SUPA: A Lightweight Diagnostic Simulator for Machine Learning in Particle Physics

Anonymous Author(s)

Abstract

Deep learning methods have gained popularity in high energy physics for fast modeling of particle showers in detectors. Detailed simulation frameworks such as the gold standard GEANT4 are computationally intensive, and current deep generative architectures work on discretized, lower resolution versions of the detailed simulation. The development of models that work at higher spatial resolutions is currently hindered by the complexity of the full simulation data, and by the lack of simpler, more interpretable benchmarks. Our contribution is SUPA, the SUrrogate PArticle propagation simulator, an algorithm and software package for generating data by simulating simplified particle propagation, scattering and shower development in matter. The generation is extremely fast and easy to use compared to GEANT4, but still exhibits the key characteristics and challenges of the detailed simulation. The proposed simulator generates thousands of particle showers per second on a desktop machine, a speed up of up to 6 orders of magnitudes over GEANT4, and stores detailed geometric information about the shower propagation. SUPA provides much greater flexibility for setting initial conditions and defining multiple benchmarks for the development of models. Moreover, interpreting particle showers as point clouds creates a connection to geometric machine learning and provides challenging and fundamentally new datasets for the field.

1 Introduction

In order to understand the fundamental building blocks of nature, High Energy Physics (HEP) experiments involve highly energetic particle collisions. These collisions cause particles to decay, and the identification of the resulting decay particles is of key importance to develop and confirm new physics/theories. This is enabled through an electromagnetic or hadronic calorimeter. As particles interact with the calorimeter, they split into multiple other particles, forming a particle shower. The nature of the generated shower depends on the specific material and geometry of the calorimeter, which are carefully selected and driven by physics theory and practical considerations. The generated shower deposits energy in active layers of the calorimeters, and provides energy and location measurements from the particles produced in the cascade.

Precise computer simulation is central to a better understanding of experimental results in HEP. Simulation is especially useful for the task of event reconstruction, where the deposited energy in the calorimeter is used to identify the particle that originated a given shower. The pattern of the deposited energy depends on the specific type of the (intermediate) particles, the initial energy and incidence angle, as well as the specific geometry and shape of the detector material, among other factors. The

state-of-the-art for this kind of simulations is GEANT4 [Agostinelli et al., 2003], a Monte Carlo toolkit for modeling the propagation of particles through matter. While detailed simulation through GEANT4 provides fine-grained shower generation and captures the underlying distributions accurately, this software consists of more than 3.5 million lines of C++, is computationally very expensive, and requires specific domain knowledge to be set up and tuned. This makes it cumbersome and expensive to produce the amount and diversity of data needed to speed up machine learning research on these topics. Table 1 compares GEANT4 with our proposed simulator SUPA.

Our key contributions to ease these issues are:

- We introduce SUPA: a fast, easy to use simulator for simple particle propagation, scattering and shower development in matter (see §3).
- We structure the simulator in a way that allows to easily change the complexity of the underlying shower development as well as the properties of the showers in terms of multiplicity, energy range, structure (see §4.1).
- We use SUPA to generate data that highlights the limitations of current machine learning approaches for fast shower simulation (see §5.2). We experimentally demonstrate that SUPA can be used as a proxy for detailed simulation, while being easy to tune and up to 6 orders of magnitude faster than GEANT4.

1.1 Image Representation

Existing approaches for generating particle showers [Paganini et al., 2018a, Krause and Shih, 2021] focus on a quantized representation of the calorimeter hits, binning them into discrete pixels, or cells, and setting the pixel intensity to the sum of the energy deposited. The upside of this approach is that, due to the structure of existing calorimeters, the data takes the form of low resolution images and standard computer vision models can be directly applied. However, as we increase the resolution of the quantized representations in order to preserve more information, the resulting images become exceedingly sparse, leading to difficulties in training existing machine learning architectures. Figure 1b, 1c, and 1d show the downsampled image representations at resolutions of 3x, 2x and 1x respectively for the shower shown in Figure 1a. We choose 1x to be the same resolution as used in CaloGAN [Paganini et al., 2018a] (i.e. $12 \times 12$ for Layer 1).

It is also important to notice that, when decreasing the resolution of the data and looking at showers, while the signal to model is of lower dimension and less sparse, the underlying structure and relationships between hits are lost. This requires models to implicitly learn equivariances from multiple examples, while also introducing artificial structure arising from the chosen detector geometry. For example, with a shower incident on the surface of a detector, a slight translation in either direction does not change the properties of the shower. However it still has an impact on the resulting datasets, where for small shifts it is not translationally equivariant. Having access to the underlying energy depositions would therefore greatly reduce the dependence on the geometry of the detectors in the models, and allow generative models to learn the underlying structure from the physical process directly.

![Figure 1: An example shower (single layer) shown at different resolutions](image-url)
1.2 Point Cloud Representation

Many popular point-cloud datasets [Wu et al., 2015, Chang et al., 2015] in the machine learning community contain shape representations of physical objects, such as points sampled uniformly from the surfaces of chairs, tables, etc. These datasets enjoy many nice properties, e.g. locality and redundancy, that calorimetry datasets do not. Moreover, existing generative models for point clouds rely on these simplifying assumptions and therefore their performance on more diverse datasets is still relatively unexplored. In the case of particle showers in calorimeters, dealing directly with the point cloud representation provides an optimal description of the underlying physics processes. However, due to the detailed simulation resulting from GEANT4, the number of points per shower represents a huge computational challenge.

A key objective of our synthetic simulator is to bridge this gap by allowing for quicker ground truth data generation for different initial settings, thus providing a way of modulating the complexity of the dataset, leading to more interpretable benchmarks. This would also make it possible to take gradual steps in developing the models and work out which of the many complexities in the GEANT4 datasets are hardest to model.

The code for SUPA, and all the models, along with all the canonical datasets will be available under an open source license at the time of the publication.

2 Related Work

Shower simulation with GEANT4 requires a complete description of the geometry and material of the detector and simulates all interactions between the initial particle, any subsequently produced particles through decays or emissions, and the detector material. The upside of using GEANT4 for simulation is that it is very accurate and represents the underlying physical phenomena precisely, but it comes at the cost of being computationally very expensive. Moreover, it requires expertise to define the configuration and a multitude of expert hours to set it up for new detector geometries. To reduce the computational resources required to simulate particle physics collisions and their energy depositions in detectors, deep generative models have been applied to generate particle showers. Most of these models [Abhishek et al., 2021, Krause and Shin, 2021, Paganini et al., 2018b] employ the dataset first introduced in CaloGAN [Paganini et al., 2018b] which is based on a simplified calorimeter inspired by the ATLAS Liquid Argon (LAr) electromagnetic calorimeter.¹

2.1 Datasets and calorimeter structure

For the CaloGAN dataset, the calorimeter is cubic in shape with each dimension being 480 mm and no material in front of it. The volume is divided into three layers along the radial (z) direction with varying thicknesses of 90 mm, 347 mm, and 43 mm, respectively. These layers are further segmented into discrete cells, with different sizes for each layer: 160 mm × 5 mm (first layer), 40 mm × 40 mm (second layer) and 40 mm × 80 mm (third layer). Each layer can be represented as a single-channel two-dimensional image with pixel intensities representing the energy deposited in the region. The final read-out has the resolution of 3 × 96, 12 × 12, and 12 × 6. For this dataset, three different particle types were considered, namely, positrons, charged pions and photons. Further, the particles were configured to be incident perpendicular to the calorimeter with initial energies uniformly distributed in the range between 1 GeV and 100 GeV.

Several other calorimeter geometries and configurations are employed across the literature. Erdmann et al. [2019] consider a calorimeter configuration motivated by the CMS High Granularity Calorimeter (HGCAL) prototype, while Belayneh et al. [2020] study a calorimeter based on the geometric layout of the proposed Linear Collider Detector (LCD) for the CLIC accelerator. Buhmann et al. [2021b,a] investigate the prototype calorimeter for the International Large Detector (ILD) which is one of the

¹The ATLAS calorimeter has e.g. a more complex geometry, with the cells having accordion shaped electrodes to maximise the active volume
two proposed detector concepts for the International Linear Collider (ILC). In a different application setting, Erdmann et al. [2018] model a calorimeter response for cosmic ray-induced air showers in the Earth’s atmosphere, producing signals in ground based detector stations. In contrast to the CaloGAN calorimeter, it has a single readout layer with $9 \times 9$ cells, each cell corresponds to a detector unit placed with a spacing of 1500 m. Although many different detector technologies are considered in the various models, the common aspect between the models and the corresponding datasets is the projection of the spatial signal onto discretized cells in 2D planes or a generalized 3D volume. The projection is done either directly while designing the calorimeter geometry, or as a post processing step. Further, the number of cells and thus the resolution of the final read-out is usually kept small. This is done either to simplify the data for training the models, or in order to speed up the simulations. Due to the different geometries used for generating the datasets, it is not a straightforward task to compare the performance of the various models. As the developments of each shower are completely dependent on the detailed model of the detector in GEANT4, although the same initial particles can be simulated, it is very difficult to draw parallels between two different datasets, as it is not possible simply to change the representation of the data from one model to look like that of another.

2.2 Deep Generative Models

The applications of deep generative modeling to calorimeter simulation have so far almost entirely focused on Generative Adversarial Networks (GANs, Goodfellow et al. [2014] and Normalizing Flows [Rezende and Mohamed, 2016]. CaloGAN [Paganini et al., 2018a] was the first application of GANs to a longitudinally segmented calorimeter. The approach is based on LAGAN [de Oliveira et al., 2017], a DCGAN [Radford et al., 2016]-like architecture, that is able to synthetize the shower images. The generator outputs a gray-scale image for each layer in the calorimeter, with each output pixel representing the energy pattern collected at that location. The architecture is also complemented with an auxiliary classifier tasked with reconstructing the initial energy $E$.

Krause and Shih [2021] improve on CaloGAN with CaloFlow, a normalizing flow architecture to generate shower images. CaloFlow is able to generate better samples than previous approaches based on GANs and VAEs, while also providing more stable training.

Previous work by the ATLAS collaboration [2018] proposes to use a Variational Autoencoder on the flattened images [Kingma and Welling, 2014]. Both the encoder and decoder consist of an MLP conditioned on the energy of the initial particle. In addition to the reconstruction and KL-terms, the authors also optimize for the overall energy deposit in both in the individual layers and in the overall system.

3 Propagation Model

Figure 2: Schematic summary of a propagation event generated by SUPA. (left) Splitting events can happen at the black vertical lines, called slices. See §3 for a detailed explanation of the process. (right) Each time a particle crosses a pre-defined “slice”, it either splits in two, stops and deposits energy, or passes through unperturbed.
SUPA generates data that imitates the propagation and splitting/scattering of point particles. To generate an event, a point particle is initialized at the origin of the 3D space with an initial energy value and a velocity parallel to the z-axis. We then define a sequence of 2D planes orthogonal to the z-axis at \{z_0, \ldots, z_N\} that we call “slices” (or “sub-layers”, interchangeably). We assume that the slices are evenly spaced along the z-axis, with a gap of \Delta z between consecutive slices. As the initial particle reaches the first predefined z-slice at \(z_0\), it has 3 options (see Figure 2): stop and deposit its energy with stopping probability \(p_{\text{stop}}\), split into two with splitting probability \(p_{\text{split}}\), or simply continue moving with the same velocity with pass-through probability \(p_{\text{pass}} = 1 - (p_{\text{stop}} + p_{\text{split}})\).

Then:

- If the particle splits, two particles are generated at the splitting location with energy and velocity features that ensure momentum conservation, i.e., that the sum of momentum vectors of the new particles is equal to the decaying one. The original particle is removed from the propagation model and the two new ones continue propagating with their respective velocities. The splitting angle \(\theta\) and deviation angle \(\epsilon\) are sampled from distributions \(p_{\theta}(\cdot)\) and \(p_{\epsilon}(\cdot)\), respectively.

- If the particle stops, its energy is deposited and the energy value and the location of the hit are recorded.

- If the particle passes through, it simply continues with its original velocity.

See Algorithm 1 for the pseudocode of this process. We can think of the process as a detector layer being positioned at the \(z_0\)-slice recording energy deposits. After the slice at \(z_0\), the model either contains 0, 1 or 2 particles still propagating. As they reach the additional slices at \{\(z_1, \ldots, z_N\}\} the above procedure is repeated: whenever a particle is at a \(z\)-slice, it either passes, deposits its energy or decays.

After all particles crossed the last slice at \(z_N\) we collect all energy deposits (energy value and 3D location). Instead of building a dataset from the exact 3D location of these deposits, we partition the slices into subsets we call layers. We refer to the partitioning scheme as layer configuration.

We then build an event from the energy deposit \(E\) and the \((x, y)\)-coordinate of all hits for all slices within a given layer \(L_i\), effectively projecting the z-axis into a single point. An event then consists of \(N\) 2D point clouds, one for each layer, with a single scalar feature per point. Intuitively, the slices \(z_i\) are the positions where the particle can split and stop (granularity of the simulation), and the layers \(L_i\) correspond to the detector layers, where the readout happens. Figure 2 contains a schematic summary for the propagation process.

Having complete control over the algorithm, we can increase/decrease the complexity of the generated data by adjusting the distribution of the properties of the initial particle, the number and location of the slices \(z_i\), the probabilities \(p_{\text{stop}}, p_{\text{pass}}, p_{\text{split}}\) and the distribution of the properties of the children after a splitting event. This is especially useful to debug and understand the behaviour of machine learning models at different levels for complexity.

<table>
<thead>
<tr>
<th></th>
<th>GEANT4</th>
<th>SUPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>codebase</td>
<td>3.6m lines in C++</td>
<td>few hundred lines in Python</td>
</tr>
<tr>
<td>target usage</td>
<td>detailed physics</td>
<td>fast data generation</td>
</tr>
<tr>
<td></td>
<td>simulations</td>
<td></td>
</tr>
<tr>
<td>speed</td>
<td>1772 ms/shower</td>
<td>0.1-100 ms/shower</td>
</tr>
</tbody>
</table>

\[\text{A continuous attribute obeying simple conservation laws at each splitting, which can be interpreted as an equivalent for the energy of the particles in a shower. As such, we refer to this property as the energy.}\]
Algorithm 1 SUPA - Particle Splitting

Parameters:
- Stopping probability: $p_{\text{stop}}$
- Splitting probability: $p_{\text{split}}$
- Layer configuration:
  - $L_1 = \{z_0, \ldots, z_1\}$
  - $L_2 = \{z_1+1, \ldots, z_2\}$
  - \ldots
  - $L_n = \{z_{n-1}+1, \ldots, z_N\}$
- Slice Gap: $\Delta z$
- Splitting angle: $\phi_s(\cdot)$
- Deviation angle: $\phi_d(\cdot)$

Input:
- Tree $T$, Position $p = (\Delta \eta, \Delta \phi)$
- Energy $E$, Layer $l$

Output:
- a collection of $\{p_i, E_i\}_N$ of generated particles

1: $(p, E, l) \leftarrow \text{Input}, N \leftarrow p(\delta(n = 2))$

2: $\theta \sim p_{\theta}(\cdot), \epsilon \sim p_\epsilon(\cdot), \alpha \sim \mathcal{U}(0, 1)$
3: if $\alpha < p_{\text{stop}}(l)$ and $l > 0$ then
4:  $\text{Active} \leftarrow \text{False}$
5: else if $p_{\text{stop}}(l) < \alpha < p_{\text{stop}}(l) + p_{\text{split}}$ then
6:  $\omega \leftarrow \mathcal{U}(0, 2\pi)$
7:  $p_0 \leftarrow p + \Delta z \cdot (\sin(\theta/2 + \epsilon), \cos(\theta/2 + \epsilon))$
8:  $p_1 \leftarrow p - \Delta z \cdot (\sin(\theta/2 + \epsilon), \cos(\theta/2 + \epsilon))$
9:  $E_{(0,1)} \leftarrow E \cdot \theta/2 \pm \epsilon/\theta$
10: $T.add(p_0, l + 1, E_0), T.add(p_1, l + 1, E_1)$
11: else
12:  $T.add(p, l + 1, E)$
13: end if
14: Return $T$

Notation. We denote a single shower as $x = \{x_{L_1}, x_{L_2}, \ldots, x_{L_N}\}$, where $x_{L_i}$ denotes the $i$th layer and $N$ is the total number of layers considered. For the point cloud representation, $x_{L_i}$ can be further expanded as $x_{L_i} = \{x_{i1}, x_{i2}, \ldots, x_{iN_i}\}$, where $N_i$ denotes the total number of points in layer $i$ of the shower (can vary across showers). Further, each point is a vector $x_j = [\eta_j, \phi_j, E_j]$, where $\eta$ and $\phi$ are the coordinates and $E$ denotes the energy.

4 Using SUPA

SUPA provides various meta-parameters which affect the data generation and various characteristics of the generated data. This allows to generate shower data with varying levels of complexity and thus enables incremental benchmarking. For instance, $p_{\text{split}}$ affects the number of emerging secondary particles and thus implicitly the number of hits (or points), while $p_{\text{stop}}$ controls the position (in depth) of the observed hits. $\theta$ controls the overall spread of the hits in the lateral plane and $\alpha$ controls the dynamic range of the observed energy values. The different configurations for these parameters is analogous to different detector geometry/material or perhaps a different particle.

4.1 SUPA Datasets

We propose five standard configurations and the resulting datasets SUPAv1-5 for our analysis and benchmarking, summarized in Table 2. All configurations have the same initial energy of 65 GeV, initial angle of $\pi/2$ and initial impact position $(0, 0)$, i.e. particles are incident perpendicular and at the center of the calorimeter. Section A.2 in the appendix gives more details and also shows the interpretation of these variations with respect to different summary variables. SUPAv1 is deterministic with respect to splitting and stopping probabilities as well as in $\theta$ and $\alpha$. It has the same number of points across all events and all the hits have the same energy value. SUPAv2-4 have the same configuration for splitting and stopping probabilities (see Sec. A.2.1), while SUPAv2 is deterministic in $\theta$ and $\alpha$. SUPAv3 is naturally more spread out than SUPAv2 (see Fig. 4b vs. Fig. 4c and also distributions for $\eta, \phi, r$ in Fig. 5 and $(\eta), (\phi), (r)$ in Fig. 6). SUPAv4 is similar to SUPAv3, except for the layer configuration. SUPAv4 only has a subset of the sub-layers by dropping the last 8 sub-layers from SUPAv3. While SUPAv3 captures the total initial energy in Layer 0, SUPAv4 has some energy leakage which is apparent from the distribution of layer energy $E$ in Fig. 5.

\footnote{The exact simulation time depends on the incident particle and its kinematic properties, as well as the detailed composition of the detector geometry in the GEANT4 model. These numbers are taken for the CaloGAN geometry introduced in Paghami et al. [2018].}

\footnote{Reflects the time to generate directly the point cloud representation, which is more granular than the corresponding GEANT4 generation.}
Table 2: Summary of different datasets generated with SUPA

<table>
<thead>
<tr>
<th>Dataset</th>
<th>α</th>
<th># Points</th>
<th>Layer Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPAv1</td>
<td>$\frac{\pi}{4}$</td>
<td>128</td>
<td>L0 : [5, 10]</td>
</tr>
<tr>
<td>SUPAv2</td>
<td>$\frac{\pi}{4}$</td>
<td>1, 91</td>
<td>L0 : [7, 20]</td>
</tr>
<tr>
<td>SUPAv3</td>
<td>$U(\frac{\pi}{6}, \frac{\pi}{10})$</td>
<td>1, 91</td>
<td>L0 : [7, 20]</td>
</tr>
<tr>
<td>SUPAv4</td>
<td>$U(\frac{\pi}{6}, \frac{\pi}{10})$</td>
<td>1, 84</td>
<td>L0 : [7, 12]</td>
</tr>
<tr>
<td>SUPAv5</td>
<td>$U(\frac{\pi}{6}, \frac{\pi}{10})$</td>
<td>[5, 280]</td>
<td>L0 : [7, 12]</td>
</tr>
</tbody>
</table>

interesting structure in shower data is the dynamic range of the energy values in a given shower. A high dynamic range of energy makes learning generative models difficult [Krause and Shih, 2021]. With SUPA, we can control the dynamic range via the parameter $\alpha$. SUPAv1 and SUPAv2 have fixed $\alpha(= 0)$, thus all the splits are symmetric leading to a low dynamic range (see Fig. 3d for histogram of variance in energy values). The other factor which affects the dynamic range is the layer configuration, more number of slices in a layer would lead to more dynamic range. SUPAv5 has multiple layers and has a higher $p_{\text{split}}$ in the initial slices, thus it is more complex in terms of multiplicity or total number of hits. Please refer to the appendix Sec. A.2 for more details and other canonical SUPA datasets.

4.2 Extensions

SUPA is very flexible and can be easily extended. The slices can be further divided into different regions each with their own meta-parameters providing more granular control over the scattering and splitting process. Another variation requires having some slices that are not merged into layers to mimic dead material. These extensions can possibly bring it closer to more realistic detectors, and lead to unique structures in the dataset and thus present challenges for model development and evaluation. It is also possible to condition the data generation on various attributes, for example: initial impact angle, impact energy, impact position, etc. This encourages interesting studies and benchmarks regarding extrapolation and generalization capabilities of generative models for unseen conditioning variables during training.

5 Evaluation and Analysis

The evaluation of generative models for high dimensional data is a non-trivial problem. While reconstruction losses and/or data likelihood are available for some model architectures (e.g. auto-encoders or flows), judging the sample quality is still difficult. Moreover, the average log-likelihood is difficult to evaluate or even approximate for many interesting models. For instance, in computer vision, evaluating generative models for images is done via proxy measures such as the Frechet Inception Distances (FID, Heusel et al., 2017). Point cloud generative models are evaluated [Yang et al., 2019] using metrics such as Minimum matching distance (MMD), Coverage (COV) and 1-Nearest Neighbour Accuracy (1-NNA), where similarity between a set of reference point clouds and generated point clouds are measured using a distance metric such as the Chamfer distance (CD) or Earth mover’s distance (EMD) based on optimal matching.

For calorimeter simulation, we cannot judge performance using individual examples due to the stochastic nature of shower development. The standard practice to estimate the quality of generative models for calorimetry data (shower simulations) is to use histograms of different metrics called shower shape variables [Paganini et al., 2018] in addition to images of calorimeter showers. These shower shape variables are domain-specific and physically motivated, and matching densities over these marginals is indicative of matching salient aspects of the data. These marginals capture various aspects of how energy is distributed within individual layers and across different layers.
5.1 Shower Shape Variables

In this section, we will present the extension of the shower shape variables for the point cloud representation of showers.

Point level marginals. Marginals of each point feature by considering the set all the points from all the point clouds together.

Feature means $\langle \eta_i \rangle, \langle \phi_i \rangle, \langle r_i \rangle, \langle E_i \rangle$. Mean of each feature.

$$
\langle \eta_i \rangle = \frac{\sum_j \eta^i_j}{\sum_j 1}, \quad \langle \phi_i \rangle = \frac{\sum_j \phi^i_j}{\sum_j 1}, \quad \langle r_i \rangle = \frac{\sum_j r^i_j}{\sum_j 1}, \quad \langle E_i \rangle = \frac{\sum_j E^i_j}{\sum_j 1}
$$

where $r^i_j = \sqrt{(\eta^i_j)^2 + (\phi^i_j)^2}$ denotes the distance of the point in the lateral plane from the center.

Feature variances $\sigma(\eta_i), \sigma(\phi_i), \sigma(r_i), \sigma(E_i)$. Variance of each feature. $\sigma(\eta_i) = \sqrt{\frac{\sum_j (\eta^i_j)^2}{\sum_j 1} - (\langle \eta_i \rangle)^2}$

Layer Energy $E_i$. Denotes the total energy deposited in layer $i$ of the shower. $E_i = \sum_{j \in N_i} E^i_j$.

Total Energy $E_{tot}$. Total energy across all layers of the shower. $E_{tot} = \sum_{i \leq N} E_i$.

Layer Centroids $\langle \eta_i \rangle_E, \langle \phi_i \rangle_E, \langle r_i \rangle_E$. Energy weighted mean of the features ($\eta, \phi, \text{or } r$).

$$
\langle \eta_i \rangle_E = \frac{\sum_j E^i_j \eta^i_j}{E_i}, \quad \langle \phi_i \rangle_E = \frac{\sum_j E^i_j \phi^i_j}{E_i}, \quad \langle r_i \rangle_E = \frac{\sum_j E^i_j r^i_j}{E_i}
$$

The layer centroids can be interpreted as the center of energy in the lateral plane in respective dimensions.

Layer Lateral Width $\sigma(\eta_i)_E, \sigma(\phi_i)_E, \sigma(r_i)_E$. Denotes the standard deviation of the layer centroids.

$$
\sigma(\eta_i)_E = \sqrt{\frac{\sum_j E^i_j (\eta^i_j)^2}{E_i} - (\langle \eta_i \rangle_E)^2}
$$

The layer lateral widths can be interpreted as the spread around the center of energy in the lateral plane in respective dimensions. We drop the layer notation $i$ from the above metrics when working with a single layer for brevity.

5.2 Performance Analysis of Generative Models

In order to summarise the discrepancy into a scalar value, we consider the Wasserstein-1 distance between the histograms of the ground truth and generated sample’s marginal. Further, we compute the mean discrepancy over different groups of marginals to summarize the performance of models.

We train point cloud generative models, PointFlow [Yang et al. 2019], SetVAE [Kim et al. 2021], and a transformer-based flow model (which we call Tranflowmner, see Sec. 5.3), on SUPA data.

Table 5 summarizes the performance of different models across different variations of SUPA datasets. Pointflow performs better than SetVAE for all marginals not involving energy feature such as point level marginals, feature means and variances for location parameters, etc., and worse for marginals which depend on energy such as, layer centroids, and layer energy, etc. (with the exception of layer lateral widths and feature variance $\sigma(\langle E_i \rangle)_E$, which are both second order moments), across all datasets.

Roughly speaking, PointFlow struggles at modeling the energy feature while it is competitive for the location features, and SetVAE behaves opposite. It is interesting to observe that Tranflowmner is either the best performing or very close to best performing model (either SetVAE or PointFlow) across all datasets and marginals, indicating relatively greater flexibility at modeling a variety of features. The instances with relatively high values in Table 5 (feature means for SUPAv1), and layer

---

Footnote: For computing the Wasserstein distance, we normalize the shower shape variables for the ground truth and the generated sample according to the mean and standard deviation of the ground truth.
energy for SUPAv2 and SUPAv3) are where the marginals had a very narrow spread, indicating the difficulty in modeling discrete distributions. The models also have the tendency to predict points with very high energy values unseen during training (see Fig. 2a). We provide more details in the Appendix Section 4.4 on the training, results and analysis, along with plots of various shower shape variables to visually compare the goodness of fit.

These observations highlight the need for adapting point cloud generative models such that they are able to model points with special features (different from location parameters), such as physical features like energy, momentum, etc. Although the location parameters are correlated with these special features, the overall structure of this correlation can be very different from domain to domain and also from how they are correlated with other location features. To highlight more on this, we did experiments with a point-cloud shower dataset generated with GEANT4 and with SetMNIST which is a representative point-cloud shape dataset. We provide more details on the same in the Appendix.

Table 3: Performance benchmarks across different datasets with SetVAE, PointFlow and Transformer. The distance metric is Wasserstein-1. The reported numbers are averages over a group of marginals as indicated in the top row. Lower numbers are better.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Point Features : $\eta$, $\phi$, $r$</th>
<th>Point Feature : $E$</th>
<th>$\langle \eta \rangle$, $\langle \phi \rangle$, $\langle r \rangle$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPAv1</td>
<td>SV 0.044, PF 0.037, TF 0.028</td>
<td>SV 0.001, PF 0.000, TF 0.000</td>
<td>SV 16.254, PF 21.921, TF 5.789</td>
</tr>
<tr>
<td>SUPAv2</td>
<td>SV 0.290, PF 0.167, TF 0.029</td>
<td>SV 0.253, PF 0.462, TF 0.145</td>
<td>SV 0.517, PF 0.256, TF 0.091</td>
</tr>
<tr>
<td>SUPAv3</td>
<td>SV 0.510, PF 0.144, TF 0.044</td>
<td>SV 0.304, PF 0.484, TF 0.048</td>
<td>SV 0.769, PF 0.075, TF 0.105</td>
</tr>
<tr>
<td>SUPAv4</td>
<td>SV 0.209, PF 0.122, TF 0.029</td>
<td>SV 0.111, PF 0.459, TF 0.006</td>
<td>SV 0.460, PF 0.094, TF 0.073</td>
</tr>
<tr>
<td>SUPAv5</td>
<td>SV 0.166, PF 0.042, TF 0.078</td>
<td>SV 0.090, PF 0.151, TF 0.154</td>
<td>SV 0.592, PF 0.031, TF 0.189</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$\sigma_{\langle \eta \rangle}, \sigma_{\langle \phi \rangle}, \sigma_{\langle r \rangle}$</th>
<th>$\langle E \rangle$</th>
<th>$\sigma_{\langle E \rangle}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPAv1</td>
<td>SV 0.513, PF 0.585, TF 0.359</td>
<td>SV 0.001, PF 0.000, TF 0.000</td>
<td>SV 0.000, PF 0.000, TF 0.000</td>
</tr>
<tr>
<td>SUPAv2</td>
<td>SV 0.648, PF 0.154, TF 0.130</td>
<td>SV 0.302, PF 0.077, TF 0.087</td>
<td>SV 0.153, PF 0.177, TF 0.165</td>
</tr>
<tr>
<td>SUPAv3</td>
<td>SV 1.114, PF 0.109, TF 0.126</td>
<td>SV 0.320, PF 0.064, TF 0.071</td>
<td>SV 0.500, PF 0.152, TF 0.144</td>
</tr>
<tr>
<td>SUPAv4</td>
<td>SV 0.634, PF 0.092, TF 0.051</td>
<td>SV 0.263, PF 0.047, TF 0.022</td>
<td>SV 0.355, PF 0.156, TF 0.038</td>
</tr>
<tr>
<td>SUPAv5</td>
<td>SV 0.799, PF 0.040, TF 0.223</td>
<td>SV 0.251, PF 0.059, TF 0.414</td>
<td>SV 0.377, PF 0.047, TF 0.421</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$\langle \eta \rangle_{E}, \langle \phi \rangle_{E}, \langle r \rangle_{E}$</th>
<th>$\sigma_{\langle \eta \rangle_{E}}, \sigma_{\langle \phi \rangle_{E}}, \sigma_{\langle r \rangle_{E}}$</th>
<th>$E$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPAv1</td>
<td>SV 16.244, PF 21.921, TF 5.766</td>
<td>SV 0.517, PF 0.591, TF 0.379</td>
<td>SV 0.101, PF 0.000, TF 0.039</td>
</tr>
<tr>
<td>SUPAv2</td>
<td>SV 1.336, PF 1.933, TF 0.137</td>
<td>SV 0.779, PF 0.134, TF 0.190</td>
<td>SV 23.387, PF 69.814, TF 10.468</td>
</tr>
<tr>
<td>SUPAv3</td>
<td>SV 1.365, PF 1.627, TF 0.217</td>
<td>SV 1.226, PF 0.147, TF 0.151</td>
<td>SV 36.116, PF 79.062, TF 5.916</td>
</tr>
<tr>
<td>SUPAv4</td>
<td>SV 1.369, PF 1.346, TF 0.083</td>
<td>SV 0.645, PF 0.169, TF 0.067</td>
<td>SV 3.997, PF 12.895, TF 0.659</td>
</tr>
<tr>
<td>SUPAv5</td>
<td>SV 2.373, PF 1.615, TF 0.463</td>
<td>SV 0.740, PF 0.058, TF 0.317</td>
<td>SV 6.132, PF 16.742, TF 9.273</td>
</tr>
</tbody>
</table>

6 Conclusion

We introduced SUPA, a lightweight pseudo-particle simulator that is inspired by the physics governed development of particle showers and qualitatively resembles the data generated by the gold standard GEANT4. By allowing to easily change the underlying parameters of the propagation model, we introduce the freedom to define new benchmark datasets of varying complexity with a fidelity and simplicity not available using the GEANT4 toolkit.

We believe that the ease of use, flexibility and speed of this simulator could be a MNIST moment for the field, allowing the rapid exploration of models in various regime of functioning as discussed in § 5.1. We also highlight the gaps in the ability of existing point cloud generative models on calorimetry data in § 5.2 and highlight the need for more powerful point cloud generative models.

Additionally, we showed in § A.5.1 that with the grid representation of the shower data, the performance of deep generative models estimated on data produced with this simulator is a good proxy for their performance on the standard data used historically in machine learning for particle physics. We highlight the flexibility of SUPA in § A.5.2 allowing us to train and evaluate at different resolutions.
Moreover, by giving easy access to the fully recorded event, we open the way to applying methods from point clouds and geometric deep learning research, with the goal of possibly devising end-to-end architectures able to reconstruct a full event from the energy deposits by solving the complete inverse problem, something currently impossible with data from Geant4.

References


Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]