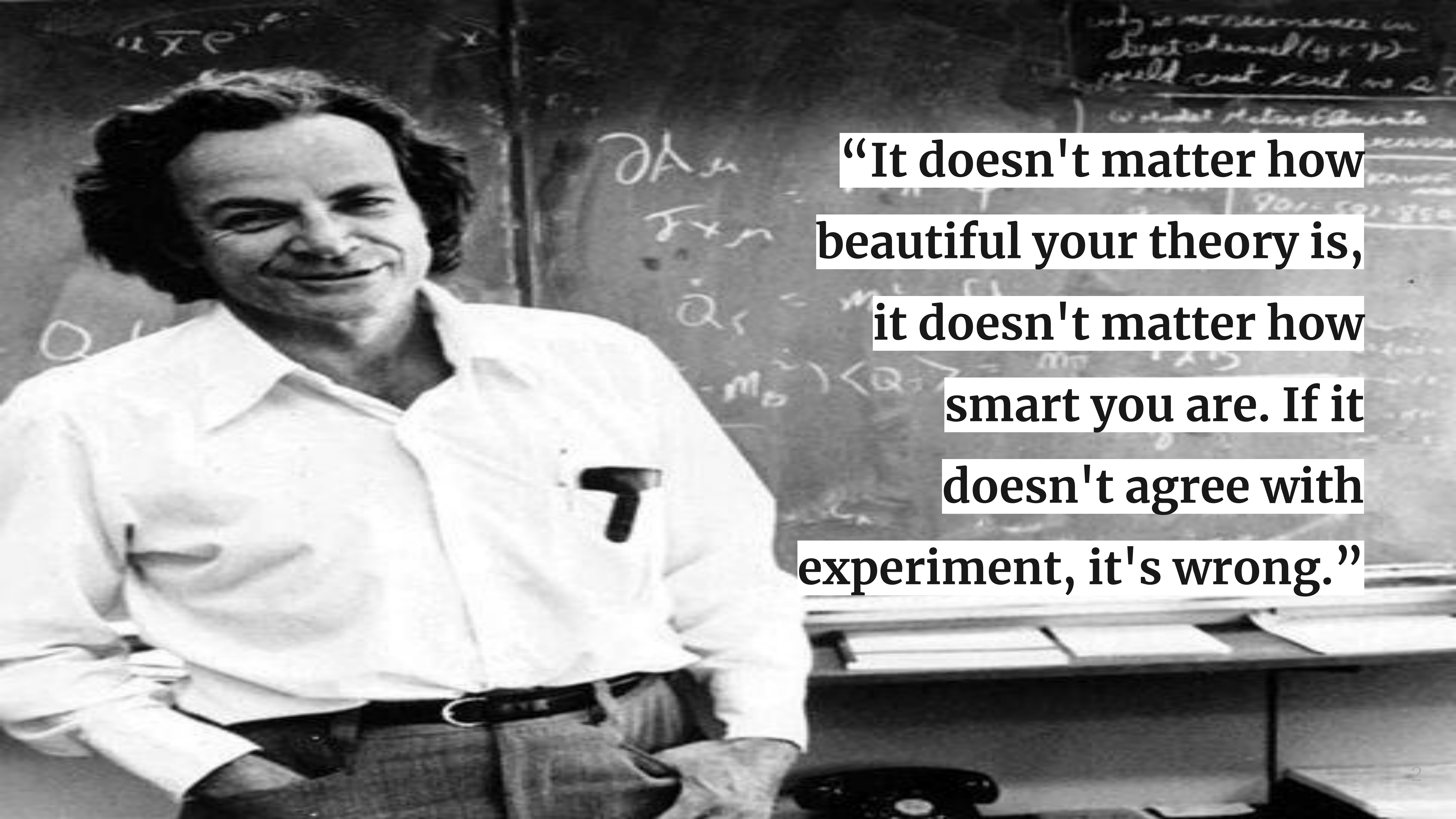
IMPROVING VIRTUAL COLLIDERS

PROJECT BY :
Chaitanya Paranjape, Aakanksha Dwivedi

TUTORS :
dr hab. Andrzej Siódmok (IFJ PAN)
dr Benjamin Nachman (Lawrence Berkeley National Laboratory)



“It doesn't matter how beautiful your theory is, it doesn't matter how smart you are. If it doesn't agree with experiment, it's wrong.”

MOTIVATION & AIM

Search for BSM Physics

Need of Precise MC Predictions.

Limited Understanding of QCD.

So, what can we do ?

Improve the phenomenological model of hadronization which is currently used at MC event generators.

STAGE 1

Setting up the training data
➔ For Further Machine Learning Analysis

Herwig 7.2

General-purpose MC event generator.

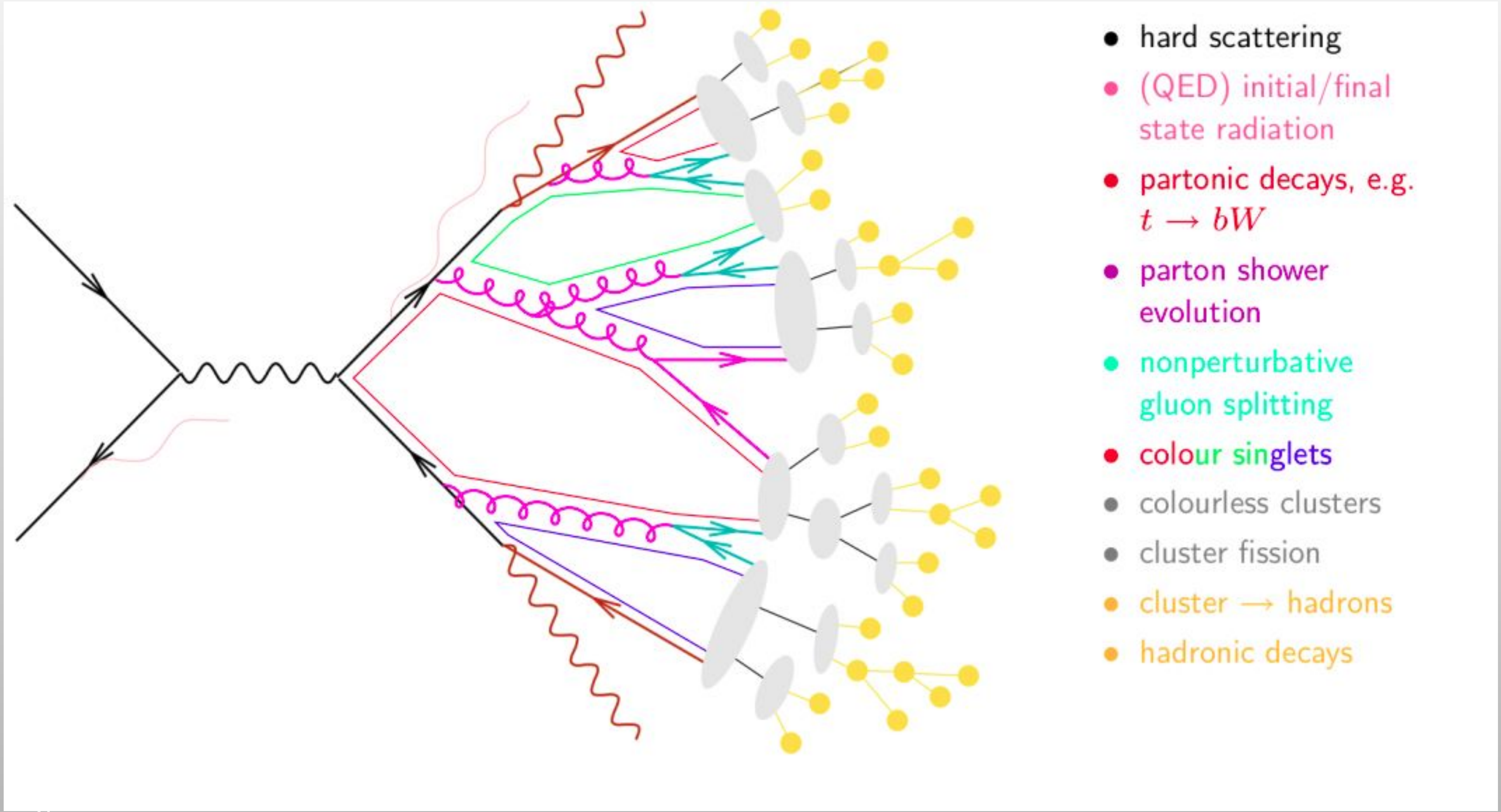
Simulation of high-energy hadron/lepton collisions.

Specializes Simulation of QCD interactions.

It currently uses Cluster Hadronization Model.

- *Which we aim to improve/replace.*

CLUSTER HADRONIZATION MODEL



● hard scattering
● (QED) initial/final state radiation
● partonic decays, e.g. $t \to bW$
● parton shower evolution
● nonperturbative gluon splitting
● colour singlets
● colourless clusters
● cluster fission
● cluster $\to$ hadrons
● hadronic decays

Setting up Analysis Handler

- Herwig uses various Analysis Handlers.
- We designed our own Analysis Handler
  - To access data of all Clusters and their decay processes.

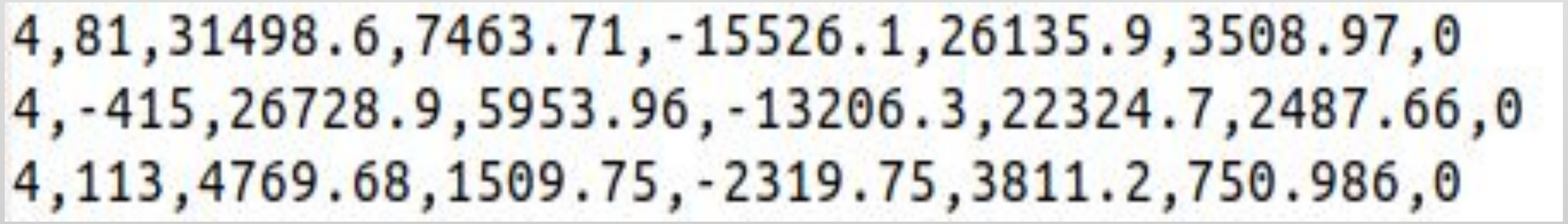
For the time being, concerned ourselves with the decay of primary clusters into two particles.

Setting up the Training Data

- Simulating the electron-positron collision at around 91 GeV, similar to LEP.
- Number of events generated = 1,00,000
- Cluster decays observed = 2,75,000
- Using Custom analysis handler, we saved the required data into numpy file format.

- ➔ We stored the elementary data of each cluster and it's two decay products.
- ➔ Elementary data for each component looks like :
  Event No , PDG ID, Energy, Px, Py, Pz, Mass, On shell condition

CLUSTER ->
1ST PARTICLE ->
2ND PARTICLE ->



```
4,81,31498.6,7463.71,-15526.1,26135.9,3508.97,0
4,-415,26728.9,5953.96,-13206.3,22324.7,2487.66,0
4,113,4769.68,1509.75,-2319.75,3811.2,750.986,0
```

- ❑ PDG ID - Every particle has a unique ID assigned by the Particle Data Group. (For Cluster it is 81)
- ❑ On Shell Condition value is $E^2 - P^2 - m^2$

STAGE 2

Using the data To Train a GAN Model

What is GAN?

A generative adversarial network (**GAN**) is a machine learning (**ML**) model in which two neural networks compete with each other to become more accurate in their predictions.

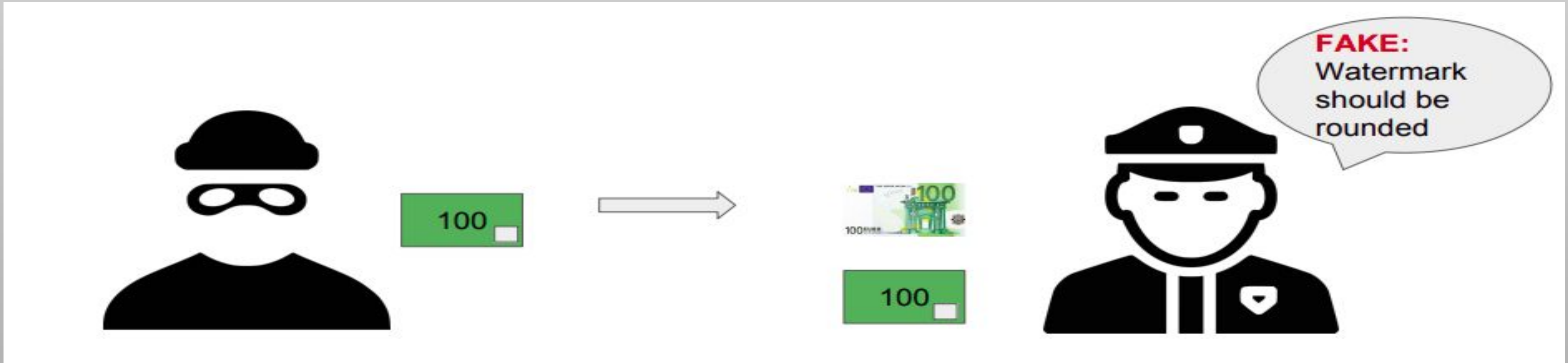
- ➔ GAN was designed by IAN Goodfellow and his colleagues in 2014.

GENERATOR

- The generator learns to generate plausible data.
- The generated instances become negative training examples for the discriminator.

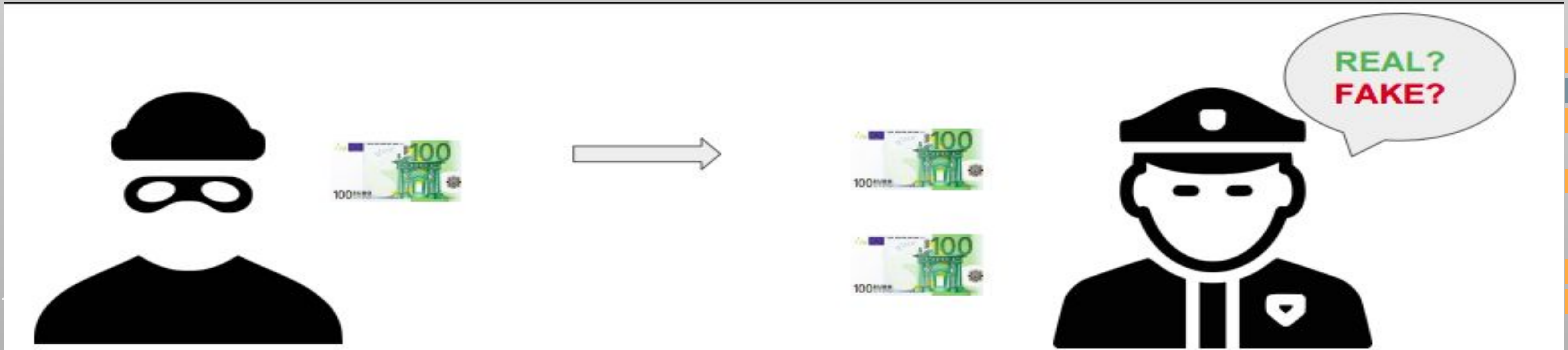
DISCRIMINATOR

- The discriminator learns to distinguish the generator's fake data from real data.
- The discriminator penalizes the generator for producing implausible results.

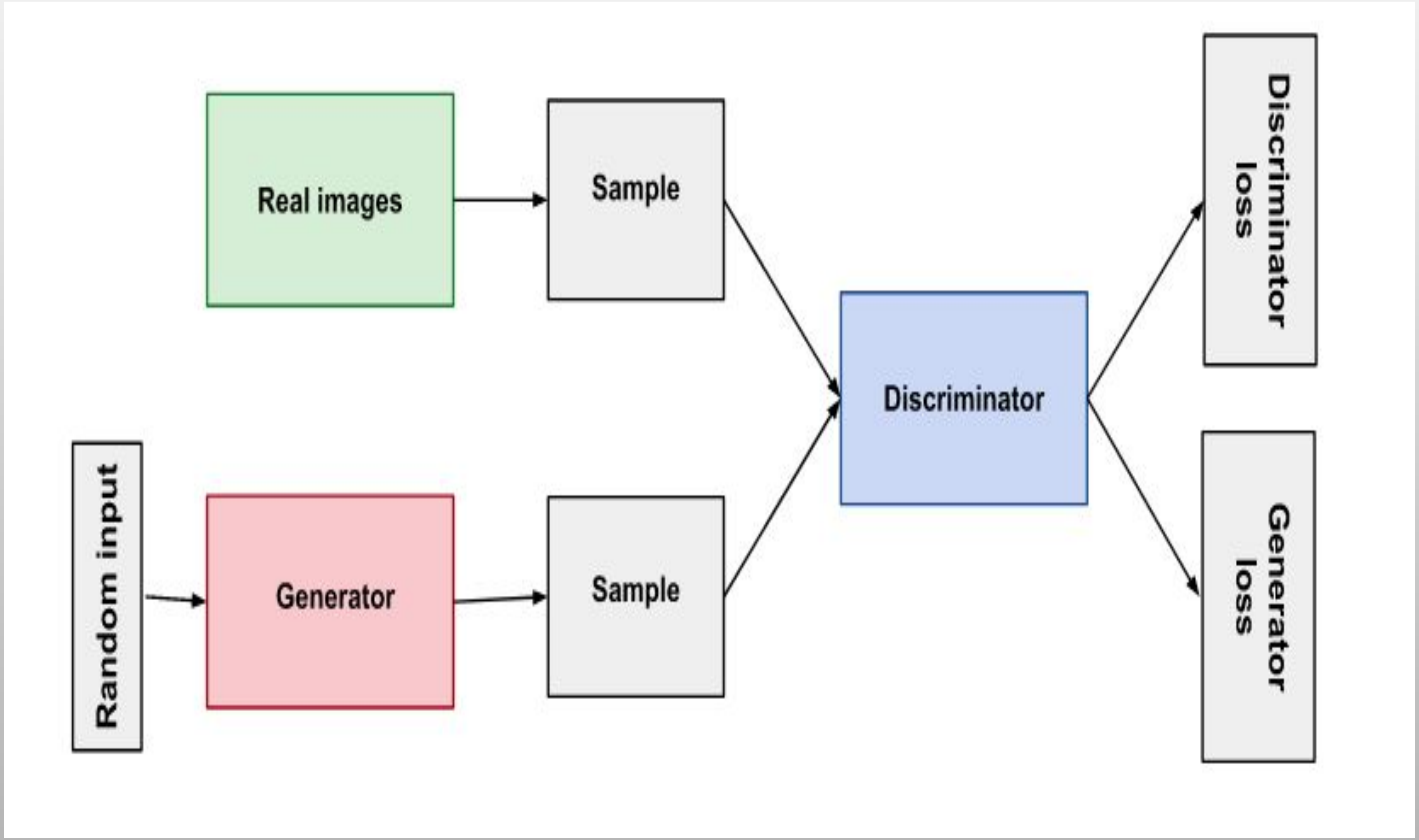


How does GAN work?

- ➔ When training begins, the generator produces fake data, and the discriminator quickly learns to tell that it's fake.
- ➔ As training progresses, the generator gets closer to producing output that can fool the discriminator.
- ➔ Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.



GENERATIVE ADVERSARIAL NETWORK



Why are we using GAN?

- ➔ GANs provide a viable strategy for speeding up numerically intensive simulation - extends to other disciplines (medicine, weather, nuclear)
- ➔ Generative models trained on the detailed GEANT simulation can be used for Calorimeter Simulation (ex. For ECAL)

- ➔ **GANs are very good at replicating given Distribution, so it would be good to train our model based on the data of existing model, and then proceed further.**

Our Work

Approach

- Sample Size of Training data is too large and diverse for direct GAN implementation.
  - PDG IDs are int, Energy is float etc.
- Focused on specific types of decay events only.

  In our case,

  cluster -> π+ π-

  cluster -> π0 π0

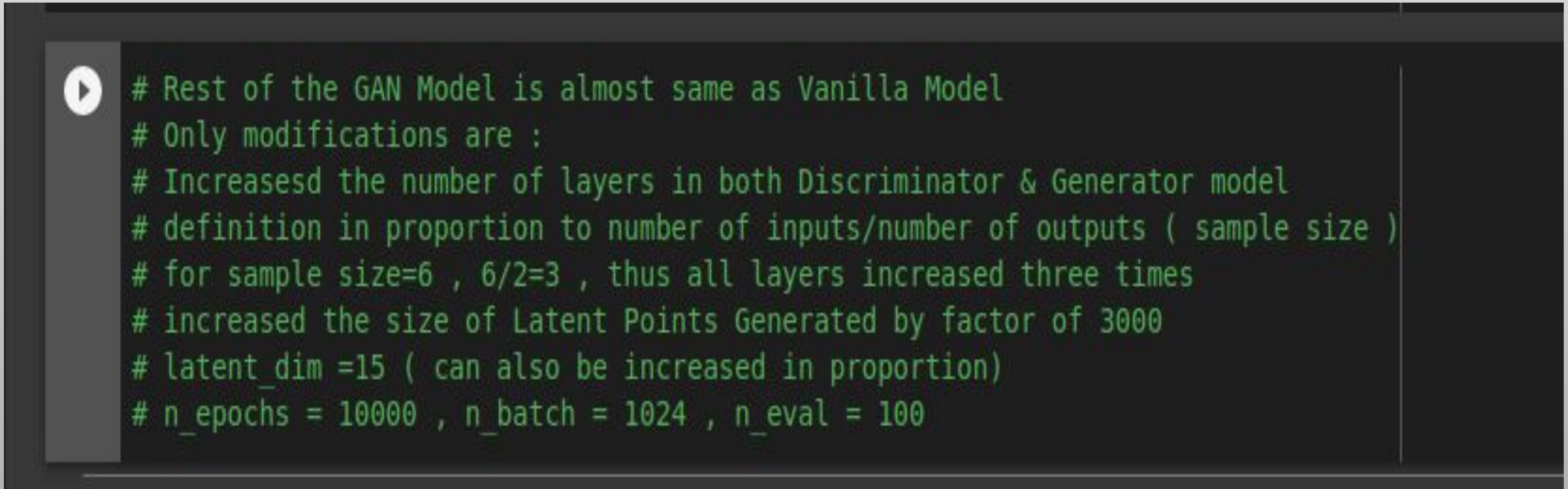
MODEL DETAILS

➔ Sample Data Variables : (Variables used to Train GAN)

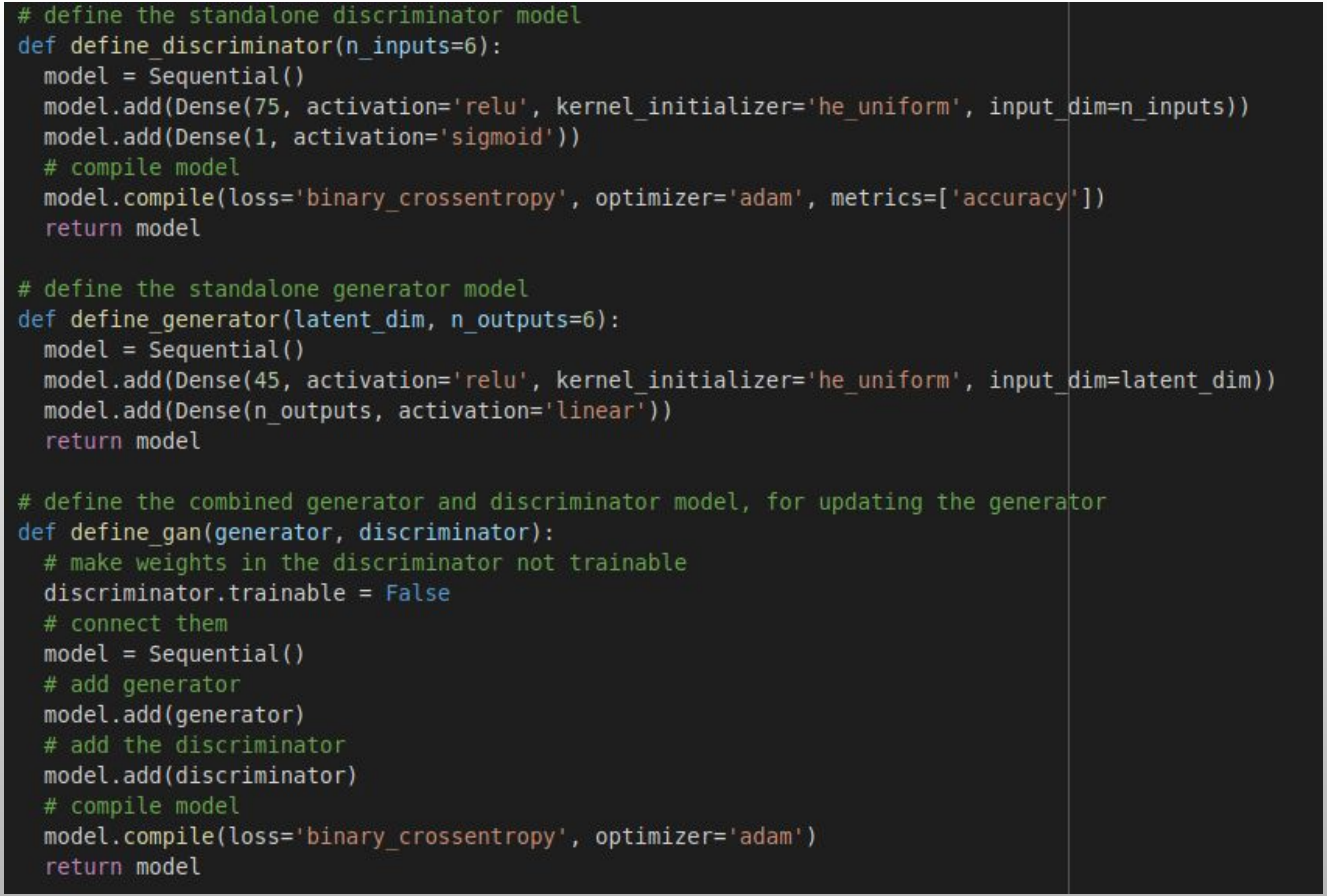
Px, Py, Pz (for cluster)

Px1, Py1, Pz1 (for one of the decay products)

➔ Platform used - Google Colaboratory



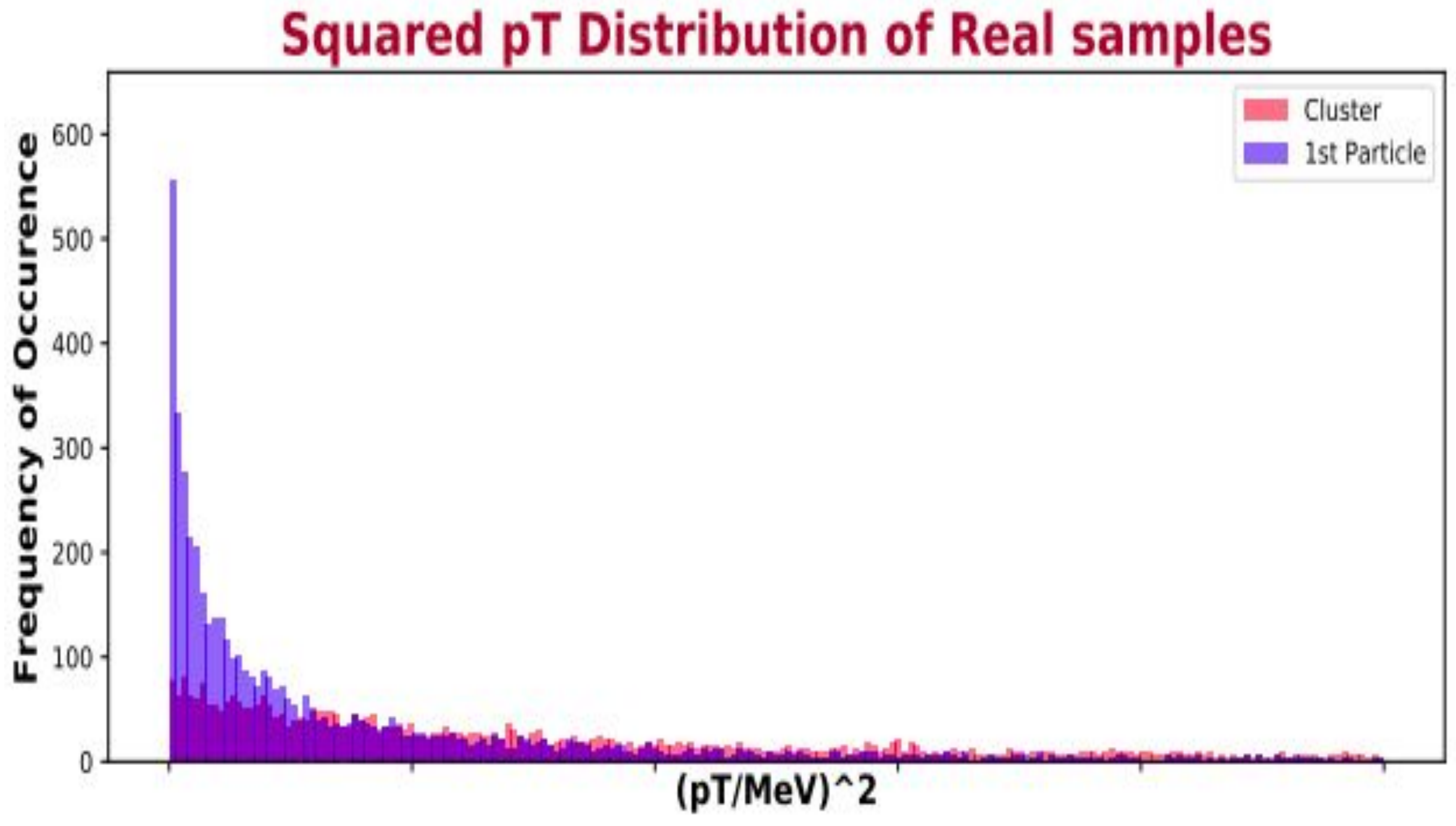
```
# Rest of the GAN Model is almost same as Vanilla Model
# Only modifications are :
# Increasesd the number of layers in both Discriminator & Generator model
# definition in proportion to number of inputs/number of outputs ( sample size )
# for sample size=6 , 6/2=3 , thus all layers increased three times
# increased the size of Latent Points Generated by factor of 3000
# latent_dim =15 ( can also be increased in proportion)
# n_epochs = 10000 , n_batch = 1024 , n_eval = 100
```

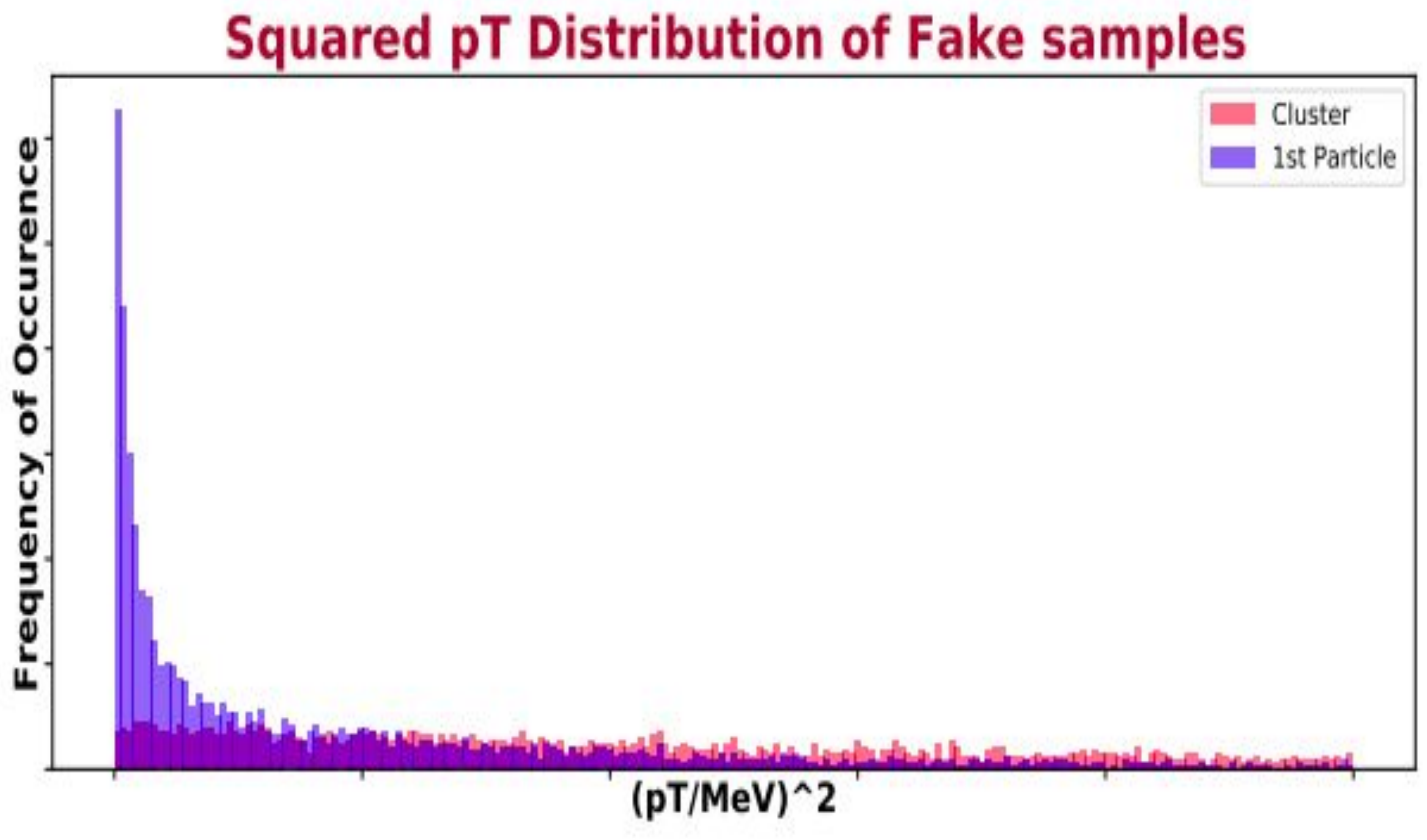

```
# define the standalone discriminator model
def define_discriminator(n_inputs=6):
  model = Sequential()
  model.add(Dense(75, activation='relu', kernel_initializer='he_uniform', input_dim=n_inputs))
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
  return model

# define the standalone generator model
def define_generator(latent_dim, n_outputs=6):
  model = Sequential()
  model.add(Dense(45, activation='relu', kernel_initializer='he_uniform', input_dim=latent_dim))
  model.add(Dense(n_outputs, activation='linear'))
  return model

# define the combined generator and discriminator model, for updating the generator
def define_gan(generator, discriminator):
  # make weights in the discriminator not trainable
  discriminator.trainable = False
  # connect them
  model = Sequential()
  # add generator
  model.add(generator)
  # add the discriminator
  model.add(discriminator)
  # compile model
  model.compile(loss='binary_crossentropy', optimizer='adam')
  return model
```

RESULTS

cluster -> π+ π-

Total Decay Events : 6000





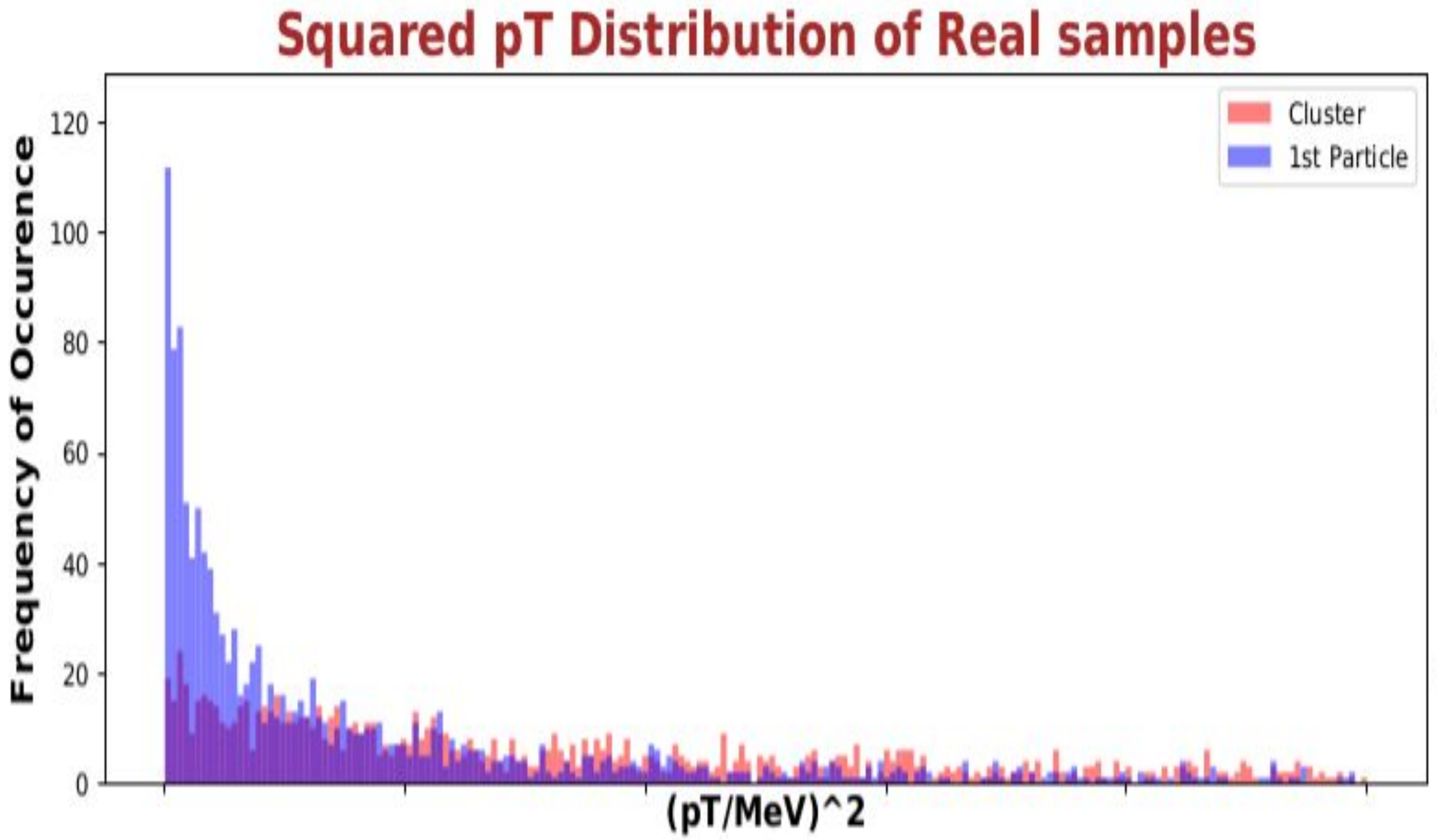
Accuracy Real : 0.43

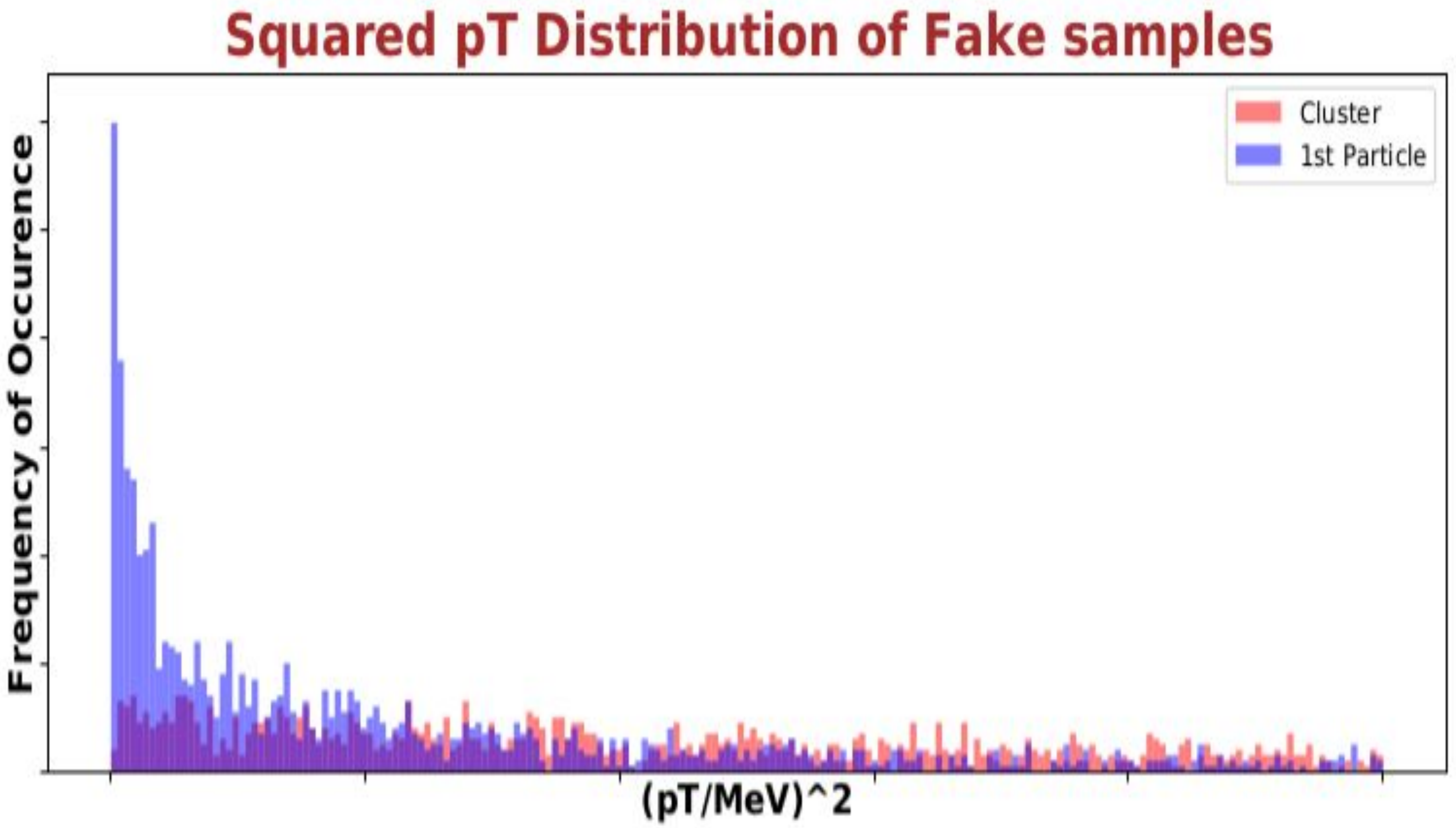
Accuracy Fake : 0.63

- The accuracy with which Discriminator predicts if its real or fake.
- Here, both are close to half, which is a good sign!!! WHY???

cluster -> π0 π0

Total Decay
Events : 1500





Accuracy Real : 0.44

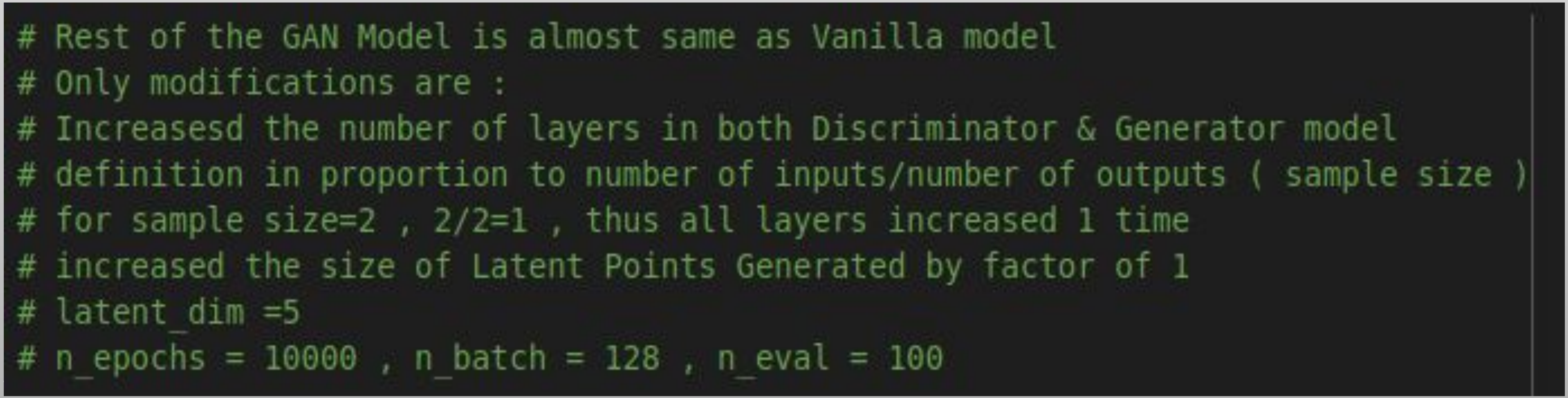
Accuracy Fake : 0.68

The GAN models we showed just now, generated momentum variables by assuming the decay particles produced.

THEN

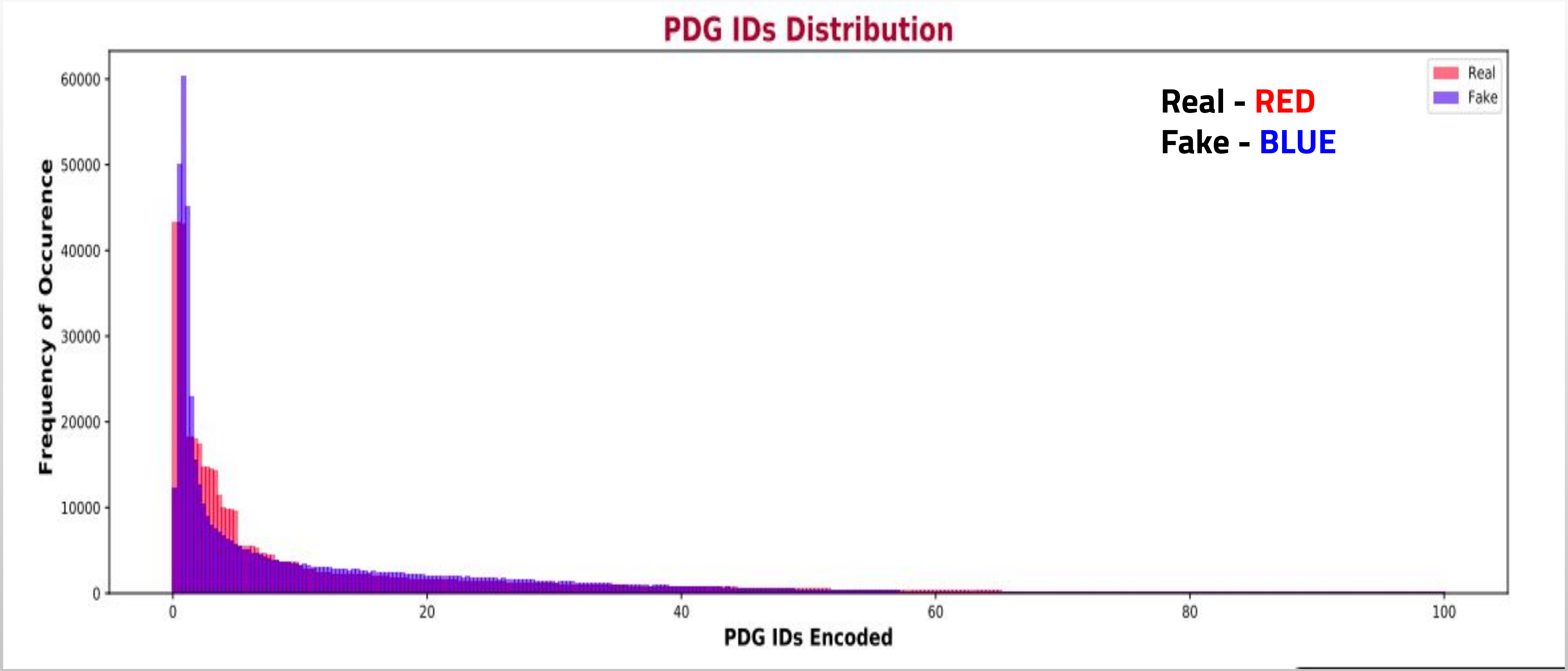
We also tried to come up with a GAN model which implements the PDG IDs produced (not concerning other variables).
Whose details are -
Sample Variables Used

- PDGID1, PDGID2
  - Encoded in range (0,100) : for ease of analysis
  - Highest occurence are close to 0 and least occurence near100



```
# Rest of the GAN Model is almost same as Vanilla model
# Only modifications are :
# Increasesd the number of layers in both Discriminator & Generator model
# definition in proportion to number of inputs/number of outputs ( sample size )
# for sample size=2 , 2/2=1 , thus all layers increased 1 time
# increased the size of Latent Points Generated by factor of 1
# latent_dim =5
# n_epochs = 10000 , n_batch = 128 , n_eval = 100
```

Total Decay Events : 275000



Accuracy Real : 0.34

Accuracy Fake : 0.55

CONCLUSIONS & OUTLOOK

- ➔ We need to work towards a Complete GAN Implementation Including all the variables, to achieve our Goal.

- ➔ We got good results with Kinematics. Now we need to turn our attentions to PGD IDs.
  - ➔ Our approach is to be extended to taking all clusters into account and including all decay processes.

- ➔ GAN Models have a promising future in the HEP Analysis for Simulation of Events, Reconstruction,etc.

REFERENCES :

- ❑ **Herwig 7.2 Documentation** : https://herwig.hepforge.org/
- ❑ **GAN Original Research Paper** : https://arxiv.org/abs/1406.2661
- ❑ **GAN Vanilla Model** : https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-1-dimensional-function-from-scratch-in-keras/
- ❑ **Google Colab** : https://colab.research.google.com/notebooks/intro.ipynb

ACKNOWLEDGEMENT

We owe a deep sense of gratitude towards **The Henryk Niewodniczanski Institute of Nuclear Physics in Cracow, Polish Academy of Sciences (IFJ PAN)** for giving us opportunity to take part in PPSS-2020 in spite of difficult circumstances due to COVID-19..

We are grateful to **dr hab. Andrzej Siódmok** for giving us excellent scientific guidance and constant encouragement towards the project.
http://siodmok.web.cern.ch/siodmok/research.html

Sincere thanks to **dr Benjamin Nachman**, who is the leading researcher working in the interface of particle physics and machine learning.
https://bnachman.web.cern.ch/bnachman/

THANKS! <3