Physics Department ”E. Fermi”
University of Pisa

EM-shower-simulator-with-NN

An EM shower simulation toolkit based on Generative Adversarial Network

Daniele Passaro
d.passaro1@studenti.unipi.it

Dario Cafasso
d.cafasso@studenti.unipi.it

March 4, 2022

Contents

Introduction

Package structure

Geant4 simulations
- Code and data pre-processing

GAN structure
- Generator
- Discriminator
- GAN class
- Losses implementation

Analysis
- Performances evaluation algorithm
- Analysis results

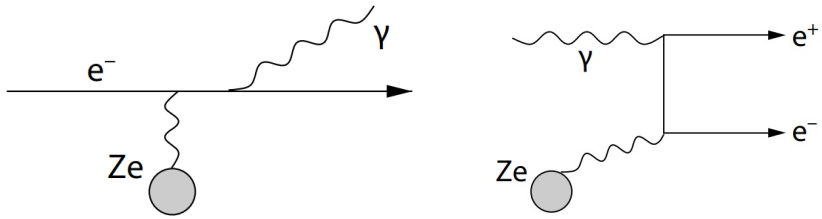
Conclusions

Introduction

Physics and reasons for our work

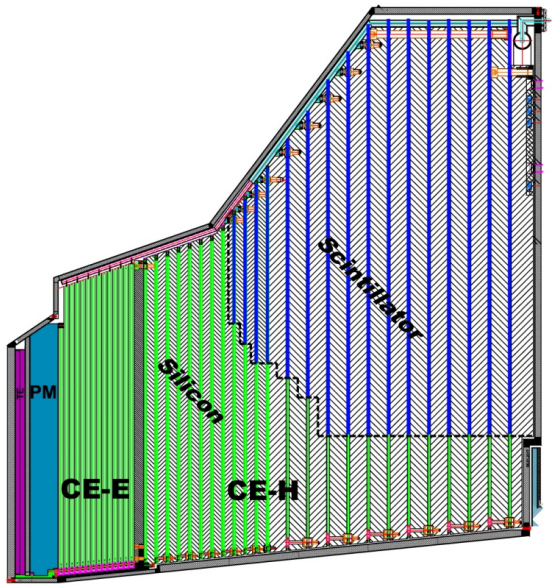
EM showers develops through bremsstrahlung and pair production processes. Main physical paramaters:

- X_0 = radiation lenght;
- λ_γ = photon absorption lenght;
- $R_M = \frac{E_S}{E_c} X_0$;



(a) Bremsstrahlung. (b) Pair production.

Calorimeters : homogeneous vs sampling design. Recently, many experiments are redesigning their calorimeters in order to achieve high energy resolutions (high segmentation).



Package structure

We developed the project on GitHub using a package structure. In particular:

Package structure

We developed the project on GitHub using a package structure. In particular:

- the documentation has been created with Sphinx using an API and the README files in each folder.

Package structure

We developed the project on GitHub using a package structure. In particular:

- the documentation has been created with Sphinx using an API and the README files in each folder.
- the compatibility of the various parts of the project is ensured using a test-driven development (TDD) approach and the unittest module.

Package structure

We developed the project on GitHub using a package structure. In particular:

- the documentation has been created with Sphinx using an API and the README files in each folder.
- the compatibility of the various parts of the project is ensured using a test-driven development (TDD) approach and the unittest module.
- the versioneer module allows automatic version detection.

Package structure

We developed the project on GitHub using a package structure. In particular:

- the documentation has been created with Sphinx using an API and the README files in each folder.
- the compatibility of the various parts of the project is ensured using a test-driven development (TDD) approach and the unittest module.
- the versioneer module allows automatic version detection.
- the setup.cfg file contains the package metadata and access-points (CLI), while the setup.py file finds dependencies and modules to install.

Package structure

We developed the project on GitHub using a package structure. In particular:

- the documentation has been created with Sphinx using an API and the README files in each folder.
- the compatibility of the various parts of the project is ensured using a test-driven development (TDD) approach and the unittest module.
- the versioneer module allows automatic version detection.
- the setup.cfg file contains the package metadata and access-points (CLI), while the setup.py file finds dependencies and modules to install.

This package can be installed through pip as showed in the documentation: `https://em-shower-simulator-with-nn.readthedocs.io/en/latest/?badge=latest`.

Geant4 simulations
Code and data pre-processing

Geant4 simulations are performed with the precompiled executable GEARS (released for Windows, macOs and Linux):

```
gears.exe simulazione.mac
```

Parameters of the simulation:

- Detector: CsI material, $12 \times 25 \times 25$ cells, $1 \times 1 \times 5\,\mathrm{cm}^3$ wide;
- Particle source: 50% γ, 50% $e^{\pm}$; energy range $[1\,\mathrm{GeV}, 30\,\mathrm{GeV}]$;
- Analysis manager: tree filled with track ID, xyz position, physical processes involved at each step, ...

```
*********************************************************************************************************
* G4Track Information:   Particle = gamma,   Track ID = 1,   Parent ID = 0
*********************************************************************************************************

Step#    X(mm)    Y(mm)    Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0     -200      120      120  2.29e+03        0        0         0       world initStep
    1     -191      120      120  2.29e+03        0     8.98      8.98    step0(S) CoupledTransportation
    2     -189      120      120  2.29e+03        0        2        11       world CoupledTransportation
    3      -25      120      120  2.29e+03        0      164       175 cella1313(S) CoupledTransportation
    4    0.879      120      120         0        0     25.9       201 cella1313(S) conv
    :----- List of 2ndaries - #SpawnInStep=  2(Rest= 0,Along= 0,Post= 2), #SpawnTotal=  2 --------------
    :     0.879      120      120       474                 e-
    :     0.879      120      120  1.82e+03                 e+
    :---------------------------------------------------------- EndOf2ndaries Info --------------

*********************************************************************************************************
```

Geant4 simulations
Code and data pre-processing

Geant4 simulations are performed with the precompiled executable GEARS (released for Windows, macOs and Linux):

```
gears.exe simulazione.mac
```

Parameters of the simulation:

- Detector: CsI material, $12 \times 25 \times 25$ cells, $1 \times 1 \times 5\,\mathrm{cm}^3$ wide;
- Particle source: 50% γ, 50% $e^{\pm}$; energy range $[1\,\mathrm{GeV}, 30\,\mathrm{GeV}]$;
- Analysis manager: tree filled with track ID, xyz position, physical processes involved at each step, ...

Data pre-processing :

- overall normalization:

$$E^{\mathrm{norm}}_{l,i,j} = \log E_{l,i,j} \cdot \left(\max_{l,i,j} \{\log E_{l,i,j}\} \right)^{-1} \quad , \quad -1 \text{ if } E_{l,i,j} = 0$$

- new tree containing a $12 \times 25 \times 25 \times 1$ vector, $\mathrm{E_{in}}$, $\mathrm{E_{mis}}$, $\mathrm{P_{ID}}$.

GAN structure
Generator

Generator specifications

- Inputs:

latent_space : R^{1024} noise vector

energy_label : float energy value, from 1 to 30 GeV

particle_label : integer particle class, 0, 2 for $e^{\pm}$ and 1 for γ

- Combination of the noise and the particle ID
- Concatenation with the energy information
- Dense and Conv3DTranspose layers
- Output:

fake_image : tanh, tensor with shape (12, 25, 25, 1)

GAN structure
Discriminator

Discriminator specifications

- Input:

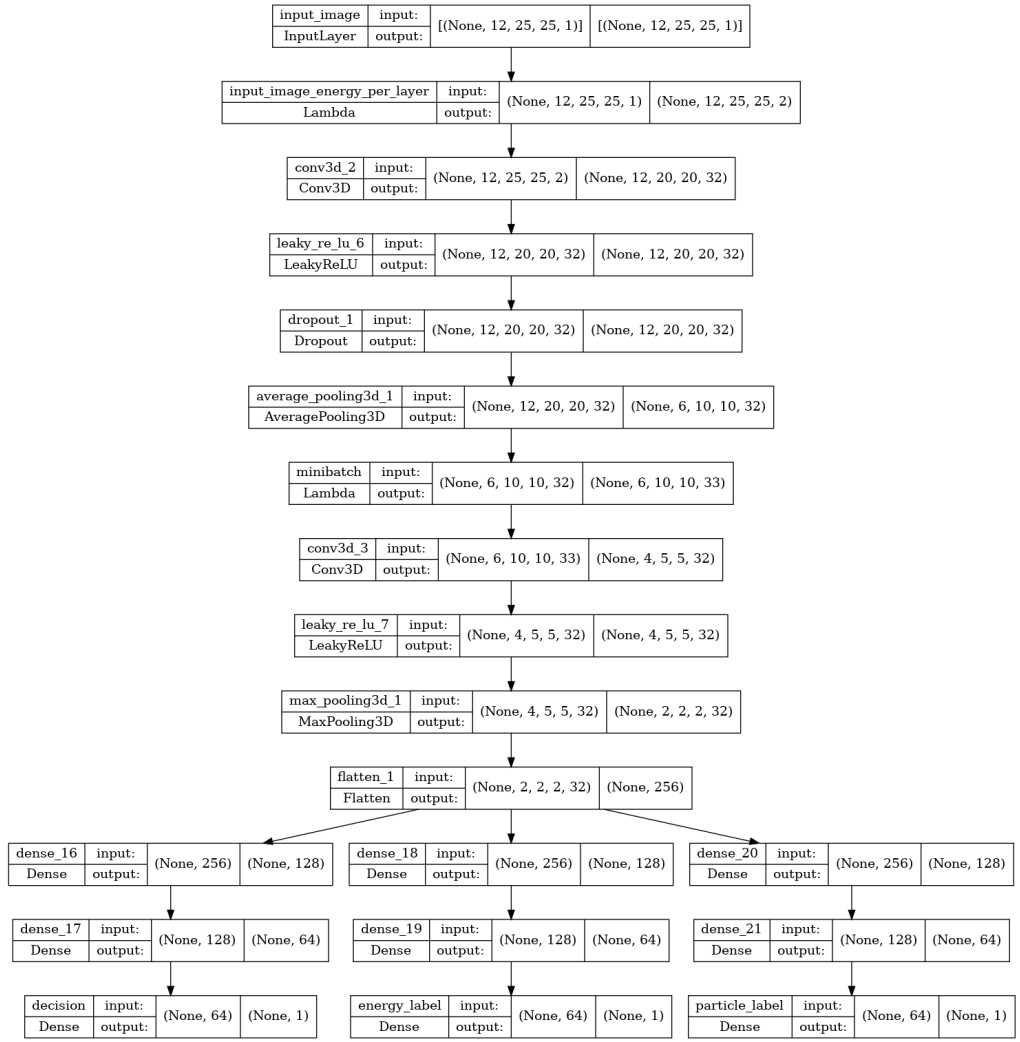
images : shape $(12, 25, 25, 1)$ and pixels $\in [-1, 1]$

- Concatenation of layer energies with features
- Pooling Layers
- Minibatch Std Deviation discrimination
- Outputs:

decision : sigmoid, decides whether the shower is true or fake

energy_label : relu, decides energy of the shower

particle_label : sigmoid, decides primary particle ID



GAN structure
GAN class

The class GAN inherits keras.Model properties and functions. The **constructor, compile, fit and summary** methods have been wrapped using the super function, while new important methods have been implemented, like:

GAN structure
GAN class

The class GAN inherits keras.Model properties and functions. The **constructor, compile, fit and summary** methods have been wrapped using the super function, while new important methods have been implemented, like:

- **generate_and_save_images**
  Use the current status of the NN to generate images from the noise, plot, evaluate and save them.

GAN structure
GAN class

The class GAN inherits keras.Model properties and functions. The **constructor, compile, fit and summary** methods have been wrapped using the super function, while new important methods have been implemented, like:

- **generate_and_save_images**
  Use the current status of the NN to generate images from the noise, plot, evaluate and save them.
- **train** and **train_step**
  Train the generator and the discriminator simultaneously, save checkpoints and print examples of fake_images.

GAN structure
GAN class

The class GAN inherits keras.Model properties and functions. The **constructor, compile, fit and summary** methods have been wrapped using the super function, while new important methods have been implemented, like:

- **generate_and_save_images**
  Use the current status of the NN to generate images from the noise, plot, evaluate and save them.
- **train** and **train_step**
  Train the generator and the discriminator simultaneously, save checkpoints and print examples of fake_images.
- **restore**
  Restore the last checkpoint and return generator and discriminator models for further operations.

Generator:

- $\text{gener\_loss} = \mathcal{L}_{BCE}(1, \mathcal{D}(\mathcal{G}))$

Discriminator:

- $\text{discr\_loss} = \mathcal{L}_{BCE}\left(0, \mathcal{D}(\mathcal{G})\right) + \mathcal{L}_{BCE}(1, \mathcal{D}(\mathcal{I}))$

Generator:

- $\text{gener\_loss} = \mathcal{L}_{BCE}(1, \mathcal{D}(\mathcal{G}))$
- $\text{comp\_en} = \lambda_E \cdot \mathcal{L}_{MSE}(E_{in}, E_{GAN}^{meas})$
- $\text{fake\_label\_en} = \lambda_E \cdot \mathcal{L}_{MSE}(E_{in}, E_{GAN}^{label})$
- $\text{fake\_part\_id} = \mathcal{L}_{BCE}(P_{in}, P_{GAN}^{label})$

Discriminator:

- $\text{discr\_loss} = \mathcal{L}_{BCE}(0, \mathcal{D}(\mathcal{G})) + \mathcal{L}_{BCE}(1, \mathcal{D}(\mathcal{I}))$
- $\text{real\_label\_en} = \lambda_E \cdot \mathcal{L}_{MSE}(E_{in}, E_{GEANT}^{label})$
- $\text{real\_part\_id} = \mathcal{L}_{BCE}(P_{in}, P_{GEANT}^{label})$

Generator:

- $\text{gener\_loss} = \mathcal{L}_{BCE}(1, \mathcal{D}(\mathcal{G}))$
- $\text{comp\_en} = \lambda_E \cdot \mathcal{L}_{MSE}(E_{in}, E_{GAN}^{meas})$
- $\text{fake\_label\_en} = \lambda_E \cdot \mathcal{L}_{MSE}(E_{in}, E_{GAN}^{label})$
- $\text{fake\_part\_id} = \mathcal{L}_{BCE}(P_{in}, P_{GAN}^{label})$

Discriminator:

- $\text{discr\_loss} = \mathcal{L}_{BCE}(0, \mathcal{D}(\mathcal{G})) + \mathcal{L}_{BCE}(1, \mathcal{D}(\mathcal{I}))$
- $\text{real\_label\_en} = \lambda_E \cdot \mathcal{L}_{MSE}(E_{in}, E_{GEANT}^{label})$
- $\text{real\_part\_id} = \mathcal{L}_{BCE}(P_{in}, P_{GEANT}^{label})$

We introduced two *unbiased metrics* to evaluate GAN performances in emulating shower samples during the training. The unbiased metric we use are physical properties of EM showers: **shower longitudinal depth** and **shower lateral width**

The performance evaluation is based on:

1. Deposited energy vs initial energy
2. Mean energy deposition per layer
3. Mean energy deposition per cell in each layer
4. Shower mean depth per primary particle ID:

$$\hat{d}(E_{in}, P_{ID}) = \frac{1}{N_{P_{ID}}} \sum_{P_{ID}} \left(\sum_{l=0}^{11} \frac{l \cdot E_l}{E_{in}} \right)$$

5. Shower depth's mean width per primary particle ID:

$$\hat{w}_{long}(E_{in}, P_{ID}) = \frac{1}{N_{P_{ID}}} \sum_{P_{ID}} \left(\sqrt{\sum_{l=0}^{11} \frac{l^2 \cdot E_l}{E_{in}} - \left(\sum_{l=0}^{11} \frac{l \cdot E_l}{E_{in}} \right)^2} \right)$$

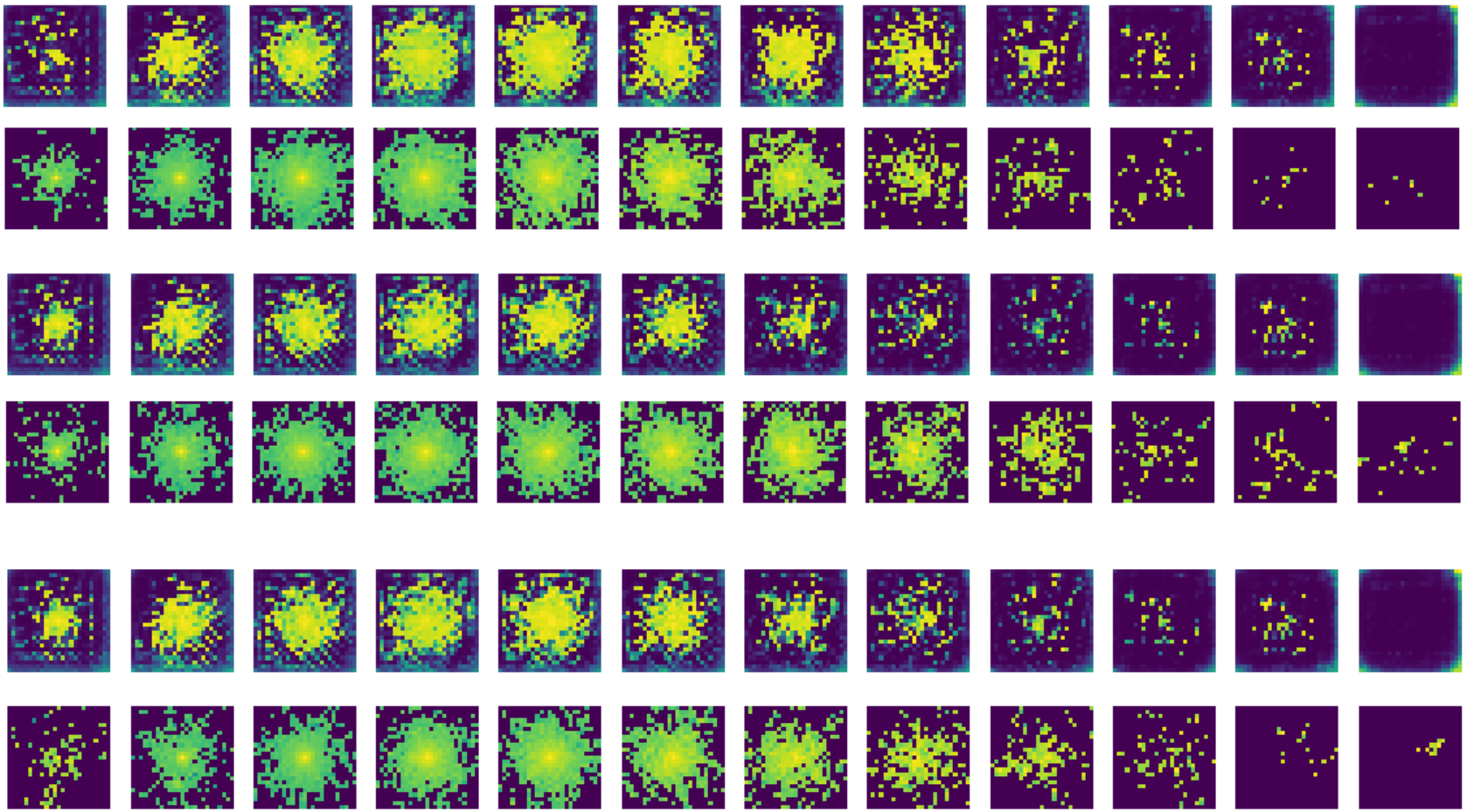
6. Shower mean lateral width per primary particle ID:

$$\hat{\sigma}(E_{in}, P_{ID}) = \frac{1}{N_{P_{ID}}} \sum_{P_{ID}} \frac{1}{12} \sum_{l=0}^{11} \left(\sqrt{\frac{l^2 \cdot E_l^w}{E_{in}} - \left(\frac{l \cdot E_l^w}{E_{in}} \right)^2} \right) \quad , \quad E_l^w = \sum_{n_x, n_y = 0}^{24} E_{l, n_x, n_y} \cdot (n_x - 12)$$

Analysis

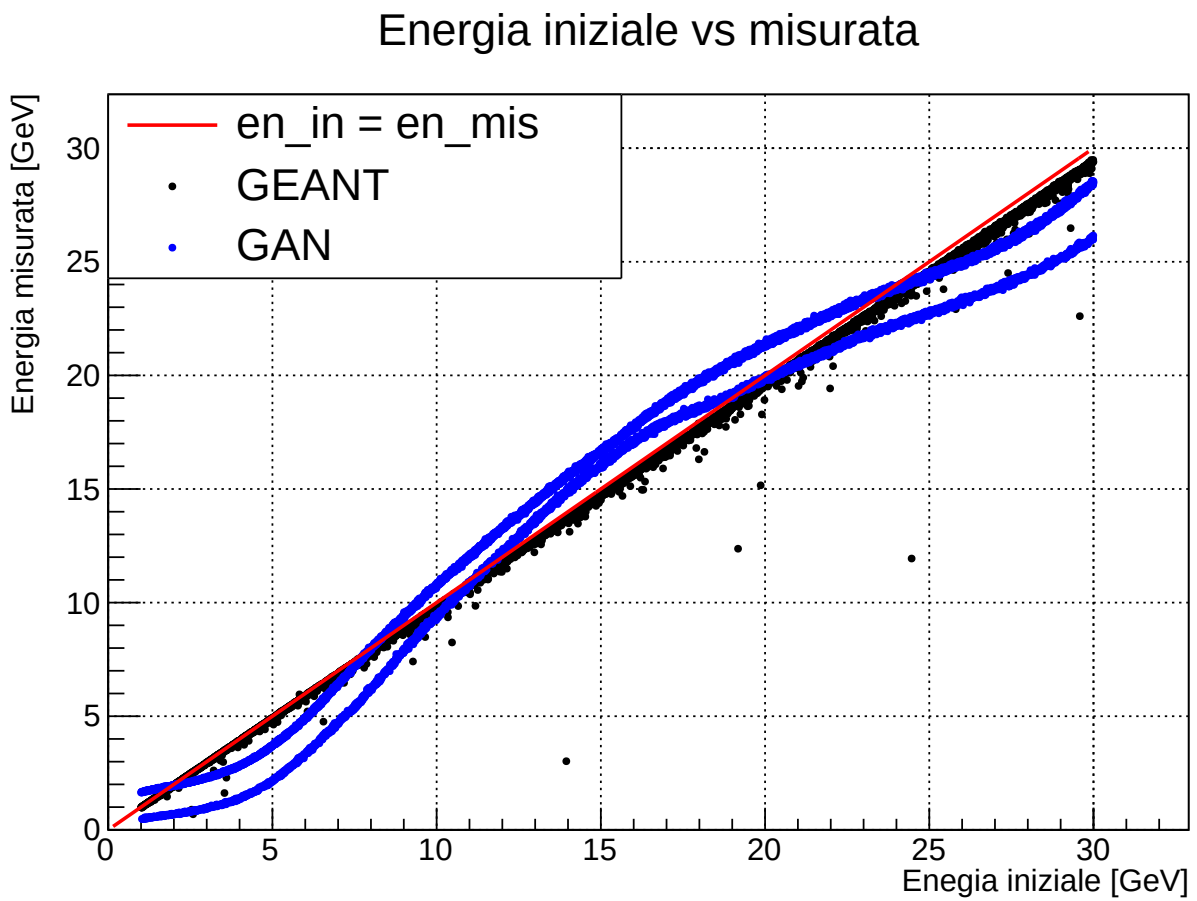
Results : shower comparison

GAN vs GEANT4



Analysis

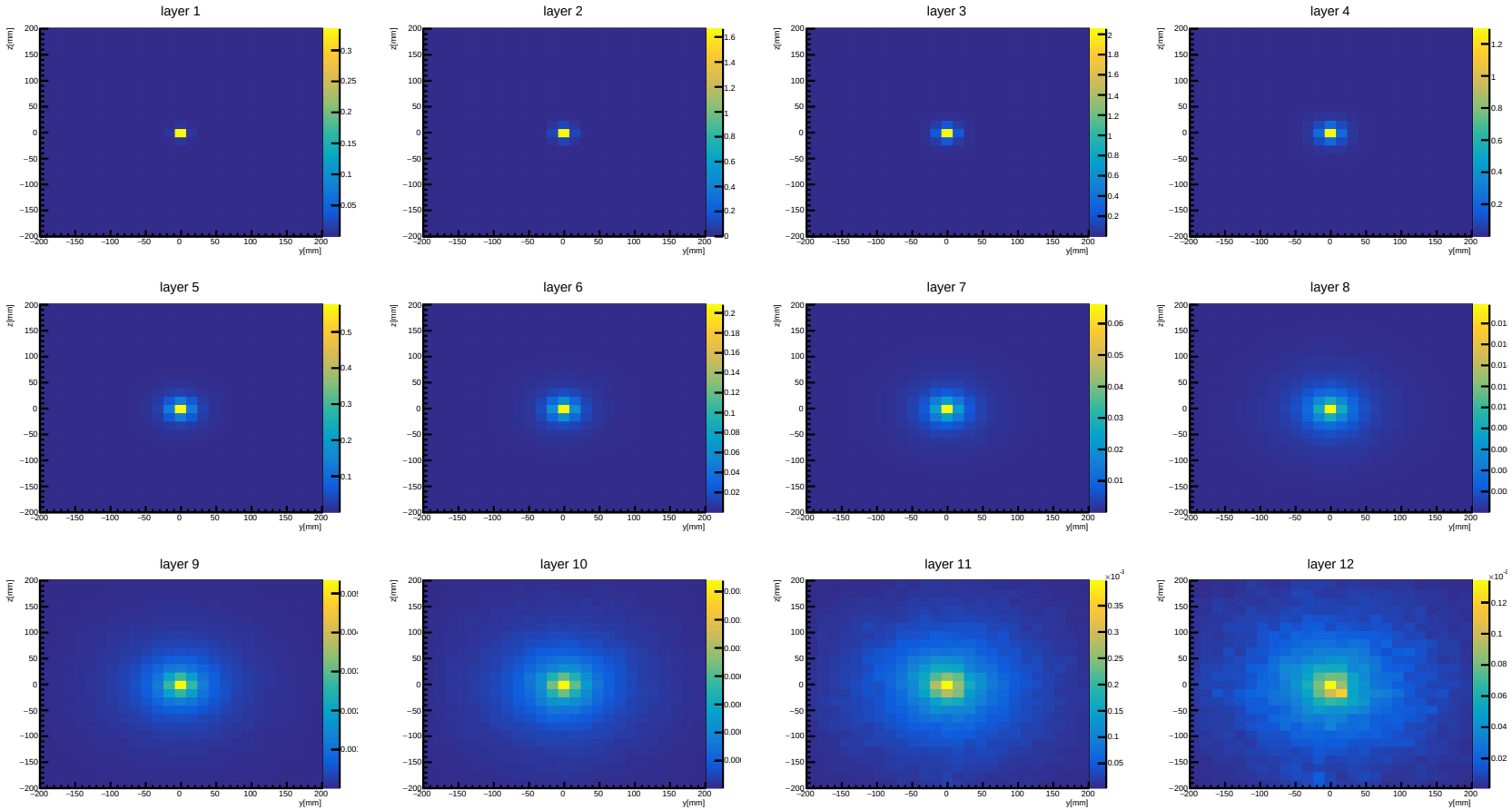
Results : Deposited energy vs initial energy



Analysis

Results : Mean energy deposition per cell per layer

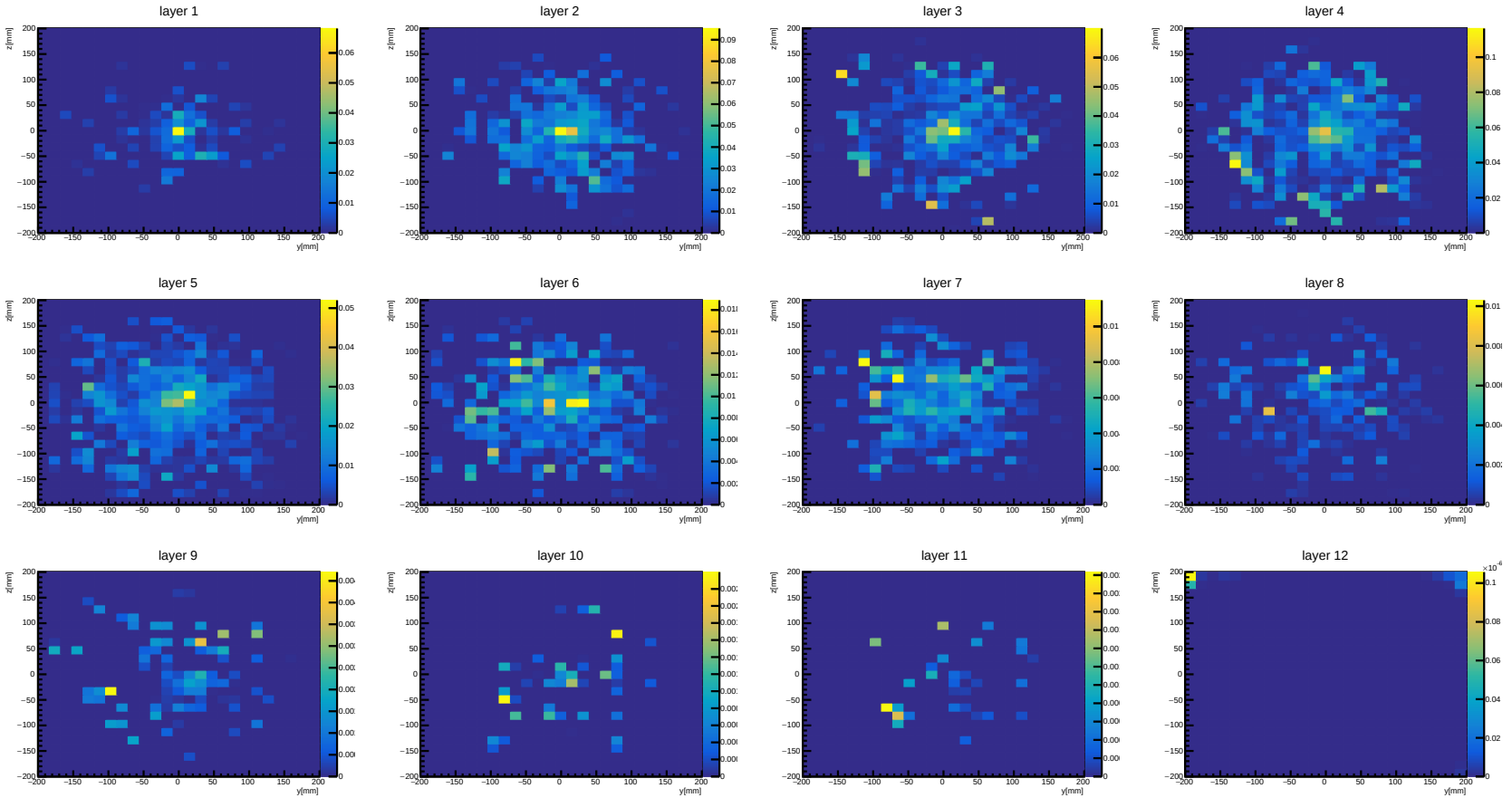
Geant4 samples:



Analysis

Results : Mean energy deposition per cell per layer

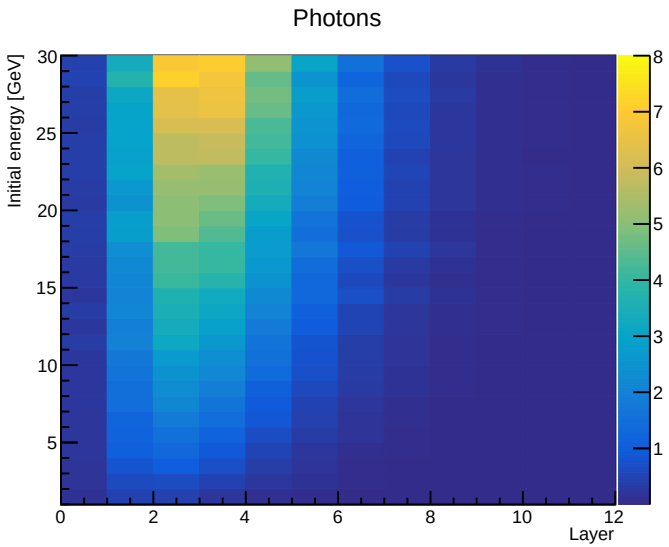
GAN samples:

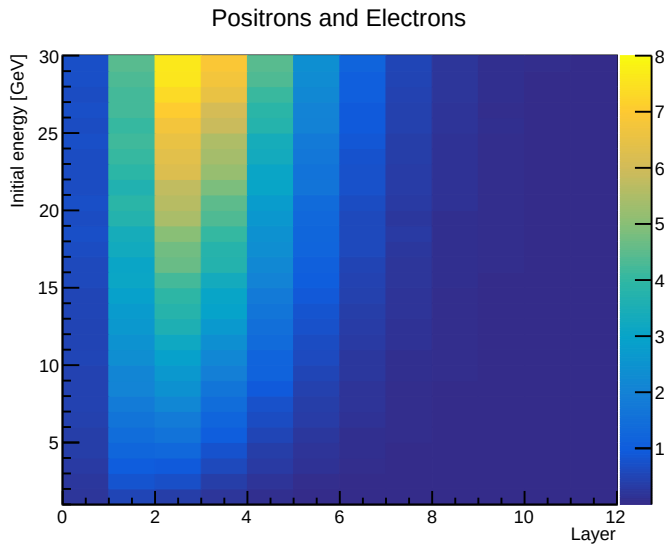


Analysis

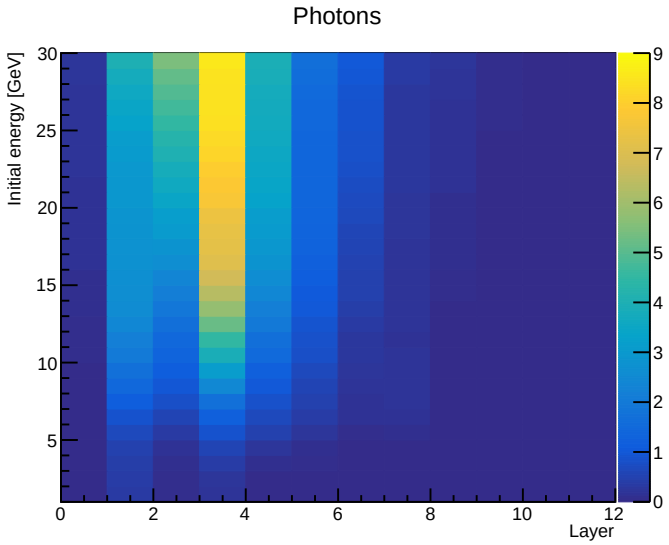
Results : Mean energy deposition per layer per primary particle energy

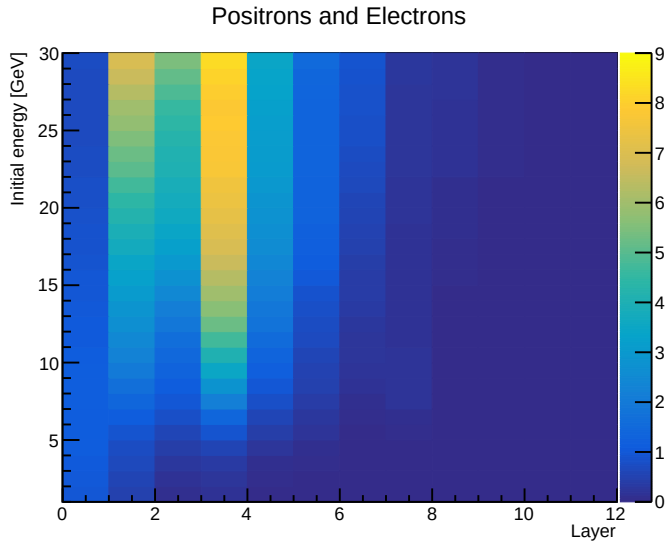
Geant4





GAN

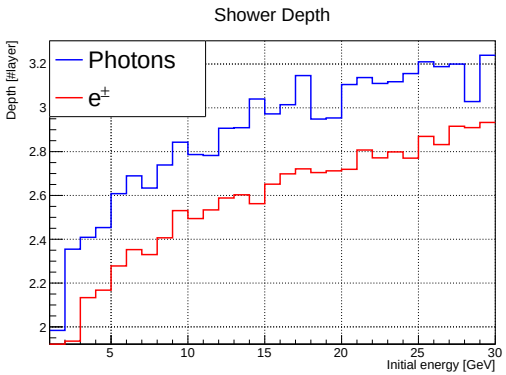


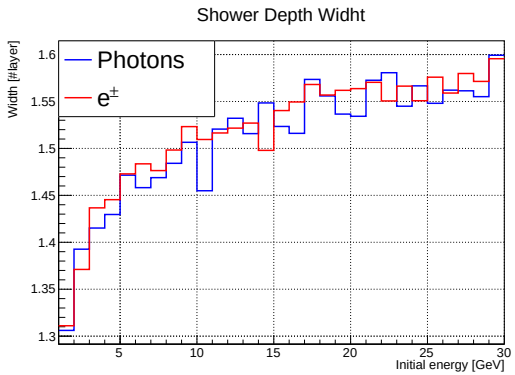


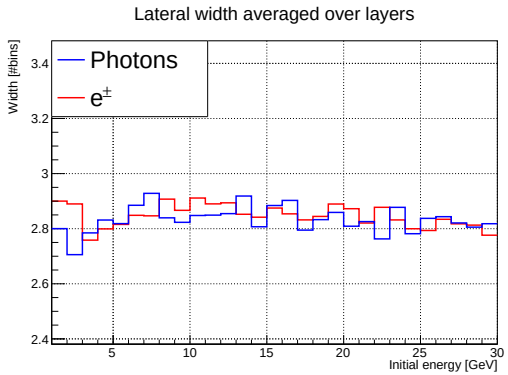
Analysis

Results : Shower mean depth; Shower depth width; Shower lateral width

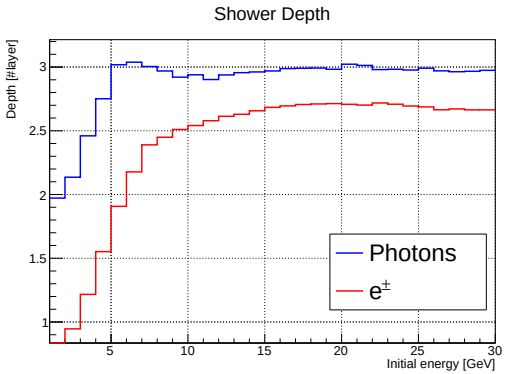
Geant4

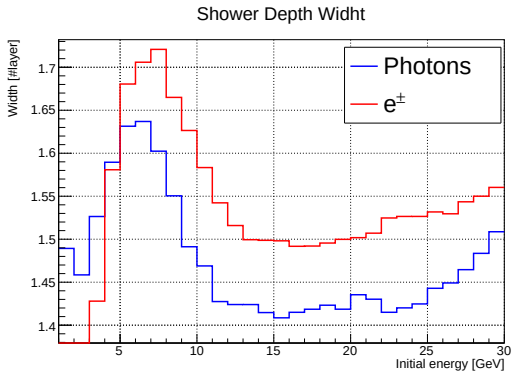


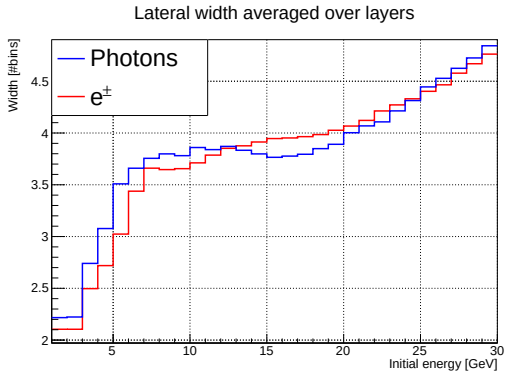




GAN







Conclusions
Package usage

It's possible to clone the repository and install the package typing:

```
$ git clone
https://github.com/Dario-Caf/EM-shower-simulator-with-NN.git
$ cd EM-shower-simulator-with-NN
$ python3 -m pip install -e .
```

Once installed, it can be used typing:

from bash

```
$ simulate-EM-shower -f 10. 1
```

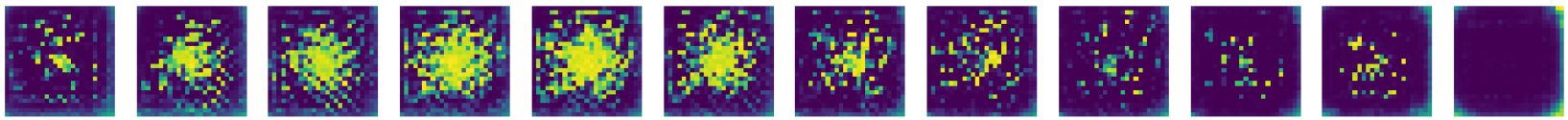
from Python

```
>>> import em_shower_simulator as em
>>> em.simulate([10., 1.], verbose=0)
```

Conclusions
Package usage

The output is...

```
simulating event with features: [10.0, 1.0]
Restored from ./checkpoints/ckpt-14
Example 1 Primary particle = 0 Predicted particle = 0.1849
Initial energy = 10.0 Generated energy = 9.3354
Predicted energy = 8.5973 Decision = 0.0021
The work is done.
```



Conclusions

What we achieved:

- the synthesis of different ideas and perspectives
- a complete package structure
- a basis for further ML works
- an evidence that ML techniques can be useful in future HEP

Conclusions

What we achieved:

- the synthesis of different ideas and perspectives
- a complete package structure
- a basis for further ML works
- an evidence that ML techniques can be useful in future HEP

What we learned:

- collaborative coding
- how to create a package
- create and train on purpose neural-networks models with personalized train_step
- how to use the ROOT toolkit to evaluate performances

Conclusions

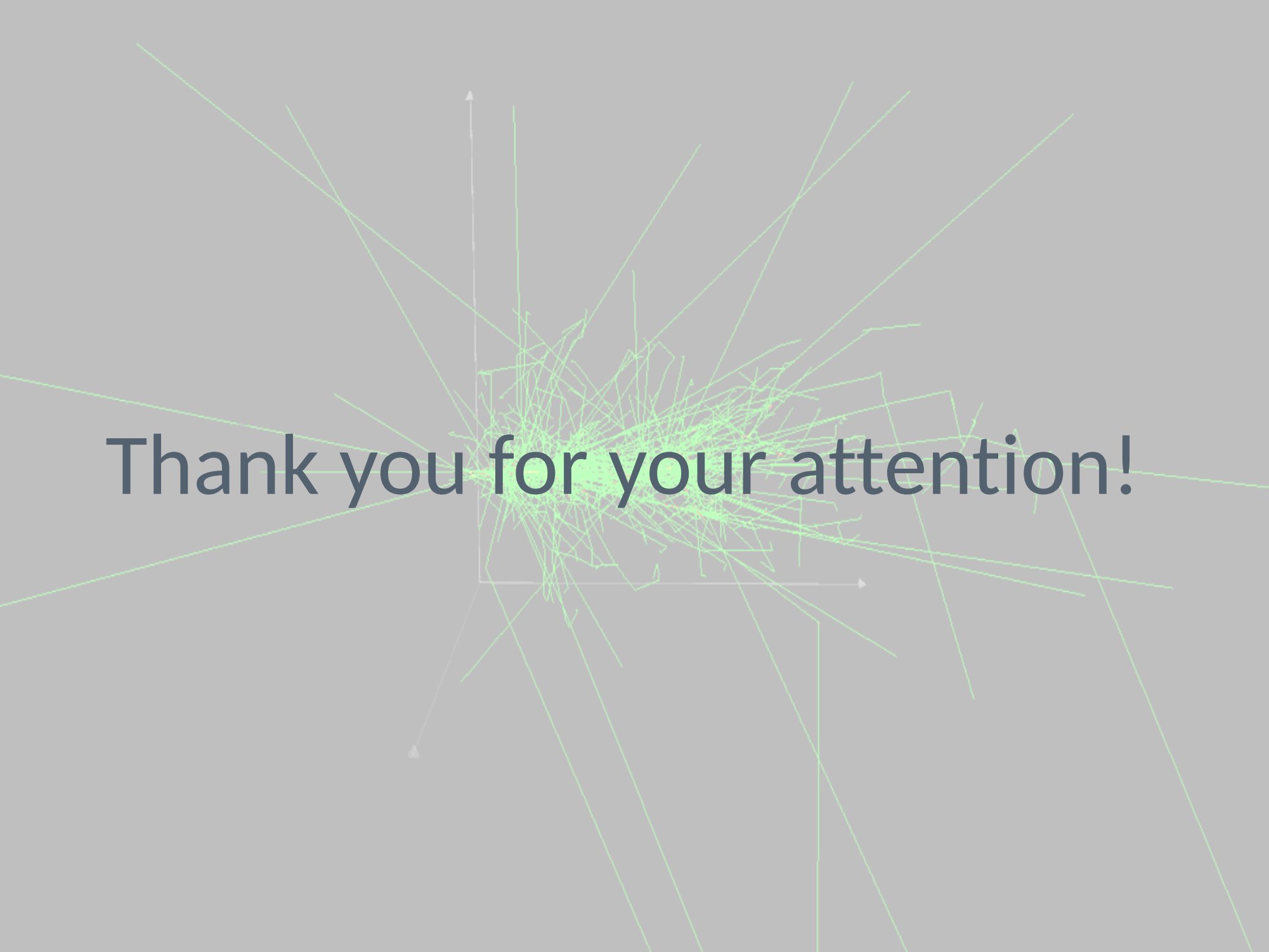
What we achieved:

- the synthesis of different ideas and perspectives
- a complete package structure
- a basis for further ML works
- an evidence that ML techniques can be useful in future HEP

What we learned:

- collaborative coding
- how to create a package
- create and train on purpose neural-networks models with personalized train_step
- how to use the ROOT toolkit to evaluate performances

- **Simulating physics is very difficult !!**


Thank you for your attention!