Alma Mater Studiorum - Università di Bologna

Department of Physics and Astronomy
Master Degree Programme in Physics

Hyperparameter tuning of BoloGAN: a CWGAN-GP model for fast simulation of the calorimeter of the ATLAS experiment

Graduation Thesis

March 26, 2025

Presented by:
Muhammad Wisal Abdullah

Supervisor:
Prof. Lorenzo Rinaldi

Co-supervisor:
Federico Andrea Guillaume Corchia

Academic year 2023-2024
Graduation date March 2025

Acknowledgements

I extend my gratitude to my supervisor Professor Lorenzo Rinaldi and co-supervisor Federico A.G. Corchia for their invaluable contributions and insights through numerous engaging discussions. I am deeply thankful to my family, friends, and a brother-in-spirit Hamza Shafique for their unwavering support and patience throughout this endeavor.

Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry

---

Richard P. Feynman

Abstract

The analysis of current high energy experiments at the LHC requires simultaneous comparisons with simulations for the purposes of event reconstruction, background estimation and detector design and optimization, etc. Geant4 is the traditional simulation toolkit and has been widely used to carry these simulations. It has problems related to computational expense and slow speed. For fast simulation, Generative Models are being employed to approximate detector responses and event generation. BoloGAN is a fast simulation technique built using a conditional Wasserstein GAN. The aim of this thesis is to tune the hyperparameters of the model using random search based on the validation metric of reduced chi squares between Geant4 data and generated energy distributions by simulating the ATLAS calorimeter, on HPC systems. The tuning will be executed on LXBATCH at CERN remotely. This is done in view to optimize the performance of the model, to improve the detector response simulation, helping in the recostruction of the physics object in the events collected by the ATLAS experiment.

The structure of this thesis is the following: an introduction to detector simulation is presented in Chapter 1 along with an broad overview and context of the project. Chapter 2 and Chapter 3 describe GANs and WGANs. Chapter 4 describes the hypothesis testing to compare Geant4 and BoloGAN, while Chapter 5 focuses on the optimization of hyperparameters. The results and analysis are finally demonstrated in Chapter 6 and Chapter 7.

Summary

Le analisi degli esperimenti di Fisica delle Alte Energie presso l'acceleratore LHC richiedono un continuo confronto con le simulazioni ai fini della ricostruzione dell'evento, della stima del background e della progettazione e ottimizzazione del rivelatore. Tradizionalmente, il toolkit di simulazione Geant4 è ampiamente utilizzato per eseguire tali simulazioni. Attualmente, Geant4 presenta alcuni problemi legati al consumo delle risprse computazionali e alla bassa velocità di esecuzione. Per una simulazione rapida, possono essere utilizzati modelli generativi per approssimare le risposte dei rilevatori e la generazione di eventi. BoloGAN è una tecnica di simulazione veloce sviluppata utilizzando un rete neurale GAN-Wasserstein condizionale. Lo scopo di questa tesi è quello di ottimizzare gli iperparametri del modello utilizzando una grid-search basata sulla confronto basato su una validazione del chi-quadrato ridotto tra le distribuzioni di energia generata simulando su sistemi HPC il calorimetro di ATLAS con Geant e con Bologan. L'ottimizzazione è eseguita su LXBATCH al CERN. Lo scopo del lavoro è di ottimizzare le prestazioni del modello, per migliorare la simulazione della risposta del rivelatore, aiutando nella ricostruzione fisica degli eventi raccolti dall'esperimento ATLAS.

Contents

List of Figures

List of Tables

1

Introduction

The Large Hadron Collider (LHC) at CERN is the world's most powerful particle accelerator, designed to collide protons at extremely high energies to explore fundamental physics. The ATLAS experiment is one of the four major detectors at the LHC, built to study a wide range of physics phenomena, including the Higgs boson, dark matter, and new particles beyond the Standard Model. ATLAS relies on accurate simulations to interpret the vast amounts of data generated by particle collisions. Geant4 is a toolkit for simulating the passage of particles through matter and is widely used in high-energy physics experiments, including ATLAS. At ATLAS, Geant4 is used to model the detailed interactions of particles with the detector's components, such as the Inner Detector, Calorimeters, and Muon Spectrometer. Geant4 provides a high level of accuracy but is computationally expensive, making it challenging to use for large-scale simulations. To address the computational cost of full Geant4 simulations, fast simulation tools like Atlfast and FastCaloSim were developed. These tools use parameterized models or precomputed shower libraries to approximate the detector response, significantly reducing the time and resources required for simulations. Fast simulations are particularly useful for generating large datasets for physics analysis, detector optimization, and machine learning applications. However these simulations lack in accuracy, Generative Adversarial Networks (GANs) are a class of machine learning models that have been adapted for fast simulation in high-energy physics. GANs can learn the complex patterns of particle showers and energy deposits from full Geant4 simulations and generate synthetic data that mimics the real detector response. GAN-based fast simulations offer several advantages namely speed, accuracy, and flexibility.

Simulations based on first principles serve as a crucial bridge between theory and experiment, playing a central role in the success of these facilities' physics programs. As the LHC enters future data-taking runs, the volume of generated data will significantly increase, necessitating a corresponding rise in the number of simulated events required for accurate and sensitive analyses. Consequently, the computational resources needed to produce these simulations will also grow.

Figure 1.4 illustrates the projected CPU requirements for the two general-purpose experiments, ATLAS ATLAS Collaboration et al. (2008) and CMS CMS Collaboration et al. (2008), with similar challenges faced by other experiments such as LHCb LHC Collaboration et al. (2008). A substantial fraction of CPU consumption is dedicated to simulation, particularly in modeling detector responses, with calorimeter simulations being especially demanding. These detectors pose significant computational challenges due to the need to track numerous secondary particles generated in extensive showers as primary particles interact with dense materials.

State-of-the-art physics-based simulations rely on Geant4 Agostinelli et al. (2003), which represents a major computational bottleneck. Projections indicate that simulation demands will surpass the available computing resources, potentially exceeding the computing budgets of both current and future experiments.

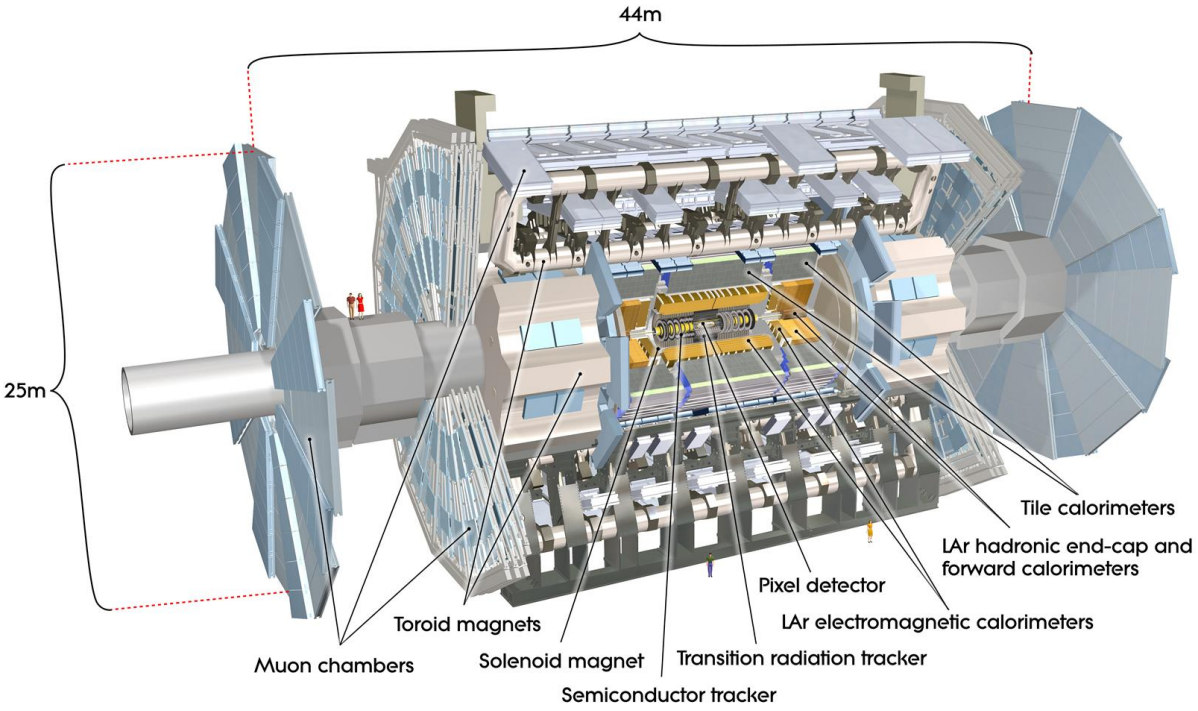


Figure 1.1. ATLAS detector layout ATLAS Collaboration (2023)

1.1 CERN and LHC

The Large Hadron Collider (LHC) stands as the world's most powerful and extensive particle accelerator. It began operations on September 10, 2008, and continues to be the most recent addition to CERN's accelerator facilities. The LHC features a 27-kilometer ring equipped with superconducting magnets and various accelerating structures designed to increase the energy of particles as they travel through the system.

Within the accelerator, two beams of high-energy particles travel at nearly the speed of light before being directed to collide. These beams move in opposite directions through separate beam pipes—tubes maintained at ultrahigh vacuum. They are steered around the accelerator ring by a powerful magnetic field generated by superconducting electromagnets. These electromagnets are constructed from coils of specialized electric cable that operate in a superconducting state, allowing electricity to flow without resistance or energy loss. To achieve this, the magnets must be cooled to -271.3°C—a temperature even colder than outer space. Consequently, much of the accelerator is linked to a distribution system of liquid helium, which cools the magnets, along with other essential supply services.

The Large Hadron Collider (LHC) utilizes thousands of magnets of various types and sizes to guide the beams around the accelerator. This includes 1,232 dipole magnets, each 15 meters long, which are responsible for bending the beams, and 392 quadrupole magnets, ranging from 5 to 7 meters in length, which focus the beams. Just before collision, an additional type of magnet is employed to "squeeze" the particles closer together, thereby increasing the likelihood of collisions. Given the minuscule size of the particles, achieving a collision is comparable to

firing two needles from 10 kilometers apart with such precision that they meet exactly halfway.

All the controls for the accelerator, including its services and technical infrastructure, are centralized under one roof at the CERN Control Centre. From this location, the beams within the LHC are directed to collide at four specific points around the accelerator ring, which correspond to the positions of the four particle detectors: ATLAS, CMS, ALICE, and LHCb.

1.2 ATLAS: General Features and structure of detector

The ATLAS Detector (A Toroidal LHC Apparatus) is one of the four major experiments at the Large Hadron Collider (LHC) at CERN, the European Organization for Nuclear Research. It is a general-purpose particle detector designed to explore a wide range of physics phenomena, including the search for the Higgs boson, dark matter, extra dimensions, and other new physics beyond the Standard Model. The ATLAS detector is one of the largest and most complex scientific instruments ever built.

The ATLAS detector is composed of several layers, each designed to detect different types of particles and measure their properties. The detector is cylindrical in shape and is divided into several subsystems. *Inner Detector*'s purpose is to track the paths of charged particles close to the collision point. It tracks charged particles and measures their momentum. *Calorimeter* measures the energy of particles by absorbing them. It has two components the electromagnetic calorimeter (EMcal) and hadronic calorimeter (Hcal). EMcal measures the energy of photons and electrons, and Hcal measures the energy of hadrons (e.g. protons, neutrons, pions). *Muon Spectrometer* detects and measures muons (through their momenta and trajectory), only charged particles that can penetrate the calorimeters. The *magnet system* bends the paths of charged particles to measure their momentum. The *Trigger and Data Acquisition System* selects interesting collision events from the billions of collisions per second and records the data.

There are two flavors of calorimeters: electromagnetic and hadronic. Electromagnetic calorimeters are designed to stop electrons and photons ,which have shallower and narrower showers compared with protons, neutrons,and charged pions. Hadronic calorimeters are thicker and deeper in order to capture penetrating radiation that forms irregular showers from nuclear interactions. In this first application of GANs to along longitudinally segmented calorimeter, we choose to focus only on electromagnetic showers. In addition to already providing the capability to simulate electrons and photons,the electromagnetic shower contains all of the new challenges described.

1.3 Detector Simulation at ATLAS

Detector simulation at the ATLAS experiment is a sophisticated and essential process that enables physicists to model and interpret the behavior of particles produced in high-energy proton-proton collisions at the Large Hadron Collider (LHC). The simulation workflow begins with event generation, where tools like Pythia or Herwig simulate the initial collisions and produce primary particles. These particles are then tracked through a detailed model of the ATLAS detector using Geant4, which simulates their interactions with detector materials, including

energy deposits in the calorimeters and hits in the muon spectrometer. For faster simulations, tools like Atlfast and FastCaloSim provide parameterized approximations of the calorimeter response. The simulated energy deposits are digitized to mimic the detector's electronic readout, and the ATLAS trigger system is simulated to select events of interest. The resulting simulated data is used for physics analysis, detector optimization, and training machine learning algorithms. Detector simulation plays a crucial role in ATLAS's research program, from the discovery of the Higgs boson to ongoing searches for new physics beyond the Standard Model. Despite challenges such as computational cost and the complexity of accurately modeling the detector, simulations are indispensable for interpreting experimental data, testing theoretical predictions, and advancing our understanding of the fundamental nature of the universe.

Geant4 calorimeter simulation

Geant4 is a comprehensive toolkit designed for simulating the passage of particles or radiation through matter. Applications developed with Geant4 can model any experimental setup, detector configuration, or radiation source, capturing specific physical quantities resulting from the interactions of primary and secondary particles with the materials in the setup. The toolkit provides complete functionality for all aspects of particle transport simulation, enabling users to create detailed geometric models with defined shapes and materials, locate points and navigate particle tracks within these models, and apply physics interactions to simulate particle-matter effects and generate secondary particles. It also allows users to record selected information as tallies or hits, which can be used to simulate detector responses, visualize the geometry of the setup and particle trajectories, and interact with the application through an extensible command-line or graphical user interface. Geant4 includes an extensive library of physics processes covering electromagnetic, strong, and weak interactions of particles within matter, supporting a wide energy range from milli-electron volts (meV) for thermal neutrons, electron volts (eV) for electrons, and typically kilo-electron volts (keV) for hadrons, up to hundreds of GeV—and in some cases, even extending to 100 tera-electron volts (TeV). This makes Geant4 a powerful and versatile tool for simulating particle interactions across a broad spectrum of energies and applications.

Several terms used within GEANT4 have meanings that differ slightly from their general usage. While these terms are defined in other documentation, they are briefly reviewed here for clarity and convenience. An *event* consists of the decay or interaction of a primary particle and a target, and all subsequent interactions, produced particles and four-vectors. G4Event objects encapsulate primary vertices and particles, and they may also include hits, digitizations, and trajectories. A *trajectory* is defined as a series of track snapshots that capture the particle's path as it moves through the simulated environment. A *run* comprises of a series of events. A *step* is defined by two endpoints that mark the boundaries of the fundamental unit of propagation in space or time. The distance between these points is determined by a combination of transportation and physics processes, and it can be constrained to a fixed size by the user when smaller step lengths are required. The G4Step object stores the changes in track properties that occur between these two endpoints. Allison et al. (2016)

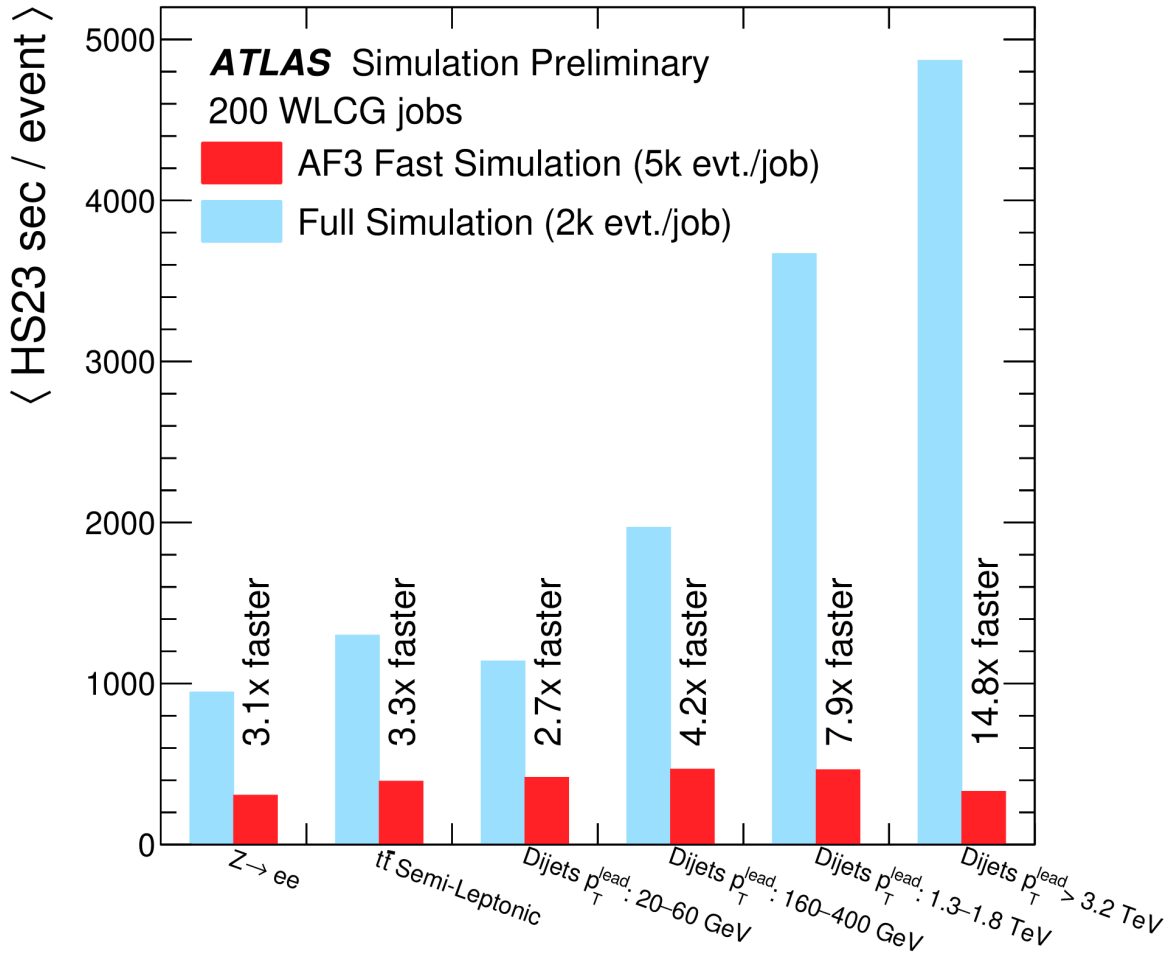


Figure 1.2. Geant4 Full simulation vs Fast simulationATLAS Collaboration (2023)

Fast calorimeter simulation: AtlFast and FastCaloSim

Over the past decade, ATLAS has developed and utilized tools to replace the most CPU-intensive component of simulation—calorimeter shower simulation—with faster simulation methods. AtlFast3 is the next generation of high-accuracy fast simulation in ATLAS. AtlFast3 combines parameterized approaches with machine-learning techniques and is deployed to address current and future computing challenges, as well as the simulation needs of the ATLAS experiment. With highly accurate performance and significantly improved modeling of substructure within jets, AtlFast3 can simulate large numbers of events for a wide range of physics processes. AtlFast3 is already in production for Run 3. Depending on particle type and energy, AtlFast3 employs the fast simulation system returning the most accurate simulation with respect to Geant4 (image below Performance: 3-15 times faster than Geant4 (depending on the process). It combines two fast simulation systems: - FastCaloSim, based on parametrisations; - FastCaloGAN, based on GANs (Generative Adversarial Networks).

The aim of the FastCaloSim package is to provide a parametrized simulation of the particle energy response and of the energy distribution in the ATLAS calorimeter and hence reduce the calorimeter simulation time to a few seconds per event. The parametrization is based on the Geant 4 simulations of single photons, electrons and charged pions in a fine grid of simulated particle energies and directions.

The fast simulation of showers in the calorimeter can be broken down into several components: the total shower energy, the distribution of energy between calorimeter layers, the average lateral development of showers within a layer, the uncorrelated energy fluctuations in individual showers compared to average showers, and, for hadronic showers, the correlated fluctuations between longitudinal and lateral energy distributions. The energy deposited in the calorimeter depends on the particle's kinetic energy (E_{kin}), which is used as the basis for parameterization unless specified otherwise. Here, E_{kin} is defined as the particle's total energy minus its mass. For antiprotons and antineutrons, the rest mass is added instead of subtracted, as their annihilation results in additional energy deposition in the calorimeter. The simulation of the total shower energy and its longitudinal distribution across calorimeter layers, including correlations, provides an approximate representation of jets, electrons, photons, and τ-leptons, though it tends to overestimate reconstruction and identification efficiencies. The simulation of the average lateral energy spread is crucial for the accurate reconstruction and identification of physics objects. To optimize computational speed, AtlFast3 employs a simplified geometry for the calorimeter cells , where each cell is assigned to a longitudinal sampling layer. The detector geometry consists of concentric cylinders with particles propagating along the z-axis. In this model, cells are represented as cuboids: in η, ϕ, and r for the barrel layers; in η, ϕ, and z for the endcap layers up to $|\eta| < 3.2$; and in x, y, and z for the forward calorimeter layers. η is the *Pseudorapidity* which is a measure of the angle of a particle's trajectory relative to the beam axis. Its invariance under Lorentz boosts and uniform coverage in solid angle make it a practical coordinate for high-energy physics experiments.

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

Where θ is the polar angle measured from the beam axis (z-axis). ϕ is the azimuthal angle that describes the angular position of a particle or energy deposit around the beam axis. It is used together with η to define the position of particles or energy deposits in the calorimeter. the angle measured in the plane perpendicular to the beam axis (z-axis). The radial distance is the distance from the beam axis (z-axis) to a point in the x-y plane. z is the longitudinal position along the beam axis.

For the case of CaloChallenge, the detector geometry is composed of concentric cylinders, with particles propagating along the z-axis. The detector is segmented along the z-axis into discrete layers. Each layer is further divided into bins along the radial direction (r), and some layers also include bins in the angular direction (α). The number of layers, as well as the number of bins in r and α, is specified in the `binning.xml` files and is read by the `HighLevelFeatures` class of helper functions. The coordinates $\Delta\phi$ and $\Delta\eta$ correspond to the x- and y-axes in cylindrical coordinates.

The image below provides a 3D view of a geometry with 3 layers, where each layer contains 3 radial bins and 6 angular bins. The right image shows the front view of the geometry, as seen along the z-axis.

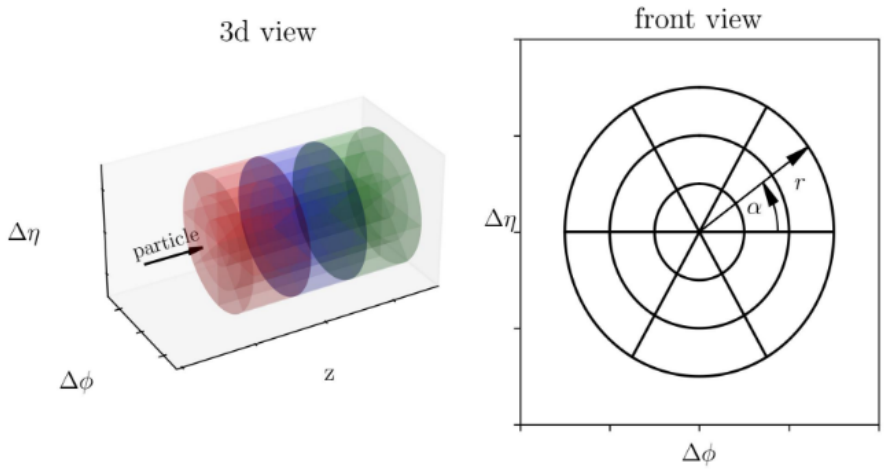


Figure 1.3. Schematic view of the voxelization in all datasets. Along the direction of the incoming particle (z), the volume is segmented in N_z layers. Each layer has N_r radial and N_α angular bins. Left: 3-dimensional view. Right: Front view.Krause et al. (2022a)

This simplification requires emulating the accordion structure of the ATLAS liquid-argon electromagnetic calorimeter. By improving the average shower energy distribution and incorporating corrections for the accordion structure, AtlFast3 achieves reconstruction and identification efficiencies comparable to those of the Geant4 simulation, particularly for electrons and photons. The spatial energy deposits in each layer in the Geant4 datasets are grouped into volumes called 'voxels' for parameterization of FastCaloSimV2 and FastCaloGAN. Only layers with a significant amount of energy, referred to as 'relevant layers', are considered in the parameterization. In FastCaloSimV2, only layers with energy fractions larger than 0.1% are used; this procedure is performed for each sample independently. In FastCaloGAN, all samples in the same η slice are processed with the same number of voxels. The relevant layers are determined using only the 1

TeV energy point and have an energy fraction larger than 0.1%. In addition, a layer with less energy is considered relevant if it is in front of a relevant layer.

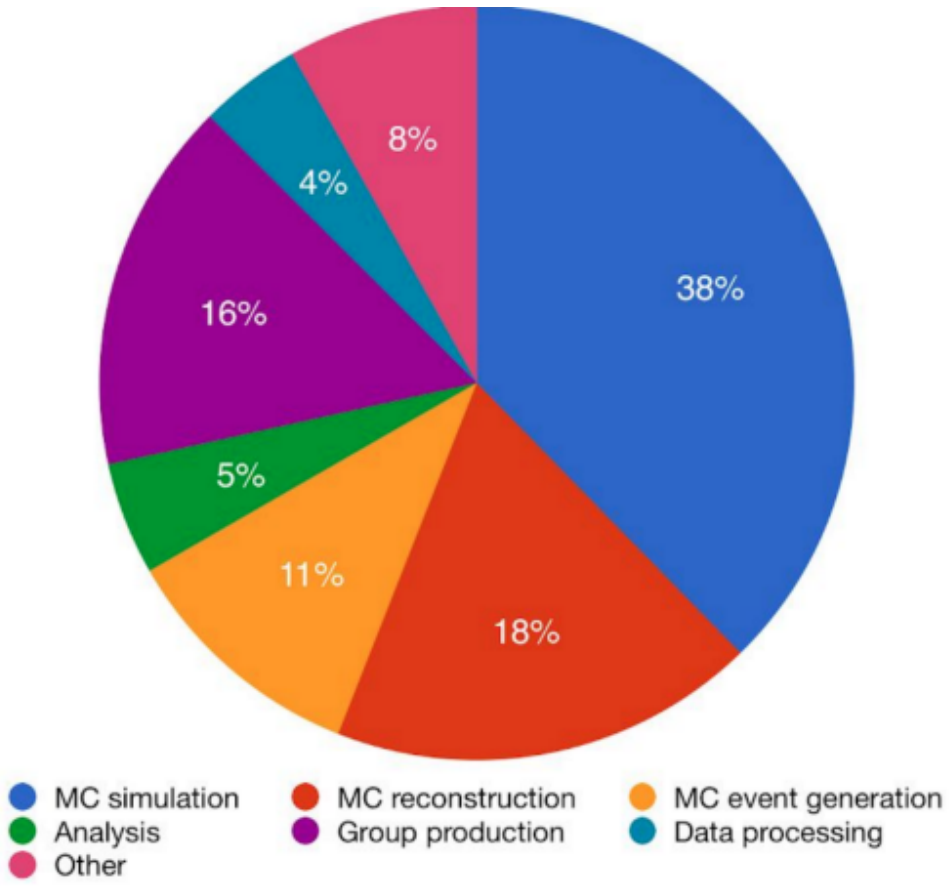


Figure 1.4. ATLAS CPU hours taken by activities ATLAS Collaboration et al. (2020)

As shown in the figure 1.5 is a properly structured schema for both the detailed (full) simulation and the fast simulation

ATLAS Computing Resources

The simulation of particle interactions within the detectors, essential for analyzing data collected during LHC Run 2, consumes approximately 40% of the computing resources allocated to the ATLAS experiment. Run 2, the second data-taking phase of the LHC, spanned from 2015 to 2018.

The intricate accordion-shaped geometry of the ATLAS electromagnetic calorimeter results in a computationally demanding simulation of shower development when employing the Geant4 toolkit (G4)

In fact, approximately 80% of the total simulation time for a typical sample of top and anti-top quark pair ($t\bar{t}$) production is dedicated to modeling the development of showers. As a result, fast simulation approaches for calorimeter systems are essential to reduce the computational demands of the ATLAS experiment and to facilitate the generation of the large numbers of simulated events required for precision physics analyses.

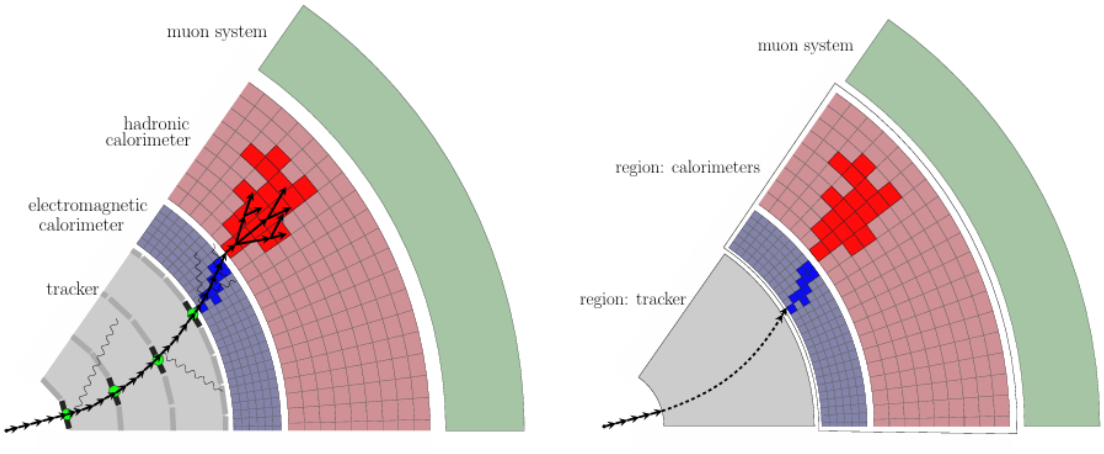


Figure 1.5. Schema of detailed full simulation (left) vs Schema of the possible configu ration of the fast simulation (right).Zaborowska (2017)

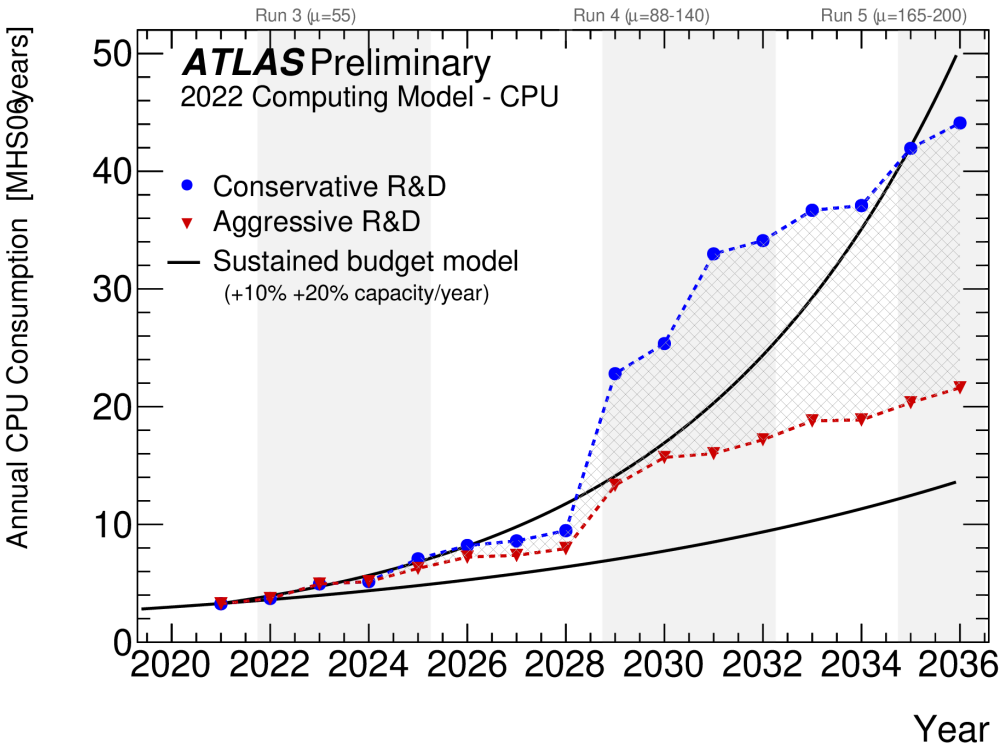


Figure 1.6. Projected evolution of compute usageATLAS Collaboration & Di Girolamo (2022)

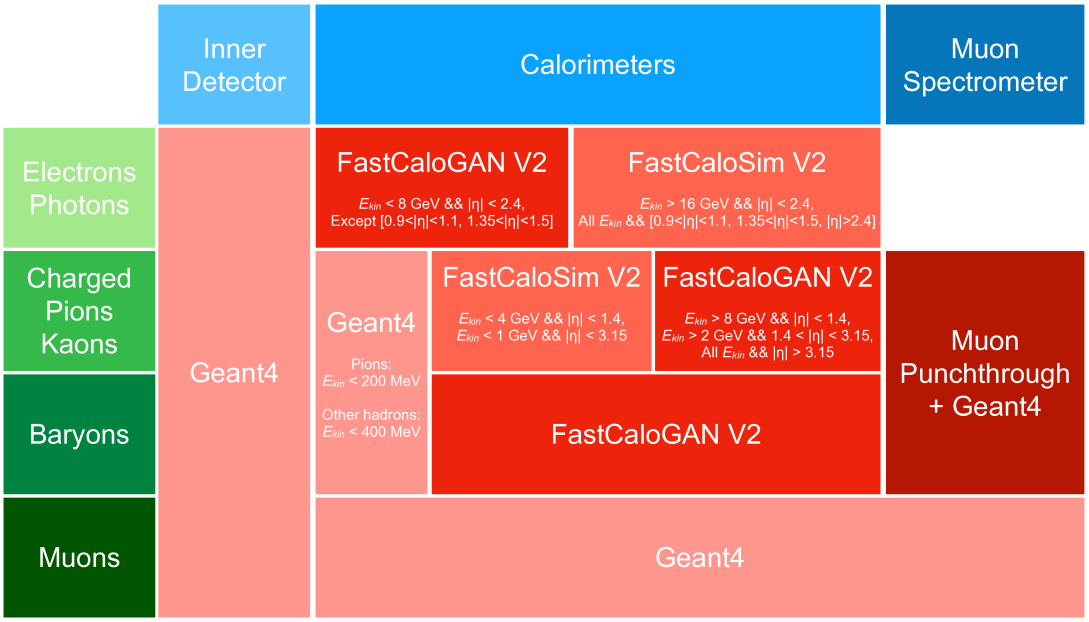


Figure 1.7. Configuration of the tools that constitute the AtlFast3 version used for Run 3 ATLAS Collaboration % (2024)

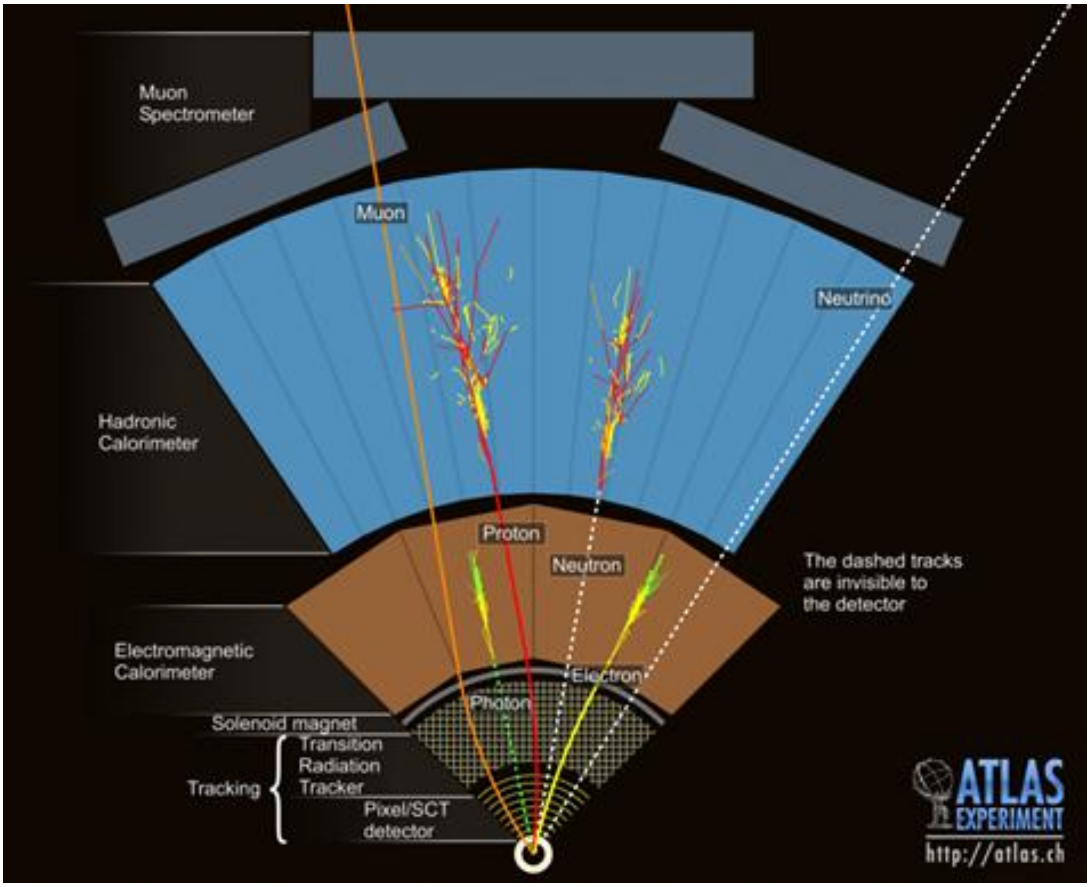


Figure 1.8. Event cross Section in a computer generated image of the ATLAS detector Pequenao (2008)

Deep Learning Solutions: Achieving accuracy

One possible approach to simulate the calorimeter response involves using deep learning techniques. Specifically, recent research Paganini et al. (2017) has demonstrated that Generative Adversarial Networks (GANs) can be effectively employed to efficiently simulate particle showers. The document Krause et al. (2022a) presents various generative models used for fast simulation in the context of the CaloChallenge 2022, a community challenge focused on fast calorimeter simulation. The generative models discussed include Generative Adversarial Networks (GANs), Normalizing Flow-based models, Diffusion-based models, Variational Autoencoders (VAEs), and Conditional Flow Matching-based models.

Generative Adversarial Networks (GANs) include models such as BoloGANKrause et al. (2022b), a derivative model of FastCaloGAN, developed at the University of Bologna.

CaloShowerGAN, an evolution of FastCaloGAN optimized for better performance with photons and pions, using multiple GANs for different energy ranges and extensive preprocessing and hyperparameter tuning.

1.4 Calochallenge

CaloChallenge 2022 was a community-driven initiative aimed at advancing the development of fast calorimeter simulation techniques using machine learning, particularly Generative Adversarial Networks (GANs) and other deep learning models. The challenge focused on improving the speed and accuracy of simulating particle showers in calorimeters, which are critical components of high-energy physics experiments like those at the Large Hadron Collider (LHC). Participants were provided with datasets generated using detailed Monte Carlo simulations of particle showers in calorimeters. The datasets included energy deposits in calorimeter cells, labeled by particle type, energy, and incident angle. Submissions were evaluated based on accuracy, speed and generalization.

The challenge provides three datasets, categorized by difficulty levels: easy, medium, and hard. The difficulty is determined by the dimensionality of the calorimeter showers, specifically the number of layers and the number of voxels within each layer. For the purpose of this thesis the dataset 1 (easy) is used with 533 voxels for pions and 368 for photons.The dataset in CaloChallenge is stored in one or more .hdf5 files, created using Python's h5py module with gzip compression. Each file contains two HDF5 datasets: *incident_energies* with the shape (num_events, 1) and *showers* with the shape (num_events, num_voxels). *incident_energies* stores the energy of the incoming particle for each event, measured in MeV, whereas *showers* stores the energy depositions in the calorimeter for each event. The energy deposits (in MeV) are flattened into a 1D array for each event. For the purpose of this thesis,

The dataset is derived from the ATLAS open datasets Faucci Giannelli et al. (2023) and includes simulations of single photons and single charged pions, which are generated at the surface of the ATLAS calorimeter system and directed towards the center of the detector. The particle interactions within the calorimeters were simulated using the official ATLAS software, which is based on Geant4. Each file containing the voxelised shower information obtained from single particles produced at the front of the calorimeter in the $|\eta|$ range (0.2-0.25) simulated in

the ATLAS detector. The information in each file is a table; the rows correspond to the events and the columns to the voxels.

In summary, the detailed energy deposits produced by ATLAS were converted from x,y,z coordinates to local cylindrical coordinates defined around the particle 3-momentum at the entrance of the calorimeter. The energy deposits in each layer were then grouped in voxels and for each voxel the energy was stored in the csv file. For each particle, there are 15 files corresponding to the 15 energy points used to train the GAN. The name of the csv file defines both the particle and the energy of the sample used to create the file. Each sample has 10,000 events at energies up to 256 GeV. For higher energies, the number of events decreases, reaching 1,000 events at 4 TeV due to the longer simulation time needed for high-energy showers. This approach balances the need for statistical robustness at low energies with computational efficiency at high energies.

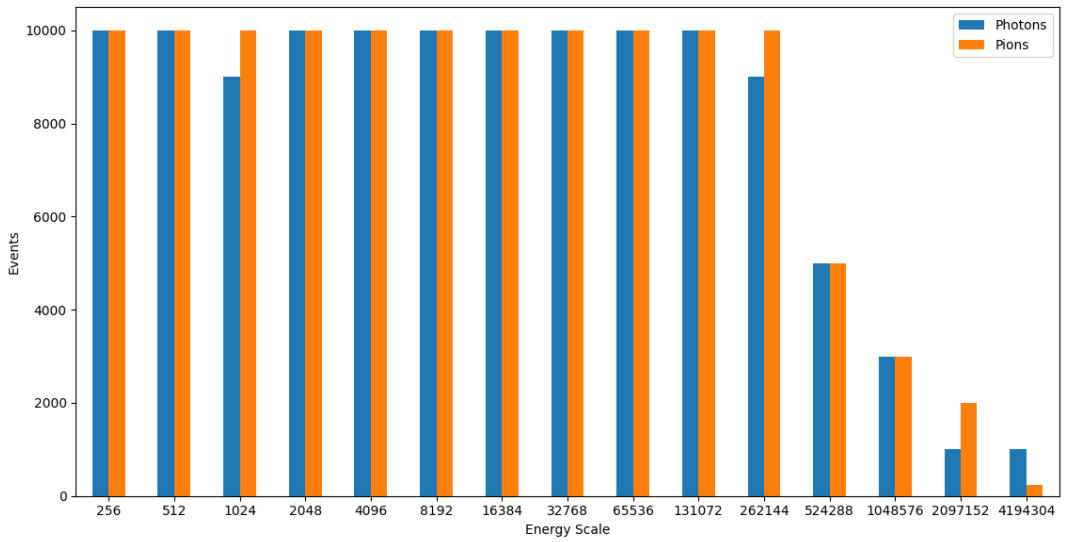


Figure 1.9. Number of events per various energy levels (in Mev) for Photons and Pions based on dataset 1 of Calochallenge 2022

1.5 BoloGAN

Model

BoloGAN Krause et al. (2022b) is a GAN-based calorimeter simulation tool, evolved from FastCaloGAN, a fast simulation framework developed within the ATLAS Collaboration at CERN. The model employs a Wasserstein GAN with gradient penalty (WGAN-GP) Goodfellow et al. (2014); Gulrajani et al. (2017) in the discriminator's loss function, ensuring stable training and high-performance results. Additionally, the model is conditioned on the kinetic energy of the incoming particle, implementing a conditional WGAN-GP architecture in TensorFlow 2.0.

Training was conducted over 1 million epochs, with TensorFlow checkpoints saved every 1000 epochs. This checkpointing strategy balances the need for monitoring training progress while avoiding excessive disk space usage. Due to the adversarial nature of GAN training, the final epoch does not necessarily yield the best model, as fluctuations during training can lead

to suboptimal performance. To select the best iteration, a χ^2 test is employed between the reference sample and the GAN-simulated sample, evaluated over the sum of the energy in all voxels—corresponding to the total energy deposited in the calorimeter by the particle. The iteration with the lowest χ^2 value is chosen for simulation. The total energy for each incident energy value serves as a natural benchmark, as it is straightforward to define yet challenging to accurately reproduce.

For each checkpoint, 10,000 events are generated per incident energy value, and the χ^2 between the reference and GAN-simulated samples is computed. The total χ^2 for a given checkpoint is obtained by summing the individual χ^2 values across all incident energy levels, and the checkpoint with the lowest total χ^2 is selected as the best-performing GAN iteration.

The current implementation of BoloGAN is capable of simulating calorimeter showers for photons, electrons, and pions, covering an energy range from 256 MeV to 4 TeV across the full detector acceptance. For the CaloChallenge, BoloGAN was applied to Dataset 1 for both photons and pions. For pion simulations, a single GAN was trained across all energy values. In contrast, for photon simulations, two separate GANs were trained: one dedicated to low-energy photons (up to 4.096 GeV) and another for high-energy photons (above 4.096 GeV). This division ensures optimal performance across different energy scales.

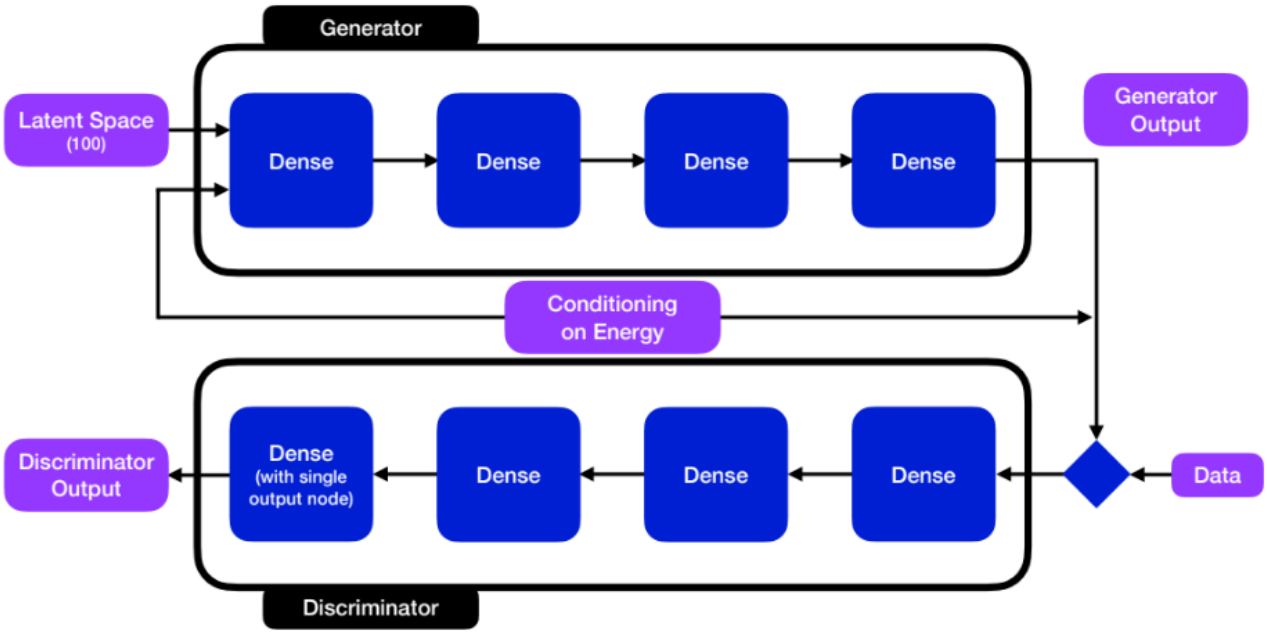


Figure 1.10. BoloGAN architecture Krause et al. (2022b)

I

First part

2

GANs

2.1 Basics of Deep learning

Deep learning is a subfield of machine learning that focuses on deep neural networks — neural networks with many hidden layers. These layers enable the network to learn hierarchical representations of data, capturing both low-level and high-level features. The Neurons *nodes* are the fundamental unit of a neural network. Each neuron performs a computation:

$$y = f(\sum_i^n x_i \omega_i + b)\,. \tag{2.1}$$

Where x_i is the input to the neuron, ω_i are the weights, Activation functions are functions, set up by the user, that introduce non-linearity to the network, enabling it to learn complex relationships. The quantity b is the bias and f is the activation function. Weights determine the importance of each input while bias shifts the activation function, helping the network learn more complex patterns. Neurons are organized into layers, forming the structure of a neural network. The *input layer* is the first layer and it is the entry point of data into the network. It does not perform any computations unlike *hidden layers*. *Hidden layers* are intermediate layers that process features. They include weights, biases, and activation functions. *Output Layer* produces the final result (e.g., class probabilities or predictions).

The *forward pass* is a fundamental step in the operation of a neural network. It refers to the process of passing input data through the network's layers to compute the output. During the forward pass, the input data is transformed layer by layer, with each layer applying weights, biases, and activation functions to produce intermediate outputs until the final output is generated. During it, input x is passed through the network to compute the output $\hat{y}$. The loss L is computed between $\hat{y}$ and the true label y.

The *backward pass* is the process of computing gradients of the loss function with respect to the model's parameters (weights and biases) and using these gradients to update the parameters. The backward pass follows the forward pass, where the model computes predictions and the loss. It starts from the output layer and moves backward through the network. The gradient of the loss is computed with respect to the outputs of each layer $\frac{\partial L}{\partial \hat{y}}$. The chain rule is then used to propagate these gradients back to the inputs of each layer, and ultimately to the parameters $\frac{\partial L}{\partial w_1}$ and $\frac{\partial L}{\partial w_2}$ (gradients for the weights in the hidden and output layers). Finally the weights are updated using the gradients (gradient descent algorithm):

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

where η is the learning rate.

This is also called optimization which refers to the process of adjusting the model's parameters (weights and biases) to minimize the loss function. This is typically done using gradient-based

optimization algorithms, which iteratively update the parameters based on the gradients of the loss function with respect to those parameters. Gradient descent is the most basic optimization algorithm. For the purpose of this thesis, we will refer to advanced optimization algorithms that improve stability and convergence. *Momentum* accelerates gradient descent by adding a fraction of the previous update to the current update. This helps overcome local minima and saddle points.

$$v_t = \gamma v_{t-1} + \eta \cdot \frac{\partial L}{\partial w}$$

$$w_{\text{new}} = w_{\text{old}} - v_t$$

where γ is the momentum term.

Root Mean Square propagation: Adapts the learning rate for each parameter based on the magnitude of recent gradients. This helps in dealing with sparse gradients.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial w}\right)^2$$

$$w_{\text{new}} = w_{\text{old}} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot \frac{\partial L}{\partial w}$$

where β is the decay rate and ϵ is a small constant for numerical stability.

Adam (Adaptive Moment Estimation): Combines the ideas of Momentum and RMSprop. It maintains both a moving average of the gradients and their squared magnitudes.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial w}\right)^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$w_{\text{new}} = w_{\text{old}} - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where β_1 and β_2 are the decay rates for the first and second moments, respectively.

As a result of training there can be problems such as *overfitting* or *underfitting*. *Overfitting* occurs when a model learns the training data too well, including its noise and outliers, resulting in poor performance on unseen data. This could happen due to a model having too many parameters that can just "memorize" the data, or when the data is insufficient or when the model is trained for too many epochs. To fix this problem, use regularization [1], data augmentation, reduction of model complexity, etc. On the other hand, *underfitting* occurs when a model is too simple to capture the underlying patterns in the training data, resulting in poor performance on both the training data and unseen data. It can happen due to too simple models, insufficient training, poor feature representation, or high bias. To address this problem, we can increase model complexity, conduct longer trainings, engineer input features via transformation or feature augmentation, reduce regularization, use a different model or tune hyperparameters.

[1] Regularization techniques add a penalty to the loss function to discourage the model from learning overly complex patterns.

Stability and convergence are key considerations when training neural networks. Stability ensures that the training process remains controlled, while convergence ensures that the model reaches an optimal or near-optimal solution. Poor stability or convergence can lead to training failures, such as exploding or vanishing gradients, or the model getting stuck in suboptimal solutions. *Batch Normalization* is such a technique used in deep learning to normalize activations in a neural network during training. It helps stabilize and accelerate training by reducing internal covariate shift.

Evaluation metrics are used to measure the performance of neural networks on specific tasks. The choice of metric depends on the type of problem (e.g., classification, regression, or object detection) and the desired outcome. For classification tasks, the goal is to predict discrete labels. Accuracy measures the proportion of correctly predicted instances out of the total instances.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Precision measures the proportion of true positives out of all positive predictions.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

F1-Score measures the harmonic mean of precision and recall.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

For regression tasks, the goal is to predict continuous values. Common metrics include: *Mean Absolute Error* measures the average absolute difference between predicted and actual values.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

MSE measures the average squared difference between predicted and actual values.

$$\text{Mean Squared Error} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

R-squared measures the proportion of variance in the target variable that is explained by the model.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

where $\bar{y}$ is the mean of the target variable.

All of this process is undertaken under the model logistics. The first concept is *pipelining* which organizes the sequence of steps required to preprocess data, train a model, and produce predictions. It helps automate and standardize the deep learning workflow. There is data preprocessing pipeline which handles tasks like normalization, augmentation, tokenization, and batching. There is then Model Training Pipeline which defines the forward pass, loss computation, backward pass, and optimization steps. Moreover, Inference Pipeline is Optimized for deploying models to production and it includes preprocessing inputs and postprocessing

outputs for real-world applications. Lastly, there is Evaluation Pipeline in which we compute metrics (e.g., accuracy, F1-score, BLEU) on validation/test datasets.

Checkpointing is a technique used during the training of neural networks to save the model's state (e.g., weights, optimizer state, and training progress) at regular intervals. This allows one to resume training from a saved checkpoint in case of interruptions, and it also enables you to select the best-performing model based on validation metrics. *Distributed* training refers to splitting the workload of training a deep learning model across multiple devices (e.g., GPUs, TPUs, or nodes). It is crucial for training large models and processing massive datasets.

There are various types of neural networks. One are *Feedforward Neural Networks* (FNNs) in which the data flows in one direction which is from input to output. They are commonly used for simple regression and classification tasks. There are also *Convolutional Neural Networks* (CNNs) specialized for processing grid-like data (e.g., images). They use convolutional layers to extract spatial features. Then there are *Recurrent Neural Networks* (RNNs) which are designed for sequential data (e.g., time series, text). They maintain a "memory" of previous inputs using loops. Then come the *transformer Networks* which are state-of-the-art for NLP tasks (e.g., GPT, BERT). They use attention mechanisms to process sequences efficiently. As used in this thesis there are *Generative Adversarial Networks* (GANs) which consist of a generator and a discriminator competing to improve each other. They are used for generating new data, like images. These however incorporate FNNs or CNNs as the generators and discriminators.

2.2 Generative Deep Learning

Generating high-quality samples from a complex distribution is one of the fundamental problems in machine learning. One approach to solve this is by using *Generative Adversarial Networks* (GANs). A GAN is a framework in which two neural networks, the generator and the discriminator, engage in a competitive process. The *generator* aims to transform random noise into data that resembles real observations, while the *discriminator* tries to distinguish between real data and the data produced by the generator. This adversarial dynamic drives both networks to improve iteratively, resulting in the generator producing increasingly realistic data. In the original GAN paper Gulrajani et al. (2017), the generator and discriminator were constructed using dense (fully connected) layers. However, subsequent researches demonstrated that convolutional layers significantly enhance the discriminator's predictive power and the generator's ability to produce high-quality outputs. This led to the development of *Deep Convolutional Generative Adversarial Networks* (DCGANs), which replaced dense layers with convolutional layers in both the generator and discriminator. Foster (2019) For this thesis we will still use the fully connected implementation in *BoloGAN*

The generator takes as input a vector, typically sampled from a multivariate standard normal distribution. This vector serves as a source of randomness, allowing the generator to produce diverse outputs. The generator then transforms this input vector into an image whose dimensions match those of the images in the original training dataset. This latent vector z contains all information needed to generate a sample, hence it can be seen as an encoding (latent representation) of the given sample. The latent values must be disseminated with a known,

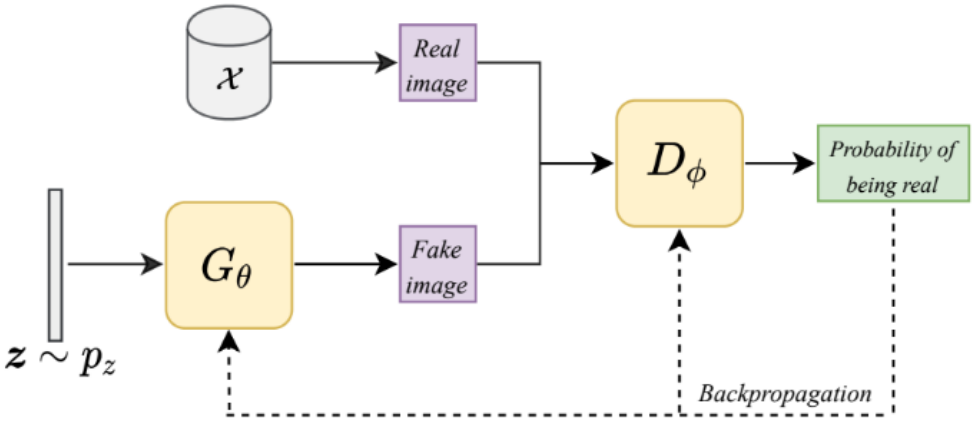


Figure 2.1. Schematic overview of GAN training process.

regular distribution in their space (the so-called prior distribution).

However, the true essence of GANs lies in their training process, which is unique and requires a deeper understanding.Training of the discriminator still makes sense from a supervised learning standpoint since we have training data that the discriminator compares the generated images with to distinguish between real and fake (generated) images. It gives 1 to real images and 0 to fake images. Training the generator is more challenging and counterintuitive because there is no explicit "correct" output for a given input (a point in the latent space). Instead, the generator's goal is to produce images that fool the discriminator, which means the discriminator should output values close to 1 for the generated images. Therefore, the generator only produces some images that it wants to fool the discriminator with. In any case, since the output of the generator is fed to the discriminator, we connect the generator to the discriminator in a single Keras model. We train the combined model. The loss function is then just the *binary cross-entropy loss* between the output from the discriminator and the response vector of 1. We train this combined model by creating training batches consisting of randomly generated 100-dimensional latent vectors as input and a response which is set to 1, since we want to train the generator to produce images that the discriminator thinks are real. Importantly, while training the weights of the discriminator are fixed so that only the weights of the generator are updated. If the weights are not frozen, there is a danger that discriminator will adjust so that it is more likely to predict generated images as real, which is not the desired outcome. It is essential that generated images to be predicted close to 1 (real) because the generator is strong, not because the discriminator is weak. After a suitable number of epochs, the discriminator and generator find an equilibrium that allows the generator to learn meaningful information from the discriminator and the quality of the images improves. Foster (2019)

2.3 Performance metrics

Evaluating the performance of Generative Adversarial Networks (GANs) is challenging because GANs generate data rather than classify or predict it. Traditional metrics like accuracy or mean

squared error (MSE) are not directly applicable. Instead, GAN performance is assessed using specialized metrics that measure the quality, diversity, and realism of the generated data. Below is a detailed explanation of the most commonly used performance metrics for GANs:

The *Inception Score* (IS) evaluates the quality and diversity of images generated by AI models, such as GANs. A pre-trained Inception v3 model (an image classification network) is used to analyze generated images. For each image, the model predicts a probability distribution over possible classes (e.g., cat, flower, car), indicating how confident the model is about the image's content. A sharp probability distribution (high confidence in one class) indicates that the image is high-quality and recognizable. The marginal distribution (aggregate of all probability distributions across the dataset) measures the variety of generated images. A uniform marginal distribution indicates high diversity (many different classes represented). The Kullback-Leibler (KL) divergence quantifies the difference between the probability distribution and the marginal distribution. A high KL divergence indicates high quality (each image is distinct and recognizable) and high diversity (the dataset contains a wide variety of classes). The IS is calculated as the exponent of the average KL divergence across all images. A higher IS indicates better performance in terms of both quality and diversity.

$$\text{IS} = \exp\left(\mathbb{E}_{x\sim p_g}\left[\text{KL}(p(y|x)\|p(y))\right]\right)$$

where $p(y|x)$ is conditional probability over classes $p(y)$ given a generated image x. $p(y)$ is the marginal probability distribution over classes y, computed by averaging $p(y|x)$ over all generated images. $KL(p(y|x)\|p(y))$ measures how different the conditional distribution $p(y|x)$ is from the marginal distribution $p(y)$. It is the *Kullback-Leibler (KL) divergence* between $p(y|x)$ and $p(y)$. $\mathbb{E}_{x\sim p_g}$ is the expectation (average) over all generated images x sampled from the generator's distribution p_g.

Fréchet Inception Distance (FID) is a metric used to evaluate the quality of AI-generated images, particularly for generative models like GANs. Unlike IS, which evaluates only generated images, FID compares generated images to real images (ground truth). This makes FID more aligned with human perception, as it directly measures how close generated images are to real ones. It uses a pre-trained Inception v3 model to extract feature representations of both real and generated images. FID models the feature distributions of real and generated images as *multivariate Gaussian distributions*. It computes the mean (μ) and covariance (Σ) for both distributions, where μ_r, Σ_r: Mean and covariance for real images and μ_g, Σ_g is the mean and covariance for generated images.FID calculates *Fréchet distance* (also called Wasserstein-2 distance) between the two Gaussian distributions using the formula

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$

where $\|\mu_r - \mu_g\|^2$ is squared Euclidean distance between the means. Tr is the trace of the matrix (sum of diagonal elements). $(\Sigma_r\Sigma_g)^{1/2}$ is the square root of the matrix product $\Sigma_r\Sigma_g$.

Lower FID indicates that the generated images are closer to real images in terms of feature distribution, implying better quality and diversity. Higher FID indicates poorer performance, with generated images less similar to real images.

2.4 Advantages of GANs and Training Issues

In the context of training Generative Adversarial Networks (GANs), the following issues are common challenges that can arise:

1. Unstable Convergence Explanation: GAN training involves a minimax optimization game between the generator (G) and the discriminator (D), which can be inherently unstable. The loss functions of G and D are adversarial, meaning their improvements are often at the expense of the other. This creates oscillations in the optimization process rather than convergence to a stable point. Symptoms: Losses for G and D fluctuate wildly or fail to stabilize. The generator produces highly inconsistent outputs across training epochs. Reasons: Non-stationary optimization dynamics: G and D update their parameters simultaneously, continuously changing the optimization landscape. Imbalanced training: If D becomes too strong, G may stop improving, and vice versa.

2. Mode Collapse Explanation: The generator learns to produce only a limited subset of the data distribution (or even just one mode), ignoring other valid modes of the distribution. This leads to a lack of diversity in the generated samples. Symptoms: The generator outputs nearly identical or repetitive samples, even for varied input noise. The GAN fails to model the full data distribution. Reasons: The generator finds a "shortcut" by exploiting weaknesses in the discriminator, repeatedly generating outputs that maximize D's loss for a specific pattern while ignoring the rest of the distribution. Solutions: Use techniques like minibatch discrimination, unrolled GANs, or mode-seeking regularization. Improve diversity by incorporating noise conditioning or feature matching objectives.

3. Two Time Scale Update Rule (TTUR) Explanation: TTUR refers to updating G and D with different learning rates or at different frequencies. This can help balance the dynamics of GAN training. Why It Matters: If D learns too quickly (due to a high learning rate or more frequent updates), it might overpower G, providing gradients that are not useful for G's improvement. If G learns too quickly, it might confuse D, preventing D from learning to distinguish real from fake samples effectively. Approach: Typically, D is updated more frequently or with a higher learning rate than G. A well-known configuration from literature (e.g., WGAN-GP) updates D five times for every G update.

4. Vanishing Gradient Explanation: When the discriminator becomes too strong, it confidently classifies fake samples as fake (or real samples as real), leading to vanishing gradients for the generator. Without meaningful gradients, G cannot learn to improve. Symptoms: The generator stops improving, producing low-quality outputs or noise. The discriminator achieves near-perfect accuracy early in training. Reasons: Poor choice of loss function, such as using the standard cross-entropy loss, which can saturate quickly. Solutions: Replace the standard loss with alternatives like Wasserstein loss or Least Squares GAN loss, which provide non-vanishing gradients. Regularize D to prevent it from becoming overly confident, e.g., through gradient penalty or label smoothing.

5. High Resource Demand Key Insights GAN training requires careful balancing of G and D's learning dynamics to avoid these pitfalls. Many solutions involve modifying the loss functions, architectures, or training schedules to encourage stability and diversity in the optimization

process.

3

Wasserstein GANs

During the training of Generative Adversarial Networks (GANs), the primary goal is to optimize the generator and discriminator in a way that the generator learns to produce realistic samples. The choice of metric for training is critical because it directly influences the stability and convergence of the model. Training Generative Adversarial Networks (GANs) requires carefully chosen metrics to ensure stability and effective optimization. The metric must be differentiable to allow gradient-based updates, as seen with the Wasserstein Distance, which provides stable gradients. Stability is crucial since GAN training is prone to issues like vanishing or exploding gradients; the Wasserstein Distance helps mitigate these problems compared to traditional loss functions. Real-time feedback is essential, as metrics like the discriminator's loss in Wasserstein GANs (WGANs) provide ongoing evaluation of the generator's performance. Ultimately, training metrics focus on distribution matching by minimizing the discrepancy between generated and real data distributions, with Wasserstein Distance serving as a direct measure of this difference.

Performance metrics are essential for evaluating the effectiveness of generative models like Conditional Wasserstein GANs with Gradient Penalty (CWGAN-GP). These metrics provide quantitative and qualitative insights into the quality, diversity, and realism of the generated samples. Common metrics include the Fréchet Inception Distance (FID), which measures the similarity between the generated and real data distributions in a feature space, and the Inception Score (IS), which evaluates both the quality and diversity of generated samples. Visual inspection is also crucial for assessing the realism of outputs, particularly in image generation tasks. Additionally, metrics like precision and recall for generative models can be used to evaluate the coverage and diversity of the generated data. In the context of fast calorimeter simulations, such as in this thesis, involving GEANT4 and CWGAN-GP, the chi-squared (χ^2) statistic is employed to compare the distributions of Geant4-simulated data and BoloGAN-generated data. Energy distribution of Geant4 is taken as the reference data in contrast with approximate (model) distribution of the generated distribution by CWGAN-GP. This metric quantifies the discrepancy between the two distributions, providing a rigorous measure of how well the generative model replicates the underlying physics. By using chi-squared, researchers can ensure that the generated data aligns closely with the ground truth provided by GEANT4, validating the model's performance in scientific applications.

3.1 Training metrics: Wasserstein Distance and Loss Function

Conventional metrics: Kullback-Leibler Divergence (Relative Entropy) and Jensen-Shannon Divergence

In deep learning and statistics, we use these metrics to train models, optimize likelihood functions, and improve generative modeling techniques like GANs and VAEs. The *Kullback-Leibler* (KL) divergence is used in information theory and variational inference in Bayesian deep learning. KL

divergence measures the loss of information when using one probability distribution $Q(x)$ to approximate another $P(x)$:

$$D_{KL}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

If P and Q are identical, $D_{KL}(P||Q) = 0$ (no difference). If $P(x)$ is nonzero where $Q(x)$ is zero, KL divergence becomes infinite (because $\log 0$ is undefined). It is not symmetric ($D_{KL}(P||Q) \neq D_{KL}(Q||P)$), meaning the direction matters.

On the other hand, *Jensen-Shannon Divergence* (JSD) is used to quantify the difference between two probability distributions. In generative modeling, JSD can measure how well the generated data distribution matches the real data distribution. It is a symmetrized version of KL divergence:

$$D_{JS}(P\|Q) = \frac{1}{2} D_{KL}(P\|M) + \frac{1}{2} D_{KL}(Q\|M)$$

where $M = \frac{P+Q}{2}$ is the average distribution.

Unlike KL divergence, JSD is always finite and bounded between $[0, \log 2]$. JSD is symmetric, meaning $D_{JS}(P||Q) = D_{JS}(Q||P)$, which is useful for comparing distributions more fairly. JSD is used in Generative Adversarial Networks (GANs) to measure how different the generator's distribution is from the real data.

Wasserstein Distance (Earth-Mover Distance, EMD)

Wasserstein distance measures the minimal cost of transforming one probability distribution into another:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

where $\Pi(P_r, P_g)$ represents all possible ways of pairing samples from P_r (real distribution) and P_g (generated distribution).

Unlike KL and JSD, *Wasserstein distance* is a true metric meaning that it satisfies symmetry and triangle inequality. It works even when distributions have disjoint support, making it more stable in GAN training. *Wasserstein distance* is used in optimal transport problems, image processing, and WGANs for improved training stability.

In conclusion, each of these measures serves a unique purpose in comparing probability distributions. KL Divergence measures how much information is lost when approximating $P(x)$ with $Q(x)$. JSD A symmetric, bounded alternative to KL divergence, used in GANs. Wasserstein distance measures the cost of transforming one distribution into another, making it more stable for training generative models.

Directly computing Wasserstein distance is computationally expensive therefore we use the *Kantorovich-Rubinstein* duality to simplify it. The duality allows us to rewrite the Wasserstein distance as a maximization problem over 1-Lipschitz functions:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq 1} \left(\mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]\right)$$

f is a *1-Lipschitz function*, meaning its slope is bounded by 1. The supremum (sup) is taken over all 1-Lipschitz functions. This duality transforms the problem into finding a function f that maximizes the difference between the expected values of f under the real and generated distributions.

In practice, the WGAN uses this duality to train discriminator to approximate the Wasserstein distance. The discriminator must be *1-Lipschitz*, meaning its output should not change too rapidly with small changes in the input. The discriminator is trained to maximize the difference between its outputs for real data and generated data:

$$Max\mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

This is done by feeding real data samples (from $\mathbb{P}_r$) and generated samples (from $\mathbb{P}_g$) into the discriminator and adjusting its weights to increase the difference between the average outputs for real and fake data. The generator is trained to minimize the Wasserstein distance, which means it tries to make the discriminator's outputs for generated data as close as possible to the outputs for real data. The generator's loss is:

$$\text{Minimize:} \quad -\mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

This encourages the generator to produce data that the critic assigns high scores to (i.e., data that looks real).To ensure the dsicriminator is *1-Lipschitz*, the WGAN uses one of two methods, *weight clipping* or *gradient penalty.* In *weight clipping* The weights of the critic are clamped to a small range (e.g., $[-0.01, 0.01]$), which limits how quickly the critic's output can change. For *gradient penalty*, penalty term is added to the discriminator's loss to encourage the gradient of the discriminator's output with respect to its input to have a norm close to 1.

Ultimately, the *Wasserstein distance* provides a smooth and continuous measure of how close the generated data is to the real data. he WGAN is less prone to issues like mode collapse and vanishing gradients. The WGAN can work with simpler architectures (e.g., MLPs) that often fail with traditional GANs.

Binary Cross Entropy Loss versus Wasserstein Loss

The *Binary Cross-Entropy* (BCE) Loss is a common loss function used in binary classification tasks, including traditional Generative Adversarial Networks (GANs). The BCE loss for a single sample is defined as Foster (2019):

$$\text{BCE Loss} = -\left(y_i \log(p_i) + (1 - y_i) \log(1 - p_i)\right)$$

For a dataset with n samples, the average BCE loss is:

$$\text{BCE Loss} = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log(p_i) + (1 - y_i) \log(1 - p_i)\right)$$

The BCE loss is used in traditional GANs. The discriminator outputs a probability p_i (between 0 and 1) that a sample is real. The loss is based on binary classification (real vs. fake).

The Wasserstein loss is used in Wasserstein GANs (WGANs). The discriminator (called the critic) outputs a scalar score $D(x_i)$ (unbounded, not a probability).

The Wasserstein loss for a batch of n samples is defined as:

$$\text{Wasserstein Loss} = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot p_i$$

where:y_i is the label for the i-th data point, ($y_i = 1$ is for real data, $y_i = -1$ is for fake data). p_i is the critic's output (scalar score) for the i-th data point. The critic outputs a scalar score p_i for each data point. For real data ($y_i = 1$), the critic aims to output a large positive score. For fake data ($y_i = -1$), the critic aims to output a large negative score.

The Wasserstein loss measures the difference between the critic's scores for real and fake data. Minimizing this loss encourages the critic to assign higher scores to real data and lower scores to fake data.

Aspect	**Binary Cross-Entropy Loss (BCE)**	**Wasserstein Loss**
Labels	$y_i = 1$ (real) $y_i = 0$ (fake)	$y_i = 1$ (real) $y_i = -1$ (fake)
Output of D	Probability $p_i \in [0, 1]$.	Scalar score $p_i \in [-\infty, \infty]$.
Activation	Sigmoid activation in the final layer.	No activation in the final layer.
Loss Function	$-\frac{1}{n}\sum_{i=1}^{n}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i))$	$\frac{1}{n}\cdot\sum_{i=1}^{n} y_i \cdot p_i$.
Gradient Behavior	Can suffer from vanishing gradients.	Provides stable gradients.
Training Stability	Prone to mode collapse and instability.	More stable training.
Lipschitz Constraint	Not required.	Required (enforced via weight clipping or gradient penalty).

3.2 Stability Issues of Wasserstein GANs (WGANs)

While *Wasserstein GANs (WGANs)* address many stability issues present in traditional GANs (e.g., mode collapse, vanishing gradients), they are not entirely free from challenges. Below is a summary of the key stability issues and their mitigation strategies. .

Critic (Discriminator) Capacity and Lipschitz Constraint The Wasserstein Distance relies on the critic being a *1-Lipschitz function*. Enforcing this constraint can lead to stability issues. Clipping weights to a small range (e.g., $[-0.01, 0.01]$) can cause optimization difficulties, vanishing gradients, or poor training dynamics. Gradient Penalty adds computational overhead and is sensitive to hyperparameters (e.g., penalty coefficient). WGANs are sensitive to hyperparameters Poor tuning can lead to overfitting, instability, or slow convergence. Mode collapse is another issue which WGAN is not immune to, it occurs if discriminator does not provide meaningful feedback. This can be mitigated by using expressive discriminatot architectures or gradient penalty. WGANs, especially WGAN-GP, are computationally expensive. Multiple discriminator updates per generator update increase training time. WGANs also have non-convergence or oscillations if the discriminator is not trained optimally. WGANs also have challenges such as sensitivity to initialization.

While WGANs improve upon traditional GANs, they face challenges related to the Lipschitz constraint, hyperparameter sensitivity, and computational cost. By employing techniques like gradient penalty, spectral normalization, and careful tuning, these stability issues can be mitigated, leading to more robust and reliable generative models.

3.3 The Lipschitz constraint: Gradient Penalty

Gradient Penalty in WGAN-GP

The *gradient penalty* is a technique used in *Wasserstein GANs with Gradient Penalty (WGAN-GP)* to enforce the *1-Lipschitz constraint* on the critic (or discriminator). This constraint is necessary for the critic to approximate the Wasserstein distance accurately. Let's break down the gradient penalty and its role in WGAN-GP: In WGANs, the critic must be *1-Lipschitz*, meaning its output should not change too rapidly with small changes in the input. Mathematically, this means:

$$|D(x_1) - D(x_2)| \leq \|x_1 - x_2\|$$

for any two inputs x_1 and x_2. This constraint ensures that the critic's gradients are bounded, which is necessary for the Wasserstein distance to be well-defined and for the training process to remain stable. In the original WGAN Goodfellow et al. (2014), the 1-Lipschitz constraint was enforced by *clipping the weights* of the critic to a small range (e.g., $[-0.01, 0.01]$). However, weight clipping can lead to undesirable behavior, such as vanishing gradients or poor approximation of the Wasserstein distance. Instead of clipping weights, WGAN-GP Gulrajani et al. (2017) introduces a *gradient penalty* to enforce the 1-Lipschitz constraint. The idea is to directly constrain the *gradient norm* of the discriminator's output with respect to its input. Specifically, the gradient norm should be at most 1 everywhere. The gradient penalty term $(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2$

encourages the critic's gradient norm to be close to 1 for random samples $\hat{x}$. By penalizing deviations from a gradient norm of 1, the critic is trained to satisfy the 1-Lipschitz constraint. To formualte mathematically, the new objective function for the critic in WGAN-GP is:

$$L = \underbrace{\mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)]}_{\text{Original critic loss}} + \underbrace{\lambda\,\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2\right]}_{\text{Gradient penalty}}$$

Where $\mathbb{P}_r$is the real data distribution,$\mathbb{P}_g$ is the generated data distribution, $D(x)$is the critic's output for input x. $\nabla_{\hat{x}}D(\hat{x})$ is the gradient of the critic's output with respect to its input $\hat{x}$. λ: A hyperparameter that controls the strength of the gradient penalty (typically $\lambda = 10$). $\mathbb{P}_{\hat{x}}$: A distribution of random samples $\hat{x}$ created by interpolating between real and generated samples:

$$\hat{x} = \epsilon x + (1-\epsilon)\tilde{x}, \quad \epsilon \sim \text{Uniform}(0,1)$$

where $x \sim \mathbb{P}_r$ and $\tilde{x} \sim \mathbb{P}_g$.

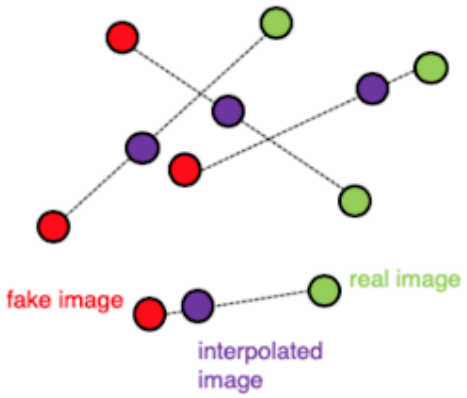


Figure 3.1. Interpolated image.

The gradient penalty is applied to samples $\hat{x}$ that lie on straight lines between real and generated samples. This ensures that the critic's gradient norm is constrained not just at real and generated points, but also in the space between them. This interpolation helps the critic generalize better and prevents it from focusing only on specific regions of the input space.

Here's how the gradient penalty is implemented in practice: Sample a batch of real data $x \sim \mathbb{P}_r$. Sample a batch of generated data $\tilde{x} \sim \mathbb{P}_g$. Create Interpolated Samples Generate random interpolation coefficients $\epsilon \sim \text{Uniform}(0,1)$. Compute interpolated samples:

$$\hat{x} = \epsilon x + (1-\epsilon)\tilde{x}$$

Compute Gradients: Compute the critic's output for the interpolated samples $D(\hat{x})$. Compute the gradient of $D(\hat{x})$ with respect to $\hat{x}$:

$$\nabla_{\hat{x}}D(\hat{x})$$

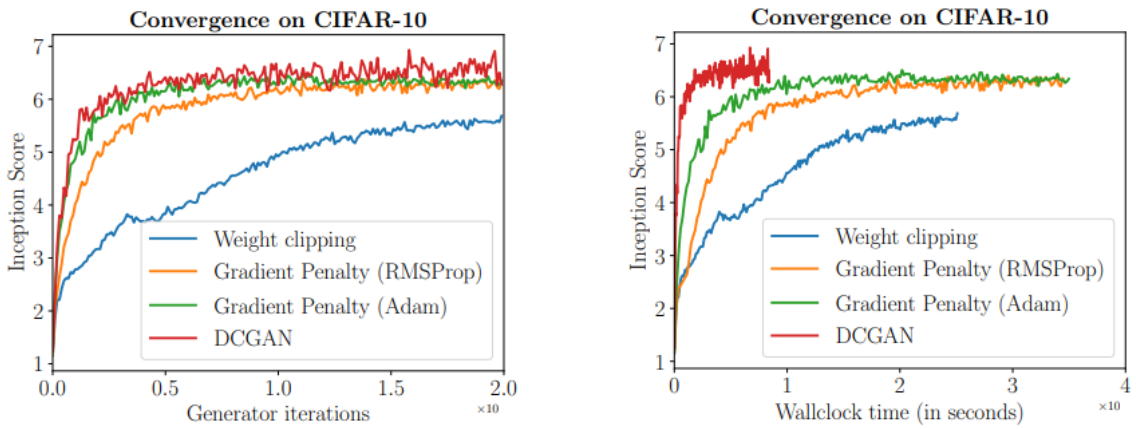


Figure 3.2. CIFAR-10 Inception score over generator iterations (left) or wall-clock time (right) for four models. WGAN-GP significantly outperforms weight clipping and performs comparably to DCGAN. Gulrajani et al. (2017)

Calculate Gradient Penalty:Compute the gradient penalty term:

$$(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2$$

Update Critic Loss: Add the gradient penalty to the original critic loss:

$$L = \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \lambda\,\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right]$$

BoloGAN Implementation Krause et al. (2022a)

```
def gradient_penalty(self, batch_size, real_samples, fake_samples, labels):
    """Calculates the gradient penalty.

    This loss is calculated on an interpolated image
    and added to the discriminator loss.
    """
    # Get the interpolated image
    alpha = tf.random.uniform([batch_size, 1], 0.0, 1.0)
    interpolated = alpha * real_samples + (1 - alpha) * fake_samples

    with tf.GradientTape() as gp_tape:
        gp_tape.watch(interpolated)
        # 1. Get the discriminator output for this interpolated image.
        pred = self.discriminator(inputs=[interpolated, labels], training=True)

    # 2. Calculate the gradients w.r.t to this interpolated image.
    grads = gp_tape.gradient(pred, [interpolated])[0]
    # 3. Calculate the norm of the gradients.
    norm = tf.sqrt(tf.reduce_sum(tf.square(grads), axis=1))
    gp = tf.reduce_mean((norm - 1.0) ** 2)
    return gp
```

Summary

The *gradient penalty* in WGAN-GP is a soft constraint that enforces the 1-Lipschitz condition on the critic by penalizing deviations of the gradient norm from 1. This approach improves the stability and performance of WGANs compared to weight clipping, making it a key component of WGAN-GP. The gradient penalty is applied to interpolated samples between real and generated data, ensuring that the critic's gradients are well-behaved across the entire input space.

3.4 Intergration of Conditionality *conditional* WGAN-GP

The ATLAS Fast Calorimeter CWGAN-GP (Conditional Wasserstein Generative Adversarial Network with Gradient Penalty) is a model designed to generate fast calorimeter shower simulations for high-energy physics experiments. Conditionality in this context is imposed on the *kinetic energy of the showers*, allowing the model to generate realistic shower patterns corresponding to specific energy levels.

Conditionality on Kinetic Energy

In high-energy physics (HEP), calorimeter showers depend on the kinetic energy of the incoming particle. The CWGAN-GP model is conditioned on kinetic energy, enabling it to generate showers consistent with the expected energy response. This conditioning is implemented using the function:

$$G(z, E_{\text{kin}}) \to X \tag{3.1}$$

where:

- G is the generator.
- z is the latent noise.
- E_{kin} is the kinetic energy.
- X is the generated calorimeter shower.

The critic $D(X, E_{\text{kin}})$ is trained to distinguish real and fake showers while considering the energy conditioning.

Conversion of Momenta to Kinetic Energy

To implement conditioning on kinetic energy, the momenta of incoming particles must be converted to kinetic energy using the relativistic energy-momentum relation:

$$E_{\text{kin}} = \sqrt{p^2 + m^2} - m \tag{3.2}$$

where:

- p is the momentum of the particle.
- m is the rest mass of the particle.
- E_{kin} is the kinetic energy.

This computation is performed using the function:

```
@staticmethod
    def momentumsToEKins(momentums, mass):
        ekins = np.sqrt(np.square(momentums) + np.square(mass)) - mass
        return ekins


@staticmethod
    def energyLabel(labelType, energy, E_min, E_max):
        if labelType == EnergyLabelDefinition.LogE:
            return math.log(energy / E_min) / math.log(E_max / E_min)
        elif labelType == EnergyLabelDefinition.MaxE:
            return energy / E_max
        elif labelType == EnergyLabelDefinition.Exp:
            return math.log(energy, 2)
```

Energy Normalization

Before training, the kinetic energy is normalized using different methods:

$$\text{Logarithmic Scaling:} \quad E' = \frac{\log(E/E_{\text{min}})}{\log(E_{\text{max}}/E_{\text{min}})} \tag{3.3}$$

$$\text{Max Normalization:} \quad E' = \frac{E}{E_{\text{max}}} \tag{3.4}$$

$$\text{Exponential Scaling:} \quad E' = \log_2(E) \tag{3.5}$$

This ensures that energy values are properly scaled before being used in the CWGAN-GP model. For this thesis, we have used the normalization of logE.

Conclusion

Conditioning the CWGAN-GP on kinetic energy enables energy-aware fast calorimeter simulation, making it an essential tool for ATLAS and other HEP experiments. This approach provides a computationally efficient alternative to full detector simulation while preserving crucial physics properties.

3.5 Evaluation metric: Reduced chi squared

After training, the focus shifts to evaluating how well the generator replicates the real data distribution. Post-training validation metrics are used to assess the quality and realism of the

generated samples. These metrics do not need to be differentiable, as they are not used for optimization, such as in the case of the Chi-Squared (χ^2) test. They often emphasize statistical significance by quantifying the similarity between the generated and real data distributions. For example, the χ^2 test measures how well the generated data matches the expected distribution, like Geant4 showers in high-energy physics. Post-training metrics should also be interpretable, providing p-values that indicate the statistical significance of the differences between distributions. They complement training metrics by offering a different perspective on the model's performance; while Wasserstein Distance ensures stable training, χ^2 provides a rigorous statistical evaluation. Additionally, validation metrics focus on the quality of the generated samples, including their realism, diversity, or adherence to physical constraints, such as matching the energy distributions and spatial patterns of real calorimeter showers in high-energy physics.

II

Second part

4

Hypothesis Testing

Hypothesis testing is a cornerstone of inferential statistics, helping make data-driven decisions across various fields like medicine, business, and science. It is a statistical method used to make decisions or draw conclusions about a population based on sample data. It helps determine whether there is enough evidence to support a specific claim or hypothesis about a population or sample data parameter.

4.1 Overview of Hypothesis Testing

Types of Hypothesis Testing and Their Metrics

Hypothesis testing is a statistical method used to make decisions about population parameters based on sample data. Below are some common types of hypothesis tests and the metrics associated with them:

Statistical hypothesis testing involves various methods depending on the data and research question. The *Z-test* is used to test the mean of a population when the population variance is known and the sample size is large ($n \geq 30$). It uses the Z-score, calculated as:

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}},$$

where $\bar{X}$ is the sample mean, μ is the population mean, σ is the population standard deviation, and n is the sample size. For smaller sample sizes ($n < 30$) with unknown population variance, the *T-test* is appropriate, using the T-score:

$$t = \frac{\bar{X} - \mu}{s/\sqrt{n}},$$

where s is the sample standard deviation. The *Chi-square test* (χ^2) is used to test the independence of categorical variables or the goodness-of-fit of observed data to expected data, with the Chi-square statistic calculated as:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i},$$

where O_i is the observed frequency and E_i is the expected frequency. For comparing the means of three or more groups, the *ANOVA (Analysis of Variance)* is used, with the F-statistic:

$$F = \frac{\text{Between-group variability}}{\text{Within-group variability}}.$$

When data is not normally distributed, non-parametric tests are employed. The *Mann-Whitney U test* compares two independent groups using the U-statistic, while the *Wilcoxon signed-rank*

test compares two related samples using the W-statistic. For three or more independent groups, the *Kruskal-Wallis test* uses the H-statistic. Finally, *Fisher's exact test* is used for small sample sizes to test the association between two categorical variables, calculating an exact p-value using the hypergeometric distribution.

4.2 Chi-squared testing: Assumptions and Methodology

The assumptions for conducting a chi-squared test include several key requirements. First, the observations must be independent, meaning each data point belongs to only one category, and there should be no overlap (e.g., a single respondent's answer in a survey cannot contribute to multiple categories). Second, the data must be categorical, such as gender, type of car, or color. Third, the expected frequency in each cell of the contingency table should ideally be at least 5; if some expected frequencies are below 5, the test may still work but with reduced reliability, and alternatives like Fisher's Exact Test are recommended for small sample sizes. Fourth, the sample data should be collected through random sampling to ensure unbiased representation. Fifth, the total sample size should be large enough to ensure reliable results, which ties into the expected frequency assumption. Finally, the observed and expected frequencies must be non-negative values.

The steps for calculating the chi-squared statistic involve several stages. First, we state the null hypothesis (H_0), which typically asserts no effect, difference, or relationship, and the alternative hypothesis (H_A), which contradicts H_0. We then choose a goodness-of-fit statistic that is sensitive to the hypothesis. From there we calculate the value of this statistic using the observed data. Then, we determine the probability distribution of the statistic under the assumption that H_0 is true, identifying regions where the statistic becomes increasingly improbable. Lastly, we compute the p-value, which represents the probability of obtaining a statistic as extreme as the observed value under H_0. If the p-value is sufficiently small (e.g., $p < 0.05$), reject H_0; otherwise, fail to reject H_0.

In the context of this thesis, the chi-squared test is used to compare energy distributions of showers generated by Geant4 (ground truth) and CWGAN-GP (a generative model). The null hypothesis (H_0) states that the energy distribution of single-particle showers (pions, photons) from CWGAN-GP is statistically indistinguishable from Geant4, while the alternative hypothesis (H_A) suggests a significant difference. To apply the test, the continuous energy values are binned into histograms, with observed frequencies (O_i) from CWGAN-GP and expected frequencies (E_i) from Geant4. The chi-squared statistic is computed as:

$$\chi^2 = \sum_{i=1}^{m} \frac{(O_i - E_i)^2}{E_i},$$

where m is the number of bins. To account for degrees of freedom, the reduced chi-squared (χ^2_{red}) is calculated as:

$$\chi^2_{\text{red}} = \frac{\chi^2}{m - k},$$

where k is the number of estimated parameters. A $\chi^2_{\text{red}} \approx 1$ indicates good agreement between

the distributions, while $\chi^2_{\text{red}} \gg 1$ or $\chi^2_{\text{red}} \ll 1$ suggests significant deviation or overfitting, respectively. The p-value, derived from the chi-squared distribution with $m - k$ degrees of freedom, determines statistical significance: if $p > 0.05$, H_0 is not rejected, but if $p < 0.05$, H_0 is rejected in favor of H_A.

BoloGAN evaluation with Geant4

In the context of high-energy physics, comparing the energy distributions of showers generated by *Geant4* (considered the ground truth) and *CWGAN-GP* (a generative model) involves *hypothesis testing* to determine whether the generated data accurately represents the real data. The *reduced chi-squared* (χ^2_{red}) test is commonly used for this purpose.

By applying the reduced chi-squared test, we quantitatively assess how well CWGAN-GP replicates the Geant4 energy distribution. This methodology ensures a rigorous statistical evaluation, guiding improvements in generative model training for accurate physics simulations.

```
def chi2test(hist1, hist2) -> Tuple[float, int]:
    """Perform a chi2 test between two histograms.

    Internally invokes the ‘TH1::Chi2Test‘ method. The test is performed
    bin-by-bin, comparing the contents of each bin in the two histograms.
    The histograms are considered as weighted histograms.

    Args:
        hist1 (TH1): First histogram.
        hist2 (TH1): Second histogram.

    Returns:
        Tuple[float, int]: The chi2 value and the number of degrees of freedom.
    """

    # Using the C convention, we're passing return values by reference
    chi2 = ctypes.c_double(0.0)
    ndf = ctypes.c_int(0)
    igood = ctypes.c_int(0)

    hist1.Chi2TestX(
        hist2,
        chi2,
        ndf,
        igood,
        "WW"
    )

    # igood=0 - no problems
    # For weighted weighted comparison
    #     igood=1'There is a bin in the 1st histogram with less then 10 effective number of
```

```
        events'
#     igood=2'There is a bin in the 2nd histogram with less then 10 effective number of
        events'
#     igood=3'when the conditions for igood=1 and igood=2 are satisfied'
if igood.value != 0:
    if igood.value == 1:
        msg = "There is a bin in the 1st histogram with less than 10 effective number of
            events."
    elif igood.value == 2:
        msg = "There is a bin in the 2nd histogram with less than 10 effective number of
            events."
    elif igood.value == 3:
        msg = "There is a bin in the 1st and 2nd histograms with less than 10 effective
            number of events."

    warnings.warn("chi2test: " + msg)

return chi2.value, ndf.value
```

```
histos_gan = []

    total_chi2 = 0
    total_ndf = 0

    for index, energy in enumerate(self.energies):

        # Create the label to condition the GAN
        labels = Label(
            nevents=NEVENTS,
            energy=energy,
            E_min=self.energy_min,
            E_max=self.energy_max,
            eta_min=self.eta_min,
            energyLabelType=self.label_definition,
        )

        # convert to a numpy array and then to a tensor
        labels = labels.GetLabelsAndPhiMod()

        # Generate data with the GAN
        data = self.model(labels)

        # Normalize the GAN output data
        if self._data_params.voxel_normalisation == VoxelNormalisation.normE:
            data = data * energy # needed for conditional
        elif self._data_params.voxel_normalisation == VoxelNormalisation.MaxVoxelAll:
            data = data * self.max_voxel
        elif self._data_params.voxel_normalisation == VoxelNormalisation.MaxVoxelMid:
```

```
            data = data * self.max_voxel
        elif self._data_params.voxel_normalisation == VoxelNormalisation.midE:
            data = data * self.mid_energy

        # Get the original histogram
        h_vox = self._histos_vox[index]

        # Create a new histogram with GAN generated data
        # https://root.cern.ch/doc/master/classTH1F.html#a958c4845a7cc935bae03b5578c4267c0
        h_gan = ROOT.TH1F(
            "h_gan", # name
            "", # title
            30, # nbinsx
            h_vox.GetXaxis().GetXmin(), # xlow
            h_vox.GetXaxis().GetXmax(), # xup
        )

        E_tot = data.numpy().sum(axis=1)
        for e in E_tot:
            h_gan.Fill(e / 1000)

        h_gan.Scale(1 / h_gan.GetEntries())
        h_gan.SetLineColor(ROOT.kRed)
        h_gan.SetLineStyle(7) # Dashed line

        range_max = [
            h_vox.GetBinContent(h_vox.GetMaximumBin()),
            h_gan.GetBinContent(h_gan.GetMaximumBin()),
        ]
        range_max = max(range_max) * 1.25

        h_vox.GetYaxis().SetRangeUser(0, range_max)
        histos_gan.append(h_gan)
        h_vox.GetYaxis().SetTitle("Entries")

        xaxis_title = "Energy [GeV]"
        h_vox.GetXaxis().SetTitle(xaxis_title)
        h_vox.GetXaxis().SetNdivisions(506)

        # Chi2 test over the two histograms. Returns chi2 and ndf (number of degrees of
            freedom)
        chi2, ndf = chi2test(h_vox, h_gan)

        total_chi2 += chi2
        total_ndf += ndf

        # Add the two histogram to the canvas
        canvas.cd(index + 1) # next graph
        self._histos_vox[index].Draw("HIST") # draw the original histogram
        histos_gan[index].Draw("HIST same") # draw the GAN histogram

        # Legend box
```

```
        if energy > 1024:
            energy_legend = f"{round(energy / 1000, 1)} GeV"
        else:
            energy_legend = f"{round(energy, 1)} MeV"

        t = ROOT.TLatex()
        t.SetNDC()
        t.SetTextFont(42)
        t.SetTextSize(0.1)
        t.DrawLatex(0.2, 0.83, energy_legend)

        t.DrawLatex(0.2, 0.73, "#chi^{2}/NDF = %.1f" % (chi2 / ndf))

    chi2_o_ndf = total_chi2 / total_ndf
```

6

4.3 Data Study

The challenge Krause et al. (2022a)provides three datasets, categorized by difficulty levels: easy, medium, and hard. The difficulty is determined by the dimensionality of the calorimeter showers, specifically the number of layers and the number of voxels within each layer. For the purpose of this thesis we work with "easy" dataset 1 as provided in Faucci Giannelli et al. (2023).

All datasets follow the same general structure. The detector geometry consists of concentric cylinders, with particles propagating along the z-axis. The detector is segmented along this axis into N_z discrete layers. Each layer is further divided into N_r bins in the radial direction and N_α bins in the angular direction α. The number of layers, as well as the bin counts in r and α, are summarized in Table 4.1.

Layer Number	0	1	2	3	...	12	13	14	...	44	total
ds $1-\gamma$	8×1 $=8$	16×10 $=160$	19×10 $=190$	5×1 $=5$	–	5×1 $=5$	–	–	–	–	368
ds $1-\pi^+$	8×1 $=8$	10×10 $=100$	10×10 $=100$	5×1 $=5$	–	15×10 $=150$	16×10 $=160$	10×1 $=10$	–	–	533
ds 2	$9\times16=144$										6480
ds 3	$18\times50=900$										40 500

Figure 4.1. Voxelization of layers in each dataset. We show Nr $\times$ Nα and the total number of voxels, Ni, per layer. For datasets 1: a "–" indicates that this layer is not in the dataset, as the numbering is based on the ATLAS detector definitions. ATLAS Collaboration et al. (2022) Krause et al. (2022a)

The coordinates $\Delta\phi$ and $\Delta\eta$ correspond to the x and y axes in cylindrical coordinates. Figure 4.2 provides a three-dimensional visualization of a geometry consisting of three layers, where each layer contains three radial bins and six angular bins. The right panel of the figure presents a front view of the geometry as observed along the z-axis.

The dataset is derived from the ATLAS open datasetsFaucci Giannelli et al. (2023) and includes simulations of single photons and single charged pions, which are generated at the surface of the ATLAS calorimeter system and directed towards the center of the detector. The particle interactions within the calorimeters were simulated using the official ATLAS software, which is based on Geant4.

A specialized configuration was employed, in which detailed hits were recorded while effects such as electronic noise and cross-talk were excluded. This approach enables the modeling of idealized showers that can be injected into the simulation chain before these effects are introduced, thereby enhancing realism. These simulated samples were utilized for training the GANs presented in the AtlFast3 paper ATLAS Collaboration et al. (2022) and the FastCaloGAN ATLAS Collaboration (2020)

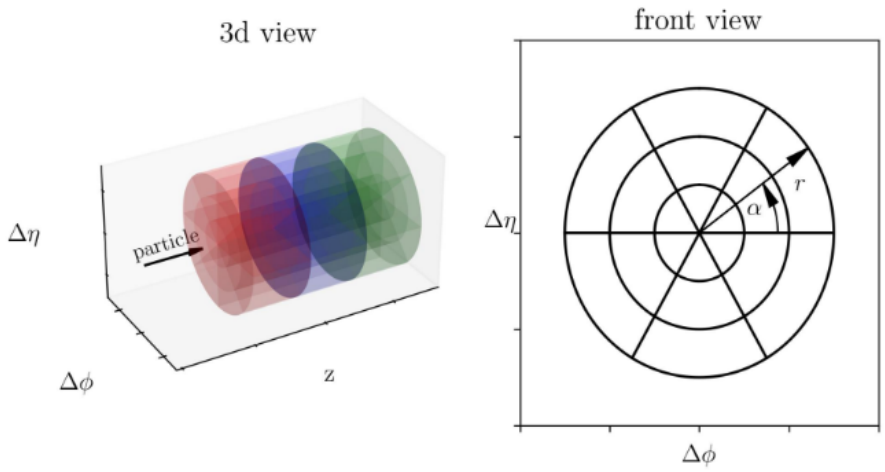


Figure 4.2. Schematic view of the voxelization in all datasets. Along the direction of the incoming particle (z), the volume is segmented in N_z layers. Each layer has N_r radial and N_α angular bins. Left: 3-dimensional view. Right: Front view.Krause et al. (2022a)

In the results section, a 1 MeV threshold is applied to all voxels to remove the low-energy tail that affects certain models but has no physical impact. This assessment is based on the transformation and calibration of energy deposited in the calorimeter into reconstructed objects (such as photons or jets), which are built from clusters of calorimeter cells ATLAS Collaboration et al. (2008).

The ATLAS calorimeters are segmented into cells to enhance granularity and improve the spatial reconstruction of showers, and this segmentation is faithfully reproduced in the simulation. The calorimeter cells have a rectangular shape for ease of construction, which differs from the cylindrical voxel geometry described earlier. Consequently, an additional step in the AtlFast3 simulation was required to reassign energy from the voxels to the actual calorimeter cells.

In offline reconstruction, ATLAS employs topological clusters, which are seeded from cells with an energy of at least four times the noise level. These clusters then grow by incorporating neighboring cells with energy above twice the noise level and are finalized by including adjacent cells that exceed the noise threshold. The minimum noise level in the considered dataset layers is approximately 10 MeV for layer 1, while other layers can have noise levels up to 50 MeV.

Thus, applying a 1 MeV energy threshold to voxels is justified, even when accounting for the fact that multiple voxels may map to the same cell. This scenario occurs predominantly in the core of the shower, where the majority of the energy is deposited, meaning the threshold does not eliminate significant energy contributions. All masked voxels correspond to peripheral regions, where they map to multiple cells, further diluting the associated energy per cell.

Each file containing the voxelised shower information obtained from single particles produced at the front of the calorimeter in the $0.2 < \eta < 0.25$ range simulated in the ATLAS detector.

The information in each file is a table; the rows correspond to the events and the columns to the voxels.

In summary, the detailed energy deposits produced by ATLAS were converted from x,y,z coordinates to local cylindrical coordinates defined around the particle 3-momentum at the entrance of the calorimeter. The energy deposits in each layer were then grouped in voxels and for each voxel the energy was stored in the csv file. For each particle, there are 15 files corresponding to the 15 energy points used to train the GAN. The name of the csv file defines both the particle and the energy of the sample used to create the file.

The size of the voxels is described in the binning.xml file.

Each sample has 10,000 events at energies up to 256 GeV. For higher energies, the number of events decreases, reaching 1,000 events at 4 TeV due to the longer simulation time needed for high-energy showers. This approach balances the need for statistical robustness at low energies with computational efficiency at high energies

1. *Energy Deposits*

    - e_0 to e_{23}: Energy deposited in 24 calorimeter cells or layers.
    - e_{tot}: Total energy deposited in the calorimeter for the event.

2. *Delta Eta and Delta Phi*

    - $\Delta\eta_1$ to $\Delta\eta_{22}$: Pseudorapidity differences between pairs of particles or calorimeter cells.
    - $\Delta\phi_1$ to $\Delta\phi_{22}$: Azimuthal angle differences between pairs of particles or calorimeter cells.

Momenta (MeV)	Events Photons	Voxels	Events Pions	Voxels
256	80000	368	100000	533
512	90000	368	100000	533
1024	99900	368	100000	533
2048	100000	368	100000	533
4096	100000	368	100000	533
8192	100000	368	100000	533
16384	90000	368	99900	533
32768	89910	368	100000	533
65536	80000	368	49900	533
131072	69860	368	29700	533
262144	89548	368	9800	533
524288	41165	368	9850	533
1048576	25660	368	9700	533
2097152	18649	368	9480	533
4194304	9874	368	9575	533

Table 4.1. Breakdown of data files CSV

Category	Branches
Energy Deposits	• e_0, e_1, e_2...,e_23, e_tot, e_totNorm
Pseudorapidity differences	• Delta_eta_1,Delta_eta_2, Delta_eta_5...,Delta_eta_13, Delta_eta_21, Delta_eta_22
Azimuthal angle differences	• Delta_phi_1,Delta_phi_2, Delta_phi_5...,Delta_phi_13, Delta_phi_21, Delta_phi_22
Deposit Centroids (1048576 MeV)	• cog_phi_1,cog_phi_2, cog_phi_5...,cog_phi_13, cog_phi_21, cog_phi_22 • cog_eta_1,cog_eta_2, cog_eta_5...,cog_eta_13, cog_eta_21, cog_eta_22
Shower Width (1048576 MeV)	• width_phi_1,width_phi_2, width_phi_5...,width_phi_13, width_phi_21, width_phi_22 • width_eta_1,width_eta_2, width_eta_5...,width_eta_13, width_eta_21, width_eta_22

Table 4.2. Structure of data files ROOT

Momenta (MeV)	**Events Photons**	**Events Pions**
256	10000	10000
512	10000	10000
1024	9000	10000
2048	10000	10000
4096	10000	10000
8192	10000	10000
16384	10000	10000
32768	10000	10000
65536	10000	10000
131072	10000	10000
262144	9000	10000
524288	5000	5000
1048576	3000	3000
2097152	1000	2000
4194304	1000	230

Table 4.3. Breakdown of data files ROOT

Element	Description
`<Bins>`	Root element, defines binning strategy
`isPolar="true"`	Uses polar coordinates for binning
`optimisedAlphaBins="false"`	Alpha bin optimization is disabled
`mergeAlphaBinsInFirstRBin="false"`	No merging of alpha bins in the first radial bin
`<Particle>`	Defines particles and their ML training parameters
`name="photon"`	Defines photon binning settings
`pid="22"`	Particle ID for photon
`latentDim="100"`	Latent space dimension (GAN)
`learningRate="0.0001"`	Training learning rate
`batchSize="1024"`	Number of events per batch
`label_definition="LogE"`	Defines the energy scale used
`<EnergyRanges>`	Defines energy binning categories
`Low12`	Uses ReLU activation, batch normalization enabled
`High12`	Uses Swish activation, different generator values
`<Bin>`	Defines bins for different η (pseudorapidity) ranges
`regionId="1"`	Bin covering $0 \leq \eta \leq 130$
`regionId="2"`	Bin covering $130 \leq \eta \leq 135$
`regionId="3"`	Bin covering $135 \leq \eta \leq 140$
`<Layer>`	Defines radial (r) and angular (α) bins per layer
`id="0"`	r-edges: (0, 5, 10, 30, ..., 600), α-bins = 1
`id="1"`	More refined binning, α-bins = 10
`id="2"`	Further segmentation, α-bins = 10
`<Particle name="pion">`	Defines pion-specific binning (pid=211)
`generator="200,400,800"`	GAN generator architecture
`batchSize="512"`	Smaller batch size than photons
`<Particle name="electron">`	Defines electron-specific binning (pid=11)
`label_definition="MaxE"`	Uses max energy instead of log energy

Table 4.4. Summary of XML binning file structure

`<Layer>`	Defines radial (r) and angular (α) bins per layer
Example Bin: regionId="1" ($0 \leq \eta \leq 80$)	
`id="0"`	r-edges: (0, 5, 10, 30, 50, 100, 200, 400, 600), α-bins = 1 (coarse angular binning)
`id="1"`	r-edges: (0, 1, 4, 7, 10, 15, 30, 50, 90, 150, 200), α-bins = 10
`id="2"`	r-edges: (0, 5, 10, 20, 30, 50, 80, 130, 200, 300, 400), α-bins = 10
`id="3"`	r-edges: (0, 50, 100, 200, 400, 600), α-bins = 1
`id="12"`	r-edges: (0, 10, 20, 30, 50, 80, 100, 130, 160, 200, 250, 300, 350, 400, 1000, 2000), α-bins = 10
Example Bin: regionId="2" ($80 \leq \eta \leq 110$)	
`id="0"`	r-edges: (0, 5, 10, 30, 50, 100, 200, 400, 600), α-bins = 1
`id="1"`	r-edges: (0, 1, 4, 7, 10, 15, 30, 50, 90, 150, 200), α-bins = 32 (finer angular binning)
`id="2"`	r-edges: (0, 5, 10, 20, 30, 50, 80, 130, 200, 300, 400), α-bins = 32
`id="17"`	r-edges: (0, 50, 100, 150, 200, 250, 300, 400, 600, 1000, 2000), α-bins = 32
Key Observations About Pion Binning	
Adaptive Binning	
	• Coarse binning ($n\_bin\_alpha = 1$) is used in deeper layers.
	• Fine binning ($n\_bin\_alpha = 32$) is used in some η regions for detailed shape resolution.
Radial Binning	
	• Some layers have many fine-grained bins, while others use only a few bins (e.g., 0, 50, 100, 200, 400, 600).
	• The presence of 1000, 2000 suggests inclusion of very high-energy deposits.

Table 4.5. Summary of XML binning file structure and pion-specific binning details

5

Hyperparameter Optimization

5.1 Literature Review and Current BoloGAN setup

Since BoloGAN is a model derived from FastCaloGAN literature study for the purpose of hyperparameter tuning has been focused on studies of BoloGAN as well as FastCaloGAN. The standard hyperparameters setups are reported for FastCaloGAN in Table 5.1 and 5.2 for BoloGAN. For FastCaloGAN, the generator is a neural network composed of five layers: one input layer, three hidden layers, and one output layer. In FastCaloGAN V1, the input layer included three nodes representing information about the incoming pion and 50 nodes constituting the latent space, whose values were randomly sampled. The values of the latent space nodes were independently sampled from a uniform distribution over the range $[-1, 1]$. Progressing forward through the generator, the number of nodes in the hidden layers was set to 50, 100, and 200, respectively. Like the generator, the discriminator is a neural network comprising three hidden layers. In the initial version of FastCaloGAN, the number of nodes in each hidden layer matched the number of voxels. However, it was found that reducing the number of nodes to 800, 400, and 200 in the three hidden layers, respectively, as the network progresses forward through the discriminator, led to improved performance. ATLAS Collaboration (2020)

For BoloGAN, the conditional WGAN-GP is implemented in TensorFlow 2.0. The model generator consists of neural network with 5 layers: one input layer, three hidden layers and one output layer (corresponding to the number of voxels). For Pion showers the voxels are 533, while for Photons there are 368 voxels. For Pions the optimized values of from the FastCaloGAN studies are used with the discriminator having 800, 400, 200, 1 architecture. While for photons the discriminator architecture is kept the same as FastCaloV1 i.e. number of nodes in hidden layers match the number of voxels. The values used at the start of this project are reported in Table 5.2. The discriminator has 829601 trainable parameters requiring a storage of 6.4MB while the generator has 848733 trainable parameters.

Hyperparameter	**Values**
Latent Space	50
Generator Nodes	50, 100, 200, NVoxel (pid and η dependent)
Discriminator Nodes	NVoxel, NVoxel, NVoxel, NVoxel, 1
Activation Function	ReLU (all layers)
Optimizer	Adam
Learning Rate	10^{-4}
Beta	0.5
Training Ratio (D/G)	5
Batch Size	128
Gradient Penalty λ	10

Table 5.1. Hyperparameters of FastCaloGAN ATLAS Collaboration (2020)

Parameter	Pions	Low Energy Photons	High Energy Photons
Latent Space	100	100	100
Generator Nodes	200,400,800, 533*	100,200,400,368*	50,100200,368*
Discriminator Nodes	800,400,200,1	368,368,368,1	368,368,368,1
Activation Function	ReLU	ReLU	Swish
Optimizer	Adam	Adam	Adam
Learning rate	10^{-4}	10^{-4}	10^{-4}
n_critic	5	8	8
Beta	0.5	0.5	0.5
Lambda	10	3	3
Batch Size	512	1024	1024
Used Batch Normalization	No	Yes	Yes

Table 5.2. BoloGAN WGAN-GP hyperparameters. Low (high) energy photons are those up to (above) 4.096 GeV. Values marked with * are equal to the number of voxels in the corresponding case.Krause et al. (2022a)

5.2 Hyperparameter Summary

Generator Width and Depth

Increasing the width of the feedforward neural network of the Generator contained in the GAN increases the neurons per layer and thus also increases parameters of the model. This increased number of parameter is particularly effective to reduce training times due to increased parallel processing of features from the input layer. The increased neuron count also mitigates risk of perturbations in the input and improves robustness of the network. The increased neurons would also allow for broader range of features to be captured and then generated (i.e. wider array of shower widths, shapes etc) allowing the model to achieve generalizability. Similarly, increasing the depth of the neural network increases the capacity to create outputs with both fine-grained details and coherent global structures. The early layers focus on low-level features and deeper layers on high-level features with finer details. Additionally pre-trained deep neural networks can be used for transfer learning. The potential issues with wider and deeper neural networks are vanishing gradients, overfitting, computational demands, diminishing returns, curse of dimensionality etc.

Batch Size

Increasing the number of batch size increases samples processed together in one forward/backward pass. Smaller batches lead to more frequent updates, speeding up initial learning but requiring more iterations to converge. Larger batches stabilize updates but may slow down overall training. Smaller batches require less memory, making them ideal for large datasets or complex models with limited computational resources. Smaller batches can improve generalization, while larger batches might lead to quicker convergence but increase the risk of overfitting. Larger batch sizes generally allow models to complete each epoch (iteration) faster due to increased parallel processing capabilities. However, excessively large batches may degrade model quality and hinder generalization on unseen data.

Learning rate

The generator learning rate is a key hyperparameter in CWGAN-GP that controls how quickly

the generator adapts to the critic's feedback. It plays a crucial role in ensuring stable training, preventing mode collapse, and achieving high-quality sample generation. If the generator learning rate is too high, the generator may overreact to the critic's feedback, leading to unstable training or mode collapse. In this scenario the generator produces limited varieties of samples (e.g., the same image repeatedly). If it is too low, the generator may learn too slowly, resulting in poor-quality samples.

Latent Space

The main objective of GAN is to generate realistic data from a random vector, which is in a much lower dimension, sampled from a prior distribution which is normal distribution in our case. This is called a "feature vector." This vector is considered to encode and condense significant characteristics of the generated sample from the generator, while the task of the GAN is to generate an image from that information.

It is common to presume that a latent space in our example encodes features of the generated single particle shower (shower width, shower shape, shower momenta, etc). Interpolation between two images can show that latent space encodes meaningful semantics. We also expect high correlation between latent variable and conditional label variable.

Training GAN is a tremendously challenging task, suffering from 4 main difficulties: non-convergence and instability, mode collapse, unstable generator gradient, and highly sensitive to hyper-parameter tuning. It has been attempted to show by studies that latent space encodes meaningful semantics by using vector arithmetic. Studies have investigated transition between two images by sampling a series of latent vectors which lies in a linear path connect two original random points from a prior distribution. However, this is an open research problem. In practice, an important point should be cleared out that the latent space is structured by the generator, it itself has no meaning.

5.3 Methodology

For the purpose of this experiment, a standard random search hyperparameter tuning technique for the single particle shower pion is implemented. After an initial experimentation on various generator architectures, the more promising architectures are chosen and the parameters of batch size, D/G and learning rate are tuned individually, ultimately combining the effects of multiple parameters. With the top performing architectures the batch sizes of 128, 512 and 1024 are studied. D/G ratio results for the numbers of 3, 5 and 8 are reported as well. Learning rates of 0.0001 and 0.00005 are experimented. At the end, the combined effect of multiple parameters is experimented and reported.

The binning.xml file is used to store and organize hyperparameters for the cWGAN-GP model. These values are passed from the binning.xml file and runs are carried out for each model with a given set of hyperparameter values, to minimize random fluctuations. The average reduced chi-squared value (for all energy levels) for a model is compared among all models.

Training is carried out on the NVIDIA H100/A100/V100 GPUs available on the CERN HTCondor system LXBATCH. The values of best reduced chi-squares are studied from the output files on the LXBATCH system. For the best performing models the Tensorboard visualizations

for all energy levels are reported. Finally, the best performing hyperparameters are used to train the model to demonstrate improved performance.

III

Third part

6 Results

6.1 General Checking: Setup and evaluation on Open Physics Hub cluster

Before starting hyperparameter tuning on LXPLUS, I conducted training on OPH cluster at University of Bologna. Due to shortage of available GPU resources, training was conducted using CPUs at OPH and short trainings of about 50 epochs were conducted using *BoloGANtainer*, a singularity container with all details to install essential epel repository, install dependencies and create virtual environment. Using *Tensorboard*, I carried out preliminary tutorial training of χ^2_{red} evaluation on OPH.

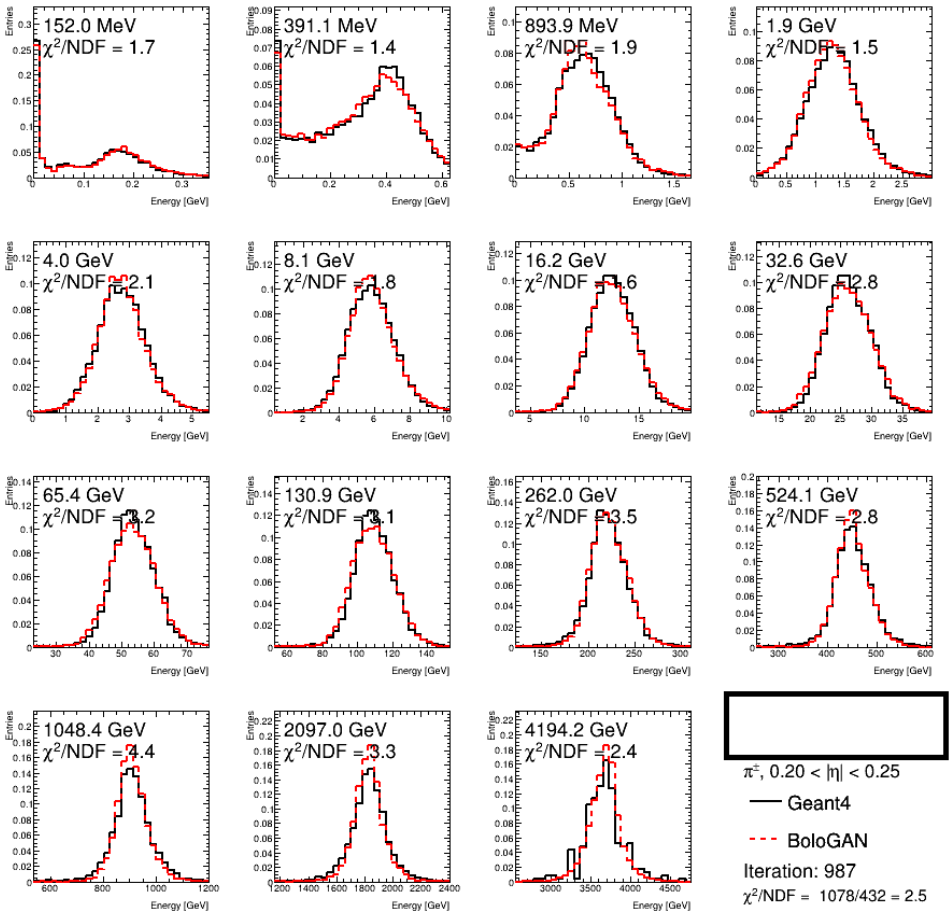
6.2 χ^2_{red} Evaluation on LXPLUS

To display the χ^2_{red} evaluation between *Geant4* and *BoloGAN* we utilize the *Tensorboard* dashboard. *TensorBoard* is a visualization tool provided by *TensorFlow* that allows us to track and visualize various metrics, models, and data during the training and evaluation of machine learning models. It provides a dashboard with several components to help monitor and debug models effectively. For the purpose of this thesis we have the plots of χ^2_{red} for all 15 energy levels for pion single particle shower. The 15 graphs are all plotted as subplots of a larger figure. The plots are normalized energy distributions of particle hits. Energy values in GeV are on x-axis and their corresponding hits as plotted on the y-axis giving the distribution of energy in the calorimeter for a single particle pion shower of a particular energy. *Geant4* data for the same distribution is plotted in black color while that generated from *BoloGAN* is plotted in red. χ^2_{red} is computed for all 15 energy values and the average of all 15 χ^2_{red} values is reported on the bottom-right of the larger figure. This value is used for hyperparameter tuning in conjunction with other considerations such as computational cost and training time.

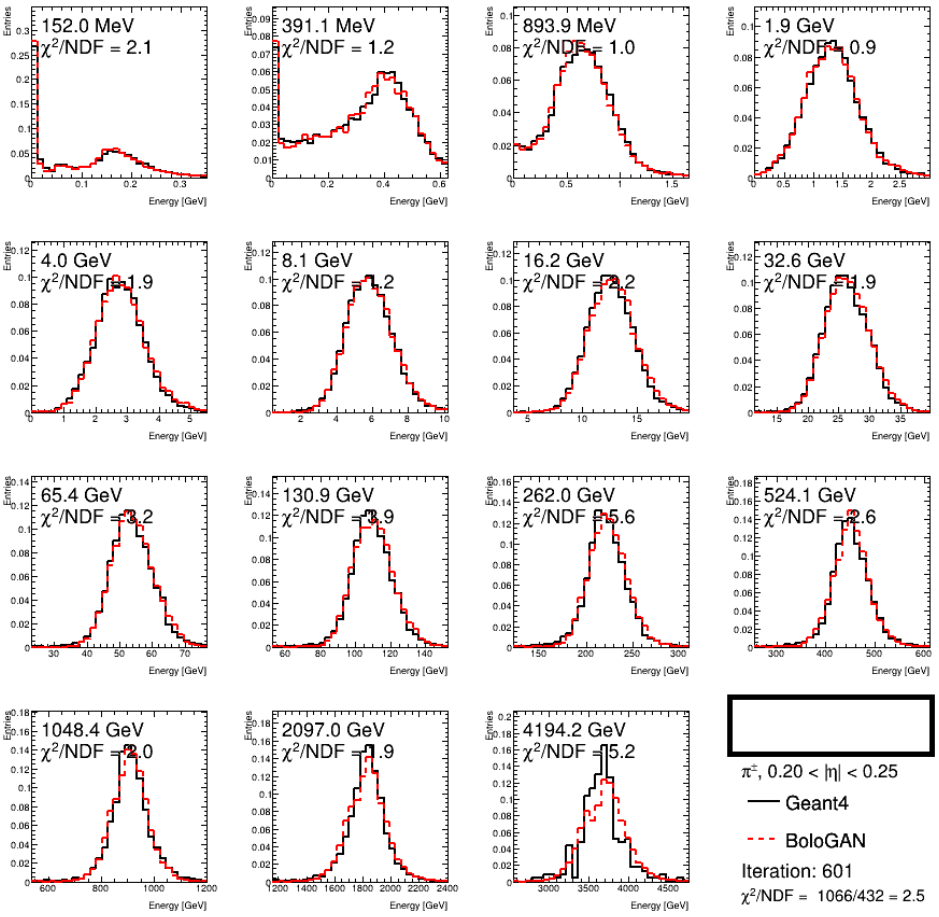
Starting from the preliminary analysis of various architectures of the generator the *Tensorboard* plots of the best 3 performing models are reported in Figure 6.1 with the baseline hyperparameters as laid down in Table 5.2 The best 2 models are with generator nodes 128, 256, 512 and 200, 400, 800, 1600. The baseline model 200, 400, 800 reported in 5.2 is also carried forward for further inspection.

The next inspection is carried out for the hyperparameter *batch size* for the three models reported above. The next hyperparameter of choice was D/G ratio for the two models with nodes 200, 400, 800 and 200, 400, 800, 1600. The best performing model is reported in Fig 6.2 (c)

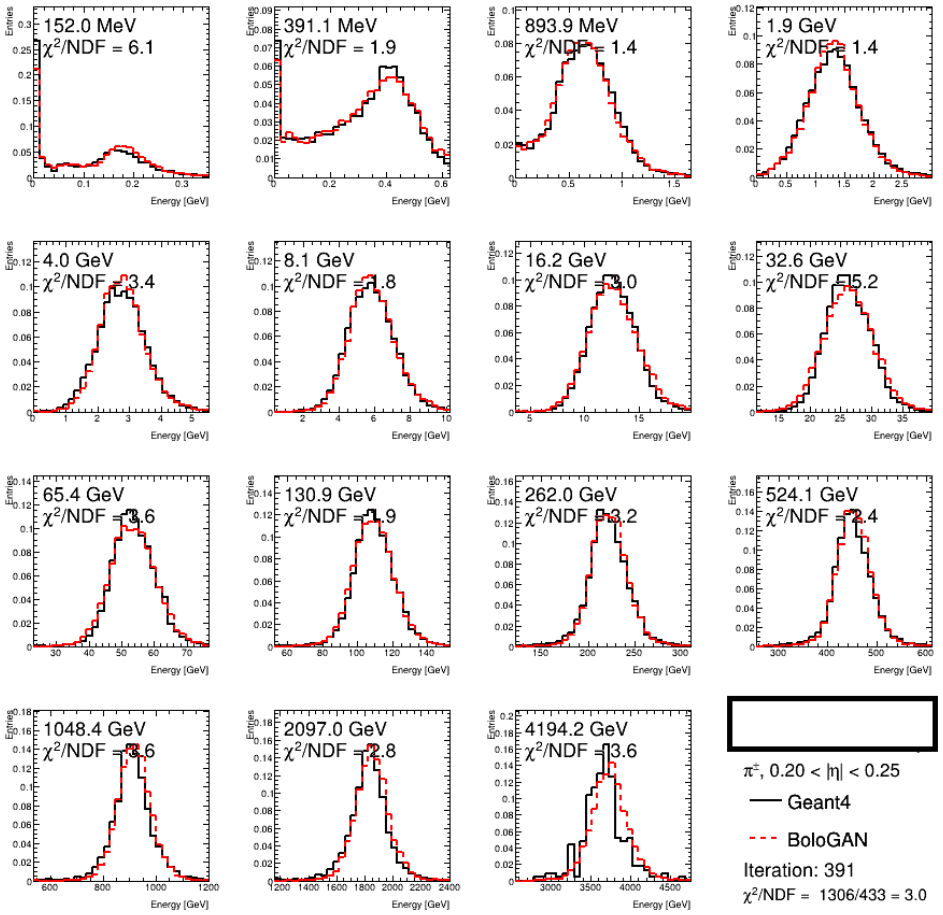
Finally combination of different D/G, *batch size* and *learning rate* were experimented with to combine the benefits of two or more hyperparameters on the performance. The results are reported in Fig 6.2.



(a) Best χ^2_{red} for the model with Generator nodes 128,256,512

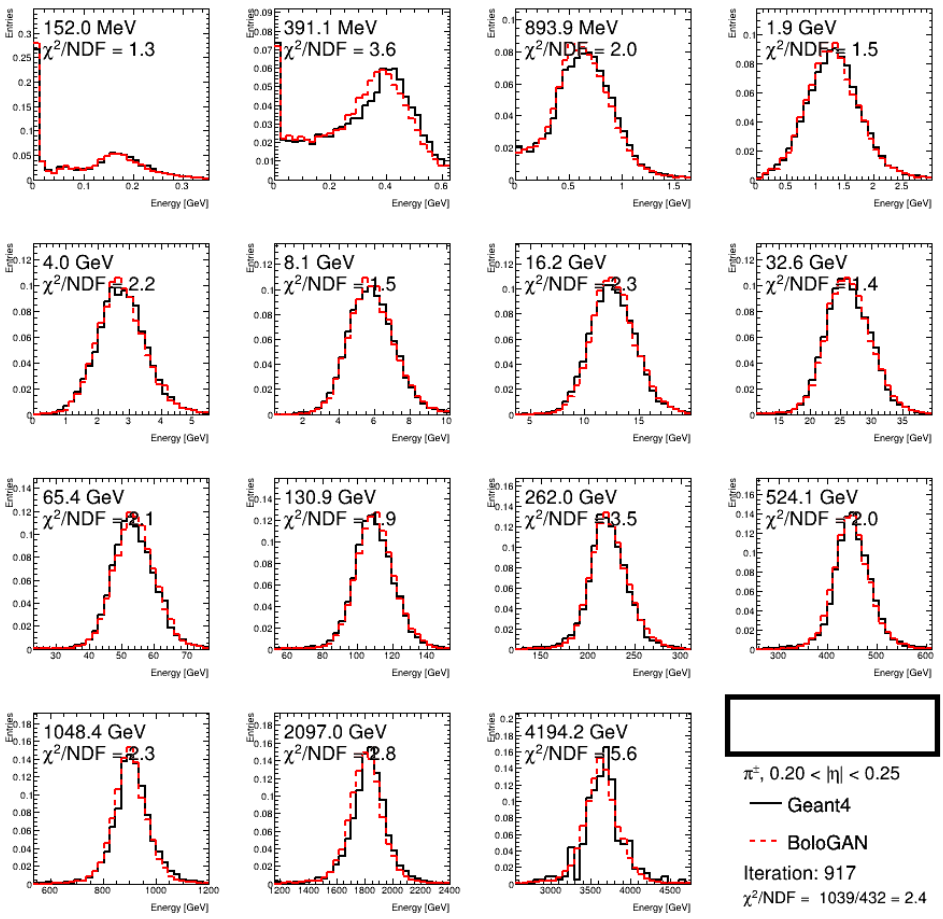


(b) Best χ^2_{red} for the model with Generator nodes 200, 400, 800, 1600

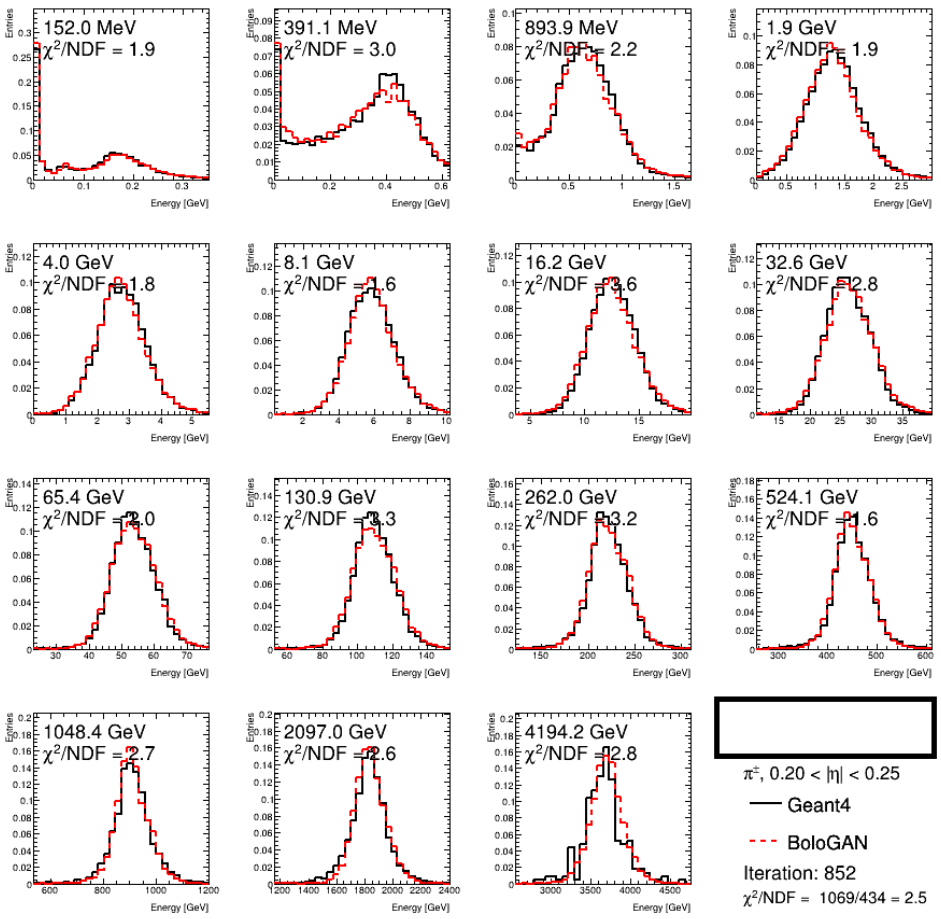


(c) Best χ^2_{red} for the model with 200, 400, 800

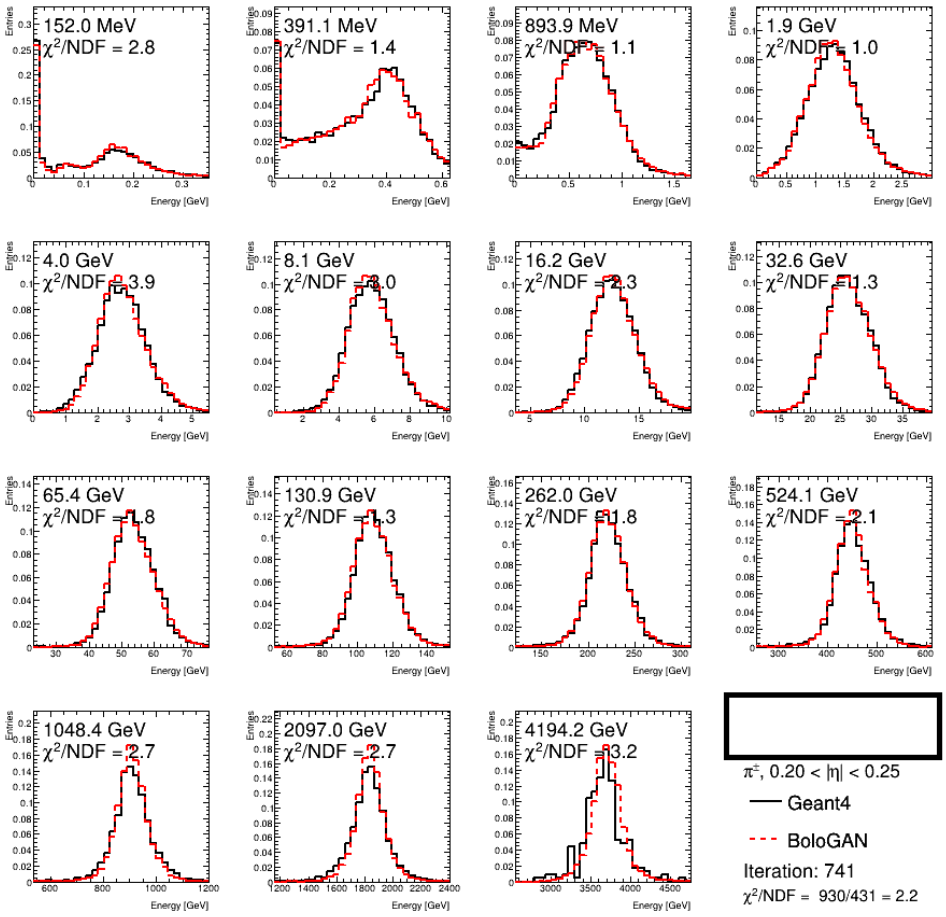
Figure 6.1. Preliminary assessment χ^2_{red} for the model with different Generator nodes with baseline hyperparameters as given in Table 5.2



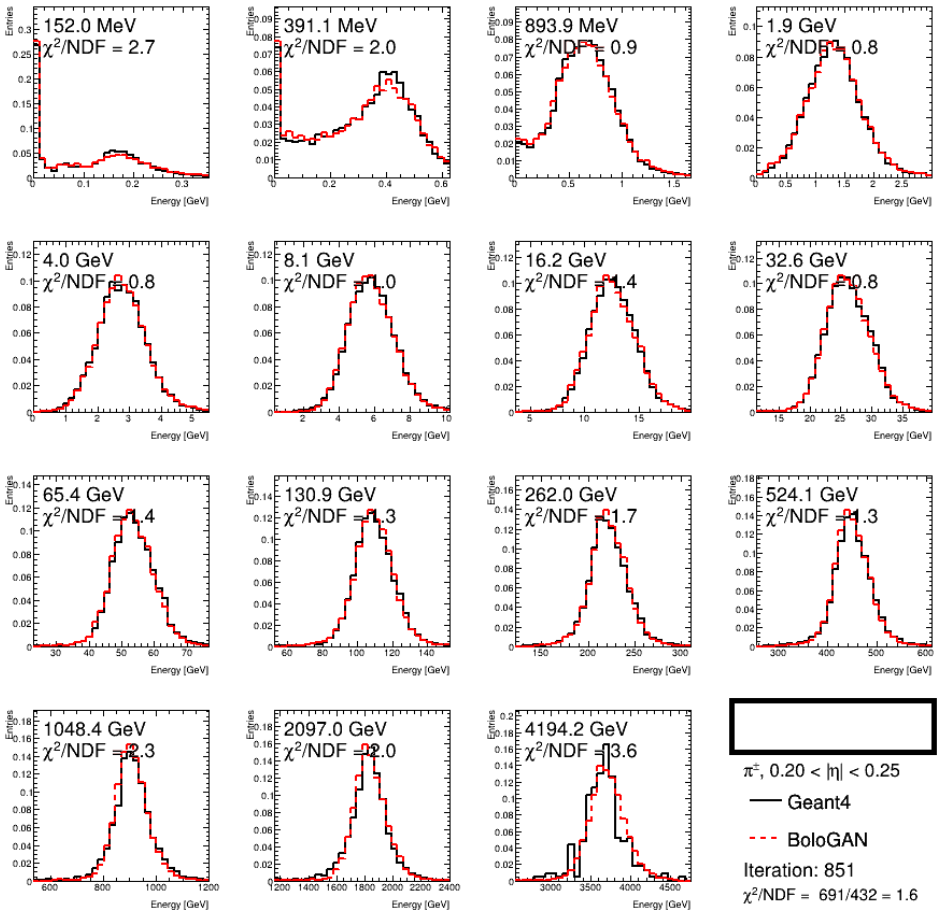
(a) Model 100-200, 400, 800, 1600 with *Batch size* 1024, *D/G* 3, *Learning rate* 0.00005, *Gradient Penalty 12*



(b) Model 100-128,256,512 with *Batch size* 1024, *D/G* 3, *Learning rate* 0.0001, *Gradient Penalty* 10



(c) Model 100-200, 400, 800, 1600 with *Batch size* 1024, *D/G* 3, *Learning rate* 0.00005, *Gradient Penalty 10*



(d) Lowest run value: Model 100-200, 400, 800, 1600 with *Batch size* 1024, *D/G* 3, *Learning rate* 0.00005, *Gradient Penalty 10*

Figure 6.2. Best χ^2_{red} for the models with combination of hyperparameters

7 Analysis

7.1 Evaluations: Training times, Reduced Chi-Squares

The goal of this thesis is to modify the architecture of *BoloGAN* so as to achieve as low of χ^2/ndf as possible with minimal computational resources. The figure of merit used to compare different models is a χ^2/ndf statistic between the binned total energy distributions of the GAN and the training data. Other factors such as training time and memory usage to store model parameters is also reported. My work was focused on simulating charged pions in one region of the detector ($0.2 < \eta < 0.25$). The number of epochs used to train the models was kept fixed at the default value of 1 million. Evaluation of the χ^2/ndf was performed every 1000 epochs. Training this model on the GPU resources available on the CERN HTCondor system takes approximately 6 hours and requires 6.4 MB of storage space.

With reference to 7.1 the analysis of generator layout is presented henceforth. By default, the generator is a neural network with 5 layers: one input layer, three hidden layers and one output layer (corresponding to the number of voxels). In BoloGAN, the input layer consists of nodes called latent space concatenated with conditional variable to give 100 nodes (with 2 additional dimensional conditional labels). These are generated from random values from a normal (Gaussian) distribution with mean and standard deviation both of 0.5. Moving to towards the internal section of the generator, there are the hidden layers that are 200,400,800 in BoloGAN whereas in In FastCaloGANV1 they were 50,100,200. In this section, we will denote the generator architecture by 100-200,400,800 where the first number represents the latent space. Starting from the 50-50,100,200 we see an improvement when using slightly different network in 100-100,150,200 with a slight change in latent dimension. The improvement is drastic when using 100-128,256,512 network. This signifies keeping the latent space a little smaller than the first hidden layers pays dividends. One possible explanation is that a relatively higher latent dimension as compared to hidden layer dimension can introduce noise or artifacts since the latent space is underutilized. Additionally the transition to the output layer is smooth with the 512 nodes leading up to 533 voxels in output layer. This is another reason for improved performance. The 100-200,400,800 and 100-256,512,1024 give comparable results however 100-200,400,800 comes out as better performing due to less number of nodes and hence lower complexity and training time. Having a deeper network promises to deliver finer high level details due to a larger number of nodes. Consequently, there is an improvement in reduced chi squares for the 4 hidden layer models 100-100,150,200,250 and 100-200,400,800,1600. 100-100,150,200, 250 performs worse than 100-200,400,800,1600 due to a lower width. The increased number of neurons per layer reduces training time due to parallel processing of features, mitigates risk of perturbations, and captures a broader range of features.

At the end of this preliminary assessment, the top 3 performing models taking into account also lower computational complexity are 100-128, 256, 512, 100-200,400,800 and 100-200,400,800,1600.

The 100-200,400,800,1600 takes significantly longer 6 hours more than 100-200,400,800 does.

Generator Layout	Reduced Chi-Squares
100- 200, 400, 800	2.80
100- 128, 256, 512	2.50
100- 256, 512, 1024	2.84
100- 100, 150, 200, 250	3.74
100- 200, 400, 800, 1600	2.52
100- 100, 150, 200	9.33
50- 50, 100, 200	12.51

Table 7.1. Preliminary assessment of models

The first parameter of choice to investigate for these models is the *batch size*. Smaller batch size 256 takes longer to train, then how much 512 or 1024 takes. Increasing *batch size* reduces the training time however it may lead to negative affects such as poor generalization, and overfitting. Reducing *batch size* to 256 improves performance of 100-200,400,800, while there is not any improvement to increase it from 512 to 1024. For 100-128,256,512 there is a decrease in performance when *batch size* is increased.

Batch Size	Architecture	Reduced Chi-squares
256	200, 400, 800	2.48
512	200, 400, 800	2.80
1024	200, 400, 800	2.76
512	128, 256, 512	2.50
1024	128, 256, 512	2.71
512	200, 400, 800, 1600	2.52

Table 7.2. Batch size tuning

The next hyperparameter we choose to evaluate is *D/G ratio*. The standard values of 3, 5 and 8 are evaluated on the two better performing models 100-200,400,800,1600 and 100-200,400,800. For the model 100-200,400,800, 3 and 8 both give better results than the current value of 5, specially for the 100-200,400,800 model.

D/G Ratio	Batch Size	Architecture	Reduced Chi-Square
3	512	200, 400, 800	2.51
5	512	200, 400, 800	2.80
8	512	200, 400, 800	2.49
3	1024	200, 400, 800, 1600	2.15
5	512	200, 400, 800, 1600	2.52

Table 7.3. D/G tuning

At the end, we combine the benefit from tuning previous parameters with the learning rate which is the same for generator as well as discriminator. It can be seen that using a smaller learning rate of 0.00005 leads to some improvement as can be seen in Table 7.4.

7.2 Discussion on best hyperparameter performances

It is to note that the model with the lowest χ^2_{red} was 100-200,400,800,1600 with D/G ratio 3, batch size 1024 and learning rate 0.00005, resulting in a value of 1.6. While this training could

Learning Rate	D/G ratio	Batch Size	Architecture	Reduced Chi-square
0.00005	3	1024	200,400,800,1600	2.16
0.00005	3	1024	200, 400, 800	2.76
0.00005	5	512	200, 400, 800	2.43
0.0001	3	1024	200, 400, 800, 1600	2.53*
0.0001	3	1024	128, 256, 512	2.46
0.00005	3	1024	200, 400, 800, 1600	2.41*
0.00005	3	1024	200, 400, 800	2.75

Table 7.4. Lowest χ^2_{red} for a combination of hyperparameters. *Dictates an increment in gradient penalty from 10 to 12

be an outlier, the gradient penalty λ was then increased from 10 to 12 to mitigate potential overfitting and the resulting value increased from 1.6 to 2.41 to prevent the large set of parameters from overfitting the data. This combination also provided the χ^2_{red} final value of 2.15. However, another consideration while doing tuning is to consider computational cost. Therefore the other two architectures 100-200,400,800 and 100-128,256,512 were experimented. In view of this 100-128,256,512 demonstrated comparable performance with the bigger 100-200,400,800 with different set of hyperparameters with the later taking 4hours to train as compared to 7 hours for the former. Both models performed around 2.43 which is comparable to the performance of larger 100-200,400,800,1600 (with increased gradient penalty). Starting from the performance of the model with the hyperparameters in Table 5.2, we had 6hr of training (6.4MB) and a χ^2_{red} value of 3.0 which has been improved to χ^2_{red} value of 2.15 with 3 hours of training with 12.91MB required to save it.

8

Conclusion

We present optimization of the cWGAN-GP model using random search to reduce incurred computational cost and to improve performance in this project. The context of the work has been reported from the point of view of high energy experiment, mathematics behind cWGAN-GP, the statistics context of testing hypotheses, technical details of BoloGAN implementation, and hyperparameters experimentation. The training and evaluation of the model on LXBATCH has been shown. Final discussion demonstrates about 28% improvement in χ^2_{red} score, proving the experimentation to be successful. This reaffirms the application of *BoloGAN* as a lightweight and accurate simulation tool for replacing current fast simulation tools, reproducing *Geant4* to such an extent that it could use the same calibrations as *Geant4*. This work can be used to see the limits of cWGAN-GP performance as the model used to build *BoloGAN* and inform the decision of either keeping with it or exploring another likely candidate DCGAN.

Bibliography

Agostinelli, S., et al. 2003, Nucl. Instrum. Meth. A, 506, 250, doi: 10.1016/S0168-9002(03)01368-8

Allison, J., Amako, K., Apostolakis, J., et al. 2016, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 835, 186, doi: 10.1016/j.nima.2016.06.125

ATLAS Collaboration % . 2024, ATLAS Simulation Plot: SIM-2024-004, Tech. rep., CERN. https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/SIM-2024-004/

ATLAS Collaboration. 2020, Fast simulation of the ATLAS calorimeter system with Generative Adversarial Networks, Tech. rep., CERN, Geneva. https://cds.cern.ch/record/2746032

ATLAS Collaboration , o. 2023, ATLAS Simulation Plot: SIM-2023-005, Tech. rep., CERN. https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PLOTS/SIM-2023-005/

ATLAS Collaboration, Aad, G., et al. 2008, JINST, 3, S08003, doi: 10.1088/1748-0221/3/08/S08003

ATLAS Collaboration, Aad, G., et al. 2022, Comput. Softw. Big Sci., 6, 7, doi: 10.1007/s41781-021-00079-7

ATLAS Collaboration, Catmore, J., Di Girolamo, A., & Al. 2020, ATLAS HL-LHC Computing Conceptual Design Report, Tech. Rep. CERN-LHCC-2020-015 ; LHCC-G-178, CERN. https://cds.cern.ch/record/2729668

ATLAS Collaboration, o. 2023, ATLAS Experiment Documentation, Tech. rep., CERN. https://opendata.atlas.cern/docs/atlas/experiment/

ATLAS Collaboration, Marshall, Z., & Di Girolamo, A. a. 2022, ATLAS Software and Computing HL-LHC Roadmap, Tech. Rep. CERN-LHCC-2022-005 ; LHCC-G-182, CERN. https://cds.cern.ch/record/2802915

CMS Collaboration, Chatrchyan, S., et al. 2008, JINST, 3, S08004, doi: 10.1088/1748-0221/3/08/S08004

Faucci Giannelli, M., et al. 2023, Fast Calorimeter Simulation Challenge 2022 - Dataset 1, 1.0, Zenodo, doi: https://doi.org/10.5281/zenodo.8099322

Foster, D. 2019, Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play (O'Reilly Media)

Goodfellow, I., Pouget-Abadie, J., Mirza, M., et al. 2014, Advances in Neural Information Processing Systems (NeurIPS). https://arxiv.org/abs/1406.2661

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. 2017, Advances in Neural Information Processing Systems (NeurIPS). https://arxiv.org/abs/1704.00028

Krause, C., et al. 2022a, Review Article

—. 2022b, Journal of Instrumentation (JINST)

LHC Collaboration, Alves, A. A., et al. 2008, JINST, 3, S08005, doi: 10.1088/1748-0221/3/08/S08005

Paganini, M., de Oliveira, L., & Nachman, B. 2017, arXiv preprint arXiv:1705.02355

Pequenao, J. 2008, Event Cross Section in a Computer-Generated Image of the ATLAS Detector, CERN Document Server, CERN. https://cds.cern.ch/record/1096081

Zaborowska, A. 2017, Journal of Physics: Conference Series, 898, 042053, doi: 10.1088/1742-6596/898/4/042053