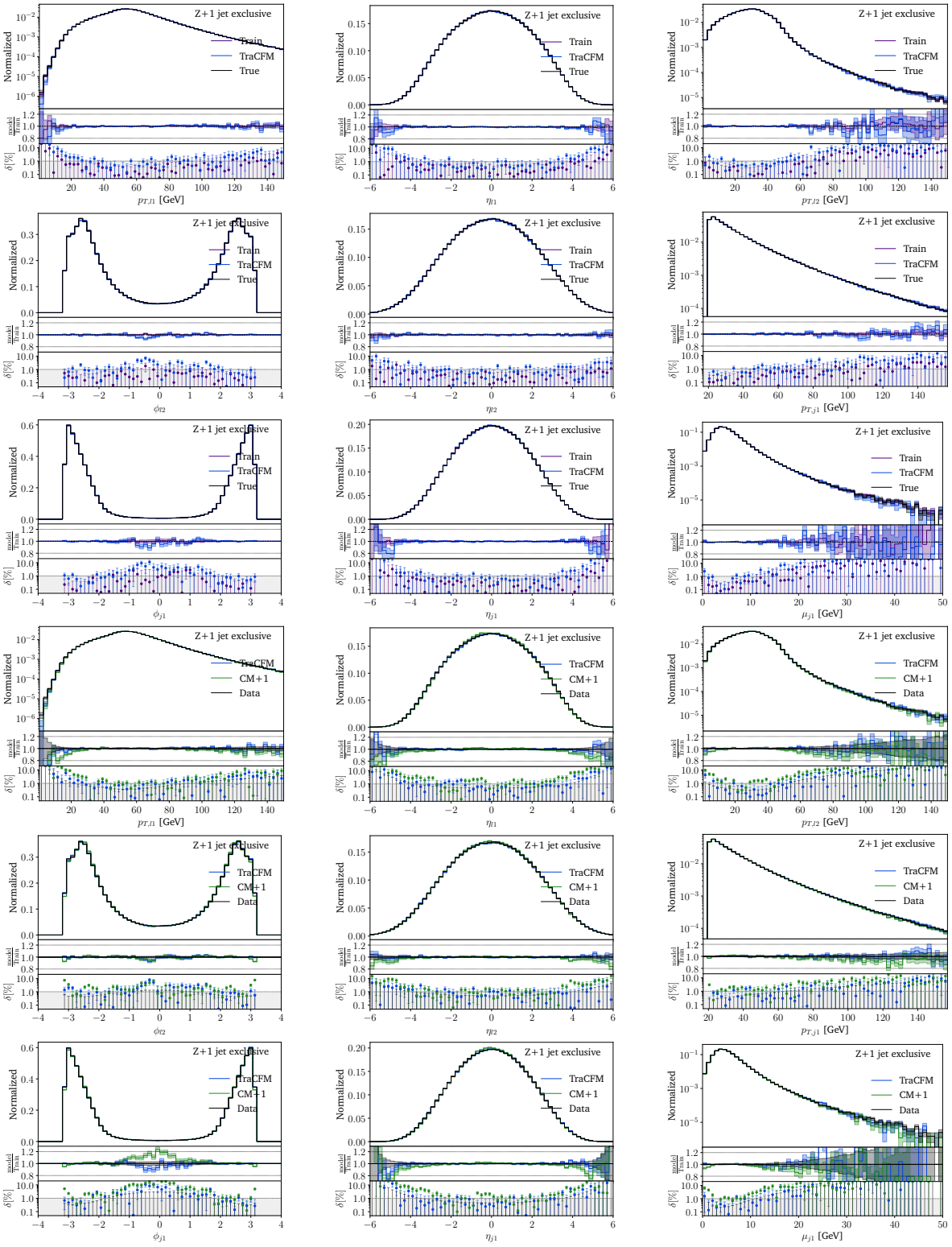


Figure 7.1: CFM and CM + 1Jets

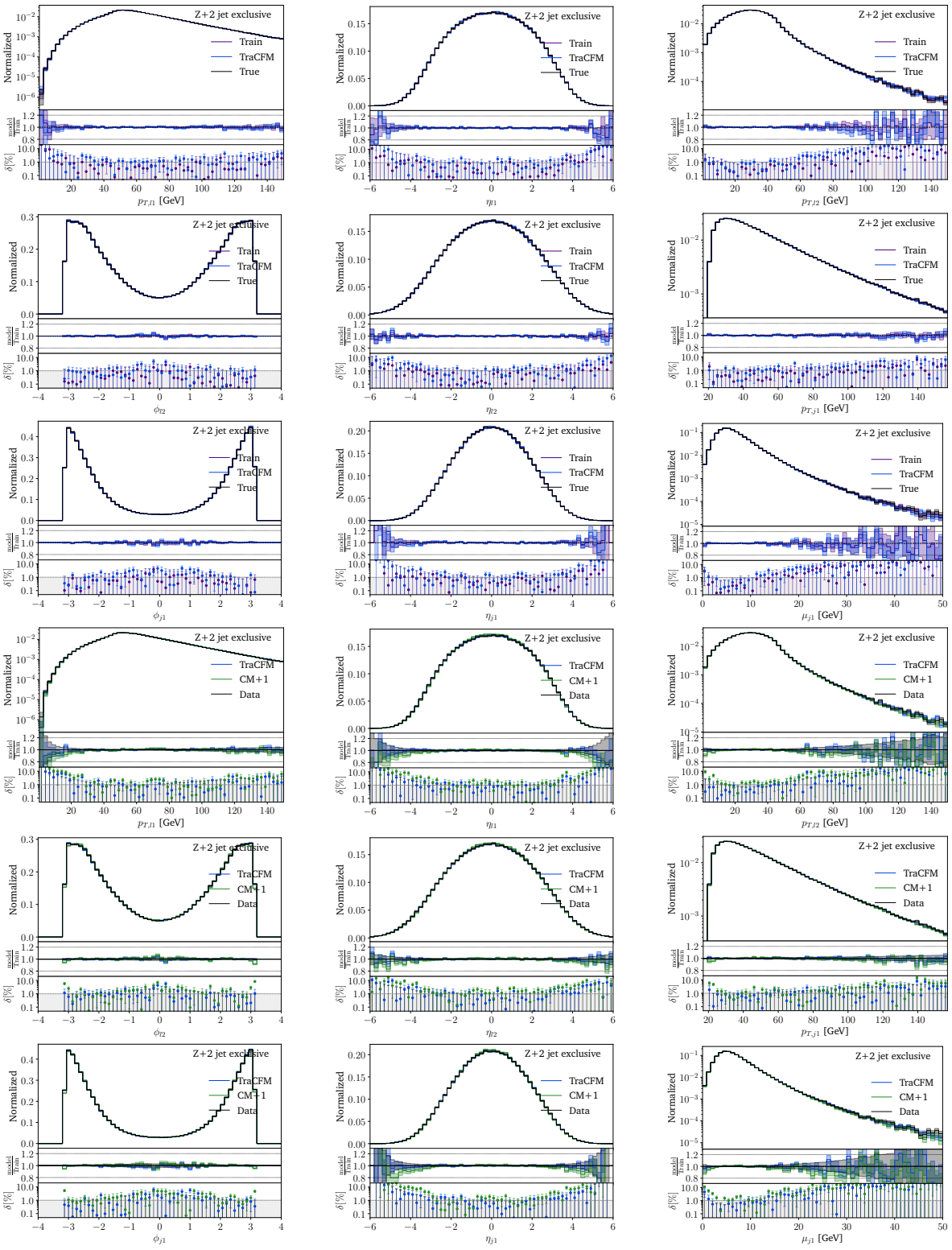


Figure 7.2: CFM and CM + 2Jets

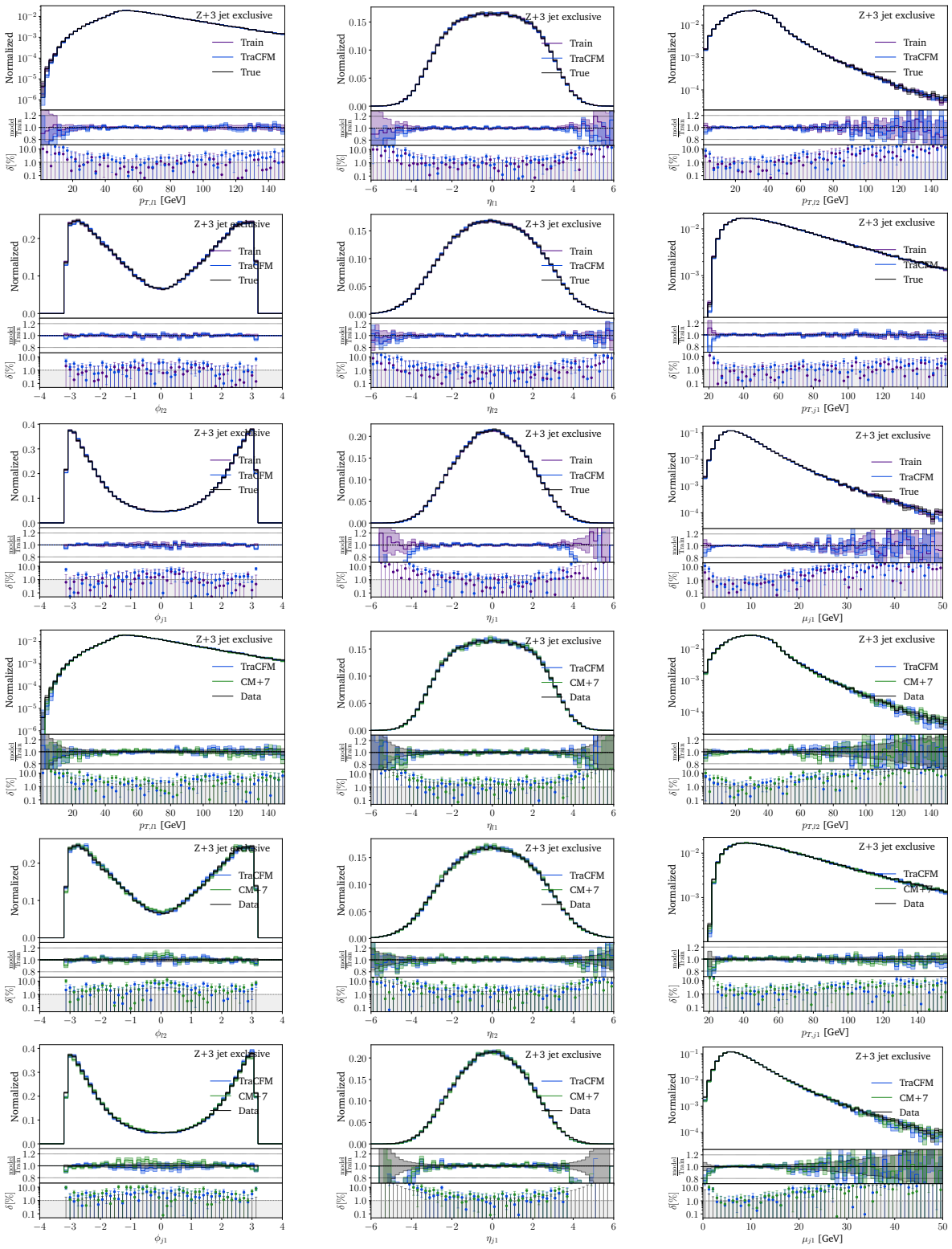


Figure 7.3: CFM and CM + 3Jets

8 Acknowledgements

ich bin sehr dankbar dafür, dass Tilman Plehn mir die Möglichkeit gegeben hat, in der Forschungsgruppe mitzuarbeiten. Es war für mich nicht nur interessant, einen ersten tatsächlichen Einblick in die Forschung zu bekommen, sondern auch die Erfahrung hat mich tief beeindruckt. Besonders dankbar bin ich dafür, dass ich mit Nathan Huetsch zusammenarbeiten konnte. Nathan hat mir bei vielen Problemen geholfen, zahlreiche Fragen beantwortet und war gleichzeitig eine Person, zu der ich aufschaue – eine Inspiration, wie ich mich weiterentwickeln möchte. Ich bin auch dankbar für die Forschungsgruppe, die trotz meiner kurzen und eher unscheinbaren Arbeit häufig nachgefragt und Interesse gezeigt hat. Ich habe mich sehr willkommen gefühlt und es war mir eine Ehre, von so vielen intelligenten und inspirierenden Menschen umgeben zu sein. Ich möchte mich bei Marius, Dominik, Valentin, Roman und Jan bedanken für die gemeinsame Zeit im Studium. Ich werde mein Leben lang auf diese Zeit zurückblicken und im kern an euch denken. Ebenso bin ich meinen Freunden aus der Heimat sehr dankbar. Besonders möchte ich meiner Familie für ihre Unterstützung danken: meiner wunderbaren Mutter und meinen liebevollen Geschwistern. Auch meinem Vater möchte ich danken, der all das nicht mehr miterleben konnte, aber immer in meinen Gedanken bei mir ist.

9 Declaration

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 16.09.2024,