Department of Physics, FMF, UL, 2023/2024

Seminar, 3rd year

Application of Diffusion Models for Simulations in High Energy Physics at the LHC

Author:
Dimitrije Pešić

Mentor:
prof. dr. Borut Paul Kerševan

Abstract

In this seminar, we delve into the use of diffusion models in high-energy physics simulations, focusing on their growing importance and use as replacements for traditional Monte Carlo methods, especially in the field of particle collisions and interactions at the Large Hadron Collider (LHC). With this research, we aim to clarify the theoretical foundations and practical consequences of diffusion models in advancing our understanding of the fundamental principles of physical phenomena and pushing the boundaries of scientific exploration. We conclude with a presentation of a concrete study involving the use of a diffusion model for simulating particle showers in a calorimeter detector.

Ljubljana, March 2024

Content

1 Introduction to High-Energy Physics at the LHC

Simulations play a crucial role in all areas of particle physics, from the initial phases of detector design to the complex analysis of collision data. They serve as a fundamental tool for understanding and predicting particle behavior in the complex environment of experiments at the *Large Hadron Collider* - LHC [1].

Traditionally, particle interaction simulations have relied primarily on Monte Carlo (MC) methods, a computational approach that involves random sampling to simulate various phenomena. These simulations model events in the collider from the initial hard scattering through hadronization[1] to the detailed responses of the tracker, calorimeter, and muon systems of the detector. Such detailed modeling is a process that requires substantial computational power. Additionally, the number of simulated events must at least equal the number of recorded collisions to ensure subsequent statistical comparisons with collision data are relevant and sufficiently accurate. As described in ref. [2], MC simulations have represented a significant load on the LHC's computing grid (*Worldwide LHC Computing Grid* - WLCG [3]), with projections indicating a substantial increase in demand, particularly with the upgrade of the LHC to high luminosity (HL-LHC) scheduled for 2025 [4].
Seeking methods capable of improving the accuracy, speed, or achievable MC statistics by two orders of magnitude is of great importance to physicists. We delve into an innovative approach — Diffusion Generative Models — and explore their potential to transform simulation techniques in the context of *high energy physics* (HEP) at the LHC. One reason for investigating the use of diffusion models is the recent surge in popularity of these models for industrial applications (DALL-E, Midjourney, SORA, Imagen, etc. [5, 6, 7, 8]).

1.1 Calorimeter Detectors

The LHC is a synchrotron collider that collides protons at extremely high energies, measured in tera-electronvolts. Upon collision, secondary particles are produced, whose energies and trajectories are detected by specialized detectors, such as CMS and ATLAS. Electromagnetic calorimeters (ECAL) specifically measure the energy of particles that create EM showers in the calorimeter material at given energies (through electron-positron pair production and bremsstrahlung). At relevant energies, these are only electrons and photons (for more massive particles, the process becomes significant at much higher energies).

In CMS, the basic unit of the ECAL is a crystal of *lead tungstate* ($PbWO_4$), coupled with a photodetector. When high-energy particles interact with these crystals, they create electromagnetic showers that include lower-energy photons, electrons, and positrons [9, 10]. These particles from the showers excite the $PbWO_4$ crystals, which emit scintillation photons during the relaxation process. These photons reach the attached photodetectors, where they trigger the release of electrons through the photoelectric effect. The resulting electric current is further amplified and reflects the energy initially deposited by the incident particle.

[1] Hadronization is the process of creating hadrons from quarks and gluons.

1.2 Simulations with Geant4

In simulations, we aim to model the calorimeter's response to a specific particle type with a given energy at the cell level (individual crystals) of the calorimeter. In principle, we generate an image of the response 1, where each crystal represents a pixel and the deposited energy is the intensity of the pixel. Calorimeter shower simulations are performed using the GEANT4 software package [11], developed by the CERN collaboration. This approach uses the MC method, where we define the geometry of our detector and specify the particles (their energy and initial direction), and the simulation develops the EM shower through pair production and bremsstrahlung [12].

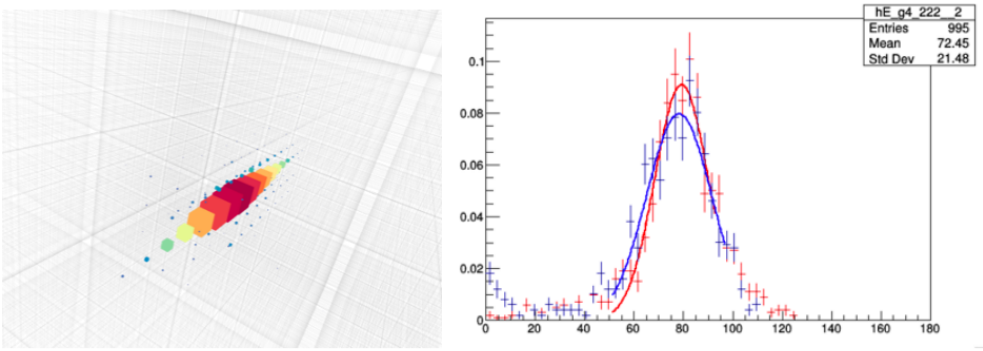

Figure 1: Typical energy showers produced by an electron with 100 GeV in the ECAL. The gray lines represent the identification of calorimeter cells. The color and size of the hit correspond to the amount of energy deposited in the cell [13].

2 Basics of Machine Learning

2.1 Neural Networks

Before delving into the specifics of diffusion models, it is important to understand the basics of machine learning, particularly neural networks (NN). A neural network consists of three types of layers:

- Input layer: The initial layer where raw data $\mathbf{x}$ is fed.
- Hidden layers: Responsible for extracting and transforming features from the input data using nonlinear transformations called activation functions. Typical activation functions are the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ and $\mathrm{ReLU}(x) = \max(x, 0)$.
- Output layer: The final layer that returns predictions or results of the model based on the learned connections from the hidden layers.

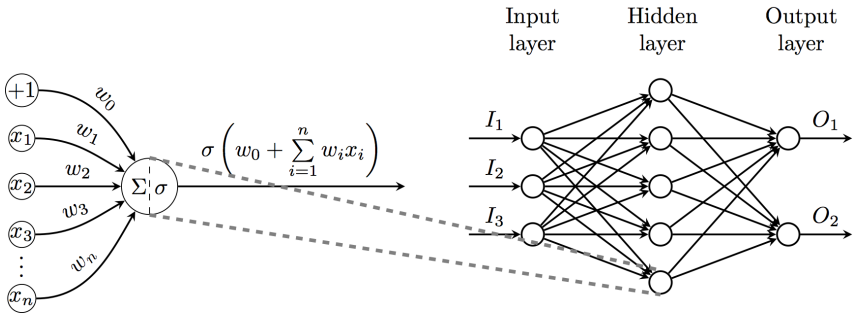

Figure 2: Diagram of a single neuron (left), diagram of a fully connected neural network (right) [14].

In Figure 2, the individual layers and the operation of the NN network on a single neuron are shown. In each layer, the neurons of the previous layer are weighted and summed through connections between them. A *bias* parameter b is added, representing an affine transformation in mathematics, which alone cannot model more complex patterns, hence it is passed through the previously mentioned activation function. This process is repeated for each layer, where the activation of the previous layer now represents the new input $\mathbf{x}$. For a single neuron, we can write the activation a_i, where index i represents the *depth* (sequential layer in the network), and index n represents an individual neuron at depth i, as:

$$a_i = \theta\left(\sum_n x_n w_{i,n} + b_i\right) , \tag{1}$$

Where θ represents our choice of activation function. The entire process can be described in matrix form, with weights written as a matrix $W \in \mathbb{R}^{m \times n}$, and $\mathbf{b}$ as the bias vector:

$$\mathbf{a} = \theta\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right) . \tag{2}$$

2.2 Network Training

Training a neural network involves iteratively adjusting its *parameters* $\theta_t = (W, b)$ (weights and biases) to minimize a predefined *loss function* Lthat quantifies the

discrepancy between the predicted outputs of the neural network and the known desired outputs. Physicists often use the mean squared error (MSE) function: $L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, which is directly related to the χ^2 test.
This process, known as *backpropagation*, involves propagating the error gradient backward through the network and updating the parameters using optimization algorithms such as stochastic gradient descent (SGD) or its variants:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t)\,, \tag{3}$$

where α represents the *learning rate*..

When training, it is important to carefully choose the loss function for the desired goal, as SGD can encounter the problem of local minima and the pitfall of *overfitting* — too few test data for too many free parameters of the model [15, 16].

2.3 Convolutional Networks

Convolutional neural networks (CNN) have emerged as a specialized class of neural networks tailored for image processing. They excel at capturing spatial hierarchies of features in images, making them particularly suitable for tasks such as image classification, object detection, and segmentation. A typical representation of a convolutional network is shown in Figure 3. architecture and key components of CNN are:

- Convolutional layers: As the name suggests, they perform the convolution of the image with *kernels* (filters), extracting local features. The convolution operation involves sliding the kernel over the input image and summing individual element-wise multiplications (analogous to weights in NN). The resulting *feature maps* capture patterns and edges present in the input image.

- Pooling layers: Used to reduce the dimensions of the feature maps while retaining important features. Pooling layers help achieve translational invariance and reduce computational complexity (fewer parameters, less problems with overfitting).

- Fully connected layers: After several convolutional and pooling layers, the extracted features are flattened and passed through one or more fully connected layers — a regular neural network.

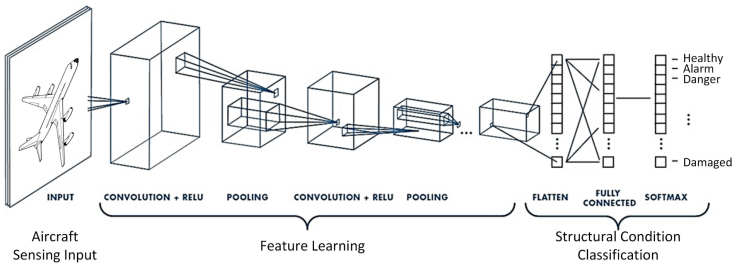

Figure 3: Diagram of individual levels of a convolutional network. Image sourced from ref. [17].

3 Introduction to Generative Models

Generative models enable the creation of new data samples that closely resemble the original data. Unlike discriminative models, which focus on predicting labels or classifying inputs, generative models focus on understanding and generating data. Several generative models are shown in Figure 4.

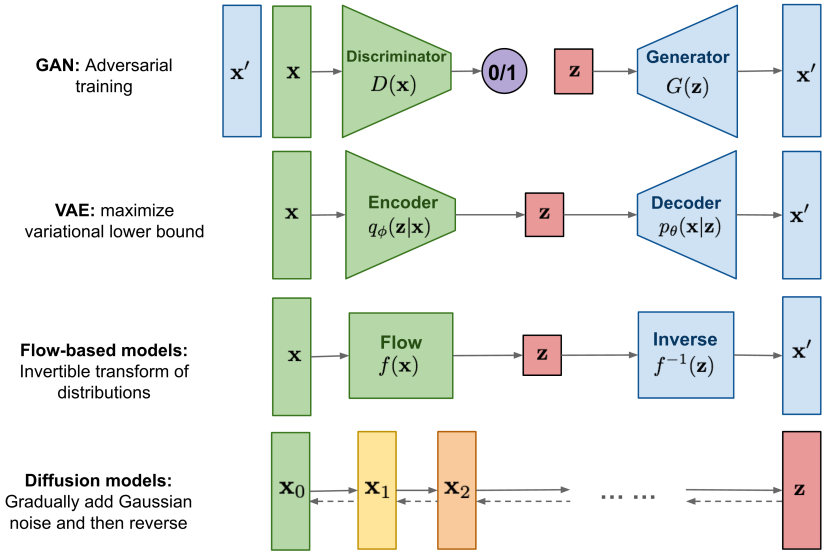

Figure 4: Overview of different types of generative models [18].

Generative Adversarial Networks (GANs) are among the most popular models. GANs consist of two neural networks, a generator and a discriminator, involved in a minimax game. The generator learns to produce realistic samples, while the discriminator learns to distinguish between real and generated samples. GANs have demonstrated remarkable success in generating high-quality images but are unstable during training and often subject to the problem of adversarial examples [19].

Another popular model is the *Variational Autoencoder* (VAE). VAEs learn a latent representation of the input data and generate new samples by sampling from the learned latent space. They are fast at sampling, easy to train, but generally produce lower-quality results.

In addition to models like VAEs and GANs, a new class of generative models based on diffusion processes has emerged in recent years. These models, inspired by the theory of stochastic processes in nonequilibrium thermodynamics [20], utilise the concept of iterative noise refinement to generate high-quality samples.

4 Diffusion Models

Several generative models based on diffusion have been proposed, and in this seminar, we focus on the implementation of denoising diffusion probabilistic models (DDPM) by Ho et al. [21]). Essentially, these implementations are similar. We start by defining a Markov chain[2] of diffusion steps to slowly add random noise to the data, and the models then learn to reverse the diffusion process to construct the desired data samples from noise (Figure 5).

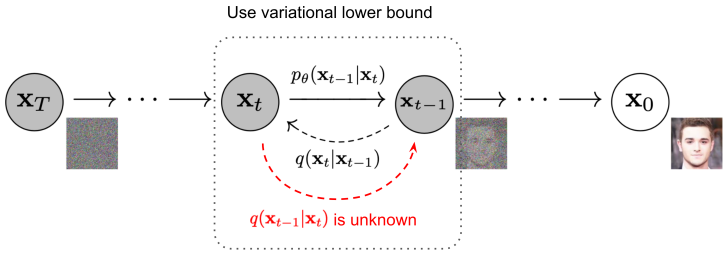

Figure 5: Markov chain of the forward (reverse) diffusion process of sample generation with gradual noise addition (removal). (Image source: Ho et al. [21]).

4.1 Mathematical Background

At a high level, diffusion models sample from a distribution by reversing a noise addition process. A detailed formulation of Gaussian diffusion models can be found in refs. [20, 21, 23]. We begin by defining our data distribution $x_0 \sim q(x_0)$ and a Markov process q,that gradually adds Gaussian noise to the data to create noisy samples from x_1 to x_T. This is mathematically described as:

$$q(x_t|x_{t-1}) \coloneqq \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}). \tag{4}$$

Here, x_t represents the output function, $\mathcal{N}$ denotes Gaussian noise sampled according to the variance schedule β_t. he noise variance schedule β_t allows applying different amounts of noise depending on the time/iteration step t. The most common schedules are linear and cosine, as they allow more gradual destruction of the image. Ho et al. [21] note that it is unnecessary to repeatedly apply the function $q(x_i|x_{i-1})$, to sample from $x_t \sim q(x_t|x_0)$. Instead, we can express $q(x_t|x_0)$ as a Gaussian distribution with $\alpha_t \coloneqq 1-\beta_t$ in $\bar{\alpha}_t \coloneqq \prod_{s=0}^{t} \alpha_s$

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{5}$$

$$x_t(x_0, \epsilon) = x_t = \sqrt{\bar{\alpha}_t}x_0 + \epsilon\sqrt{1-\bar{\alpha}_t}, \qquad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \tag{6}$$

Here, $1-\bar{\alpha}_t$ tells us the noise variance for any given time step, which can equivalently be used to define the noise schedule instead of β_t.

Using Bayes' theorem, we find that the posterior term $q(x_{t-1}|x_t, x_0)$ is also

[2]A Markov chain is a stochastic model that describes the probability of a sequence of events occurring based on a previous event [22].

Gaussian with mean $\tilde{\mu}_t(x_t, x_0)$ and variance $\tilde{\beta}_t$ defined as follows:

$$\tilde{\mu}_t(x_t, x_0) \coloneqq \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t \tag{7}$$

$$\tilde{\beta}_t \coloneqq \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{8}$$

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t \mathbf{I}) \tag{9}$$

To sample from the data distribution $q(x_0)$, we can first sample from $q(x_T)$ and then sample the reverse steps $q(x_{t-1}|x_t)$ until reaching x_0. With reasonable settings for β_t and T, the distribution $q(x_T)$ is almost isotropic Gaussian, so sampling x_T is trivial. All that remains is to approximate $q(x_{t-1}|x_t)$ using a neural network, as it cannot be calculated exactly when the data distribution is unknown. For this purpose, [20] considers that $q(x_{t-1}|x_t)$ approximates a diagonal Gaussian as $T \to \infty$ and $\beta_t \to 0$, so it suffices to train neural networks (denoted by subscript θ) to predict the mean μ_θ and diagonal covariance matrix Σ_θ:

$$p_\theta(x_{t-1}|x_t) \coloneqq \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \,. \tag{10}$$

To train this model such that $p(x_0)$ learns the true data distribution $q(x_0)$, we can optimize the following variational lower bound L_{vlb} for $p_\theta(x_0)$:

$$L_{\text{vlb}} \coloneqq L_0 + L_1 + ... + L_{T-1} + L_T \,, \tag{11}$$

$$L_0 \coloneqq -\log p_\theta(x_0|x_1) \,, \tag{12}$$

$$L_{t-1} \coloneqq q(x_{t-1}|x_t, x_0) p_\theta(x_{t-1}|x_t) \,, \tag{13}$$

$$L_T \coloneqq q(x_T|x_0) p(x_T) \,. \tag{14}$$

While the above objective is well-founded, [21] found that a different objective produces better samples in practice. Specifically, they do not directly parameterize $\mu_\theta(x_t, t)$ as a neural network but instead train a model $\epsilon_\theta(x_t, t)$ to predict the noise ϵ from equation 6. This simplified objective, which can be derived from interpreting the diffusion model as denoising as in VAE, is defined by the loss function:

$$L_{\text{simple}} \coloneqq E_{t\sim[1,T], x_0\sim q(x_0), \epsilon\sim\mathcal{N}(0,\mathbf{I})} \left[||\epsilon - \epsilon_\theta(x_t, t)||^2 \right] \,, \tag{15}$$

where, as the indices in the formula indicate, time $t \in [1, T]$, x_0 is from the initial data distribution, and $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ is noise sampled from the normal distribution. During sampling, we can use the substitution to derive $\mu_\theta(x_t, t)$ from $\epsilon_\theta(x_t, t)$:

$$x_{t-1} = \mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) \tag{16}$$

Note that L_{simple} does not provide any signal for learning the variance $\Sigma_\theta(x_t, t)$ from equation 10, so in ref. [21] they found that instead of learning $\Sigma_\theta(x_t, t)$ they can set it to a constant $\beta_t \mathbf{I}$. The value matches the upper and lower bounds for the true variance of the reverse step. Training with this objective and using their corresponding sampling procedure is equivalent to the *score-matching* model for denoising from the work of Song and Ermon [24, 25]. This model uses Langevin dynamics for sampling from the denoising model, which goes beyond the scope of this

seminar. We often use "diffusion models" as shorthand for both classes of models.

In short, we train a U-Net architecture 9 (detailed in A) by optimizing formula 15 using the backpropagation algorithm (chapter 2.2). Once we have a trained model, we sample an image from random noise using formula 16. The entire process is schematically presented in Figure 10 as the pseudocode of algorithms for training and sampling diffusion models.

4.2 Results

With some architectural improvements described in the paper [23], the authors enable diffusion models to significantly outperform the best GAN models while maintaining greater distribution coverage. Figure 6 compares random samples from the best deep model BigGAN with our diffusion model in the paper [23]. While the samples are of similar perceptual quality, the diffusion model contains more variety than the GAN model, such as larger ostrich heads, individual flamingos, various orientations of hamburgers, and a fish not held by a human. The metric used to evaluate the quality of images generated by the generative model is called the *Fréchet Inception Distance* (FID) [26].

Figure 6: Samples from BigGAN-deep (FID 6.95, left), compared to samples from our guided diffusion model (FID 4.59, middle), and samples from the training set (right).

5 Application of Diffusion Models in HEP

Now that we understand diffusion models, we can demonstrate their use for practical application. We focus on three models, specifically CaloClouds [27] and its upgrades CaloClouds II and CaloClouds II (CM) *Consistency Model* [28].

CaloClouds I is structured as a standard DDPM diffusion model [21]. The goal was to create high-fidelity representations of *point clouds* of electromagnetic showers in calorimeters, offering an alternative to computationally expensive fixed-structure approaches. CaloClouds II , which builds on the foundations laid by CaloClouds I, introduces several improvements. These are based on the Langevin dynamic models for score matching we mentioned in the previous chapter 4, fewer model evaluations, and the introduction of the consistency distillation technique.

These changes are designed to improve the *fidelity* and computational efficiency of the simulation process, meeting the requirements of experiments like the future International Large Detector (ILD [29]) and the high-luminosity Large Hadron Collider (HL-LHC [4]).

The models were evaluated based on their ability to accurately simulate various physical properties of electromagnetic showers in calorimeter detectors (as described in chapter 1.2). These properties include energy distribution per cell, radial and longitudinal shower profiles, centroid calculations, visible energy distribution, and the number of hits above the threshold. Comparative analyses with Geant4 , which represents the ground truth simulations, have shown that while both models exhibit high fidelity in representing these physical properties, CaloClouds II generally performs better than its predecessor, especially in terms of fidelity results based on the Wasserstein distance metric.[3].

5.1 Physical Performance

We compare different calorimeter shower distributions from ref. [27] between the test set Geant4 and datasets generated using all three models. First, we compare various observed values at the cell level and showers calculated from the model-generated showers with Geant4 simulations with incoming photon samples with energies uniformly distributed between 10 and 90 GeV (also called the full spectrum). In Figure 7, we explore three representations of energy distributed in calorimeter cells. The energy per cell distribution contains the energy of cells from all showers in the test dataset. All models relatively well describe the cell energy distribution.

Next, we explore the reliability of the models for individual incoming photon energies of 10, 50, and 90 GeV (Figure 8). The total energy is well represented in all three models. The number of hits is one of the most challenging distributions for a generative point cloud model. Here, high fidelity is still achieved using hit count calibration [28].

[3]The Wasserstein metric is a distance function defined between probability distributions on a given metric space M [30].

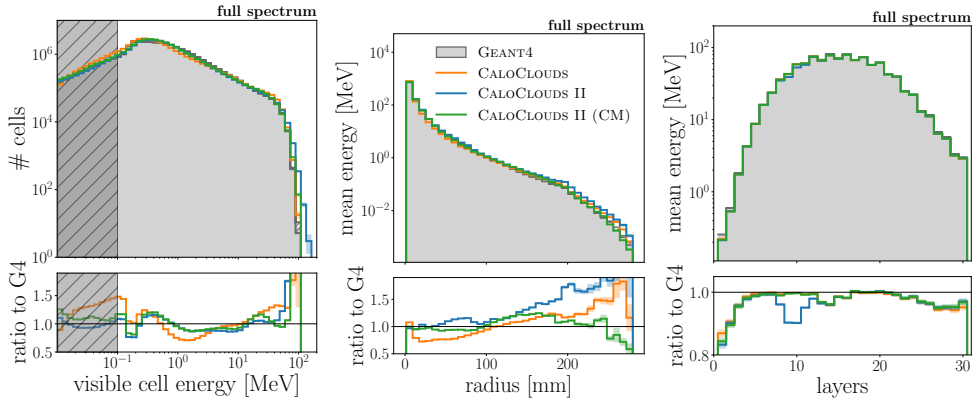

Figure 7: Histogram of cell energy distribution (left), radial profile (center), and longitudinal shower profile (right) for GEANT4 , CALOCLOUDS , CALOCLOUDS II , and CALOCLOUDS II (CM).

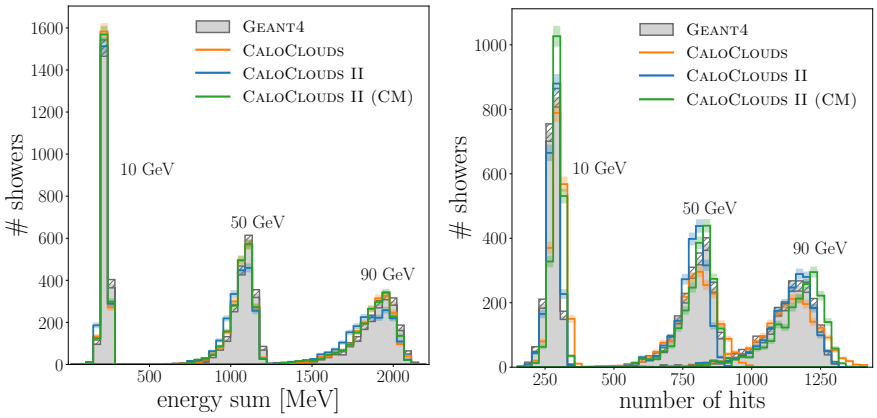

Figure 8: Visible distribution of total energy (left) and number of hits (cells with deposited energy above half the threshold) (right) for showers at 10, 50 and 90 GeV. 2000 splashes are shown for each energy and model. The error band corresponds to the statistical uncertainty in each interval (*bin*).

5.2 Evaluation Results

Table 1 presents a performance comparison of models based on 1-Wasserstein distance for various standardized shower observables. The presented values are the average and standard deviation of 10 calculated results comparing 50k GEANT4 and 50k generated showers, ref. [28].

Simulator	$W_1^{N_{\text{hits}}}$ ($\times 10^{-3}$)	$W_1^{E_{\text{vis}}/E_{\text{inc}}}$ ($\times 10^{-3}$)	$W_1^{E_{\text{cell}}}$ ($\times 10^{-3}$)	$W_1^{E_{\text{long}}}$ ($\times 10^{-3}$)	$W_1^{E_{\text{radial}}}$ ($\times 10^{-3}$)	$W_1^{m_{1,X}}$ ($\times 10^{-3}$)	$W_1^{m_{1,Y}}$ ($\times 10^{-3}$)	$W_1^{m_{1,Z}}$ ($\times 10^{-3}$)
GEANT4	0.7 ± 0.2	0.8 ± 0.2	0.9 ± 0.4	0.7 ± 0.8	0.7 ± 0.1	0.9 ± 0.1	1.1 ± 0.3	0.9 ± 0.3
CALOCLOUDS	**2.5 ± 0.3**	11.4 ± 0.4	15.9 ± 0.7	**2.0 ± 1.3**	38.8 ± 1.4	4.0 ± 0.4	8.7 ± 0.3	1.4 ± 0.5
CALOCLOUDS II	3.6 ± 0.5	26.4 ± 0.4	**15.3 ± 0.6**	3.7 ± 1.6	11.6 ± 1.5	**2.4 ± 0.4**	**7.6 ± 0.2**	3.9 ± 0.4
CALOCLOUDS II (CM)	6.1 ± 0.7	**9.8 ± 0.5**	16.0 ± 0.7	**2.0 ± 1.4**	**8.3 ± 1.9**	3.0 ± 0.4	9.5 ± 0.6	**1.2 ± 0.5**

Table 1: Performance comparison of models based on Wasserstein distance.

5.3 Timing

In this section, we compare the average time to generate a single calorimeter shower and explore the speedup relative to the baseline . Both on a single processor and an NVIDIA® A100 GPU, 25×2000 howers were generated with the same uniform energy distribution between 10 and 90 GeV. Notably, the timing on a single CPU is of interest for current applications of generative models in HEP, as CPUs are much more accessible than GPUs, and the current computing infrastructure relies on simulations running on CPUs. The timing measurement results are presented in Table 2. Note that GEANT4 is currently not GPU-compatible, and GPUs are significantly more expensive than CPUs. Nonetheless, we observe a drastic improvement in the speed of simulation production, with CALOCLOUDS II (CM)being 46× faster on a CPU and 1873× faster using a GPU.

Hardware	Simulator	NFE	Batch Size	Time per Shower [ms]	Acceleration
CPU	GEANT4			3914.80 ± 74.09	×1
	CALOCLOUDS	100	1	3146.71 ± 31.66	×1.2
	CALOCLOUDS II	25	1	651.68 ± 4.21	×6.0
	CALOCLOUDS II (CM)	1	1	84.35 ± 0.22	×46
GPU	CALOCLOUDS	100	64	24.91 ± 0.72	×157
	CALOCLOUDS II	25	64	6.12 ± 0.13	×640
	CALOCLOUDS II (CM)	1	64	2.09 ± 0.13	×1873

Table 2: Computational performance comparison of all three models.

6 Conclusion

In summary, the use of generative AI models such as CALOCLOUDS represents a significant advancement in simulating complex physical phenomena such as particle showers in detectors like the International Large Detector (ILD) and the High-Luminosity Large Hadron Collider (HL-LHC). These models offer promising paths for accelerating simulations, thereby enabling new discoveries in fundamental physics. Furthermore, beyond the academic world and industry, there is significant potential for applying generative AI in various other fields, including healthcare, materials science, and autonomous systems.

Literatura

[1] Lyndon R Evans and Philip Bryant. "LHC Machine". In: *JINST* 3 (2008), S08001. 164 p. DOI: 10.1088/1748-0221/3/08/S08001. URL: https://cds.cern.ch/record/1129806.

[2] S.Vallecorsa. "Generative models for fast simulation". In: *Journal of Physics: Conference Series* Volume 1085, Issue 2 (2018), 10 p. DOI: 10.1088/1742-6596/1085/2/022005. URL: https://indico.cern.ch/event/567550/papers/2656673/files/5841-SofiaVallecorsa_Plenary.pdf.

[3] CERN. *Worldwide LHC Computing Grid*. 2024. URL: https://wlcg.web.cern.ch/ (visited on 03/21/2024).

[4] CERN. *High-Luminosity Large Hadron Collider (HL-LHC)*. 2024. URL: https://home.cern/science/accelerators/high-luminosity-lhc (visited on 03/21/2024).

[5] OpenAI. *DALL·E: Creating images from text*. 2024. URL: https://openai.com/research/dall-e (visited on 03/21/2024).

[6] Midjourney. *Midjourney*. 2024. URL: https://www.midjourney.com/home (visited on 03/21/2024).

[7] OpenAI. *Sora: Creating video from text*. 2024. URL: https://openai.com/sora (visited on 03/21/2024).

[8] Brain Team Google Research. "Imagen unprecedented photorealism × deep level of language understanding". In: (2024). URL: https://imagen.research.google/.

[9] J. G. Layter. *The CMS electromagnetic calorimeter project : Technical Design Report*. Tech. rep. Geneva, 1997. URL: https://cds.cern.ch/record/349375.

[10] Elijan Jakob Mastnak. "End-to-End Particle Classification in High-Energy Physics". In: (2021). URL: https://ejmastnak.github.io/projects/seminar/paper.pdf.

[11] CERN. *Geant4: Toolkit for the simulation of the passage of particles through matter*. 2024. URL: https://geant4.web.cern.ch/ (visited on 03/21/2024).

[12] S. Cutajar, D. Oborn, and B. Rosenfeld. "Introduction to the Geant4 Simulation Toolkit". In: *Journal of Instrumentation* 6 (2011), T10002. DOI: 10.1088/1748-0221/6/10/T10002. URL: https://ro.uow.edu.au/engpapers/1703/.

[13] S. Vallecorsa. "Generative models for fast simulation". In: *Journal of Physics: Conference Series* 1085.2 (Sept. 2018), p. 022005. DOI: 10.1088/1742-6596/1085/2/022005. URL: https://dx.doi.org/10.1088/1742-6596/1085/2/022005.

[14] Petar Veličković. *Multilayer perceptron (MLP)*. 2016. URL: https://github.com/PetarV-/TikZ/tree/master/Multilayer%20perceptron (visited on 03/21/2024).

[15] Rahul$_{Roy}$. *ML | Stochastic Gradient Descent (SGD)*. 2024. URL: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/ (visited on 03/21/2024).

[16] *Overfitting*. 2024. URL: https://en.wikipedia.org/wiki/Overfitting (visited on 03/21/2024).

[17] Hailing Fu by Iuliana Tabian and Zahra Sharif Khodaei. "Generative models for fast simulation". In: *Sensors 2019* 19(22), 4933 (2019). DOI: https://doi.org/10.3390/s192249335. URL: https://www.mdpi.com/1424-8220/19/22/4933.

[18] Lilian Weng. "What are diffusion models?" In: *lilianweng.github.io* (July 2021). URL: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

[19] *Adversarial Examples: Definition and importance in machine learning*. 2023. URL: https://datascientest.com/en/adversarial-examples-definition-and-importance-in-machine-learning (visited on 03/21/2024).

[20] Jascha Sohl-Dickstein et al. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *CoRR* abs/1503.03585 (2015). arXiv: 1503.03585. URL: http://arxiv.org/abs/1503.03585.

[21] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *CoRR* abs/2006.11239 (2020). arXiv: 2006.11239. URL: https://arxiv.org/abs/2006.11239.

[22] *Markov chain*. 2024. URL: https://en.wikipedia.org/wiki/Markov_chain (visited on 03/21/2024).

[23] Prafulla Dhariwal and Alex Nichol. "Diffusion Models Beat GANs on Image Synthesis". In: *CoRR* abs/2105.05233 (2021). arXiv: 2105.05233. URL: https://arxiv.org/abs/2105.05233.

[24] Jiaming Song, Chenlin Meng, and Stefano Ermon. "Denoising Diffusion Implicit Models". In: *CoRR* abs/2010.02502 (2020). arXiv: 2010.02502. URL: https://arxiv.org/abs/2010.02502.

[25] Yang Song and Stefano Ermon. "Improved Techniques for Training Score-Based Generative Models". In: *CoRR* abs/2006.09011 (2020). arXiv: 2006.09011. URL: https://arxiv.org/abs/2006.09011.

[26] *Toggle the table of contents Fréchet inception distance*. 2024. URL: https://en.wikipedia.org/wiki/Fr%C3%A9chet_inception_distance (visited on 03/21/2024).

[27] Erik Buhmann et al. "CaloClouds: fast geometry-independent highly-granular calorimeter simulation". In: *Journal of Instrumentation* 18.11 (Nov. 2023), P11025. ISSN: 1748-0221. DOI: 10.1088/1748-0221/18/11/p11025. URL: http://dx.doi.org/10.1088/1748-0221/18/11/P11025.

[28] Erik Buhmann et al. *CaloClouds II: Ultra-Fast Geometry-Independent Highly-Granular Calorimeter Simulation*. 2024. arXiv: 2309.05704 [physics.ins-det].

[29] The ILD Concept Group. *The International Large Detector: Letter of Intent*. 2010. arXiv: 1006.3396 [hep-ex].

[30] *Wasserstein metric*. 2024. URL: https://en.wikipedia.org/wiki/Wasserstein_metric (visited on 03/21/2024).

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[32] *U-Net Architecture Explained*. 2023. URL: https://www.geeksforgeeks.org/u-net-architecture-explained/ (visited on 03/21/2024).

A U-Net Architecture

U-Net is a widely used architecture (shown in Figure 9) first introduced in the paper [31] to address the challenge of limited labeled data in the medical field. This network was designed to efficiently leverage smaller amounts of data while maintaining speed and accuracy.

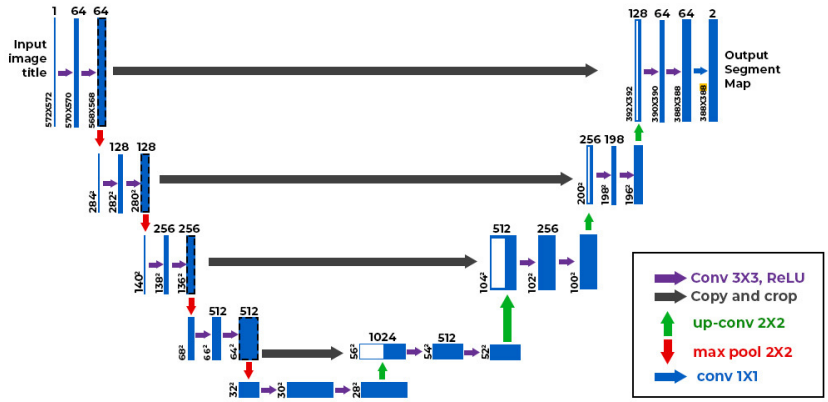

Figure 9: Diagram of the U-Net network [32].

It consists of a *contracting* path and an *expansive* path. The contracting path includes encoder layers that identify relevant features with convolutional operations and progressively abstract representations of the input. This captures contextual information and reduces spatial resolution, while the expansive path includes decoder layers that decode the data and use information from the contracting path through *skip connections* to create a segmentation map. Skip connections retain spatial information lost during the contraction path, helping the decoder layers accurately localize features.

B Pseudocode of Algorithms

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 10: Pseudocode of the algorithm for training the diffusion model (left) and pseudocode of the algorithm for sampling using the diffusion model (right). Adapted from [21].