UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Computer Systems and Computation

Improving particle physics event simulation by using Variational Autoencoders in the context of LHC experiments

Master's Thesis

Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

AUTHOR: Balanzá García, Raúl

Tutor: Gómez Adrian, Jon Ander

External cotutor: SALT CAIROLS, JOSE FRANCISCO

Experimental director: RUIZ DE AUSTRI BAZAN, ROBERTO

ACADEMIC YEAR: 2022/2023

Agradecimientos

Este trabajo representa para mí el final de otra dura etapa de gran esfuerzo, durante la cual he podido formarme más extensamente en mi vocación, pero también a la vez crecer como profesional. En este año me he podido demostrar que, con motivación y ganas, puedo cumplir la mayoría de objetivos que me proponga.

Pero, sin duda, poder terminar una etapa académica y trabajar durante el mismo periodo de tiempo con el esfuerzo que ello requiere no habría sido posible sin el apoyo de numerosas personas y organizaciones, a las cuales quiero mostrar mi agradecimiento.

En primer lugar, quisiera agradecer a mi tutor en la UPV Jon Ander Gómez Adrián su constante apoyo y dedicación, que me ha aportado gran cantidad de conocimiento. Estoy seguro de que, sin su confianza y ayuda, desarrollar un trabajo como este no me habría sido posible.

En segundo lugar, a mis dos tutores externos del IFIC (Centro Mixto CSIC-Universitat de València): José Salt Cairols y Roberto Ruiz de Austri Bazan, que han aportado gran cantidad de conocimiento en relación con la física y han resuelto todas las dudas que he tenido, además de proporcionar el conjunto de datos que ha permitido realizar la experimentación que se expone en el trabajo.

En tercer lugar, a mis padres, por apoyarme de forma incondicional durante todas las etapas académicas y profesionales de mi vida, animándome día a día hasta finalizar mis estudios de máster con este trabajo, y dotarme de todos los recursos y facilidades que he necesitado.

En cuarto lugar, agradezco a mis compañeros de trabajo y amigos de todas las etapas que siempre han creído en mí (en ocasiones más que yo mismo) su apoyo y motivación diarios, que ha sido determinante en muchos momentos del camino que me ha llevado hasta aquí.

Finalmente, quisiera agradecer a *ValgrAI - Valencian Graduate School and Research Network for Artificial Intelligence*, fundación de la que soy miembro, y a la *Generalitat Valenciana* su apoyo a través de los recursos que me han proporcionado para facilitar mis estudios de máster.

Resum

ATLAS és el detector de partícules de propòsit general més gran del Large Hadron Collider, un accelerador de partícules situat al CERN. Aquest experiment, entre d'altres, va ser creat perquè el Model Estàndard, que és el marc teòric més amplament acceptat en física, no pot explicar alguns comportaments fonamentals de la matèria a causa d'algunes limitacions.

Per a resoldre aquestes qüestions, es van crear detectors en cerca de senyals de Nova Física utilitzant el LHC. Aquests senyals són extensions del model actual que podrien explicar potencialment fenòmens desconeguts. Comparant dades experimentals reals amb dades simulades de models teòrics, podria ser possible detectar aquests senyals i validar nous descobriments. No obstant, simular dades amb mètodes tradicionals és temporalment i computacionalment molt costós. Per a reduir aquests costos i fer que el descobriment de Nova Física siga més accessible, és possible utilitzar models generatius basats en Deep Learning com generadors ràpids d'esdeveniments de Monte Carlo en el context del LHC.

Aquesta aproximació pot reduir significativament el temps i l'energia requerits respecte a les tècniques existents a l'estat de l'art actual. Anteriorment, diversos models generatius han sigut explorats tractant d'obtindre el millor generador d'esdeveniments possible. En aquest treball, profunditzem en l'ús de Variational Autoencoders amb eixe objectiu, i també usem mètriques per a comparar el rendiment d'aquests models amb altres mètodes.

Paraules clau: Esdeveniments de Monte Carlo, Models generatius basats en Deep Learning, Nova Física, Variational Autoencoder, Large Hadron Collider

Resumen

ATLAS es el detector de partículas de propósito general más grande del Large Hadron Collider, un acelerador de partículas situado en el CERN. Este experimento, entre otros, fue creado porque el Modelo Estándar, que es el marco teórico más ampliamente aceptado en física, no puede explicar algunos comportamientos fundamentales de la materia debido a algunas limitaciones.

Para resolver estas cuestiones, se crearon detectores en busca de señales de Nueva Física utilizando el LHC. Estas señales son extensiones del modelo actual que podrían explicar potencialmente fenómenos desconocidos. Comparando datos experimentales reales con datos simulados de modelos teóricos, podría ser posible detectar estas señales y validar nuevos descubrimientos. Sin embargo, simular datos con métodos tradicionales es temporal y computacionalmente costoso. Para reducir estos costes y hacer que el descubrimiento de Nueva Física sea más accesible, es posible utilizar modelos generativos basados en Deep Learning como generadores rápidos de eventos de Monte Carlo en el contexto del LHC.

Este enfoque puede reducir significativamente el tiempo y la energía requeridos respecto a las técnicas existentes en el estado del arte actual. Anteriormente, varios modelos generativos han sido explorados intentando obtener el mejor generador de eventos posible. En este trabajo, profundizamos en el uso de Variational Autoencoders con ese objetivo, y también utilizamos métricas para comparar el rendimiento de estos modelos con otros métodos.

Palabras clave: Eventos de Monte Carlo, Modelos generativos basados en Deep Learning, Nueva Física, Variational Autoencoder, Large Hadron Collider

Abstract

ATLAS is the largest general-purpose particle detector experiment at the Large Hadron Collider, a particle accelerator located at CERN. This experiment, among others, was created because the Standard Model, which is the most widely accepted theoretical framework in physics, cannot fully explain certain fundamental behaviors of matter due to some limitations.

To address these questions, detectors were created to search for signals of Physics Beyond the Standard Model using the LHC. These signals are extensions of the current model that could potentially explain unknown phenomena. By comparing real experimental data to simulated data from theoretical models, it may be possible to detect these signals and validate new discoveries.

However, simulating data with traditional methods is time-consuming and computationally intensive. To reduce these costs and make the discovery of BSM physics more accessible, generative models based on Deep Learning can be used as fast Monte Carlo event generators in the LHC context.

This approach can significantly reduce the time and energy required with respect to existing state-of-the-art techniques. Previously, several Deep Generative models have been explored trying to obtain the best event generator possible. In this work, we delve into the use of Variational Autoencoders for that purpose, and we also use numerical metrics to compare the performance of these models to other methods.

Key words: Monte Carlo event generator, Deep Generative Models, New Physics, Variational Autoencoder, Large Hadron Collider

Contents

List of Figures

List of Tables

Acronyms

AE Autoencoder. ix, 11, 12

ANN Artificial Neural Network. 85

BSM Beyond the Standard Model. viii, ix, xi, xii, 1, 2, 5, 6, 9, 11, 24, 66, 67, 69, 71–73, 75

CERN European Organization for Nuclear Research. 1, 3, 5

CPU Central Processing Unit. 15

DL Deep Learning. 3

DNN Deep Neural Network. 11

GAN Generative Adversarial Network. 5, 6

GPU Graphics Processing Unit. 15

LHC Large Hadron Collider. 1, 5

MC Monte Carlo. 2, 3, 5, 13, 75

NF Normalizing Flows. 6

PRHLT Pattern Recognition and Human Language Technology. 15

SM Standard Model. ix, xii, 5, 6, 9, 23, 24, 72, 75

VAE Variational Autoencoder. vii, ix, x, 1, 5, 6, 11–13, 26–28, 35, 47, 48, 54, 56–58, 66, 75

CHAPTER 1
Introduction

The **Standard Model** stands as the most widely accepted theoretical framework in physics at present, providing an explanation for the composition of matter and the fundamental forces that govern physical processes. Nevertheless, certain observed phenomena do not conform to its theoretical predictions, and there exist limitations in the model's ability to address fundamental aspects such as neutrino masses, gravity, and dark matter. Additionally, there exist unexplained parameters, including the surprisingly low Higgs boson mass regardless of its quantum corrections, evidencing gaps in our understanding.

Because of that, at the Large Hadron Collider (LHC), situated at the European Organization for Nuclear Research (CERN), particles are propelled to high velocities to produce collisions, enabling the meticulous examination of their interactions and providing detailed insights into the fundamental laws that govern the workings of nature. The LHC is considered as the most powerful and largest collider in the world.

At CERN, the aforementioned particle collider is used to acquire **real data** of proton-proton collisions (which make up a small portion of the total), that is then selected and recorded as events. Online selection of events is made from a trigger, and then events are processed and reconstructed. The data collection process takes place at four distinct collision points along the 27-kilometer circular ring of the LHC, employing seven detectors utilized by various experiments [1].

During the course of an experiment, simulated data is generated through various methods to facilitate a comparison between data from different theoretical models and the actual event characteristics. Subsequently, the findings derived from analyzing the accumulated data are discussed. These investigations aim to identify novel theoretical frameworks, known as Physics Beyond the Standard Model (BSM), that may provide explanations for the aforementioned limitations of the current model.

Up to this point, a substantial volume of data has been collected across four distinct experiments. Within the scope of this study, our attention centers on the ATLAS experiment (Figure 1.1), which is a general purpose experiment with the objective of researching about the origin of masses and BSM physics. Although there is no clear sign of confirmed new physics models for now, further experimentation could yield promising results.

Our proposal is to use Deep **Generative Models**[1] to provide a faster way to simulate physical events when compared to traditional Monte Carlo approaches. In previous works [3], we already explored this kind of models to approach the problem, obtaining a solid baseline. Now, we focus on different variants of a particular type of model: Variational Autoencoders (VAEs), improving their architecture and presenting some metrics to quantify the precision of each model.

[1] Unsupervised learning tasks in machine learning that involve learning patterns in input data so that the model can be used to generate new samples that follow the same patterns found in the original dataset.

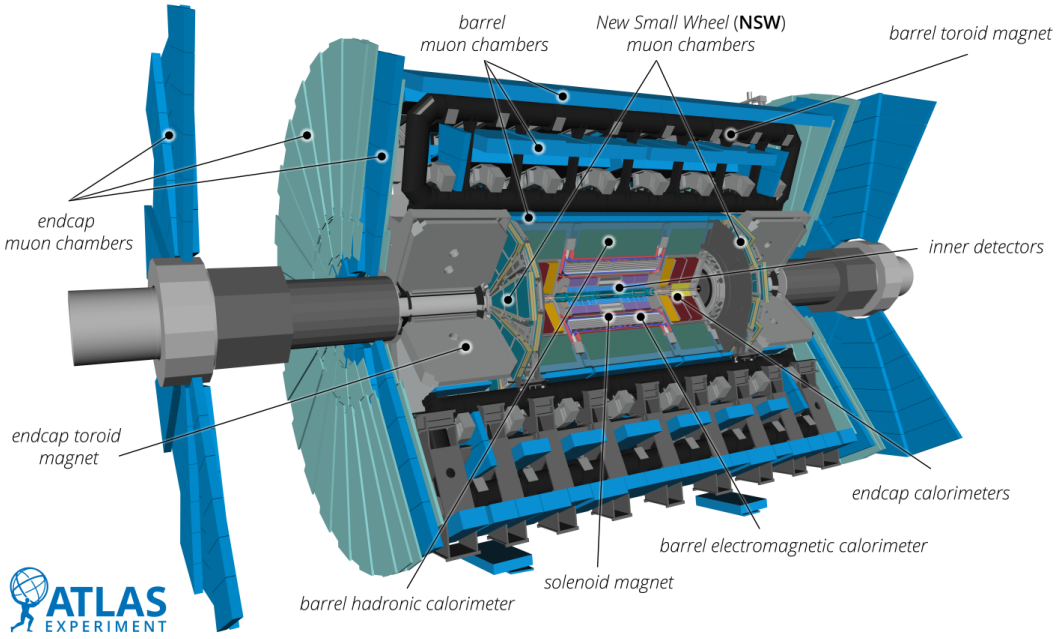


Figure 1.1. Schematic of the ATLAS detector located at CERN. [2]

1.1 Problem description and motivation

Without the use of Deep Learning models, the simulation of physical processes is mostly done following the Monte Carlo method, that is usually performed in two steps:

1. Firstly, pseudo-random numbers from a probability distribution are sampled.
2. Secondly, the generated numbers are transformed by an algorithm into simulated physical events, which in this case are high energy particle collisions.

However, a significant challenge associated with numerical simulations lies in their extensive demand for computational resources. Consequently, scientific advancements in this field are constrained by the limitations of simulation speed and available budgets. As an example, the generation of an MC event in particle physics experiments, including the response of the detector, can consume up to 10 minutes of computation time because it relies on non-optimal algorithms of toolkits like GEANT4 [4].

In upcoming experiments, the accumulation of real data generated with the collider will require the generation of billions of events to compare with. However, the current methods employed for event generation pose two significant challenges. Firstly, the required generation times using these methods will be impractical. Secondly, if these methods were to be utilized, they would impose a substantial environmental burden in terms of energy consumption. Therefore, optimizing event generation becomes an imperative task to enhance efficiency and address these concerns.

While a significant speedup for studies of signals of BSM physics was already achieved in our previous work by redesigning the event generation pipeline with the help of Deep Generative Models, the obtained results did not fully adjust to the quality of events that were generated using MC-based methods.

Moreover, accurate numeric measures of the quality of our event generation models were not defined, so if several models produced very similar results, we did not have any precise way to compare them due to the lack of **metrics**.

The use of generative models has already proven to be very helpful when producing simulated data in large quantities while saving temporal, energy, and economic resources during physics studies and analysis. Furthermore, it will facilitate the assessment of systematic errors. Estimating these errors is crucial whenever measurements of various physical quantities are conducted.

One additional concern pertains to the challenge of accurately defining the properties of simulated events. In data analysis, it is frequently necessary to generate events that exhibit similar kinematic characteristics to those observed in the actual data. Currently, event generators generate a substantial quantity of events and subsequently filter out the interesting ones while discarding the remaining events. Exploring methods to minimize the generation of irrelevant events is also a topic of interest.

1.2 Objectives

After fully describing the problem and its motivation, and taking into account our previous study that already involved using Deep Generative Models, we establish several objectives to be accomplished in this work, aiming to address and overcome some of the existing limitations.

First, we require to refine our current DL framework for event generation taking as a base existing Monte Carlo data (known as *ground truth*), making sure that it is capable of obtaining results with similar properties than the current methods.

Then, the obtained results need to be compared with the classical approaches using appropriate metrics to verify that our newly adjusted framework performs similarly regarding data quality (i.e., events are generated with a similar distribution) while being more efficient in terms of time and/or required resources.

Finally, several approaches will be evaluated with the help of our metrics to decide which one is more similar to the *ground truth* events, obtaining a balance of efficiency and quality. It is important to obtain event distributions that are similar to real MC events.

1.3 Structure of this work

The structure of this work consists of the following parts: Chapter 1 begins with a brief explanation of the problem and ongoing experiments at CERN. It is followed by an overview of the current shortcomings and the defined objectives to address them.

Moving on to Chapter 2, we delve into an analysis of the current state-of-the-art, describing classical approaches as well as machine learning solutions that closely align with our goals.

Chapter 3 provides a comprehensive description of the dataset utilized in our research. We outline the various types of events and particles present in the dataset, elaborate on the recorded information for each event and particle, and describe the format in which the data is provided.

In Chapter 4, we explain the theoretical foundations of the models that we use during the experimentation, aiming to enhance the models referenced in the second chapter and to obtain valid models that fulfill the specified objectives. Prior to delving into the practical aspects, we introduce the software and hardware technologies employed in Chapter 5, which enable the execution of all the experiments.

Then, Chapter 6 focuses on describing the metrics that will quantify the results of the performed experiments in order to determine in a more precise way which model performs better.

Chapter 7 is dedicated to presenting and describing the results of our experimentation. We present a detailed explanation of the model architecture and event generation pipeline employed for each experiment, focusing on the results obtained and providing analysis that guides the exploration of different models, ultimately leading to the identification of the most effective approach.

Finally, in Chapter 8, we conclude by interpreting the suitability of the chosen approach for the desired experiment and determining the best model architecture by comparing all the possibilities studied.

CHAPTER 2
State of the art

Many efforts have been made in the past to detect signals of new physics at the LHC through the utilization of supervised and unsupervised machine learning algorithms. As of now, an indication of BSM physics has yet to be discovered, proving the importance of developing appropriate methods that facilitate investigations and establish a solid framework for future experiments.

First of all, we highlight our previous work where we made our first attempt to use Deep Generative Models to generate SM and BSM events [3]. There, we took as a starting point some techniques used in previous studies to create a framework that was capable of generating events in a more efficient way than traditional Monte Carlo methods. Particularly, we presented several variants of generative models, being the best performing one our own variation of a Variational Autoencoder: the α-VAE, that showed the best results when using values of $\alpha \approx 0.2$.

A relevant event within the scientific community relevant to this study was *The Dark Machines Anomaly Score Challenge*, which is extensively discussed in [5]. The dataset employed in our work is the same that was used in this challenge, and its characteristics are detailed in Chapter 3. The primary objective of the challenge was to identify indications of new physics at the LHC using unsupervised machine learning algorithms. Initially, an anomaly score implementation was proposed to define model-independent signal regions. Subsequently, a wide array of anomaly detection and density estimation algorithms, specifically developed for the data challenge, were examined, and their performance was evaluated within a range of realistic analysis environments. This challenge serves as a groundwork for new benchmarks and the development of more effective approaches in the detection of BSM signals.

One experiment that closely relates to both of our works is a previous investigation focused on generating events from a physical process that already used Deep Generative Models [6]. This study aimed to explore the feasibility of learning event generation and the accurate frequency of occurrence to generate events similar to those produced by MC generators. Initially, several Generative Adversarial Network architectures and the Variational Autoencoder with a standard normal prior were unable to generate events with the correct frequency of occurrence. However, a specific VAE configuration, known as the β-VAE, yielded promising outcomes. This approach produced distributions that were very similar to real MC events and generated events significantly faster.

The ATLAS project team at CERN also investigated in this topic, defining their own strategy to find potential indications of BSM physics [7]. In their approach, events were categorized based on their final state, resulting in multiple event classes. For each event class, an automated search algorithm was employed to assess the compatibility of the data with the expected outcomes simulated by MC simulations across various distri-

butions sensitive to the effects of new physics. The significance of any deviation was quantified using pseudo-experiments. If a data selection exhibited a significant deviation, it would define a signal region for a subsequent in-depth analysis, incorporating an improved background expectation. Statistical interpretation of the signal regions derived from the data was performed on a new dataset. The sensitivity of this approach was evaluated using SM processes as well as benchmark signals representing potential instances of new physics.

To conclude, it is worth mentioning several approaches that were explored in different areas of the event processing field of study. Concerning **detector simulation**, we can find some GAN-based models [8, 9, 10], but also some VAE-based ones [11, 12, 13]. Regarding **event generation**, there have been a wide number of attempts to use GAN-based models [14, 15, 16], but the possibilities of VAEs have not been explored deeply. Also, in relation to **Monte Carlo integration**, we observe the use of regression in many studies [17, 18] and also some attempts to use Normalizing Flows [19]. Finally, recent research has also been made in the use of models based on Normalizing Flows [20, 21].

2.1 Analysis of the current situation

Once we have performed an analysis on the state of the art regarding some of the most significant and recent contributions related to our topic of interest, we obtain several noteworthy conclusions and identified specific observations that provide insights into our current status:

- The majority of physics research has predominantly employed supervised learning methods. However, there has been a recent surge in the popularity of unsupervised learning within the scientific community, indicating ample room for further exploration in this area.
- Our study performed in [3] and some others already experimented with VAEs. However, that experiment did not evaluate the models with appropriate metrics nor obtained an approach that was good enough to be used in real world use cases.
- The data challenge outlined in [5] established a robust framework for evaluating the performance of different approaches to event generation, and provided a good dataset to use as a starting point to develop new models.
- While there have been many attempts to use GANs for event generation and detector simulation, the possibilities of VAEs and NFs have been less extensively explored.

Considering the aforementioned observations, our focus in this work centers around exploring the possibilities of unsupervised learning by utilizing VAE-based models. Additionally, we keep using the same already known and diverse dataset that includes both Standard Model (SM) and Beyond the Standard Model (BSM) processes, with the goal of constructing a viable model capable of being trained for any type of process.

CHAPTER 3
Dataset details

As we already did in [3], the dataset that we used for our experimentation in this work is the one that was proposed in *The Dark Machines Anomaly Score Challenge* [5], which is available to download in [22]. Although a comprehensive explanation of the data generation procedure format can be found in [23], we provide here a concise overview, focusing on the key details that are particularly relevant to our study.

3.1 Data generation

The simulated events consist of proton-proton collisions that are similar to the ones observed at the LHC, with a center-of-mass energy of 13 TeV[1]. The signal and background processes were generated using conventional methods and tools. Once generated, the final-state physics objects as outlined in Table 3.1 were saved in a CSV text file, with each event represented in a single line.

During a collision event, a maximum of 20 objects are produced, and for each object, various properties such as total energy (E), transverse momentum (p_T), pseudo-rapidity (η), and azimuthal angle (ϕ) are measured and recorded. An event is considered valid and stored in the dataset if it satisfies at least one of the following criteria:

- One or more **jets** or ***b*-jet** with $p_T > 60$ GeV and $|\eta| < 2.8$
- One or more **electrons** with $p_T > 25$ GeV and $|\eta| < 2.47$ except for $1.37 < |\eta| < 1.52$
- One or more **muons** with $p_T > 25$ GeV and $|\eta| < 2.7$
- One or more **photons** with $p_T > 25$ GeV and $|\eta| < 2.37$

Certain additional criteria regarding the `event weight` were imposed. However, for the sake of simplicity, neither the original data challenge nor this study take that variable into account. The requirements on the final objects stored in the dataset were:

- **jet** or ***b*-jet**: $p_T > 20$ GeV and $|\eta| < 2.8$
- **electron** or **muon**: $p_T > 15$ GeV and $|\eta| < 2.7$
- **photon**: $p_T > 20$ GeV and $|\eta| < 2.37$

The objective was to generate a versatile dataset that could be used for various types of studies, each potentially requiring distinct selection criteria. The complete list of the generated processes can be found in Table 3.2.

[1] One eV is the amount of kinetic energy gained or lost by a single electron accelerating from rest through an electric potential difference of 1 volt in vacuum.

Symbol ID	Object
j	jet
b	b-jet
e-	electron (e^-)
e+	positron (e^+)
m-	muon (μ^-)
m+	antimuon (μ^+)
g	photon (γ)

Table 3.1. Definition of symbols representing final-state objects of each event.

3.2 Data format

As mentioned in the preceding section, the data is presented in a CSV file format, with each event represented by a single line. Each line varies in length and consists of three event-specifiers followed by the kinematic features of each object within the event.

Each file represents a distinct process. The number of events generated for each process is guaranteed to be equal to or greater than the minimum requirement for obtaining $10fb^{-1}$ of data[2] ($N_{10fb^{-1}}$). Additionally, when $N_{10fb^{-1}} < 20000$, a second file with 20000 lines is provided. The format of all CSV files is as follows:

`event ID; process ID; event weight; MET; METphi; obj1, E1, pt1, eta1, phi1; obj2, E2, pt2, eta2, phi2; ...`

The `event ID` serves as a unique identifier for each event, represented by an integer value. Its purpose is to facilitate event-specific debugging. The `process ID`, on the other hand, is a string that corresponds to the process responsible for generating the event, as outlined in Table 3.2. While the `event weight` depends on the cross section of a specific process and the number of events within a file, it is not considered in our experiments. These components are the event specifiers within each line of the CSV file.

In terms of the kinematic features, the `MET` and `MET`ϕ entries correspond to the magnitude E_T^{miss} and azimuthal angle $\phi_{E_T^{miss}}$ of the missing transverse energy vector within the event. These values represent the transverse energy and azimuthal angle of objects that effectively go undetected. Such objects are considered to have evaded detection.

The object identifiers (`obj1`, `obj2`, etc.) are strings that uniquely identify each object within the event, according to the specifications outlined in Table 3.1. Following each object identifier, there are four comma-separated values that comprehensively describe the characteristics of the object: `E1` (E), `pt1` (p_T), `eta1` (η), and `phi1` (ϕ). These features align with the ones mentioned in the previous section.

The arrangement of particles within each event follows the sequence of: b-jets, jets, leptons, and photons. Within each particle type, they are sorted in descending order based on their transverse momentum (p_T). Consequently, in the results section, when mentioning the *leading* particle, we are referring to the one with the highest p_T within its respective category.

The events are categorized by process and distributed across 65 files, with a total size of 66.5 GB. However, during the experimentation phase, only the largest files for each process type were selected. Specifically, for processes that achieved $10fb^{-1}$ with less than 20K events, the file containing 20K events was chosen.

[2]The inverse femtobarn is a measurement of particle-collision events per femtobarn; a measure of both the collision number and the amount of data collected ($1fb^{-1} \approx 10^{12}$ proton-proton collisions).

SM processes	
Physics process	*Process ID*
$pp \to jj$	njets
$pp \to W^{\pm}(+2j)$	w_jets
$pp \to \gamma(+2j)$	gam_jets
$pp \to Z(+2j)$	z_jets
$pp \to t\bar{t}(+2j)$	ttbar
$pp \to W^{\pm}t(+2j)$	wtop
$pp \to W^{\pm}\bar{t}(+2j)$	wtopbar
$pp \to W^{+}W^{-}(+2j)$	ww
$pp \to t + jets(+2j)$	single_top
$pp \to \bar{t} + jets(+2j)$	single_topbar
$pp \to \gamma\gamma(+2j)$	2gam
$pp \to W^{\pm}\gamma(+2j)$	Wgam
$pp \to ZW^{\pm}(+2j)$	zw
$pp \to Z\gamma(+2j)$	Zgam
$pp \to ZZ(+2j)$	zz
$pp \to h(+2j)$	single_higgs
$pp \to t\bar{t}\gamma(+1j)$	ttbarGam
$pp \to t\bar{t}Z$	ttbarZ
$pp \to t\bar{t}h(+1j)$	ttbarHiggs
$pp \to \gamma t(+2j)$	atop
$pp \to t\bar{t}W^{\pm}$	ttbarW
$pp \to \gamma\bar{t}(+2j)$	atopbar
$pp \to Zt(+2j)$	ztop
$pp \to Z\bar{t}(+2j)$	ztopbar
$pp \to t\bar{t}t\bar{t}$	4top
$pp \to t\bar{t}W^{+}W^{-}$	ttbarWW

BSM processes	
Physics process	*Process ID*
$pp \to \tilde{g}\tilde{g}$ (1 TeV)	gluino_01
$pp \to \tilde{g}\tilde{g}$ (1.2 TeV)	gluino_02
$pp \to \tilde{g}\tilde{g}$ (1.4 TeV)	gluino_03
$pp \to \tilde{g}\tilde{g}$ (1.6 TeV)	gluino_04
$pp \to \tilde{g}\tilde{g}$ (1.8 TeV)	gluino_05
$pp \to \tilde{g}\tilde{g}$ (2 TeV)	gluino_06
$pp \to \tilde{g}\tilde{g}$ (2.2 TeV)	gluino_07
$pp \to \tilde{t}_1\tilde{t}_1$ (220 GeV), $m_{\tilde{x}_1^0} = 20$ GeV	stop_01
$pp \to \tilde{t}_1\tilde{t}_1$ (300 GeV), $m_{\tilde{x}_1^0} = 100$ GeV	stop_02
$pp \to \tilde{t}_1\tilde{t}_1$ (400 GeV), $m_{\tilde{x}_1^0} = 100$ GeV	stop_03
$pp \to \tilde{t}_1\tilde{t}_1$ (800 GeV), $m_{\tilde{x}_1^0} = 100$ GeV	stop_04
$pp \to Z'$ (2 TeV)	Zp_01
$pp \to Z'$ (2.5 TeV)	Zp_02
$pp \to Z'$ (3 TeV)	Zp_03
$pp \to Z'$ (3.5 TeV)	Zp_04
$pp \to Z'$ (4 TeV)	Zp_05

Table 3.2. Generated SM and BSM signal processes with their identification.

CHAPTER 4
Proposed models

Considering the current state of the art as described in Chapter 2, and after deciding and analyzing the dataset to work with, we propose a solution based in using several Variational Autoencoder models to try to obtain relevant results when generating events.

The presented approach tries to build upon our previous work, improving its results (i.e., obtaining similar data distributions) and allowing the generation of new samples from a random normal distribution, instead of applying a permutation to already existing events.

We decided to focus in the use of VAEs due to the existence of less studies choosing this model, and also because we saw that this kind of models has great potential in tasks that are related to this one. However, Normalizing Flow models are also a popular and often used kind of Deep Generative Models in the AI field, and we believe that promising results can be obtained with both of these techniques.

While we already obtained a model that fits the data distribution of our dataset in our previous work, the objective here is to focus in generating new events without using any *ground truth* data in the generation process. A model that can satisfy this requirement will vastly improve extensive searches for signals of Physics Beyond the Standard Model, making it possible to confirm new theoretical models. A brief description of the theoretical concepts that define autoencoders and its evolution into the architecture of VAEs follows.

4.1 Autoencoders

An autoencoder (AE) [24] is a type of Deep Neural Network that consists of a central hidden layer with a lower dimension compared to the input layer. Its target space coincides with the input space. These networks are trained to reconstruct features of the input data while incorporating a bottleneck structure to prevent them from merely learning the identity mapping. The dimension of the central layer plays a crucial role as it determines the level of feature compression.

The architecture of an autoencoder is therefore composed of:

- An **encoder** component, that compresses the input data into the latent space.
- A **decoder** component, which uses information from the lower dimensional latent space to transform it back into the input dimensions.

This type of DNN has been used before in some High Energy Physics applications. In [25], an AE is trained extensively on SM data and reaches reasonably low reconstruction

errors, that make it good enough to be used as part of an anomaly detection algorithm. The model could identify anomalous events from their higher reconstruction error.

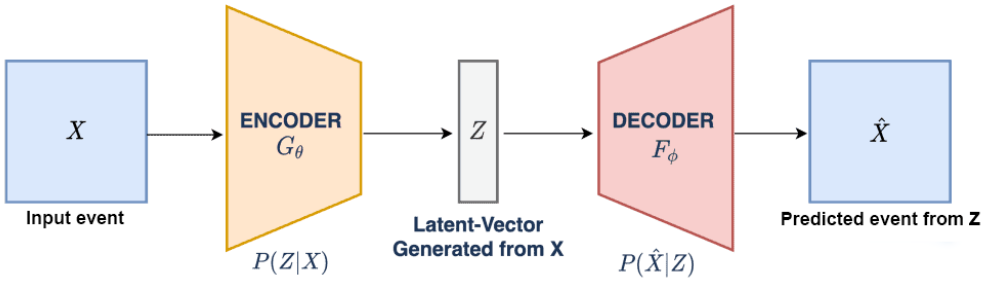


Figure 4.1. Diagram of the internal components of an Autoencoder. [26]

4.2 Variational Autoencoders

A Variational Autoencoder (VAE) [27, 24] can be seen as an autoencoder architecture that incorporates regularization techniques during training. This aims to prevent overfitting and ensure desirable properties in the latent space, enabling the generation of new data from arbitrary numbers.

Similar to a standard autoencoder, a VAE (Figure 4.2) also shares the same **encoder** and **decoder** components. However, in this approach, a small modification is made during the encoding-decoding process to introduce regularization in the latent space.

Here, instead of encoding an input as a single point, it is encoded as a distribution across the latent space. The bottleneck layer is formed by the encoder outputting two values per dimension in the latent space, typically representing the **mean** and **standard deviation** of a normal distribution. This approach ensures both local (variance) and global (mean) **regularization**.

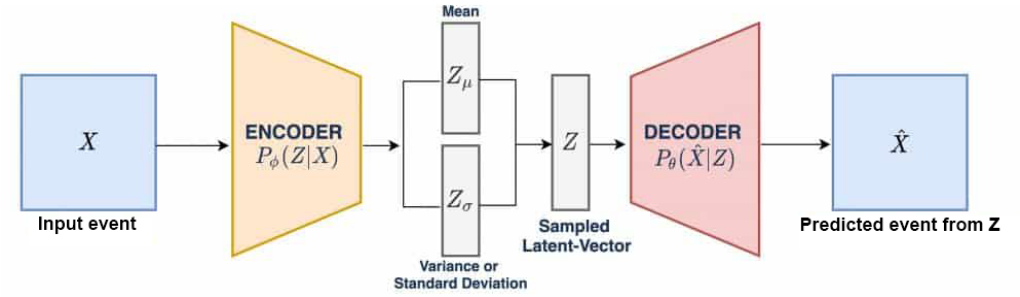


Figure 4.2. Diagram of the internal components of a Variational Autoencoder. [26]

The **training process** can be described as follows: First, the input is encoded into a distribution across the latent space. Then, a point is sampled from that distribution in the latent space. Subsequently, the sampled point is decoded, and the reconstruction loss is computed. Finally, the error is backpropagated through the network.

The **loss function** minimized during VAE training consists of two components: a *reconstruction term* (located in the output layer of the decoder), which focuses on improving the performance of the encoding-decoding process, and a *regularisation term* (located in the latent layer), which is proportional to the Kullback-Leibler (KL) divergence and is used to regularize the structure of the latent space by helping the distributions generated by the encoder to approximate a standard normal distribution.

4.2.1. α-VAE

In our previous work [3], we presented a new type of Variational Autoencoder that consisted of several differences when compared to the original version described in the previous section.

In this version, that we call α-VAE, its main feature is that after encoding the input in the usual way, we add Gaussian noise to the latent representation of the input data that follows a normal distribution with zero mean and a standard deviation with value α. This effectively replaces the Kullback-Leibler divergence layer.

We used this kind of model, as well as the β-VAE (described in the following section) for generating events by taking *ground truth* data and applying a permutation also coming from a random normal distribution with the same mean and standard deviation used in the Gaussian noise in the training process. This approach is shown as a baseline for this work in Section 7.

4.2.2. β-VAE

For some applications, giving the same importance (*weight*) to every component of the loss function does not produce the best results. To avoid this problem, it can be a good approach to balance out the relative importance of the *reconstruction term* (L_{rec}) and the *regularisation* (L_{KL}) term. To accomplish that objective, a hyperparameter β is sometimes introduced to adjust this relative importance. Then, we can formulate the loss function as follows:

$$L_{VAE} = (1 - \beta) * L_{rec} + \beta * L_{KL} \tag{4.1}$$

However, during our experiments we found out that this way of applying β to balance the weight of each component of the loss function did not help the model to learn the data distributions correctly. For that reason, we finally applied the following slightly modified version:

$$L_{VAE} = L_{rec} + \beta * L_{KL} \tag{4.2}$$

Variational Autoencoders that use this description of the loss function are known as β-VAE. We propose using this kind of model for generating accurate MC events from random numbers, as it is described in Section 7.

CHAPTER 5
Used hardware and software

In the past, the implementation of machine learning algorithms and model topologies involved creating them from scratch, precisely specifying the details of the calculations required during the training or inference process.

However, thanks to ongoing research and collaborative efforts within the scientific community, significant progress has been made over the years. Today, we have access to a multitude of libraries and tools that simplify the development and testing of models by providing high-level programming interfaces that are very straightforward to use.

These libraries and tools have revolutionized the field by abstracting away the need to delve into low-level implementation details. They allow us to focus on the core concepts and innovations in our models, rather than defining basic operations. By leveraging these tools, the process of building and testing different model architectures has become more simple.

During the experimentation performed in this work, we used several hardware and software tools (Figure 5.2) to help us build and train the designed models without requiring to dive into low-level implementation details.

5.1 Hardware components

To effectively train and test a machine learning model, a substantial number of calculations of a specific nature must be executed, often requiring multiple iterations with different configurations. When it comes to hardware, GPUs are undoubtedly the ideal devices for conducting these calculations, offering superior performance in comparison to CPUs, specially in batch calculations.

Due to that fact, the hardware infrastructure that we used to perform experiments during this work consisted of high-end servers featuring up to 128 GB of RAM memory and high-performance *NVIDIA GeForce RTX* 20, 30 and 40 series graphics cards, that include GPUs with several machine learning specifically designed cores and up to 24 gigabytes of VRAM (graphics memory).

However, the servers also had high-performance CPUs, particularly *Intel Core* i5 and i7 processors with up to 16 cores running at 5.4 GHz. This processors made it easy to parallelize the event generation pipeline when it was required to generate particles one by one.

The availability of these computing resources was possible thanks to the infrastructure provided by the Pattern Recognition and Human Language Technology (PRHLT) research center, which belongs to the *Universitat Politècnica de València*.

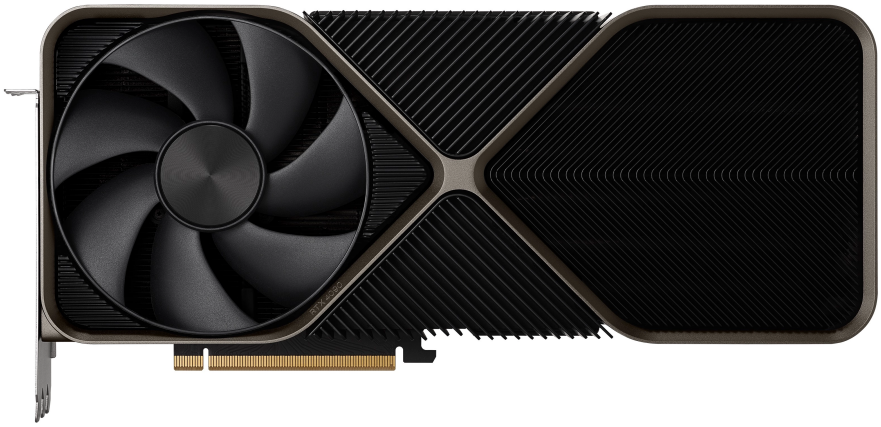


Figure 5.1. NVIDIA GeForce RTX 4090 high-end graphics card. [28]

5.2 Software libraries

In the context of our research, we chose to use a Linux-based software environment, which proved to be a reliable and efficient platform for our experimentation. To carry out our investigations, we decided to use Python as our high-level programming language. This was due to the high availability of libraries related to ML for it, that make it easy to quickly define complex topologies with a comprehensive syntax.

5.2.1. Model topology definition

We selected TensorFlow and Keras as the libraries for model definition and training in our experimentation. This choice provided a robust framework to effectively execute the entire training and generation process.

TensorFlow [29] is an open-source machine learning framework developed by Google. It provides a feature-rich set of tools, libraries, and resources for building and deploying models, making it suitable for a wide range of applications. At its core, computations are represented as computational graphs: a series of interconnected nodes, where each node represents a mathematical function. The data is represented as tensors, which are multi-dimensional arrays.

The framework offers a high-level API that allows developers to define and train ML models easily, supporting a wide variety of models. It also provides a vast collection of pre-built layers, optimizers, and loss functions that can be used to construct models efficiently. TensorFlow has become one of the most popular and widely used frameworks for machine learning and deep learning, powering numerous applications across diverse domains.

Keras [30] is a deep learning framework with a high-level interface for building and training neural networks that runs on top of TensorFlow. It is open-source and designed to be user-friendly, modular, and easy to understand, making it popular among beginners and experienced deep learning researchers. Its intuitive API abstracts away much of the complexity of building neural networks, making it easy to prototype and experiment with different network architectures, as well as customize and fine-tune models to suit their specific needs.

Keras provides a wide range of pre-built layers that can be easily stacked together to construct complex neural network architectures. Additionally, it supports multi-input and multi-output models, allowing the creation of sophisticated networks with multiple data inputs or outputs. The framework also includes various optimizers, activation functions, loss functions, and metrics, as well as utilities for data preprocessing. It also supports both symbolic and imperative programming, offering flexibility and ease of integration with other Python libraries.

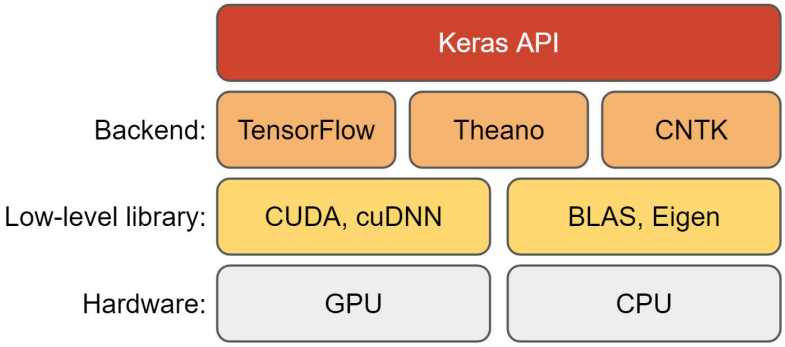


Figure 5.2. Used technology stack with Keras on top. [31]

5.2.2. Data analysis and representation

To assess the quality of the obtained results and evaluate their alignment with the desired objectives, it is necessary to create a visually accessible and easily interpretable representation of the data. In order to accomplish this, we used various libraries and software tools. Specifically, we employed Netron, for visualizing model architectures; and Matplotlib, Scikit-learn and Pandas, for calculating and visualizing statistics of our generated data.

Matplotlib [32] is a popular data visualization library in Python. It provides a wide range of tools for creating different types of visualizations of data. It is designed to be flexible, customizable, making it suitable for both simple and complex plotting tasks.

This library allows to create various types of plots and use its large set of functions and classes for controlling different aspects of them, including axes, labels, titles, legends, colors, markers, and gridlines, to enhance the visual representation of the data. It provides different APIs for creating plots, such as *pyplot*, that is a collection of functions that closely resemble the plotting functions in MATLAB; and the object-oriented API, that provides more control and flexibility by directly manipulating the plot objects. Additionally, it supports various output formats, including saving plots as image files (e.g., PNG, JPEG or SVG).

Netron is a visualization tool used for inspecting and analyzing neural network models. It allows users to view the structure and details of a deep learning model in a graphical format, aiding in model understanding and debugging.

The library supports a variety of file formats, including TensorFlow and Keras models. The tool visualizes the model's architecture, showcasing the layers, connections, and parameters in a clear and intuitive manner. It also supports visualization of intermediate outputs and activations, facilitating a better understanding of how information is processed within the model. In addition to static visualization, Netron offers dynamic capabilities, allowing users to simulate the forward pass of a model and observe the output produced by feeding in sample inputs. This can be helpful for model debugging and comprehension.

Netron is available as a standalone application and it also has a web version, that provides the same functionality as the desktop one. Overall, Netron enhances the understanding of model architectures, aids in debugging, and supports the interpretation of model behavior.

Scikit-learn [33] is an open-source machine learning library for Python. It provides a wide range of tools and algorithms for data preprocessing, feature selection, model training, and evaluation. One of the key strengths of this library is its extensive support for various distance metrics.

Distance metrics are essential in many machine learning algorithms. Scikit-learn offers a comprehensive set of built-in distance metrics that can be used with different algorithms, like the ones we describe in Chapter 6. In addition to these built-in distance metrics, it also provides a consistent API, extensive documentation, and a rich set of functionalities, making it a go-to choice for many data scientists and ML engineers.

Pandas is a Python library built on top of NumPy that provides high-performance, easy-to-use data manipulation and analysis tools. It is widely used in data science, data analysis, and machine learning workflows.

This library introduces two main data structures: the DataFrame and the Series. A DataFrame is a two-dimensional table-like data structure with labeled axes (rows and columns). A Series, on the other hand, is a one-dimensional labeled array capable of holding any data type.

Some key features of the Pandas library include the ability to read data from various file formats, a wide range of operations to manipulate data, powerful tools for exploratory data analysis, and integrations with other libraries such as Matplotlib to allow for easy data visualization.

CHAPTER 6
Evaluation metrics

When working on any machine learning task, it is important to have tools to evaluate the performance of each approach that is proposed. This allows to decide whether a proposal is suitable for performing the task providing enough confidence, and also to compare several approaches to decide which one fits best the desired objective.

While in the beginning of an experimentation process it might be enough to visualize data in a graphical way, for instance by comparing several plots that represent the experimentation results, once the development progresses and several approaches produce similar results, more accurate evaluation methods need to be employed.

In our case, we compared several models in the past by looking at plots of the data distribution of the generated events in several key variables: MET, METϕ, the first and second jets, and the first and second leptons. In this work, we propose several numerical metrics that allow us to compare our models in a more accurate way. Moreover, we extend our observed variables adding new important data to be compared, such as the total energy or the amount of discarded (invalid) events during generation.

6.1 Data distribution similarity

First, we define some metrics to allow for an accurate comparison of the previously mentioned key variables. Until now, this comparison was made by means of a visual evaluation of the generated plots that showed the original and generated data distributions. From now on, we will use the following metrics to compare those distributions:

Wasserstein (Earth-Mover) distance

The Wasserstein distance, also known as the Earth Mover's Distance, is a mathematical measure used to quantify the difference or dissimilarity between two probability distributions.

It calculates the minimum amount of work needed to transform one distribution into the other. In this context, "work" refers to the effort required to move mass from one location to another, where the mass represents the probability density. The cost is determined by the distance traveled by the mass as it is moved from one location to another.

This distance has various applications and is particularly useful when dealing with high-dimensional data or when comparing distributions that have different shapes or support. It provides a metric that captures the structural dissimilarity between distributions, which can be valuable in tasks such as shape recognition or generative modeling.

The first Wasserstein distance between two distributions can be expressed as:

$$W_1(\mu,\nu) = \inf_{\gamma\in\Gamma(\mu,\nu)} \int_{\mathbb{R}^d\times\mathbb{R}^d} \|x-y\|\, d\gamma(x,y) \tag{6.1}$$

where $\Gamma(\mu,\nu)$ is the set of (probability) distributions on $\mathbb{R}\times\mathbb{R}$ whose marginals are μ and ν on the first and second factors respectively.

Jensen-Shannon distance

The Jensen-Shannon distance is a measure of similarity between probability distributions. It is based on the concept of the Jensen-Shannon divergence, which quantifies the difference between two probability distributions and is a symmetrized and smoothed version of the Kullback–Leibler divergence $D(P \parallel Q)$.

The Jensen-Shannon distance between two probability distributions P and Q is defined as:

$$JSD(P,Q) = \sqrt{\frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)} \tag{6.2}$$

where M is the average distribution calculated as:

$$M = \frac{1}{2}(P+Q) \tag{6.3}$$

Note that the probabilities in P and Q must be non-zero for the calculation to be well-defined. Additionally, the Jensen-Shannon distance is bounded between 0 and 1, where 0 indicates identical distributions and 1 indicates completely dissimilar distributions.

Kullback-Leibler divergence

The Kullback-Leibler (KL) divergence is a measure of how one probability distribution differs from another. It quantifies the information lost when one distribution, Q, is used to approximate another distribution, P.

The KL divergence between two probability distributions P and Q is defined as:

$$D(P||Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right) \tag{6.4}$$

where $P(i)$ and $Q(i)$ are the probabilities of event i according to distributions P and Q, respectively.

The KL divergence is asymmetric, which means that $D(P \parallel Q)$ is not necessarily equal to $D(Q \parallel P)$. It is non-negative and becomes zero only when P and Q are identical (i.e., when $P(i) = Q(i) \quad \forall i$).

The KL divergence is commonly used in various fields to measure the difference between probability distributions and to optimize models in tasks like density estimation, clustering, and generative modeling. Although the previously described Jensen-Shannon distance is based on this metric, we decided to include it as it is used directly during model training so it has a more direct relationship with the experimentation process.

We calculate all the presented metrics for each plot, obtaining different measures of how well adjusted is the generated data distribution compared to the *ground truth*.

Finally, to obtain these metrics for the model as a whole, taking into account the values for each variable and effectively combining them, we calculate a weighted average whose weights are the amount of particles of each type that we find in the *ground truth*. Formally:

$$M = \frac{1}{N_{\text{part}}} \sum_i N_{\text{part}}^{(i)} * M_{\text{part}}^{(i)} \tag{6.5}$$

where M is the overall value of the metric, N_{part} is the total number of *ground truth* particles considered, $N_{\text{part}}^{(i)}$ is the amount of particles of type i in the dataset, and $M_{\text{part}}^{(i)}$ is the value of metric M for that type of particle.

In this context, `MET` and `MET`ϕ are also considered as a particle whose weight will be the number of events in the *dataset*, as they are always present. In this way, models can be compared among themselves in a global way using these metrics.

6.2 Event generation quality and efficiency

Once we have defined some basic metrics to improve our comparison methods among several model proposals, we also define two additional metrics that expose the efficiency and quality of our generated events when using each model.

Event quality

For defining a quality metric, we observe the dataset conditions for an event to be considered valid, that we already mentioned in Section 3.1.

In this case, the metric is very straightforward and describes the percentage of events that do not comply with the mentioned conditions. This metric, that we denote as Invalid Event Rate (IER), can be defined as follows:

$$IER = \frac{N_{invalid}}{N_{total}} * 100 \tag{6.6}$$

where $N_{invalid}$ is the amount of events that do not comply with one or more conditions and N_{total} is the total amount of events considered. It is assumed that the *ground truth* has $IER = 0\%$, as it was generated taking those conditions in mind.

It is important to remark that meeting the conditions of this dataset does not give any guarantee that the event is physically correct and vice versa. Therefore, it must be taken as an orientation.

Generation efficiency

To evaluate the efficiency of the event generation process, we decided to obtain the time required to generate an event. Due to the fact that events have a variable shape in terms of number of objects, different events can take different times to be generated.

For obtaining a reliable metric, we decided to generate 1K events with each model and then calculate the average generation time among that quantity. With this strategy, we ensure that events of different lengths will be generated and therefore this value can be used for comparing different models. The metric is formally defined as:

$$Generation\ time = \frac{T_{1K}}{1000} \tag{6.7}$$

where T_{1K} is the amount of time that takes to generate 1K events. It must be noted that the reason of including this metric in our comparison is that different approaches for generating events can have significant differences in the generation time depending on the possibility of benefiting from the capabilities of batch generation of the GPU.

CHAPTER 7
Experimentation

In this section, we discuss the development and experimentation conducted based on the proposed solution outlined in Chapter 4. We use the technologies described in Chapter 5 and analyze the results of each experiment using the metrics explained in Chapter 6.

During our investigation to determine the optimal model for event generation, we chose to concentrate on a specific type of event from the Standard Model: *ttbar*, which provided a substantial number of event samples. Subsequently, we applied this model to other event types. In this process, defined in Table 3.2, the proton collision generates two top quarks that decay into other particles.

Before delving into the specifics of the various approaches, it is important to highlight certain details regarding the implemented sequential data loading of our extensive dataset. Additionally, we applied preprocessing techniques based on observations made during the analysis.

7.1 Sequential data loading

The dataset of each event type used for the experiments contains a large amount of data, with all the event types adding up a total of 66.5 GB, as it was described in Chapter 3. Although the data of a single event type could technically fit into our system, which had enough RAM memory, it is always desirable to perform some optimization in order to load only the data that is useful at every moment to the model and not the whole dataset.

Because of that, we wanted to develop a strategy to process this dataset by loading small portions or **batches** of data, one at a time, so we decided to load events sequentially. To do so, we created our own implementation of the `Sequence`[1] class provided by TensorFlow. This class allows their instantiated objects to be provided as the input and output of Keras models, and provides the interface of the required functions to handle batch processing appropriately.

The class has a variable representing the **batch size**[2], another one defining the total amount of batches, and it also includes every file opened (but not loaded in memory) to extract data from it on demand. When the model requests a batch to be loaded, the specified file is accessed obtaining events one by one (Figure 7.1), until the batch reaches the desired size. Furthermore, when the end of a file is reached it is treated as a *circular array of lines*, and its internal pointer returns to the beginning. In this way, we avoid crashes during training for particles that have a low rate of appearance.

[1] https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence
[2] Amount of events included in each batch.

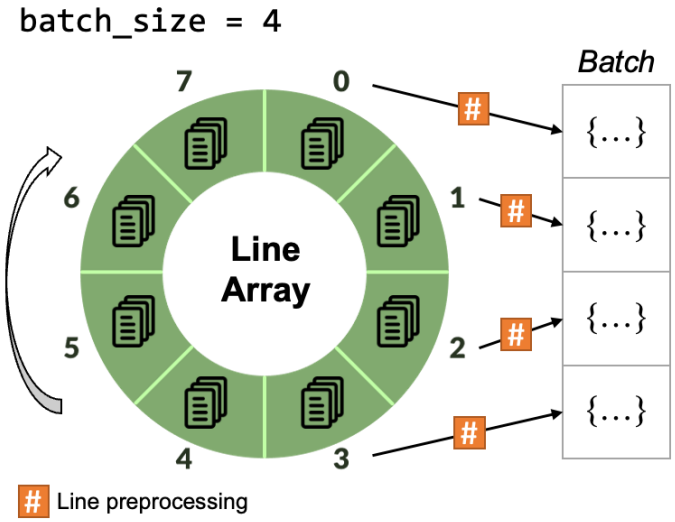


Figure 7.1. Representation of the circular array of lines of a file when creating a batch.

7.2 Data analysis

Before performing any experiment, the dataset was first analyzed to see if the different variables followed any known distribution. After considering separately all kinds of particles that can be found in the events (listed in Table 3.1) it was then observed that, among the four features that describe each particle:

- E and p_T have the shape of a log-normal distribution[3] that can be converted into a normal distribution by applying the logarithm, as seen in Figure 7.2
- ϕ followed a uniform distribution as seen in Figure 7.3
- η did not seem to describe any known shape

Then, after carrying out the described analysis, this information was taken into account to convert data into a proper format to be provided as input to each model, applying some transformations.

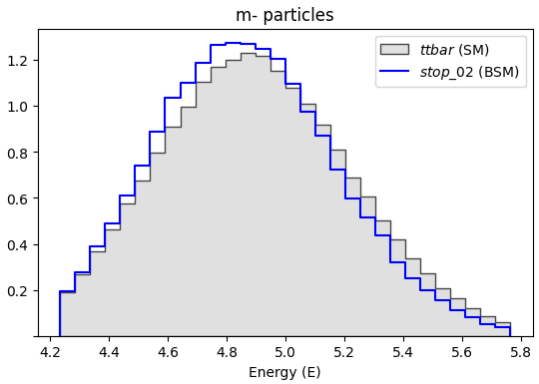


Figure 7.2. Histogram of muon object parameter E in the used SM and BSM events.

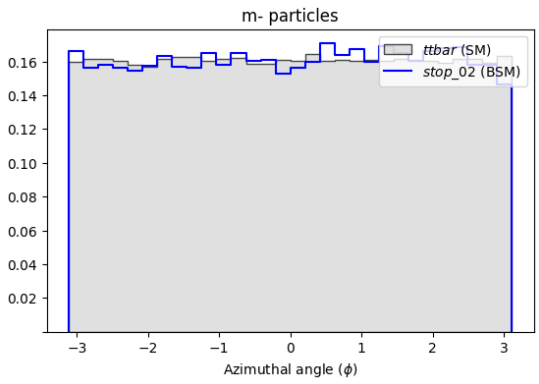


Figure 7.3. Histogram of muon object parameter ϕ in the SM and BSM dataset.

Most of the histograms that were generated during this data distribution analysis process are available in annex B.

[3]Continuous probability distribution of a random variable whose logarithm is normally distributed.

We also calculated the correlation coefficient between certain event features that are of great importance from a physical point of view (as described in [34]) using the `corr()` function of the `DataFrame` class in the Pandas library. This was done to observe if some features were related and also to allow us to check that the relations are preserved during experimentation. The results are represented in the heatmap of Figure 7.4.

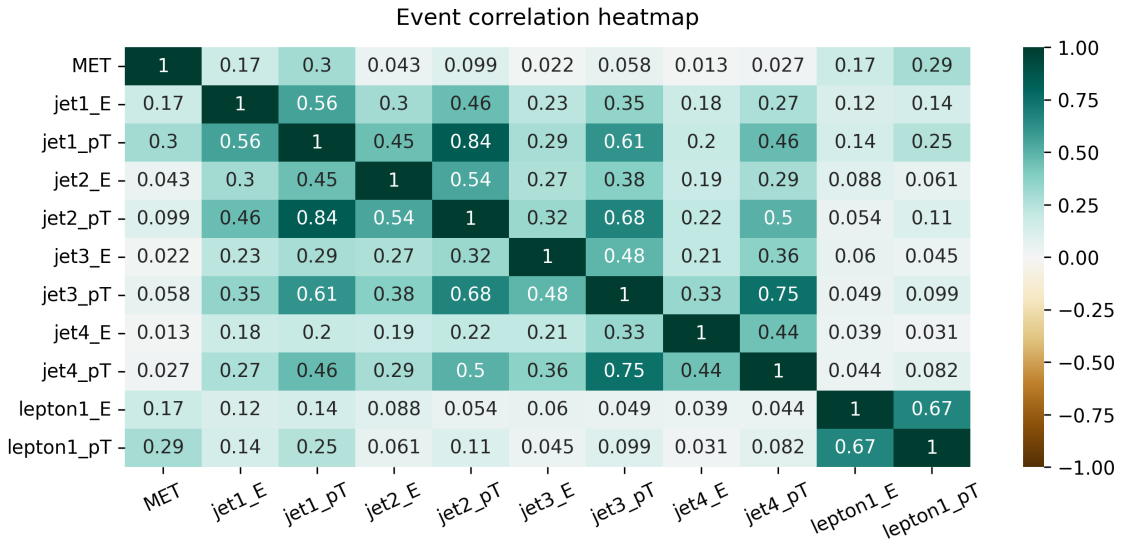


Figure 7.4. Correlation heatmap of several event features in the *ttbar* dataset.

As we can observe, the energy (E) and transverse momentum (p_T) of the first four jets present a high correlation in each particle order and also between different orders. Moreover, the first lepton's E and p_T are also correlated among them but not with respect to other particles (jets). Finally, we do not observe a significant correlation between the missing energy (`MET`) and any particle.

7.3 Experiment 0: Baseline model

Before describing the experiments that we performed and its results, we show here our baseline model, that consists on the best model that we obtained in our previous work [3]. This model is an α-VAE architecture that performed in slightly better than the best β-VAE model that we tried in that work.

It must be noted that every model in our previous work generated events by taking *ground truth* data and applying some permutation or noise to it, and not from random numbers from a probability distribution.

7.3.1. Preprocessing

Considering the dataset features from Chapter 3 and the data analysis conclusions obtained in Section 7.2, the input to the model was represented as follows.

First, it was decided that each **event** would be described as a tensor composed of 2 values (`MET` and `MET`ϕ) and 19 **objects**, each with its symbol ID and four features. If an event consisted of $k < 19$ objects, the $(19 - k)$ remaining objects were filled using a special *mask* object, with a special ID and all fields set to 0.

In each object, the features E and p_T (including `MET`) were converted into $log_{10}E$ and $log_{10}p_T$, and the symbol ID was represented using *One-Hot Encoding*[4]. The reason for this last choice instead of a numerical constant was to avoid the algorithm to misinterpret that constant as having some meaning apart from a simple identification of the object.

After applying these transformations, the final format of an event would be:

$(log_{10}MET, MET\phi)$		$MET/MET\phi$ data
$(x, x, x, x, x, x, x\ x)$	$(log_{10}E, log_{10}p_T, \eta, \phi)$,	Particle object
$(x, x, x, x, x, x, x\ x)$	$(log_{10}E, log_{10}p_T, \eta, \phi)$,	Particle object
... ,		
$(0, 0, 0, 0, 0, 0, 0, 1)$	$(0, 0, 0, 0)$	*Mask* special object

where there is a total of 19 objects, and the vectors filled with x characters represent the *One-Hot Encoding* of the different objects present in each event. The result is a tensor of dimension $1x2$ containing `MET` and `MET`ϕ, another of dimension $20x8$ containing object identification, and another one of dimension $20x4$ containing object features. With those transformations, data was already converted to the correct format for the model.

7.3.2. Model architecture

In the α-VAE of our baseline model, the whole training process is divided in different steps, and is followed by the generation part.

First, a Variational Autoencoder is trained using **all the events of the *ttbar* dataset**. This VAE has three sub-components, unlike the usual autoencoder architectures. This design choice was made for clarity, in order to make clear the difference between α and β VAEs. Our implementation is structured as follows (Figure 7.6):

- an **encoder**, which transforms the input data in the format described in Section 7.3.1 into an encoded representation in the latent space, producing the embedding of the provided input event as output.

[4] A vector with as many values as the amount of different objects where, for each object, a 1 is placed in the position in which that object ID belongs, leaving the others with 0.

The encoder of this baseline model features a *Flatten* layer, followed by two blocks composed by a *Dense* layer of 512 nodes and a ReLU activation function, all of it applied to each input separately. Then, all inputs are concatenated and two blocks of a *Dense* layer with 512 nodes + a ReLU activation function are applied after the concatenation. After that, a *Dense* layer of twice the latent dimensionality together with Batch Normalization is added to the model. Finally, we put two *Dense* layers of as many nodes as the latent dimensionality to sample the means and logarithms of the variance, providing the data that the next component requires.

- a **variator**, that receives the output of the encoder and, together with a normal distribution with $\mu = 0$ and $\sigma = \alpha$, adds Gaussian noise to the latent representation and outputs $\hat{z}$.

  The dimensions of the $\hat{z}$ tensor match the latent dimensionality. The variator component is usually part of the encoder in most implementations.

- a **decoder**, that transforms the encoded data from the latent space back into its original dimensions.

  The decoder of this approach is composed by two blocks of a *Dense* layer of 512 nodes, followed by a ReLU activation layer. Then, the output is divided into the `MET` and `MET`ϕ, particle type and particle layer tensors, with a *Dense* layer of 512 nodes in each + a ReLU activation function, followed by another *Dense* layer with linear activation of 2, 160 ($20x8$) and 80 ($20x4$) nodes respectively, connected to a *Reshape* layer to *unflatten* the data. In the case of the particle type, a final Softmax activation layer is included.

Our VAE used *Categorical Crossentropy* as the loss function for the particle type detection, and the *Mean Squared Error* for learning the different object features (including `MET` and `MET`ϕ). See appendix A.2 for a detailed description of loss functions.

Once the α-VAE is trained following the process depicted in Figure 7.5 and using a batch size of 1000 events for 50 **epochs**[5], it is used to generate encodings of events that are altered with Gaussian noise and then decoded into newly generated events, as it is shown in Figure 7.7.

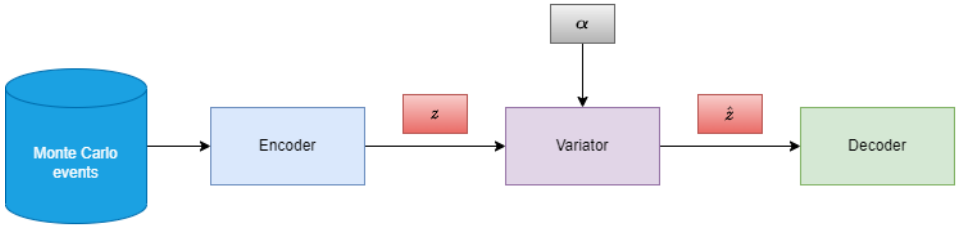
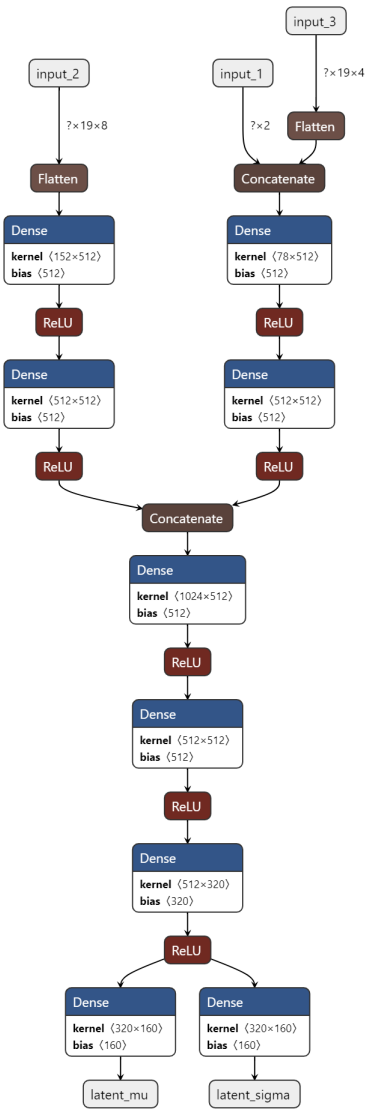
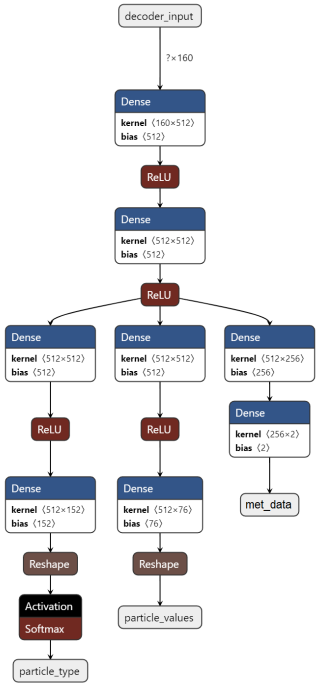


Figure 7.5. Training the α-VAE using one kind of event.

[5]Hyperparameter that defines how many times the learning algorithm goes through the training dataset.



(a) Encoder



(b) Decoder

Figure 7.6. VAE architecture scheme for the baseline model (variator omitted for simplicity).

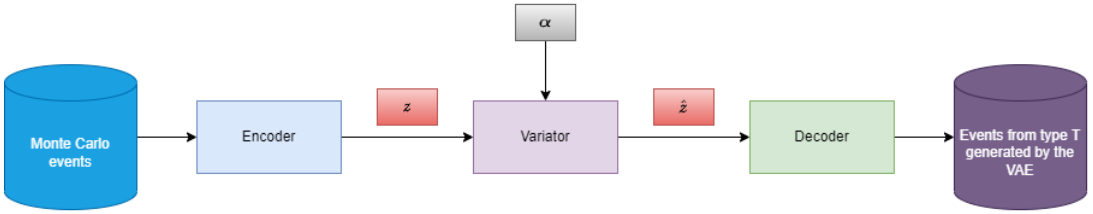


Figure 7.7. Generating events with the α-VAE.

7.3.3. Training results

After training the previously described model using several α values and generating 100K events, the best results were obtained with $\alpha = 0.2$, value with which we obtained the results depicted below. Refer to our previous work for a comparison of the different tested hyperparameters and a detailed analysis of the results.

The key aspects after observing the results of the baseline experiment, that we will take into account during the experimentation in this work, are:

- There is a great adjustment of the distributions of most parameters, including the event parameters (MET, METϕ and the total energy), as we can observe in Figure 7.8.
- We observe the biggest differences in the p_T parameter of the leptons and photons, where the values from the right and left tails of the *ground truth* events distribution are located in the central part of the generated events distribution, as we can see in Figures 7.10 and 7.12.
- The E parameter of some particles also presents some differences, being again leptons and photons the particles with the biggest difference.
- It must be taken into account that differences in the photon particle distributions with respect to the generated ones could be due to the low amount of those kinds of objects in the original *ttbar* dataset.

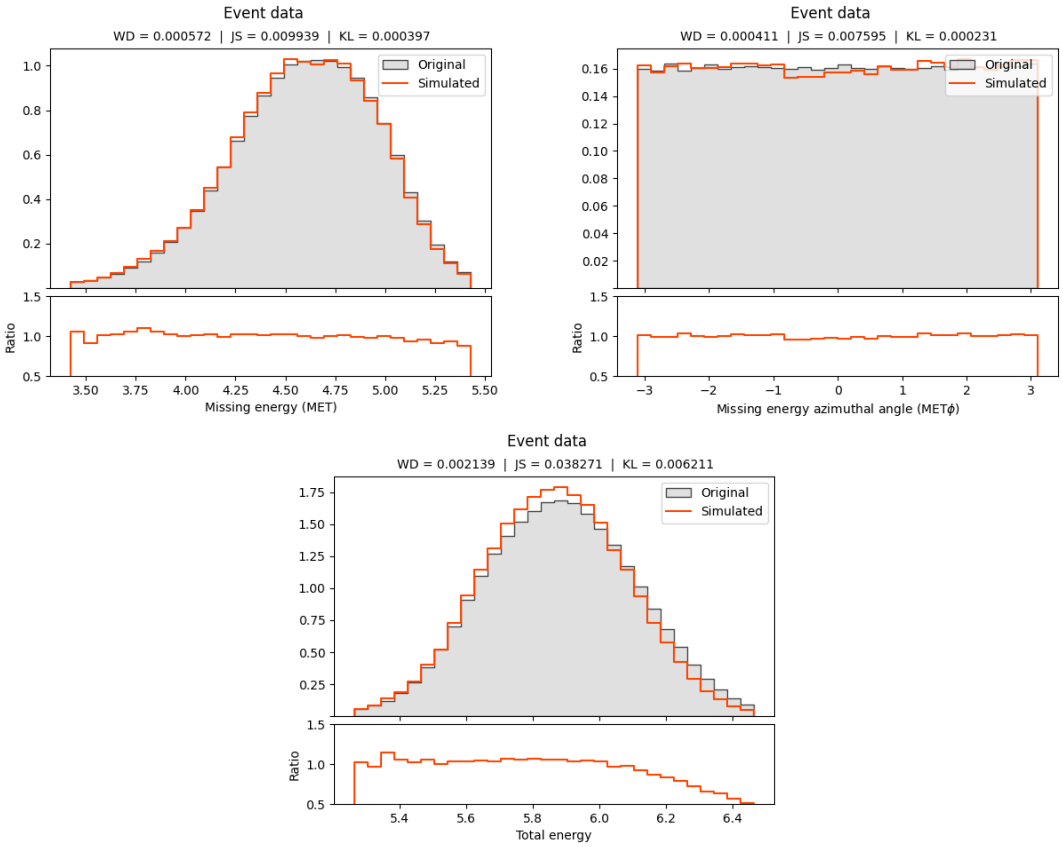


Figure 7.8. Histograms of missing energy (MET and METϕ) and total energy in each event.

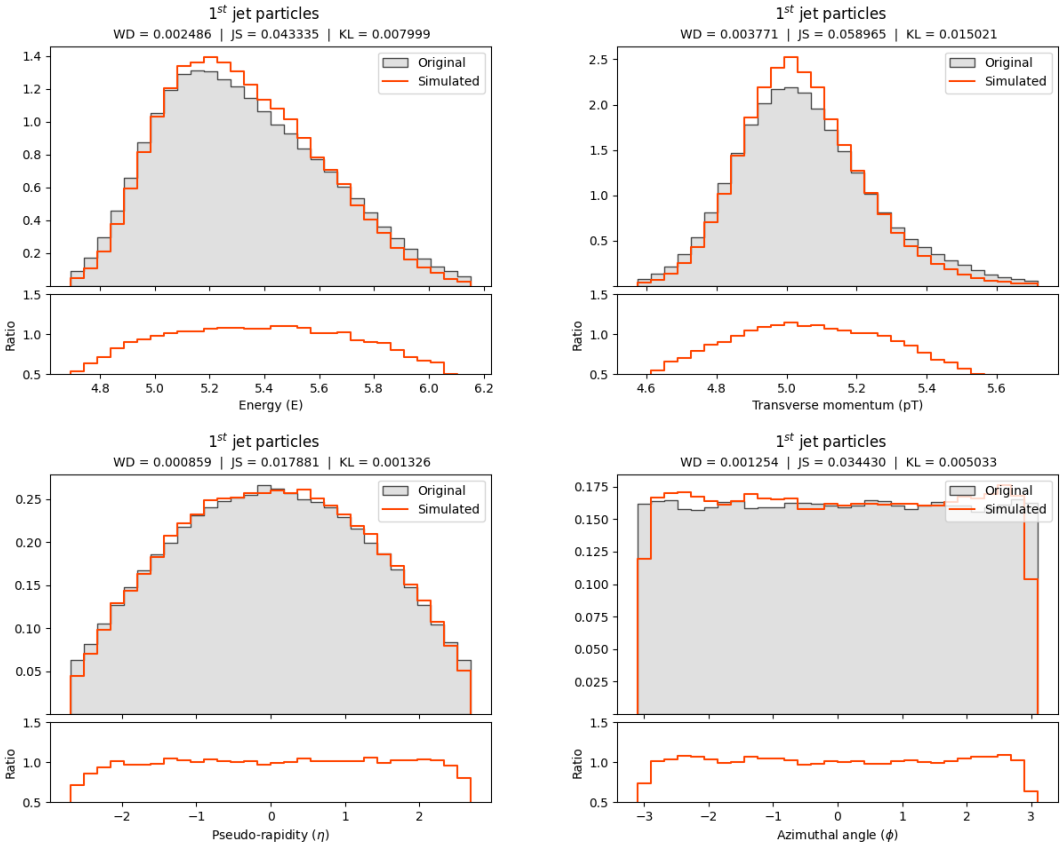


Figure 7.9. Histograms of first jet parameters in the baseline experiment ($\alpha = 0.2$).

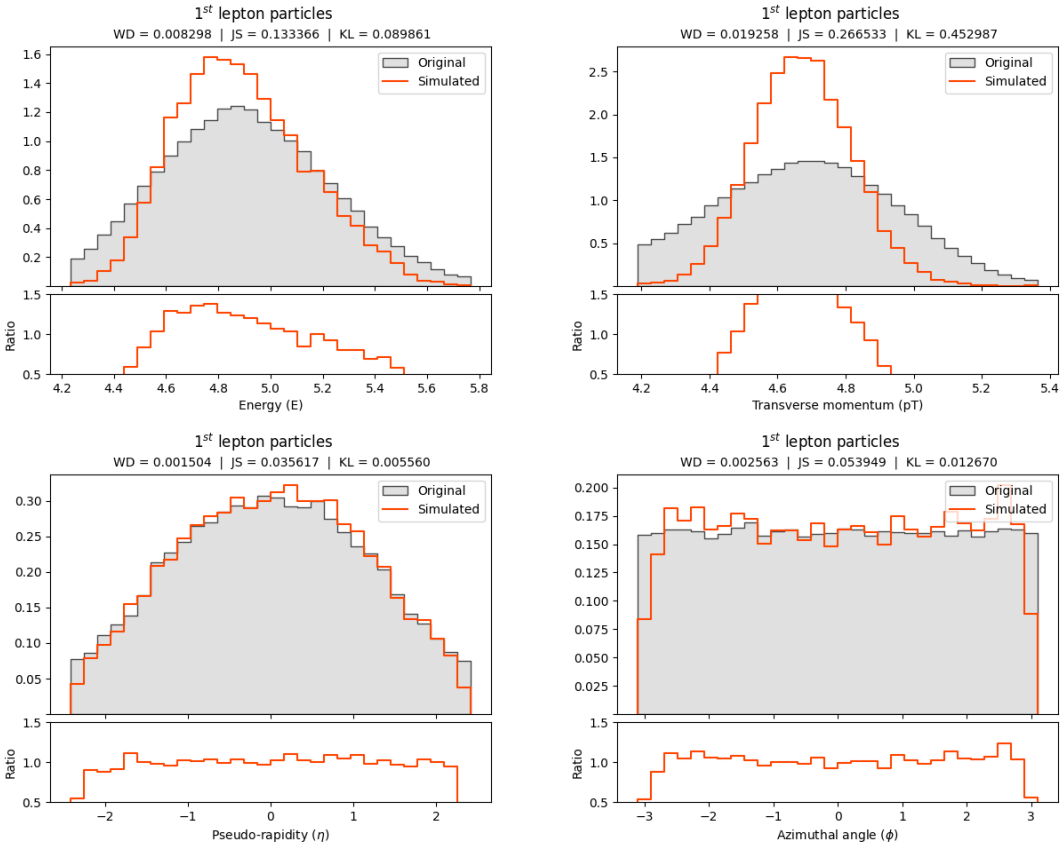


Figure 7.10. Histograms of first lepton parameters in the baseline experiment ($\alpha = 0.2$).

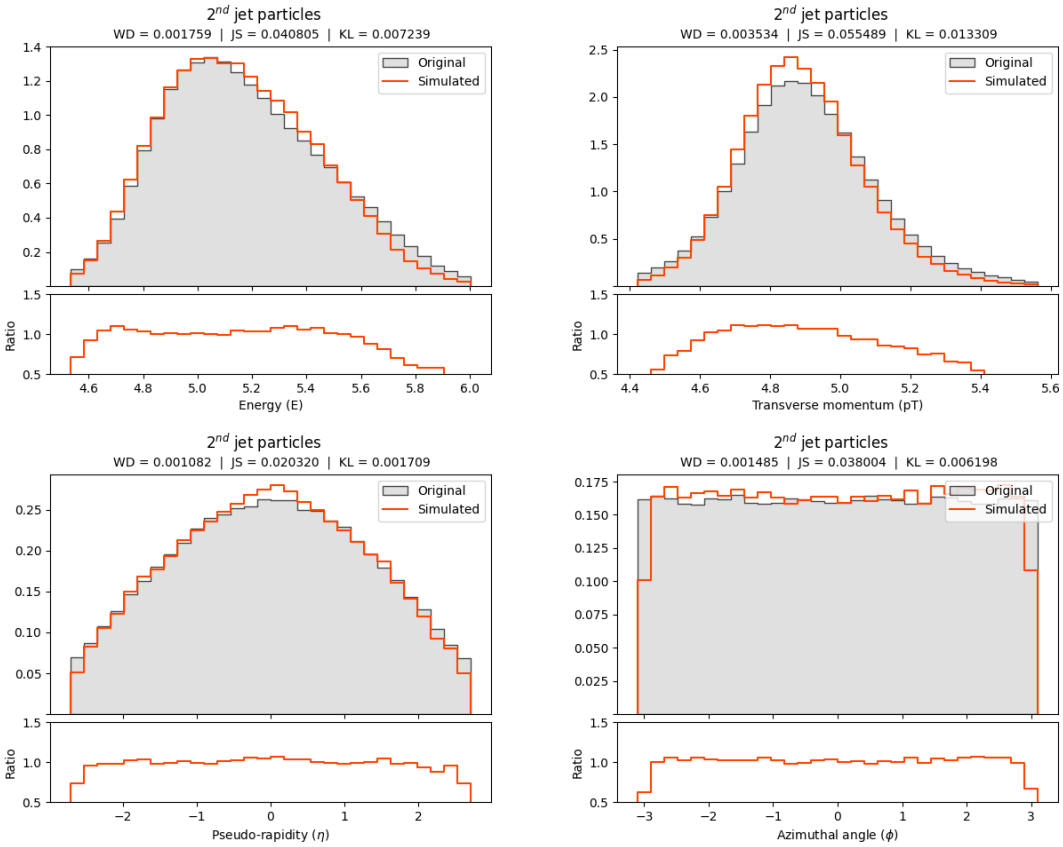


Figure 7.11. Histograms of second jet parameters in the baseline experiment ($\alpha = 0.2$).

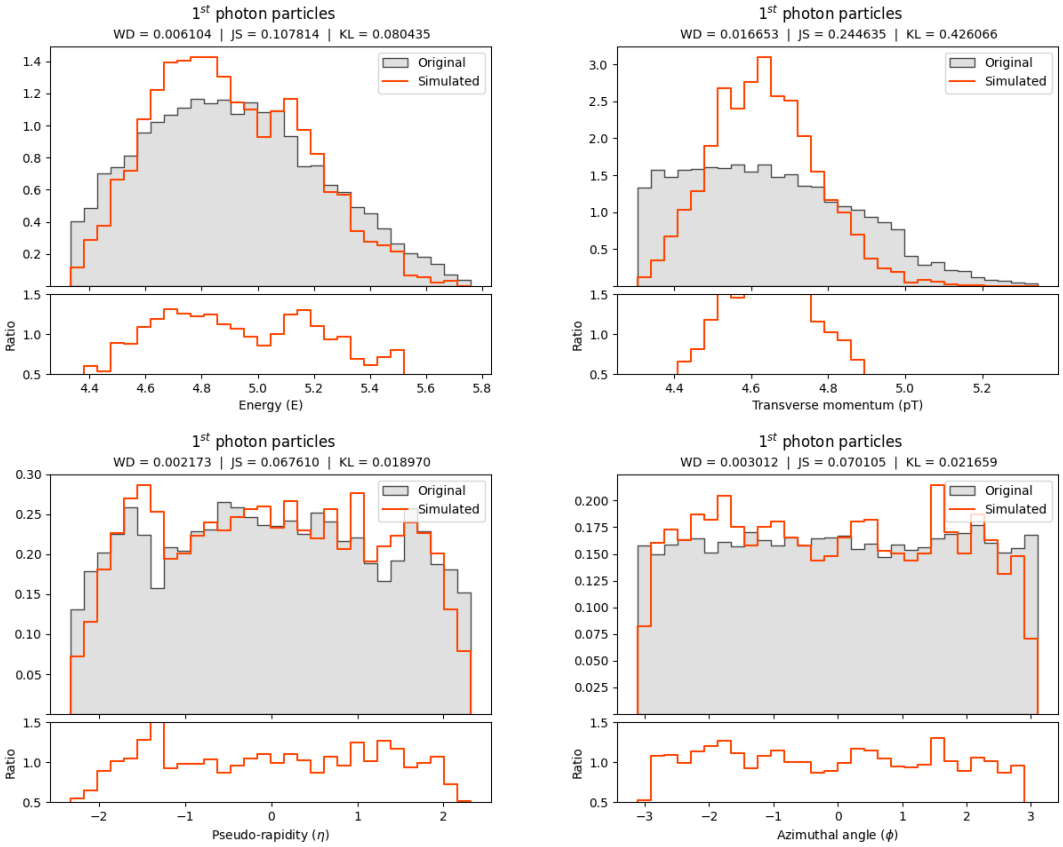


Figure 7.12. Histograms of first photon parameters in the baseline experiment ($\alpha = 0.2$).

After observing all the results from the baseline model, we can conclude that it is able to generate events with a relatively good adjustment to the original events distribution, as we can see in the metrics in Table 7.1.

Moreover, we can see in Figure 7.13 that the model is able to maintain in an accurate way most of the correlations that we observed during the data analysis process in Section 7.2, keeping the relationship among jet and lepton particles.

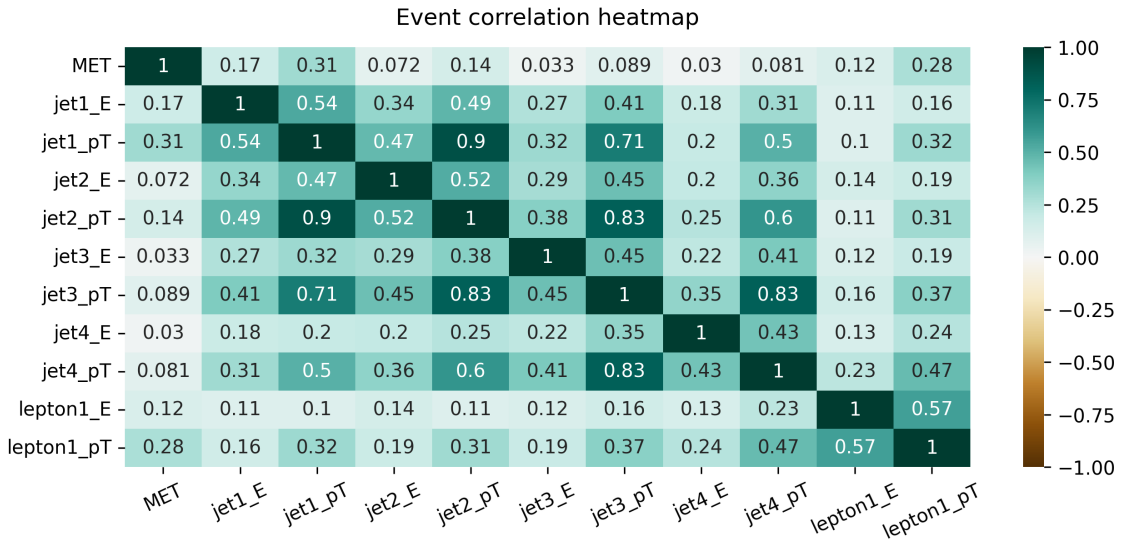


Figure 7.13. Correlation heatmap of several features on events generated by the baseline model.

We also highlight the surprisingly fast generation speed of this model, that benefits from the use of batch operations in the GPU by generating events in batches of 1K events each time, and also because of the fact that this approach only uses one model to learn the distribution of the whole event.

However, it must be taken into account that this event generation process (like all architectures that use α-VAE models) takes as its starting point *ground truth* events, and then a perturbation is applied to them in order to generate new events.

α	**Data distribution metrics**			**Event generation quality**	
	WD	*JS*	*KL*	*IER*	*Gen. speed*
0.2	0.002077	0.037879	0.013133	1.68%	1.23 ms

Table 7.1. Metrics of events generated by the baseline model.

7.4 Experiment I: Binary mask and per-particle model

Once we have analyzed our best model so far, now our two main objectives are **(1)** to correct the differences that were observed in the baseline model and **(2)** to design a generation pipeline that does not need *ground truth* events as a data source in the generation process.

Our first experiment, that focuses on achieving the first objective, is composed of several α-VAE models that focus in different parts of each event, with the aim of being capable to learn a more accurate distribution in the latent space that we can use afterwards to generate better events by combining all models.

7.4.1. Preprocessing

Due to the fact that this experiment requires to train different models separately, each event *dataset* has to be subdivided into different files corresponding to that amount of models. Therefore, this preprocessing generates the following data:

1. One file that contains, in each line, `MET`, `MET`ϕ and a binary particle mask of an event. This binary mask contains $7 * 4 = 28$ numbers where each block of 4 indicate the amount of particles of a certain type. The order of particles in the mask is the same of the rows in Table 3.1.

   Example: the mask `{1100 1100 0000 0000 0000 0000 0000}` indicates that there are 2 *jet* and 2 *bjet* particles in the event.

2. One file per particle type that also includes in each line `MET` and `MET`ϕ of the event where the particle was found, the four particle features (E, p_T, η, ϕ), and a binary mask of 4 digits indicating in which position was that particle among the particles of its same type inside the event.

   Example: in the *jets* file, the mask `{1110}` indicates that the particle was the third jet in the event where it was found.

This data separation produces 8 files in total: one for `MET`, `MET`ϕ and the binary mask; and 7 more, one for each particle type. Moreover, it is worth mentioning that particle features were standardized by removing its mean and scaling to unit variance during this process, and re-scaled after generation.

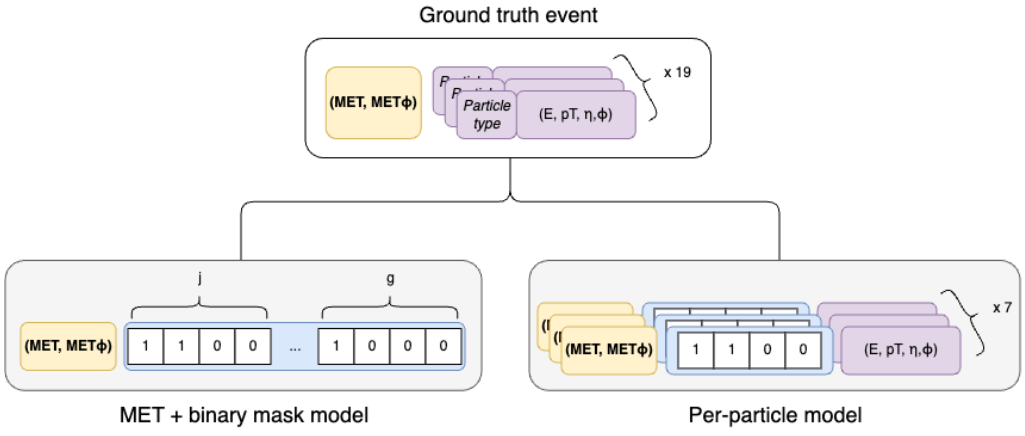


Figure 7.14. Event separation into several files for each independent model in experiment I.

7.4.2. Model architecture

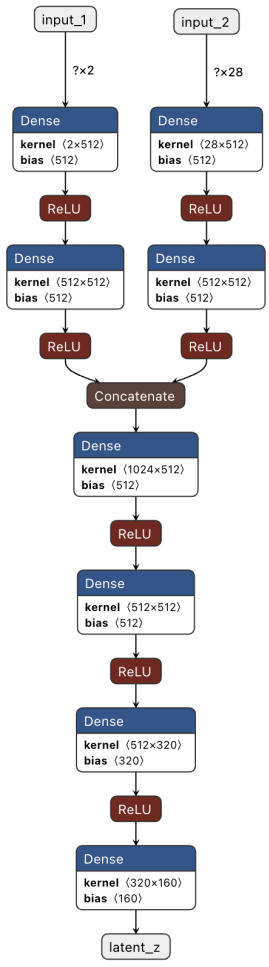
The models used in this experiment are also α-VAE based. A total of 8 models (corresponding to the files generated in the preprocessing stage) are trained, each of them sharing practically the same structure.

Regarding the **MET and binary mask** model (Figure 7.15), we observe that it consists of the following components:

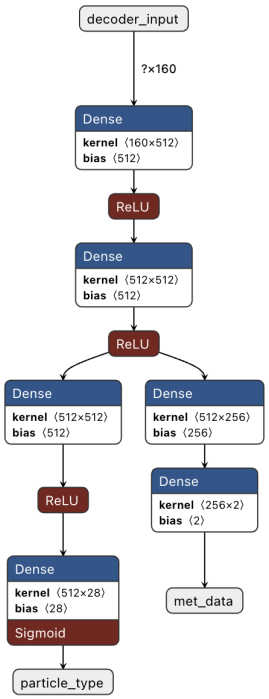
- an **encoder**, which transforms the input data in the format described in Section 7.4.1 into its encoded representation in the latent space, producing the embedding of the provided input event as output.

  The encoder of this model features two blocks composed by a *Dense* layer of 512 units and a ReLU activation function, all of it applied to each input separately.

  Then, both inputs are concatenated and two blocks of a *Dense* layer with 512 units + a ReLU activation function are applied after the concatenation. After that, a *Dense* layer of twice the latent dimensionality (320) together with Batch Normalization is added to the model. Finally, we put a *Dense* layer of as many units as the latent dimensionality to sample the latent space embedding, providing the data that the next component requires.



(a) Encoder



(b) Decoder

Figure 7.15. α-VAE architecture scheme for the MET and binary mask model of experiment I.

- a **variator**, that performs the same task in all α-VAE implementations (already described in Section 7.3.2).
- a **decoder**, that transforms the encoded data from the latent space back into its original dimensions.

  The decoder of this approach is composed by two blocks of a *Dense* layer of 512 nodes, followed by a ReLU activation layer. Then, the output is divided into the particle type + MET and METϕ tensors:

  - The **particle type** output is followed by a *Dense* layer of 512 nodes + ReLU activation, and another *Dense* layer of 28 nodes with a Sigmoid activation for the binary mask that indicates the amount of each particle type.
  - The **MET data** output is followed by a *Dense* layer of 256 units and a final one of 2 units with linear activation.

Moreover, each **particle type** model consists of the same components with an identical structure in the variator. The structure of the remaining two components are:

- the **encoder**, that features the same structure as the previously described model with two differences. First, the binary mask input only has four binary values (instead of 28), representing the order of the particle inside the event where it was found. Second, an additional input is added with four values that represent the particle features (E, p_T, η, ϕ).
- the **decoder**, which is composed by the same outputs as the MET and binary mask model, but using 4 units (instead of 28) for the output of the binary mask and adding an additional output after the common *Dense* layer of 512 units + ReLU that contains another *Dense* layer of 512 nodes + ReLU activation, and another *Dense* layer of 4 nodes with linear activation.

After defining the different models, every Variational Autoencoder is trained using **the *ttbar* dataset** with a batch size of 1000 events for 50 epochs. All the models use *Binary Crossentropy* as the loss function for the binary mask distribution, and the *Mean Squared Error* for learning the different object features (including MET and METϕ).

Once training ends, the models are used to generate encodings of *ground truth* events that are altered with Gaussian noise and then decoded into newly generated events, following this procedure per event (Figure 7.16):

1. Obtain a *ground truth* MET and METϕ with a particle binary mask and pass it as input to the encoder of the binary mask model to get its latent space encoding.
2. Apply a perturbation (Gaussian noise) to the obtained encoding and provide the result as input to the decoder, generating a new MET, METϕ and particle mask.
3. Because the generated mask is composed of floating point (non-binary) values, apply a threshold to binarize it (in our case we used 0.1 as the threshold).
4. For each particle and particle order that the mask has predicted, use the generated MET and METϕ, a mask indicating the particle order, and *ground truth* particle features as inputs to the encoder of the corresponding particle model.
5. Apply a perturbation to the encoding of the new particle and provide it as input to the decoder of the model, obtaining a newly generated particle to add to the event.
6. Repeat the process from step 4 until having at most 19 particles.

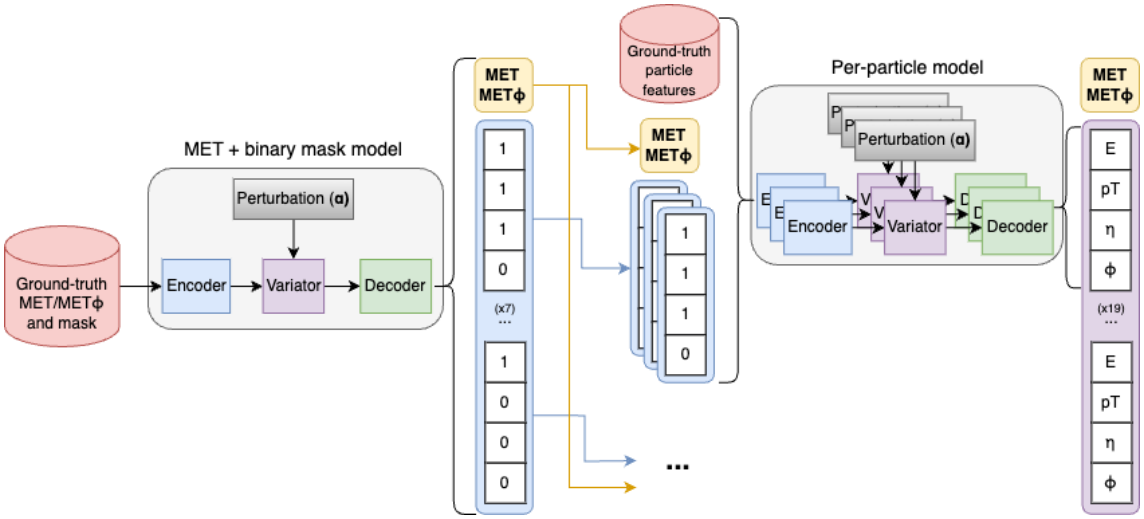
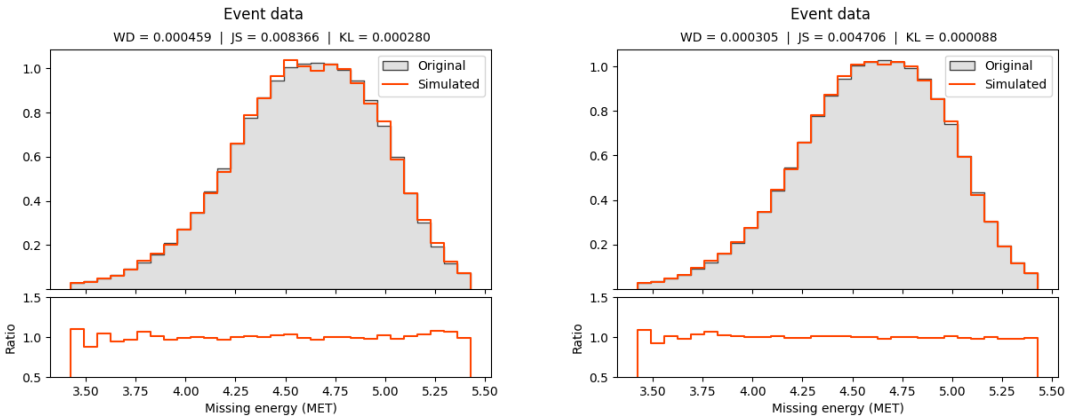


Figure 7.16. Event generation pipeline combining the use of every model in experiment I.

7.4.3. Training results

After training the models of the previously described pipeline with $\alpha \in \{0.1, 0.2, 0.3, 0.4\}$, and generating 150K events using every model trained with each α value, we obtained some promising results that we analyze while observing the following figures.

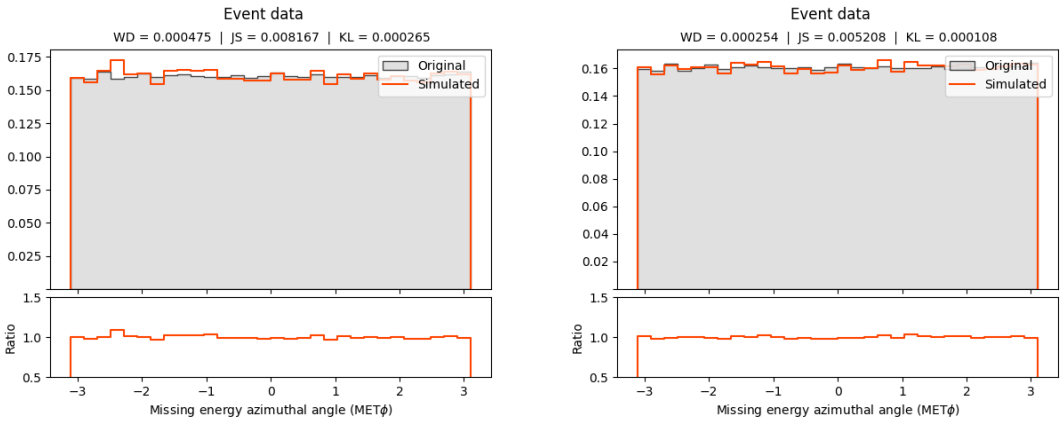
During this whole chapter, including this experiment, only the histograms for some of the tested values of α and β are shown for clarity reasons, and to avoid showing values that only have slight differences among them.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.17. Histograms of `MET` event feature with different α values in experiment I.

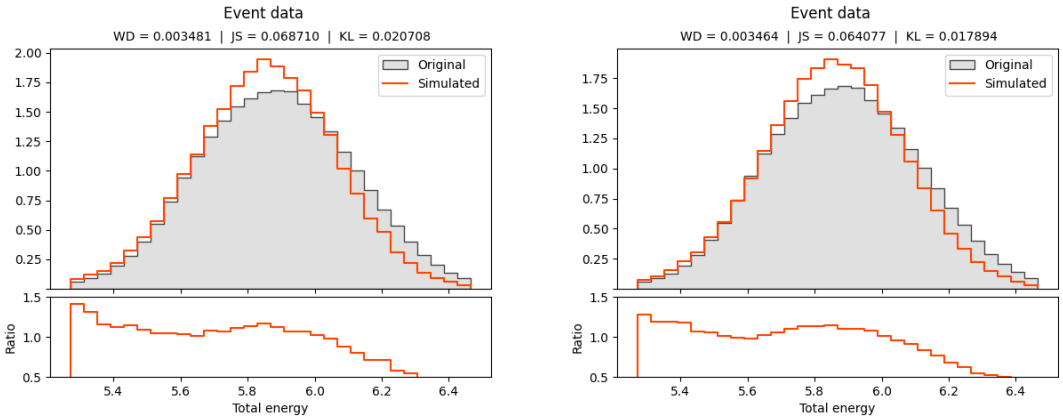
First of all, looking at the `MET` event feature histograms we see an almost perfect adjustment with respect to the original data distributions considering all α values, showing almost identical distributions with only some minor differences in the $\alpha = 0.2$ histogram. The ratio histograms show values that are very close to 1.0 along the MET bins.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.18. Histograms of METϕ event feature with different α values in experiment I.

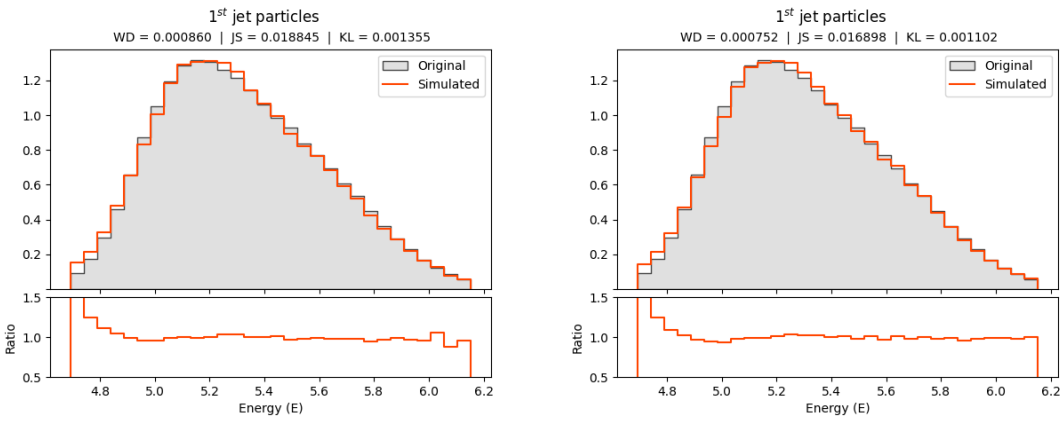
Regarding the METϕ event feature, once again the distributions with resemble the original data, with the ratio maintaining values very close to one. This result is noteworthy, as the ϕ attributes have been one of the most difficult to learn by the models so far.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.19. Histograms of total energy event feature with different α values in experiment I.

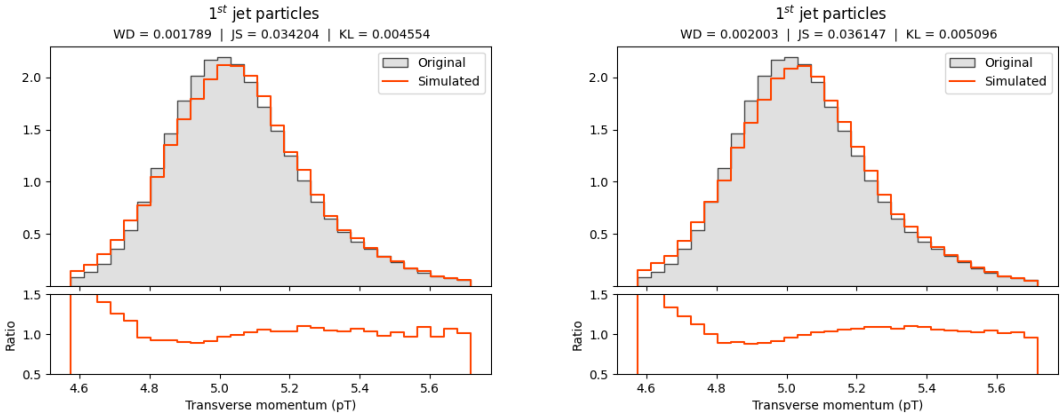
Looking at the total energy of each event feature, we observe a slight difference in the central values of the distribution, that we find more often in the simulated data when compared to the original events.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.20. Histograms of 1^{st} jet E particle feature with different α values in experiment I.

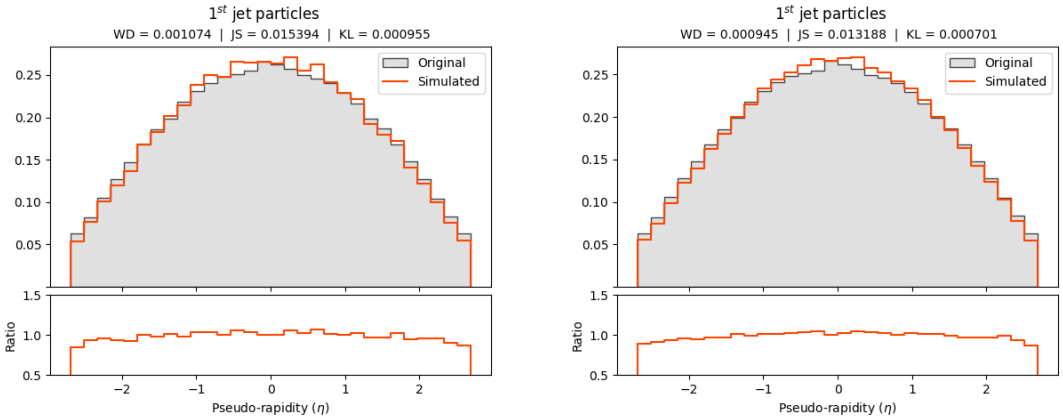
Moving on to particle features, and beginning with the E feature of the first jet, we also find a very similar distribution of the generated events when compared to the original distributions, observing the biggest difference in the lowest values of this feature, with ratios that are around 1.5. As values of the energy increase, this ratio gets very close to one again.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.21. Histograms of 1^{st} jet p_T particle feature with different α values in experiment I.

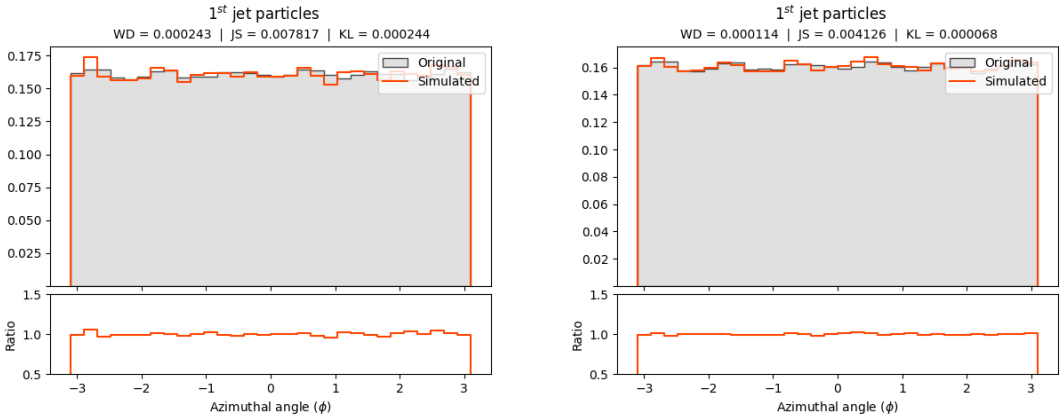
Regarding the p_T particle feature of the first jet, we find some small discrepancies that are more notable in the lowest values, but also spread over the entire value range of the aforementioned feature. However, the data distribution is also very similar to the Monte Carlo generated events.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.22. Histograms of 1^{st} jet η particle feature with different α values in experiment I.

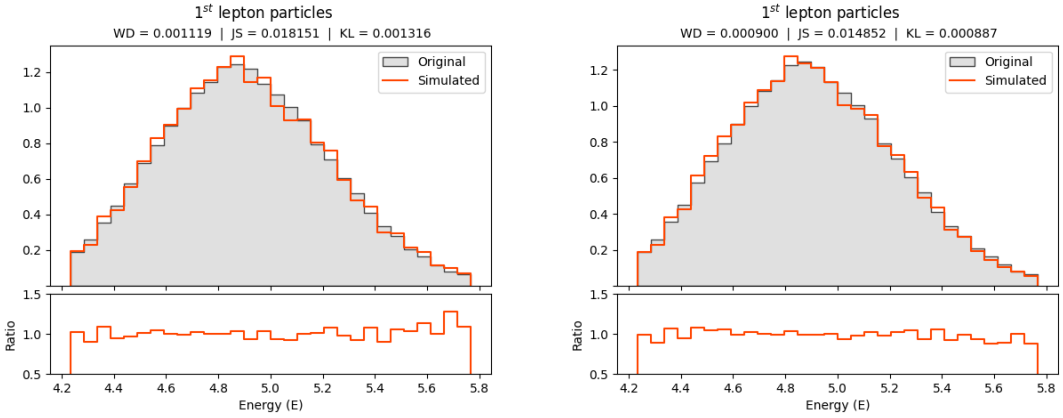
Analyzing the η feature of the first jet, we find the highest differences in the center of the distribution, where we find more simulated events with values that are close to zero. Nevertheless, this distribution, which is not exactly a Gaussian one, seems to have been correctly modeled. Moreover, we see low values of the metrics in both cases with a little improvement for high values of β.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.23. Histograms of 1^{st} jet ϕ particle feature with different α values in experiment I.

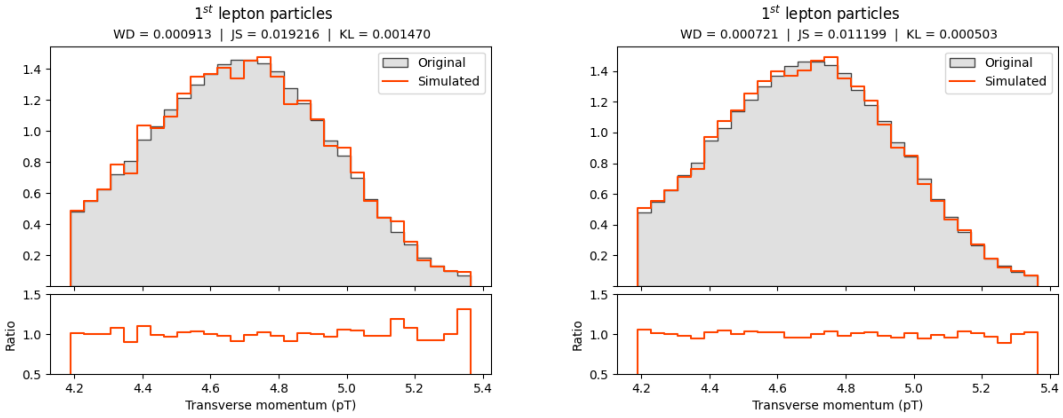
Here, in the ϕ particle feature of the first jet, we also see a great adjustment similar than what it was already observed in the $\texttt{MET}\phi$ event feature. The uniform distribution is very similar in both the generated and *ground truth* events.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.24. Histograms of 1^{st} lepton $\texttt{E}$ particle feature with different α values in experiment I.

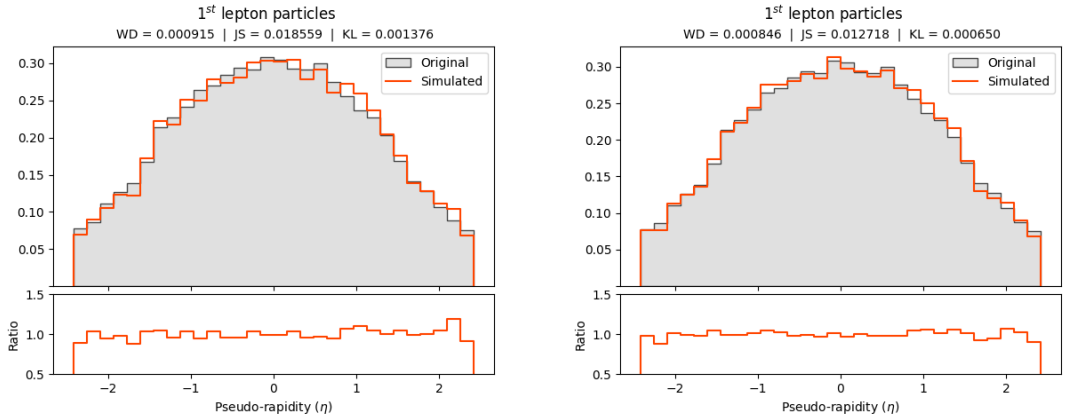
Observing the E feature of the first lepton, we see that all the values are very similar to those found in the original Monte Carlo events, observing the biggest ratio in the highest values of the model with $\alpha = 0.2$.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.25. Histograms of 1^{st} lepton p_T particle feature with different α values in experiment I.

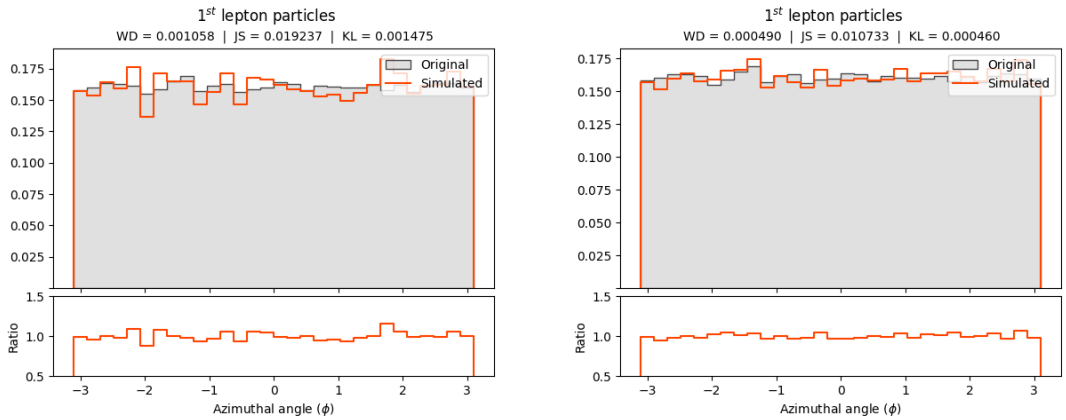
The p_T feature of the first lepton was one of the particle features that showed a higher discrepancy in our baseline model. Nevertheless, in this experiment we observe a greatly adjusted distribution with only some slight differences with respect to the original data, denoting a big improvement with the approach of this experiment.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.26. Histograms of 1^{st} lepton η particle feature with different α values in experiment I.

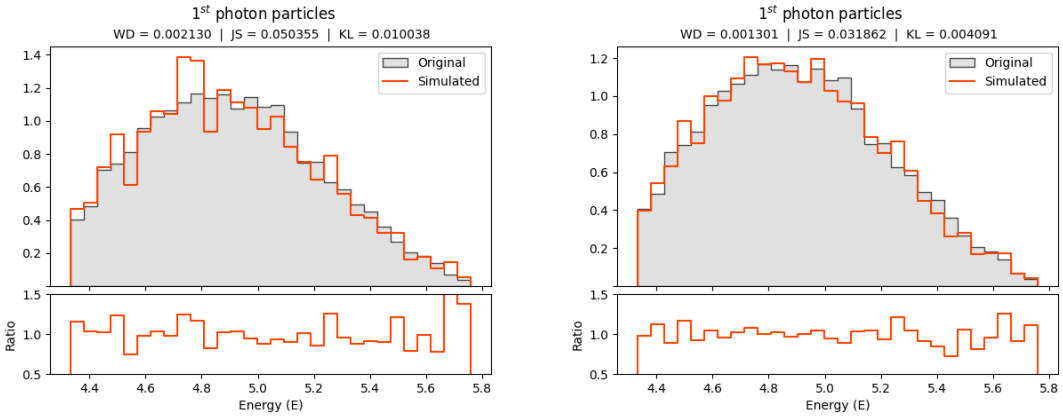
When comparing the η particle feature of the first lepton, we see a similar result to the one that we already observed in the same feature of jet particles. The distribution with an uncommon shape of the Monte Carlo events is correctly matched by the generated events, and the results improve as β increases, an effect that we can see reflected in the values of the metrics.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.27. Histograms of 1^{st} lepton ϕ particle feature with different α values in experiment I.

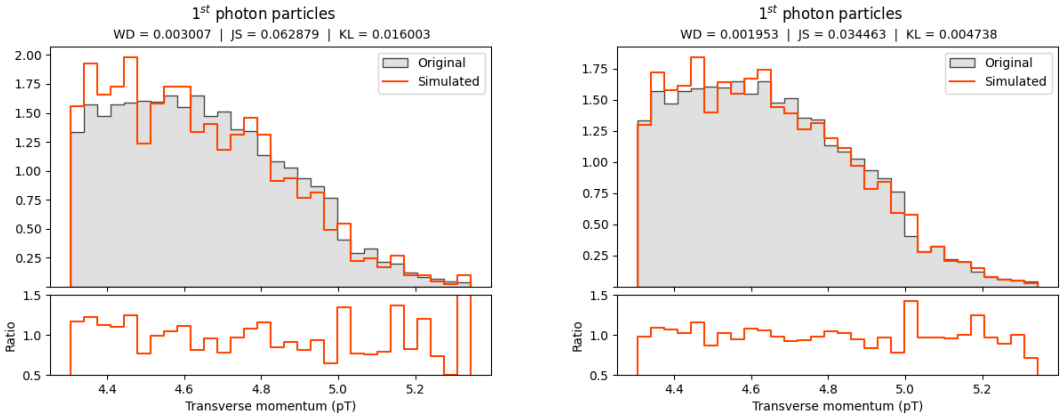
In the ϕ feature of the first lepton, we observe again a correctly defined uniform distribution that shows the biggest discrepancy when $\alpha = 0.2$, but even in that case the difference is not very remarkable as we observe in the ratio histogram. We notice that the ratio is always close to one and the effect of α does not make a big difference.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.28. Histograms of 1^{st} photon E particle feature with different α values in experiment I.

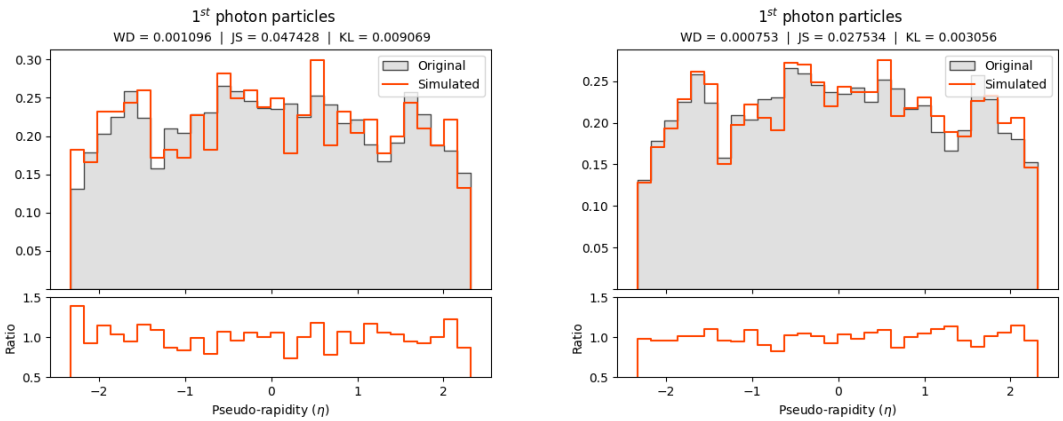
Moving on to the photon particle features, and starting with the energy (E), we see that in low α values the distribution presents a worse adjustment than when $\alpha \geq 0.3$. For those higher values, the generated data is very similar to the original distribution.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.29. Histograms of 1^{st} photon p_T particle feature with different α values in experiment I.

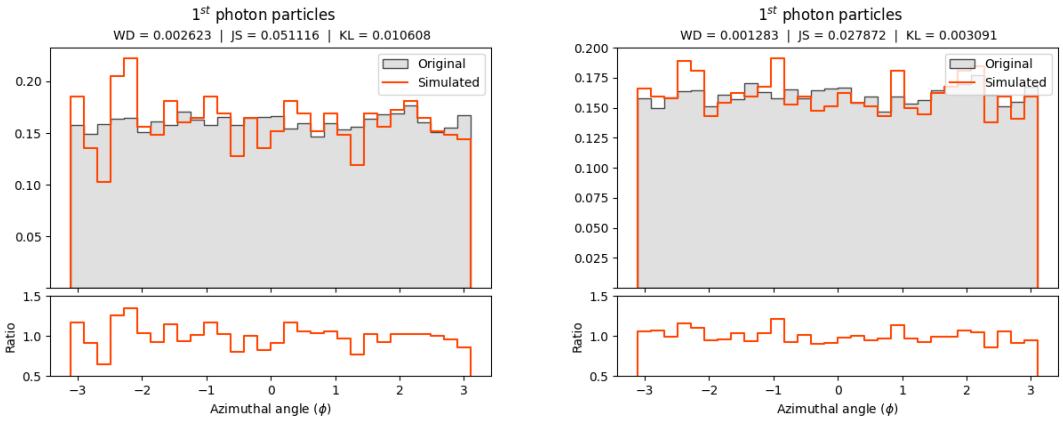
Regarding the p_T feature of the first photon, we see once again an uncommon distribution, that adds up to the fact that there are very few particles of this kind. However, the shape of the generated events resembles the original data especially when $\alpha = 0.3$.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.30. Histograms of 1^{st} photon η particle feature with different α values in experiment I.

Analyzing the η particle feature of the first photon, we notice again the small amount of existing data for this particle kind. Anyway, the shape of both the original and generated data seems to follow the same pattern.



(a) $\alpha = 0.2$ **(b)** $\alpha = 0.3$

Figure 7.31. Histograms of 1^{st} photon ϕ particle feature with different α values in experiment I.

Finally, observing the ϕ feature of the first photon, we see again a great fit for the uniform distribution, but with some more discrepancies when compared with the original Monte Carlo simulated events due to the lower amount of events with photon particles.

Overall, after analyzing several particle and event features with the models that we propose in this experiment, we can affirm that most of the shortcomings of the baseline model have improved or have been corrected by using this approach. These models keep the great adjustments to the original data that the baseline model already achieved, but also improve some aspects such as the p_T and E feature of some particles.

Looking at the generated event metrics, that we find in Table 7.2, we notice an improvement in all data distribution metrics when comparing these results to those of the baseline experiment, and also in the invalid event rate, that has been reduced 4 times from 1.68% to 0.41% in the best case. However, the generation time increases drastically due to the need of obtaining data from several models each time that an event is generated, and predicting particle data one by one instead of generating several events at once.

Taking into account the previous analysis, we conclude that the best α hyperparameter value of all the tested ones is $\alpha = 0.3$. Although this is not the best value in terms of IER, it is very close to the best value and it obtains the best numbers in all the data distribution metrics. Moreover, we observe in this experiment that the value of the α hyperparameter does not have a huge impact on the results.

α	**Data distribution metrics**			**Event generation quality**	
	WD	*JS*	*KL*	*IER*	*Gen. speed*
0.1	0.000990	0.024372	0.005365	0.45%	342.29 ms
0.2	0.001080	0.026822	0.005618	0.52%	344.69 ms
0.3	**0.000973**	**0.023365**	**0.004781**	0.46%	**339.90 ms**
0.4	0.000995	0.023622	0.004974	**0.41%**	341.35 ms

Table 7.2. Metrics of the generated events in experiment I.

Finally, we analyze the correlation between the same event features that we described during the data analysis process in Section 7.2, using the best obtained model in this experiment ($\alpha = 0.3$). In Figure 7.32, we notice that most of the correlation between particle features is maintained with a slight decrease, probably due to the fact that particle models are independent. Also, we notice that the small amount of correlation between the MET and particle features is completely lost, due to the same reason.

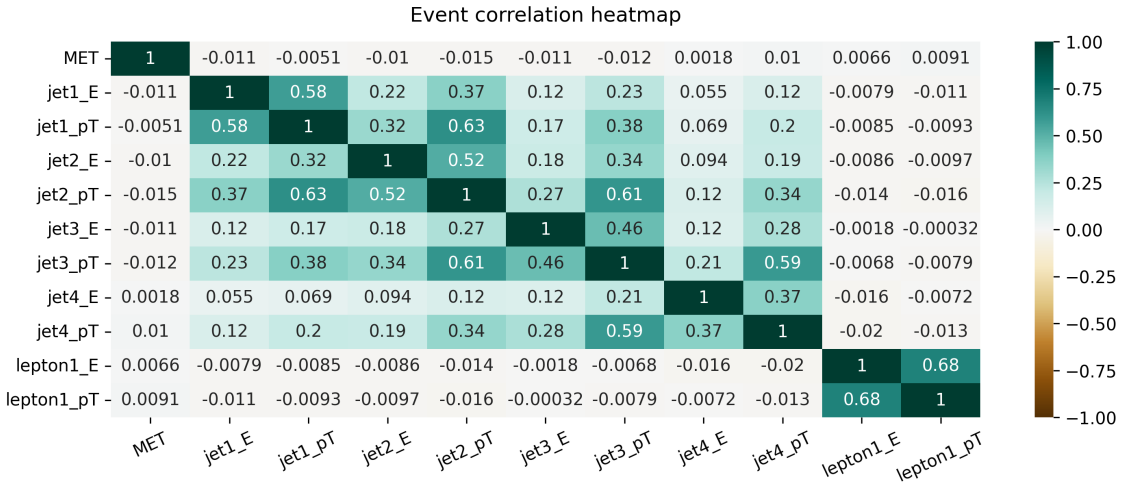


Figure 7.32. Correlation heatmap of several features on events generated in experiment I.

7.5 Experiment II: Statistical numerical mask generator

Once the first objective of obtaining a better model to generate events taking as the starting point *ground truth* events is accomplished, the following experiments try to remove the dependency of that base events replacing them by randomly sampled numbers from a normal distribution.

This next approach switches to the use of β-VAE models and also introduces a statistical component to ensure that all kinds of events are generated, trying to solve a drawback of our previous models. However, we keep the idea of training different models as we proved in the previous experiment that it yields great results.

7.5.1. Preprocessing

The preprocessing stage of this experiment is very similar to the one described in Section 7.4.1 because this experiment also requires to train different models separately. Therefore, each event *dataset* also has to be subdivided into different files. Now, we generate the following data:

1. One file that contains, in each line, `MET`, `MET`ϕ and a numerical particle mask of an event (instead of a binary one). This mask contains 7 numbers where one indicates the amount of particles of a certain type. The order of particles in the mask is the same of the rows in Table 3.1.

   Example: the mask `{2 2 0 0 0 0 1}` indicates that there are 2 *jet*, 2 *bjet* and 1 `photon` particles in the event.

2. Several files (one per particle type) that include in each line the four features of each observed particle (E, p_T, η, ϕ), without any additional information.

   This decision was taken to investigate about the independence of the position of the particle in the event, as the ordering of particles in each event does not necessarily represent the order in which the particles were generated nor observed.

This data separation produces 8 files in total: one for `MET`, `MET`ϕ and the numerical mask; and 7 more, one for each particle type. Once again, particle features were standardized by removing its mean and scaling to unit variance during this process, and re-scaled afterwards.

7.5.2. Mask generator

A negative aspect of our previous approaches that were completely based on generative models is that, due to the small amount of some rare events, the probability of generating them using this kind of models was very low, as they practically *disappeared* in the latent space distribution.

In this experiment, we will use a purely statistical mask generator that, taking into account the frequency of appearance of each possible mask in the *ground truth*, will generate new masks randomly from which events will be created. This generator (Figure 7.35) is composed of two levels: one that generates the most frequent masks (up to 85%), and another one that will generate the remaining less frequent masks.

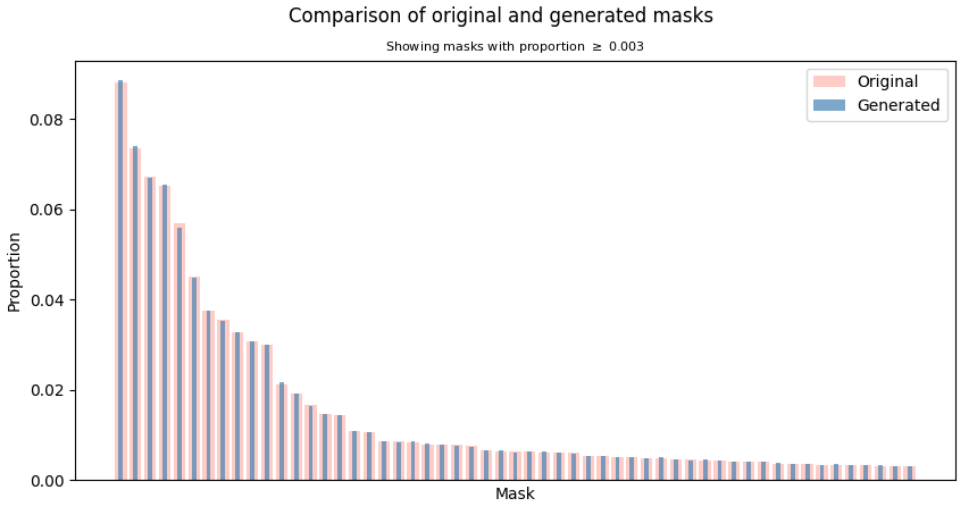


Figure 7.33. Statistically generated frequent mask proportions compared to the original *dataset*.

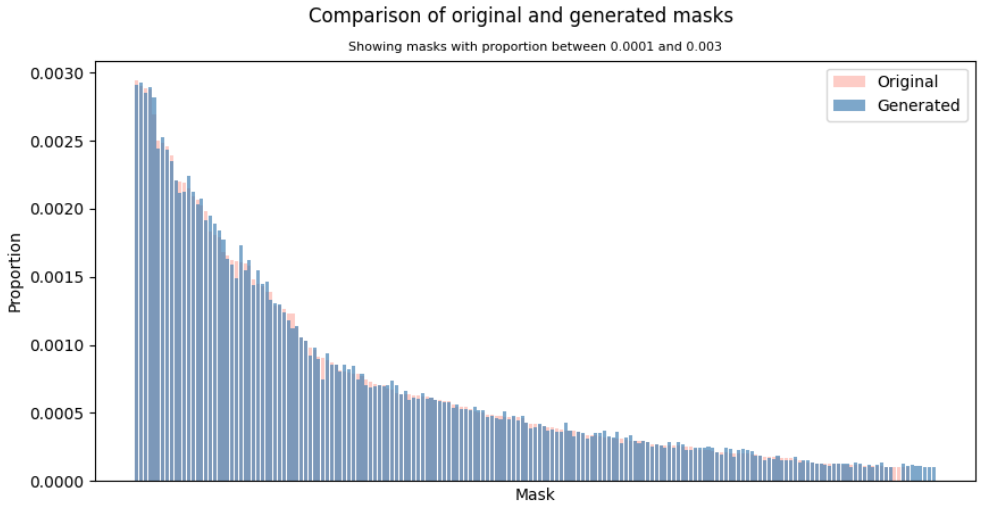


Figure 7.34. Statistically generated rare mask proportions compared to the original *dataset*.

The intuition on how this generator works can be described as follows (Figure 7.35): imagine that all possible masks are distributed in an axis that ranges from 0 to 1, and each one occupies an area in that axis that is proportional to its frequency in the *ground truth* data. The less frequent masks are represented with a unique area and not subdivided into each particular mask. Then, there is another independent axis (also from 0 to 1) where all the less frequent masks are represented, also taking an area that is proportional to its frequency inside that group. Considering that representation, to generate a new mask:

1. A random number from 0 to 1 is generated.
2. If the number is between 0 and 0.85, the mask that occupied that part of the axis with the most frequent masks is chosen as the generated mask.
3. If the number is higher than 0.85, a new random number is generated from an independent generator.
4. The mask that occupied that part of the less frequent mask axis is chosen as the generated mask.

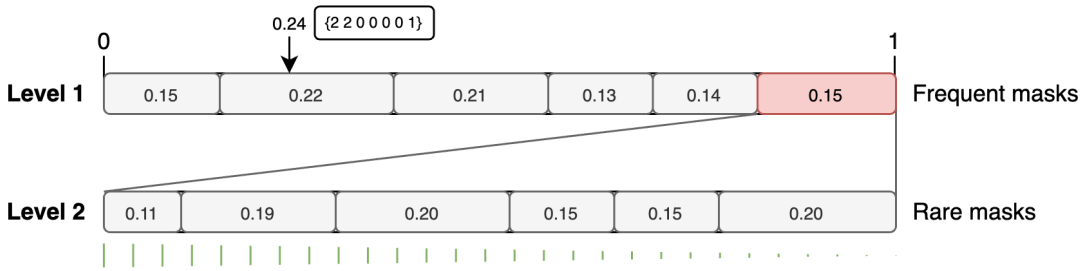


Figure 7.35. Two-level statistical mask generator with generation example.

The theoretical foundations of this procedure are known in statistics as the **alias method** [35]. Moreover, it is worth mentioning that during our experiments we noticed that the performance of generating masks was comparable to that of the VAE approaches.

7.5.3. Model architecture

As we have already stated previously, the models of in this experiment are now β-VAE based, in order to allow for a more precise generation of events from random numbers. For this experiment, a total of 8 models are trained (the same number as in the previous experiment, corresponding again to the generated files during the preprocessing stage).

Unlike in the previous experiment, now event masks are numerical instead of binary, and they are being generated by our already presented mask generator, so there is a model that is exclusively in charge of generating `MET` and `MET`ϕ.

Regarding this **MET** model (Figure 7.37), we observe that it consists of the following components:

- an **encoder**, which transforms the input data in the format described in Section 7.5.1 into its encoded representation in the latent space, producing the embedding of the provided input event as output.

  The encoder of this model features four blocks composed by a *Dense* layer of 128, 128, 64 and 64 units and a ReLU activation function, respectively. Then, those blocks are followed by a *Dense* layer of as many units as the latent dimensionality (32). Finally, we put two dense layers of as many nodes as the latent dimensionality to sample the means and logarithms of the variance, providing the data that the next component requires.

- a **variator**, that receives the outputs of the encoder and, together with a randomly generated ϵ from a normal distribution with $\mu = 0$ and $\sigma = 1$, calculates the Kullback-Leibler divergence and outputs z, calculated as follows:

$$z = (\epsilon * e^{log(\sigma^2)*0.5}) + \mu$$

  The dimensions of the z tensor match the latent dimensionality. This component is usually part of the encoder in most implementations.

- a **decoder**, that transforms the encoded data from the latent space back into its original dimensions.

  The decoder of this approach is composed by two blocks of a *Dense* layer of 64 nodes, followed by a ReLU activation layer. Finally, the output ends with another *Dense* layer of 64 units and a final one of 2 units with linear activation.

Moreover, each **particle type** model consists of the same components but with double the amount of units in each node, due to the fact that the input values are four (the particle features) instead of two (MET and METϕ). This also includes the dimensionality of the latent space.

After defining the different components, every Variational Autoencoder is trained using **the *ttbar* dataset** using a batch size of 100 events for 100 epochs. All the models use the *Mean Squared Error* for learning the different object features (including MET and METϕ).

Once training ends, the mask generator and the models are used to generate new events by decoding randomly generated numbers that are sampled from a normal distribution with zero mean and unit variance, following this procedure per event (Figure 7.36):

1. Generate a random mask using the statistical generator described in Section 7.5.2.
2. Generate a random tensor with the same size of the latent space dimensions and provide it as input to the decoder of the **MET model** to generate a new MET and METϕ pair.
3. For each particle type, if the mask predicts that k particles of that type are in the event, generate k random tensors with the same size of the latent space dimensions and provide them to that particle model to generate k particles.
4. Order the generated particles in descending order according to their p_T and insert them into the event.
5. Repeat the process from step 3 until having at most 19 particles.

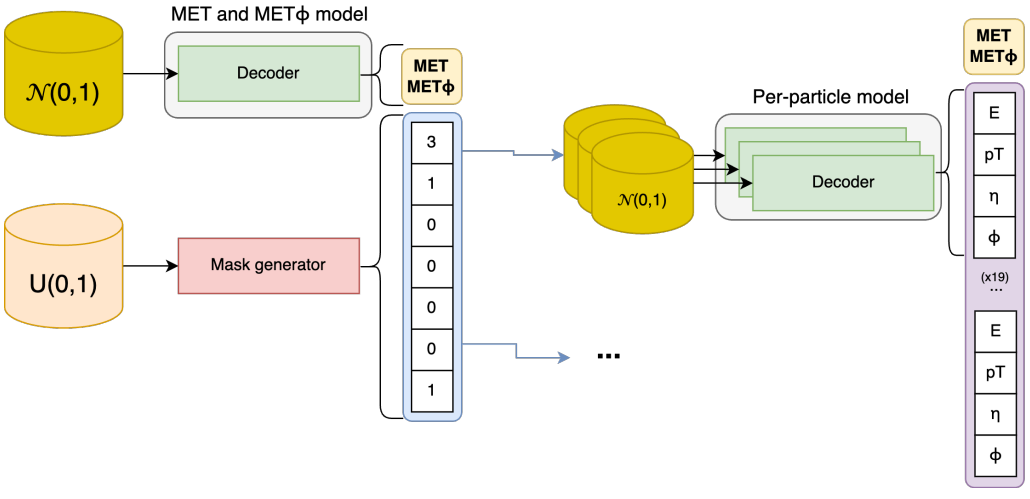
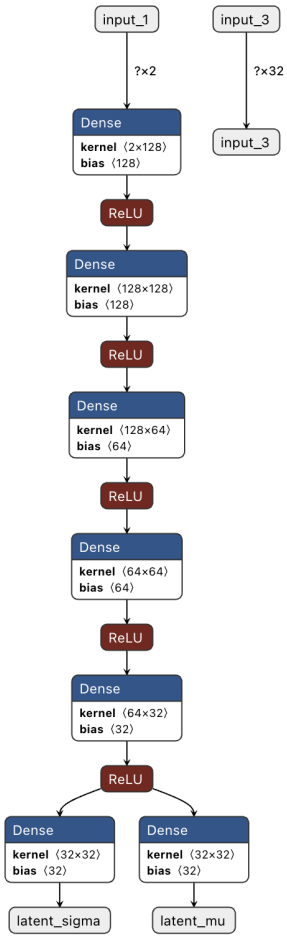


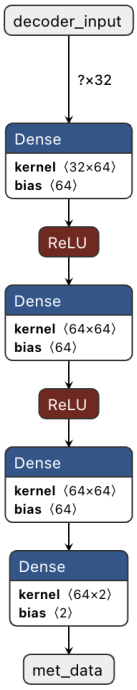
Figure 7.36. Event generation pipeline combining the use of every model in experiment II.

As we can see, this new architecture greatly simplifies the generation pipeline that we built in the previous experiment by not having to first obtain an event from the *ground truth* dataset, and then applying a perturbation to it to generate a new latent representation, that can then be decoded. Instead, we directly rely on random normal distributions.

Note that here the generation of particles of each type is completely independent of the rest of the event, so we notice that the particle models have less context about the event as a whole.



(a) Encoder

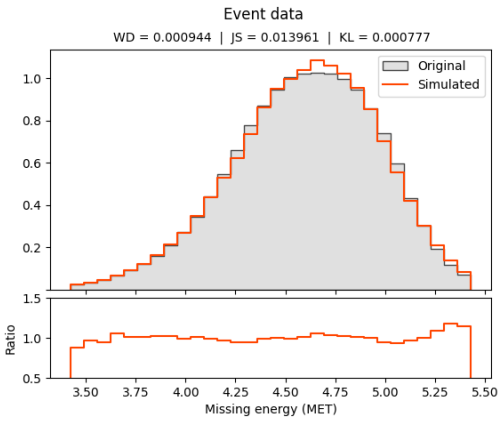


(b) Decoder

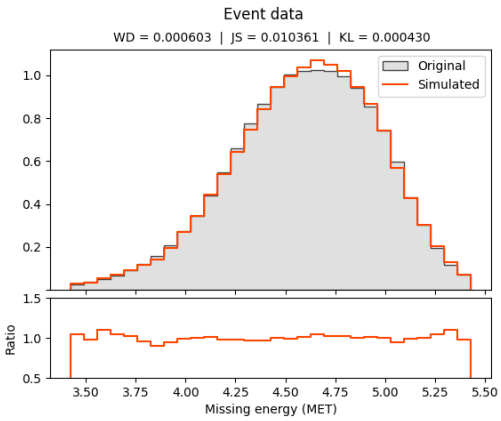
Figure 7.37. β-VAE architecture scheme for the MET model of experiment II.

7.5.4. Training results

After training the models of this new event generation pipeline with $\beta \in \{0.5, 0.7, 1.0, 1.2, 1.5\}$, and generating 150K events from random normal distributions using every model trained with each β value, we obtained interesting results that we describe with the next figures.



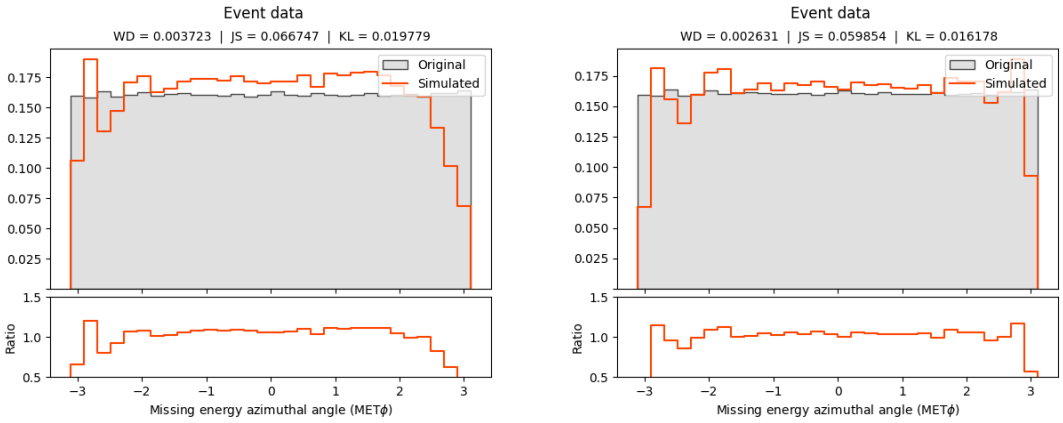
(a) $\beta = 0.7$



(b) $\beta = 1.5$

Figure 7.38. Histograms of `MET` event feature with different β values in experiment II.

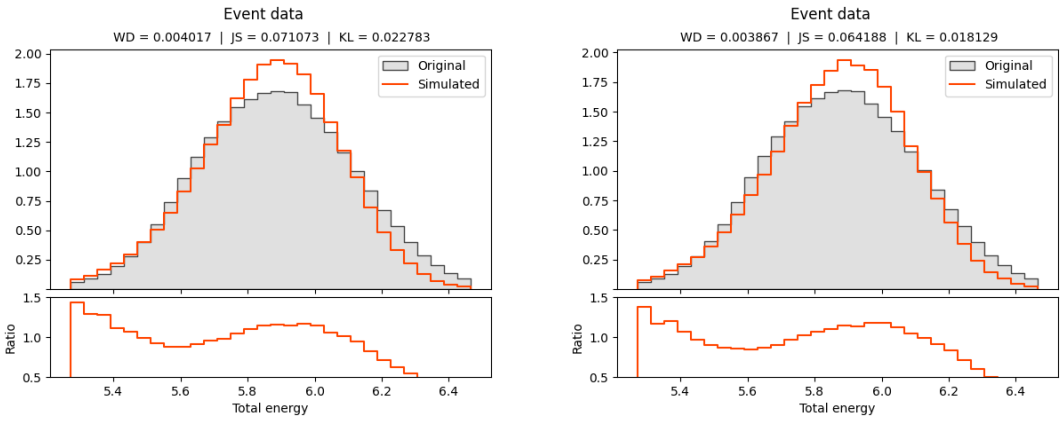
Firstly, in the MET event feature we observe a great adjustment of the distribution of the generated data. The results improve as β increases, fixing the higher ratios in the right tail of the distribution that were observed with lower β values. This is also reflected in the values of the metrics.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.39. Histograms of METϕ event feature with different β values in experiment II.

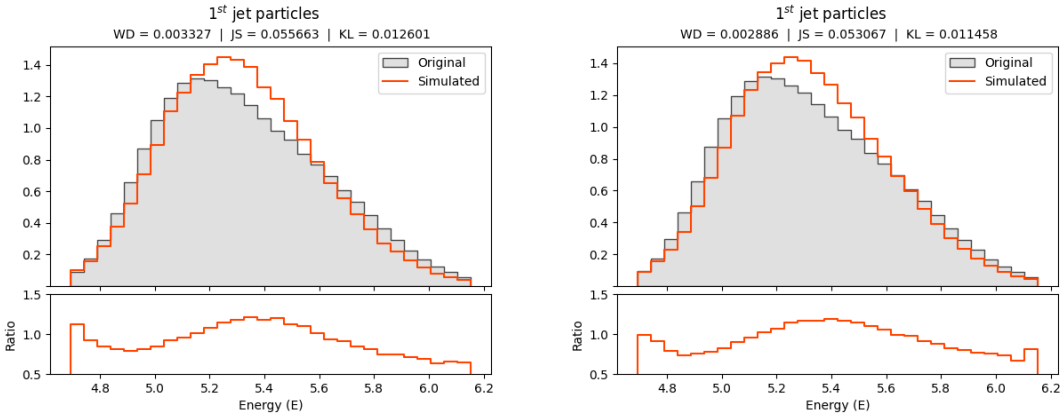
Looking at the METϕ feature of the events, we see that apparently the distribution is not correctly learnt by the model. However, looking at the scale of the histograms and analyzing the ratio evolution, we see that all values are close to 1.0, especially with high values of β, that show a great improvement as we can see in the metrics.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.40. Histograms of total energy event feature with different β values in experiment II.

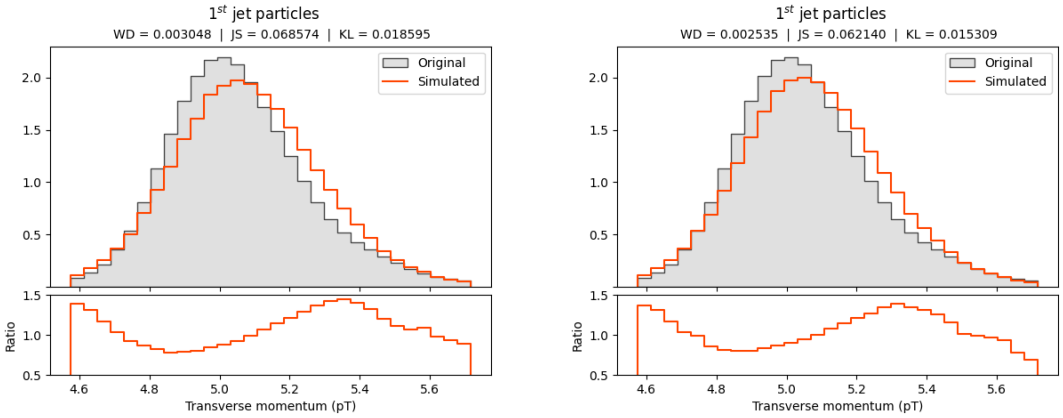
Regarding the total energy of all particles in the event, we observe a slight difference in the central values of the distribution, of which we find more occurrences in the simulated data than in the original events. The opposite happens in the right tail, where there are more occurrences of those values in the *ground truth*.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.41. Histograms of 1^{st} jet E particle feature with different β values in experiment II.

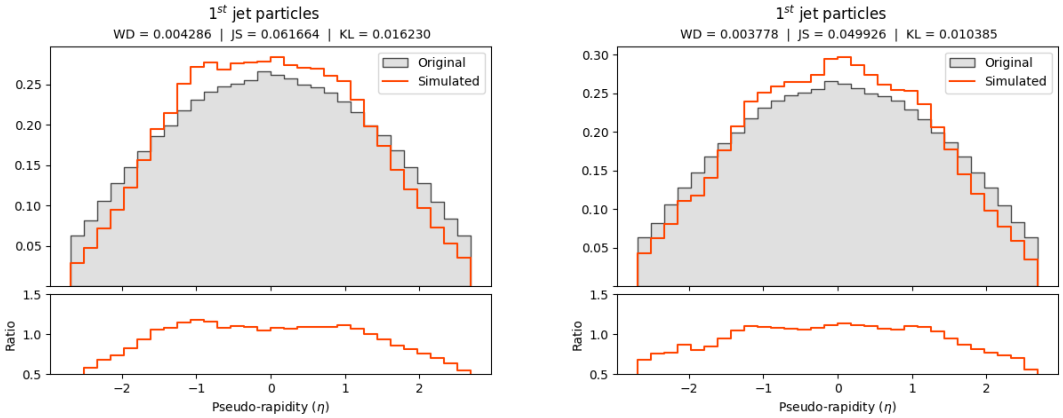
Analyzing the energy particle feature of the first jet, we find a similar effect than what happened with the total energy. However, in this case the ratio between both values is much lower as we can observe in the ratio histograms, making this result a great fit.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.42. Histograms of 1^{st} jet p_T particle feature with different β values in experiment II.

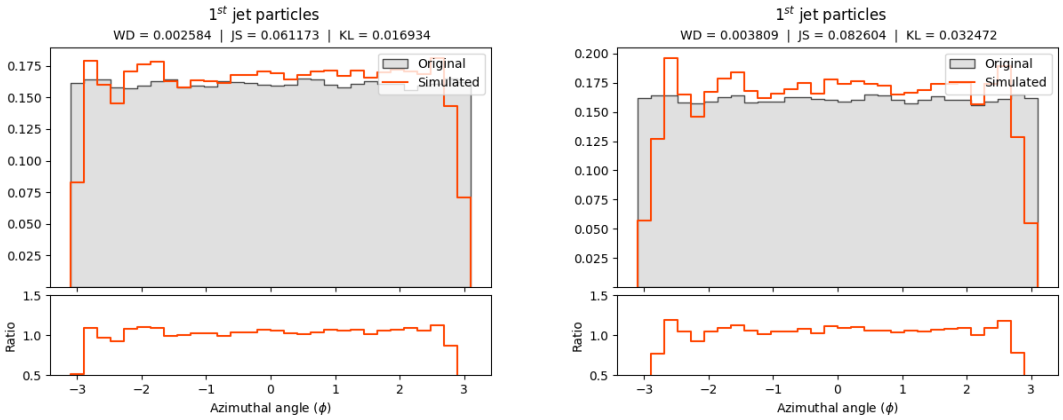
The p_T feature of the first jet shows some values of the central part of the distribution that seem to be displaced to the right tail. Moreover, we see in the ratio histograms that it is close to 1.5 in some cases, an aspect that could improve.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.43. Histograms of 1^{st} jet η particle feature with different β values in experiment II.

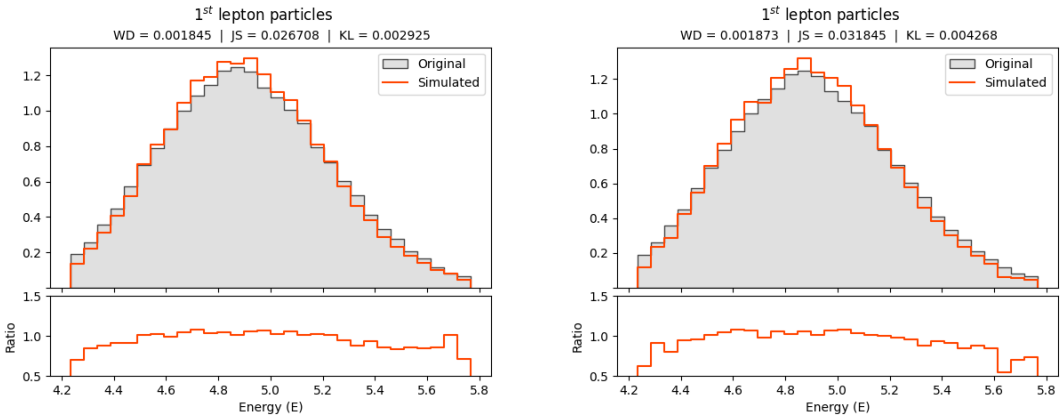
Observing the η particle feature of the first jet, and taking into account the uncommon shape of the distribution of this feature, we appreciate a great fit where the highest discrepancies are found in the left and rightmost values of the distribution.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.44. Histograms of 1^{st} jet ϕ particle feature with different β values in experiment II.

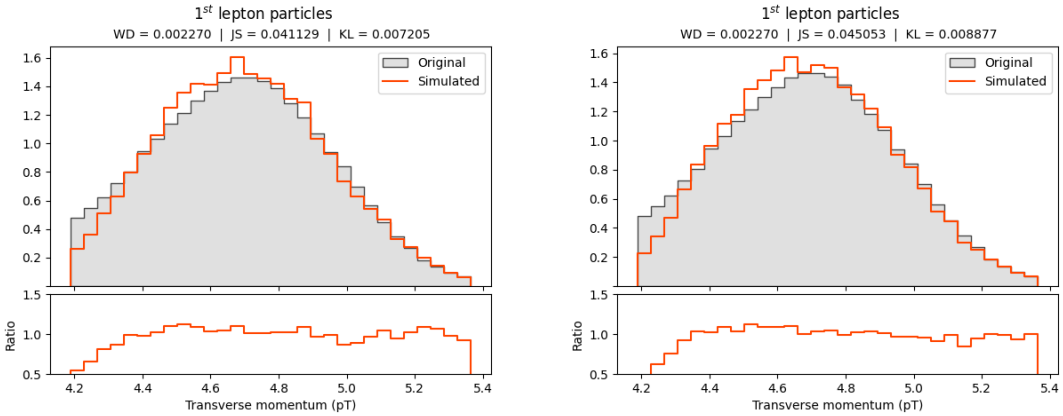
In the ϕ particle feature of the first jet we find a similar effect than what we saw in the `MET`ϕ case, with a similar difference between both distributions. Anyway, these results can be considered a great fit in both cases. In this case we can see that high values of β provide slightly worse results, as we can see in the data distribution metrics.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.45. Histograms of 1^{st} lepton `E` particle feature with different β values in experiment II.

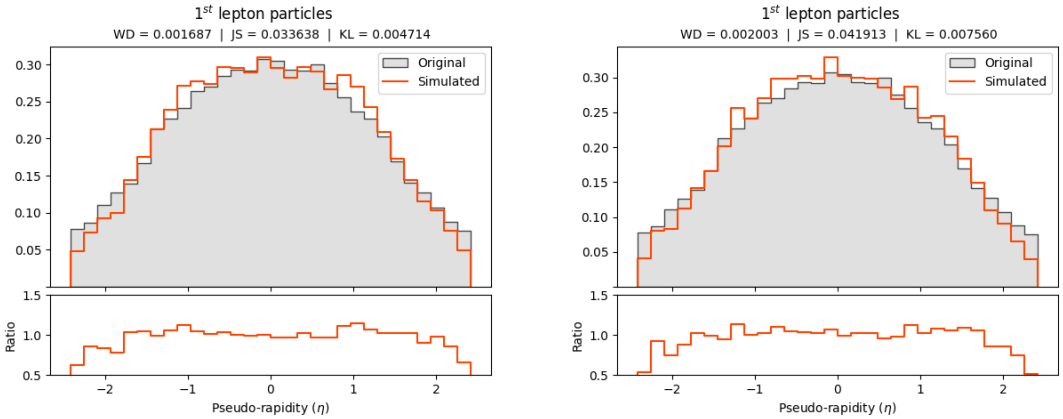
Moving on to the first lepton, we see that its E particle feature has been modelled in a great way, showing a much better adjustment than the same feature of the first jet. In this case, we see as an exception that the best β value is the lowest one due to the differences in the ratio histogram for the right tail values of the distribution with higher values of β.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.46. Histograms of 1^{st} lepton p_T particle feature with different β values in experiment II.

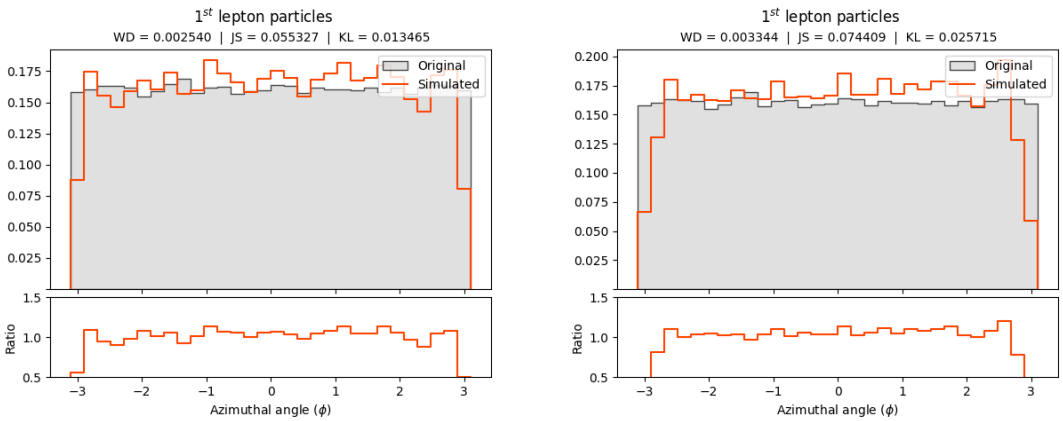
Regarding the p_T particle feature of the first lepton, we find the biggest differences in the left tail of the distribution, where the ratio falls below 0.5. However, it quickly grows and stabilizes, making this also a great adjustment that improves as β increases.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.47. Histograms of 1^{st} lepton η particle feature with different β values in experiment II.

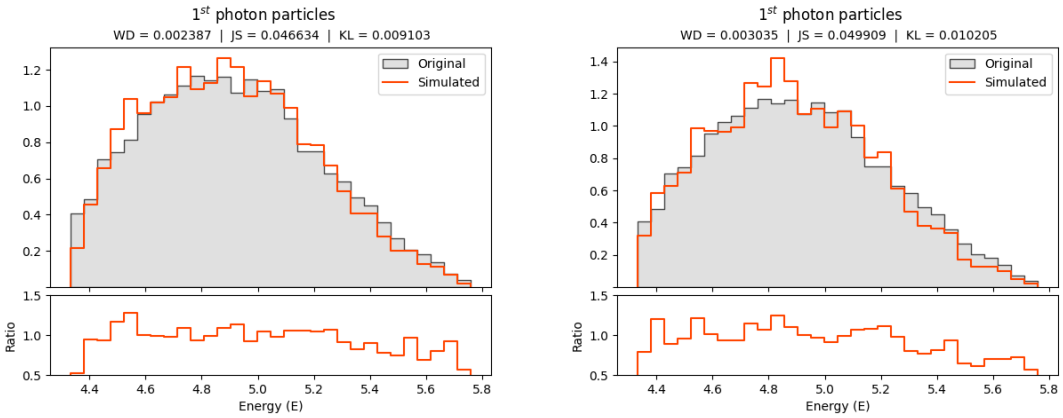
Analyzing the η feature of the first lepton, we see that both the distribution of the original *ground truth* data and the generated events are very similar, so the model learns this feature in an accurate way.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.48. Histograms of 1^{st} lepton ϕ particle feature with different β values in experiment II.

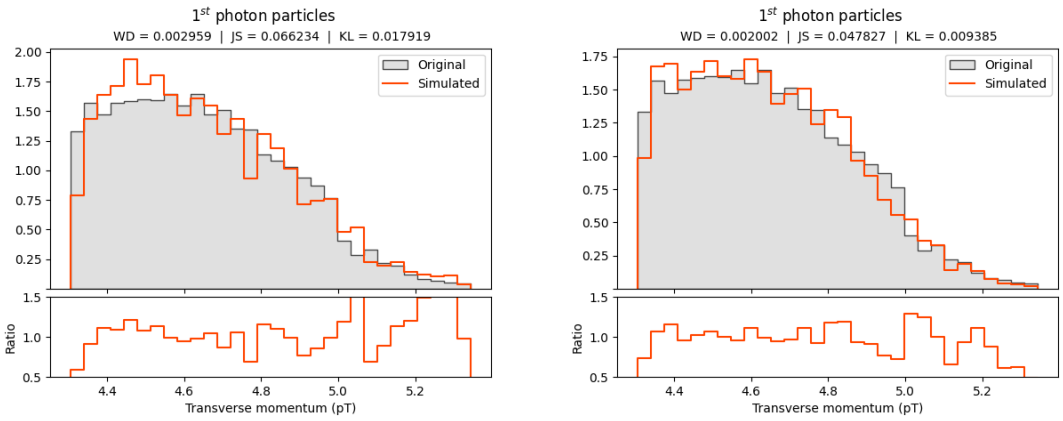
The ϕ particle feature of the first lepton is again very similar to the original data, with the ratio histograms showing a high resemblance between both distributions. We observe the highest differences in the extreme values of the uniform distribution.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.49. Histograms of 1^{st} photon `E` particle feature with different β values in experiment II.

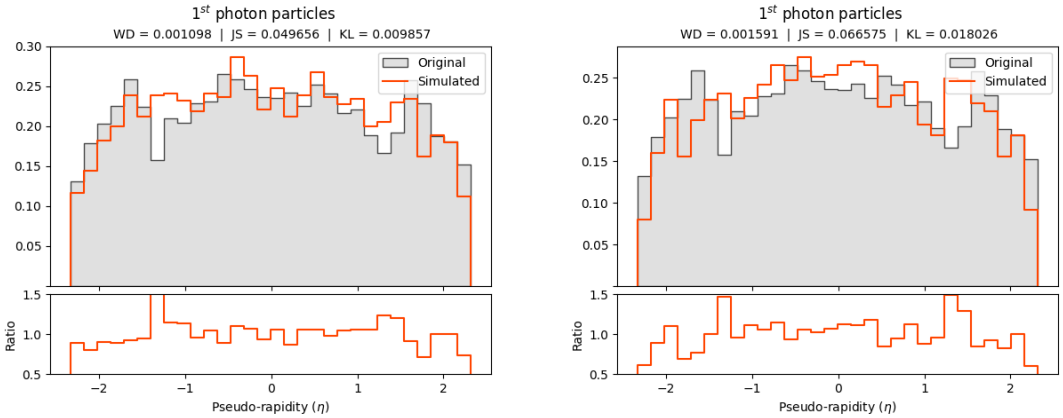
Now considering the first photon, we see that its E particle feature has been modelled with a similar shape than the original data, but not with a perfect fit. That can be due to the original distribution following an uncommon shape and also due to the low amount of data.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.50. Histograms of 1^{st} photon p_T particle feature with different β values in experiment II.

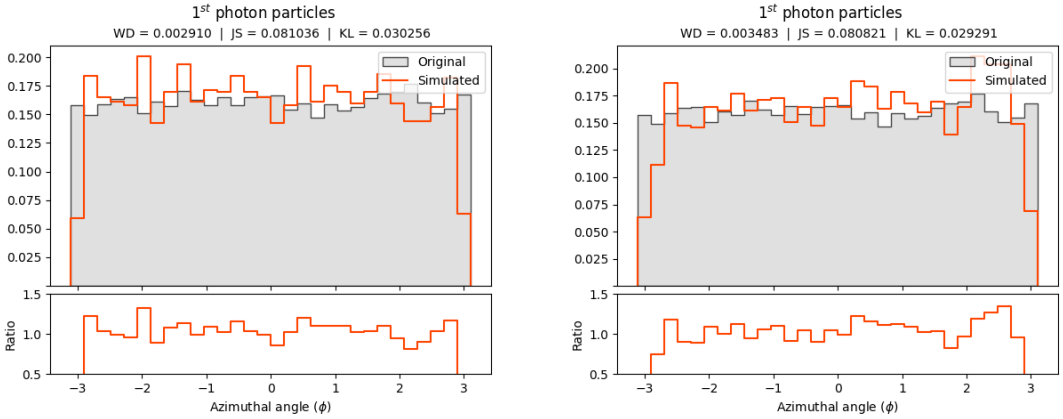
Observing the p_T particle feature of the first photon we see that, even though there is not a high amount of information, the model generates events that resemble the shape of the original distribution, showing also a great adjustment to the original *ground truth* data.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.51. Histograms of 1^{st} photon η particle feature with different β values in experiment II.

Regarding the η feature of the first photon, we find that the generated events follow a distribution that can be considered of similar shape than the one of the original data, considering the lack of sufficient data for a more accurate generation. We see that the ratio histogram oscillates but is never lower than 0.5 or higher than 1.5.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.52. Histograms of 1^{st} photon ϕ particle feature with different β values in experiment II.

To conclude, in the ϕ particle feature of the first photon we observe that the generated data follows a similar distribution than the Monte Carlo generated events of the *ground truth* data, with a ratio that stays within a reasonable interval.

After observing the particle distributions and comparing the values of the metrics that we see in Table 7.3, we confirm that these results are worse than those obtained by the previous models in terms of distribution similarity and invalid event rate. However, this was expected due to the nature of both experiments: the previous models had a solid starting point in the generation process while this experiment features event generation from random values.

In terms of generation speed, we obtain a decrease of more than 4 times with respect to the previous experiment, that is due to this pipeline not requiring to pass the original data to the encoder and variator to get an encoding in each model, and instead generating it directly and passing it into the decoder. Moreover, the mask generator is no longer a VAE model, which also makes generation a bit faster.

The effect of the β hyperparameter has a great impact on the final results in this case, improving most of the data distribution metrics as its value increases and reducing the invalid event rate in the best case by half (when compared to the worst case of this experiment).

All things considered, we can conclude that the β hyperparameter value that obtains better results is $\beta = 1.5$, that achieves the best results in 2 out of 3 distribution metrics, and the second best IER of all the tested values.

β	**Data distribution metrics**			**Event generation quality**	
	WD	*JS*	*KL*	*IER*	*Gen. speed*
0.5	0.002969	0.058086	0.016235	9.17%	75.46 ms
0.7	0.002928	0.056739	**0.015629**	7.54%	75.26 ms
1.0	0.002800	0.056514	0.015833	8.16%	**74.43 ms**
1.2	0.002785	0.056112	0.016177	**4.48%**	75.81 ms
1.5	**0.002771**	**0.055761**	0.016530	4.55%	74.64 ms

Table 7.3. Metrics of the generated events in experiment II.

Finally, regarding the analysis of the correlation between some event features using the best obtained model in this experiment ($\beta = 1.5$), that we can observe in Figure 7.53, we obtain here a similar result than what was observed in the previous experiment: most of the correlation between particle features is maintained with a slight decrease, and there is no correlation between the `MET` and particle features.

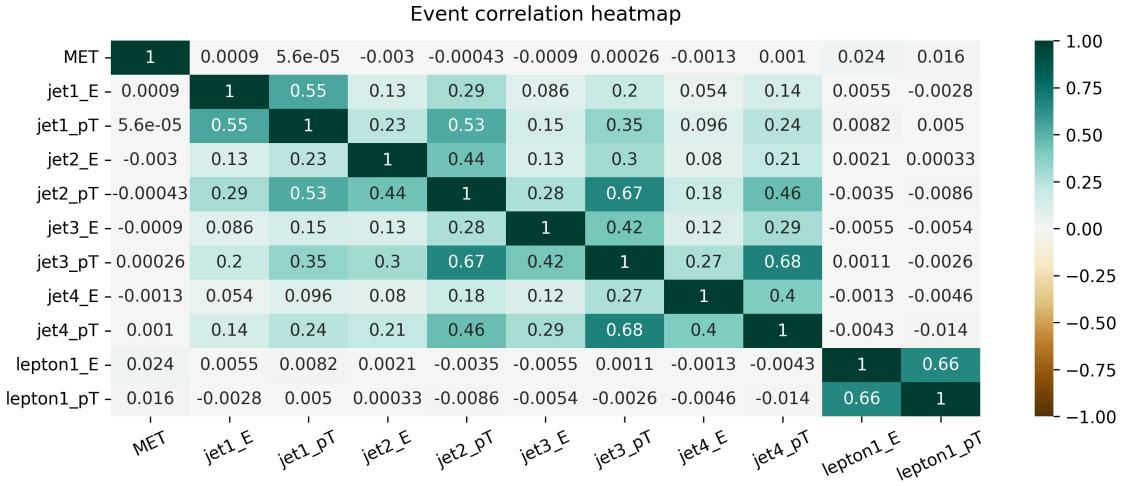


Figure 7.53. Correlation heatmap of several features on events generated in experiment II.

In conclusion, after observing many event and particle features, we see that our first model that generates events from completely random values drawn from a normal distribution obtains some results that, although not exactly equal to the original data, offer a good enough starting point that can be improved in further experiments.

7.6 Experiment III: Single model with reduced dataset

In the previous experiments, that featured independent models for each type of particle, we noticed that most of the correlation between the MET data and particle features, and some of the correlation between particles were lost. That happened due to the separation between models and the generation of each particle with little context about the rest of the already generated event.

However, as we already observed in [3], trying to generate events from random numbers using the whole *ttbar* dataset did not yield good results due to the sparsity of the data (i.e., most of the events did not have 19 particles, so there was a large amount of particle data set to zero).

Because of that, in order to tackle those weaknesses of previous approaches, we decided to design an experiment that involved using a single model but with a subset of the total data. This model would help to maintain the original data correlation, as the model would have more context when generating each particle; and also to deal with data with less sparsity, as the event characteristics would be bounded.

7.6.1. Preprocessing

During the investigation related to the mask generator that was used in the previous experiment (described in Section 7.5.2), we had the opportunity to look at the statistics of each possible particle mask that was present in the *ttbar* dataset. Then, we found out that 45.7% of the events consisted of only 4 or less jets and 2 or less bjets. Therefore, we decided to design a model with those constraints, that could generate almost half of the possibilities of the whole dataset.

Then, the preprocessing of this experiment only consisted of filtering the original dataset to take the specified events, and scaling the data to zero mean and unit variance.

7.6.2. Model architecture

In this experiment, we use again only one β-VAE model, that receives the MET, METϕ, and all particles with its features as separate inputs. Moreover, it also receives a binary mask that indicates the amount of particles in each particular event.

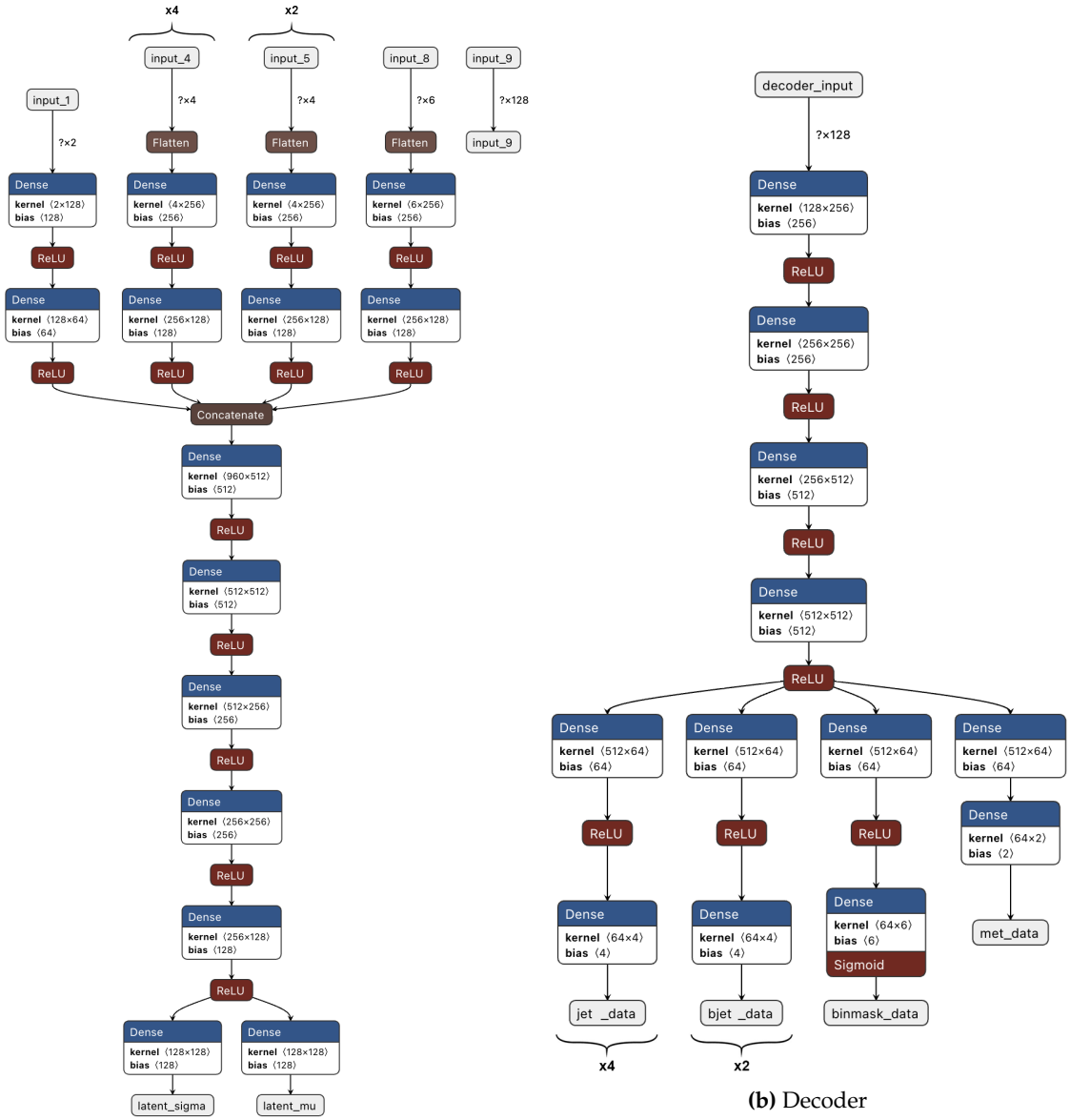
First, the Variational Autoencoder is trained with **the *ttbar* dataset**. This VAE has the same three sub-components that we have been using so far. The implementation for this particular experiment is structured as follows (Figure 7.54):

- an **encoder**, which transforms the input data in the format described previously into an encoded representation in the latent space, producing an embedding of each event. The encoder of the model for this last experiment features several inputs:

    1. an input for the MET and METϕ values of the event, composed by two blocks of a *Dense* layer of 128 and 64 units, respectively, followed by a ReLU activation.
    2. four inputs for up to four jet particles, that consist of a *Dense* layer of 256 units + a ReLU activation, and another block with the same features but with 128 units in the *Dense* layer.
    3. two inputs for up to two bjet particles, with the same features than the jet inputs.
    4. a binary mask of 6 values that indicates which jets and bjets are present in each particular event.

Then, all the inputs are concatenated and followed by five blocks of *Dense* layer + ReLU activation with 512, 512, 256, 256 and 128 units respectively. Finally, we put two *Dense* layers of as many nodes as the latent dimensionality (128) to sample the means and logarithms of the variance, providing the data that the variator requires.

- a **variator**, that receives the outputs of the encoder and, together with a randomly generated ϵ from a normal distribution, calculates the Kullback-Leibler divergence and outputs z, whose dimensions match the latent dimensionality, calculated as it was explained in the model of the previous experiment.



(a) Encoder

(b) Decoder

Figure 7.54. β-VAE architecture scheme for the model of experiment III.

- a **decoder**, that transforms the encoded data from the latent space back into its original dimensions. The decoder of the model for this experiment is composed by four blocks of *Dense* layer + ReLU activation with 256, 256, 512 and 512 units respectively. Then, the output is divided into:

  1. an output for the MET and METϕ values of the event, composed by a *Dense* layer of 64 units, followed by another *Dense* layer with linear activation of 2 units.
  2. four outputs for up to four jet particles, that consist of a *Dense* layer of 64 units + a ReLU activation, followed by another *Dense* layer with linear activation of 4 units.
  3. two outputs for up to two bjet particles, with the same features than the jet outputs.
  4. an output for the binary mask, composed by a *Dense* layer of 64 units, followed by another *Dense* layer with *Sigmoid* activation of 2 units.

Our VAE used the *Mean Squared Error* for learning the different object features (including MET and METϕ), and *Binary Crossentropy* as the loss function for the binary mask. Moreover, we use a variable learning rate strategy in the training process: starting from $1 * 10^{-4}$, a small reduction of 0.1 is applied if the global training loss does not improve in 3 consecutive epochs.

Once the β-VAE is trained using a batch size of 100 events for 100 epochs, random numbers from a Gaussian distribution with zero mean and unit variance are generated and provided as input to the decoder of the model to generate new events, as it is shown in Figure 7.55.

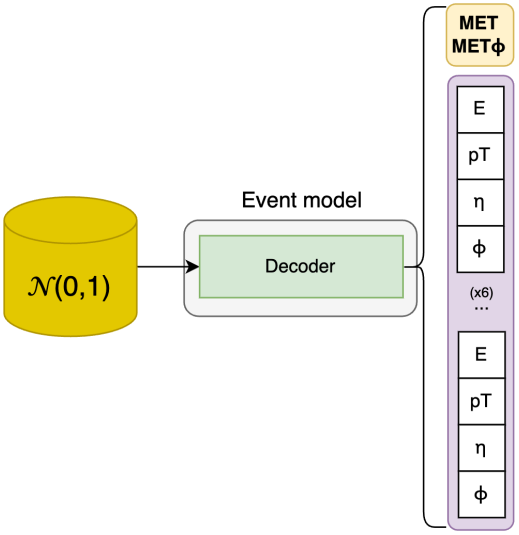
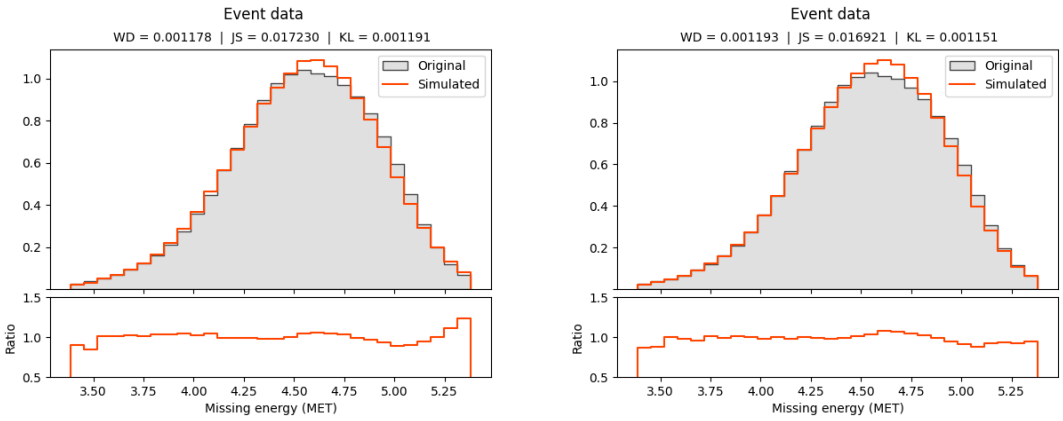


Figure 7.55. Event generation pipeline using the event model of experiment III.

7.6.3. Training results

After training the model of this experiment with $\beta \in \{0.5, 0.7, 1.0, 1.2, 1.5, 2.0\}$, and generating 300K events from random normal distributions using the model trained with each β value, we obtained the results that follow hereafter.

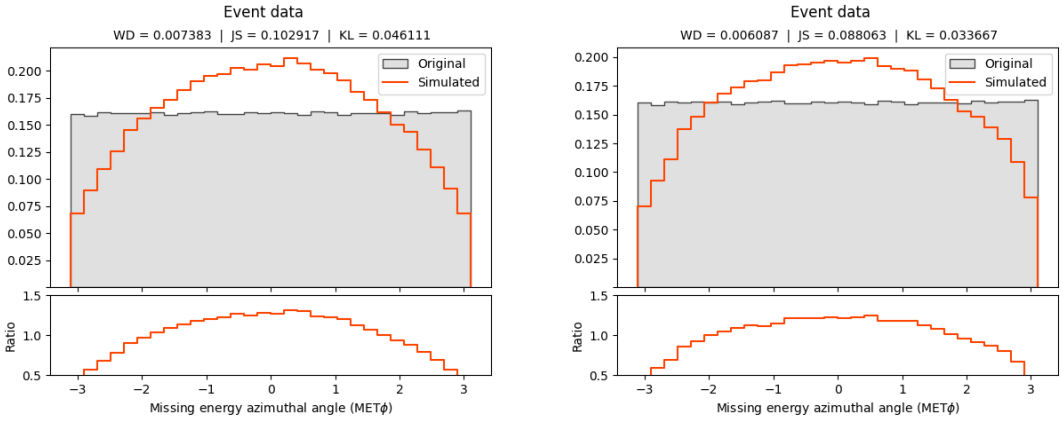
It must be noted that this model is the most complex one of all our experimentation process, as it has to generate an event all at once without any information to guide it as a starting point: only randomly generated numbers from a normal distribution.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.56. Histograms of `MET` event feature with different β values in experiment III.

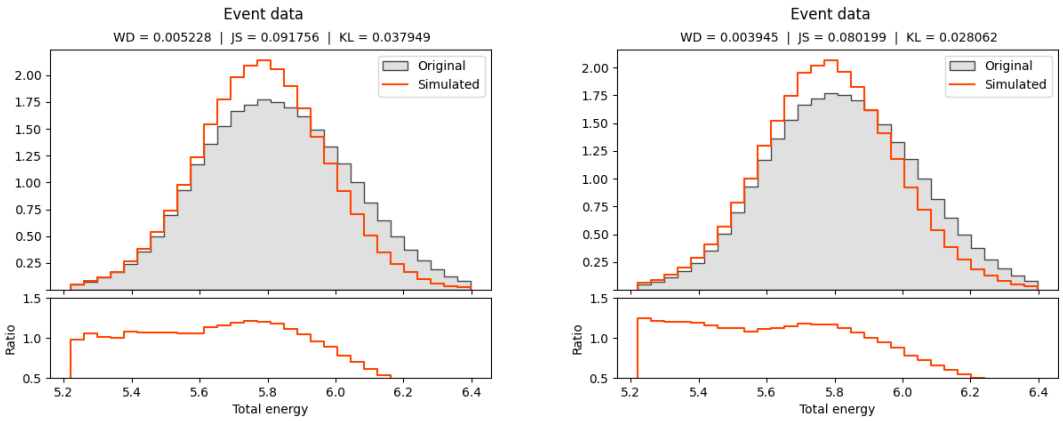
First of all, observing the `MET` event feature we can see similar distributions that have its biggest differences in the right tail of the distribution with low values of β. However, those differences are reduced when $\beta = 1.5$.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.57. Histograms of `METϕ` event feature with different β values in experiment III.

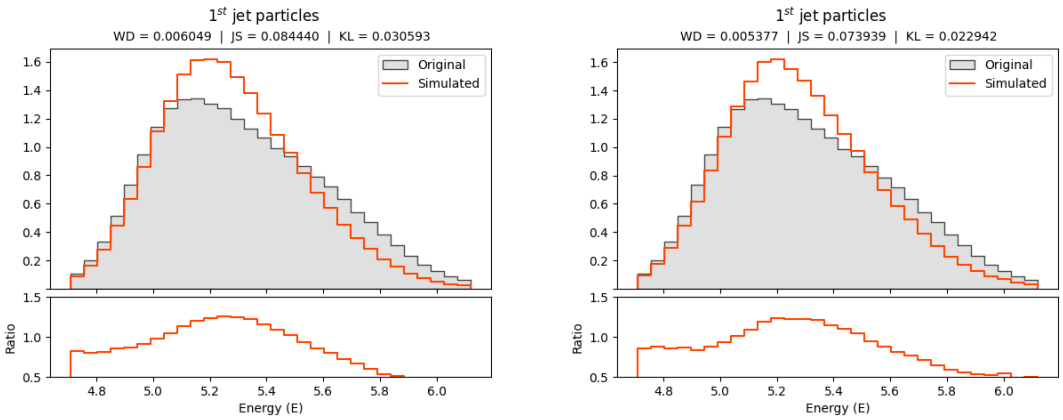
Regarding the `METϕ` feature of the events, we see that the model has not achieved a great understanding of its distribution, as it seems that this result is more similar to a normal distribution. Anyway, the ratio values are inside the 0.5 to 1.5 range.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.58. Histograms of total energy event feature with different β values in experiment III.

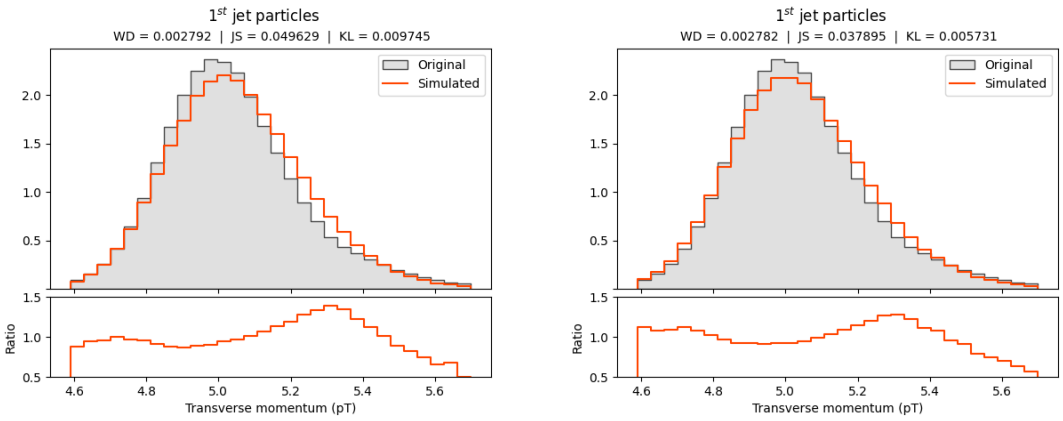
Looking at the total energy of each event, we observe a similar effect than what we saw in previous experiments: some values that belong to the right tail of the *ground truth* distribution are located in the central part of the generated distribution. We could not deduce the cause of this effect. The result in this case improves as β increases, as we can see in the small values of the metrics.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.59. Histograms of 1^{st} jet `E` particle feature with different β values in experiment III.

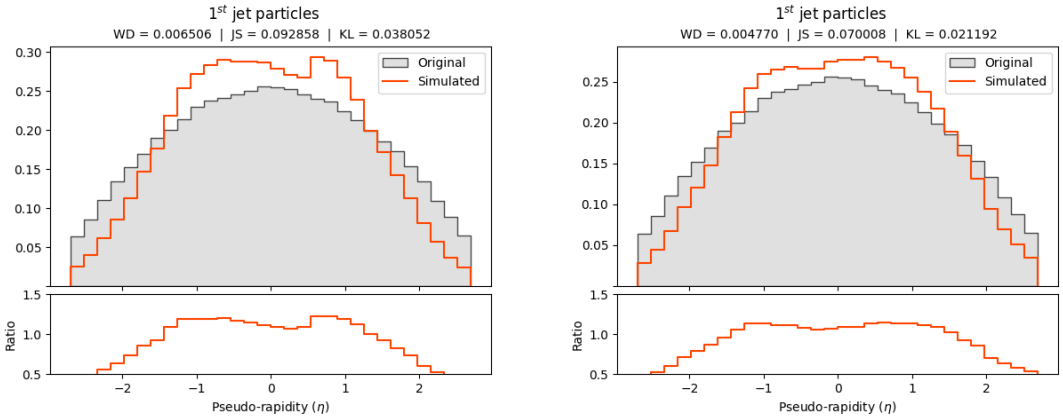
Moving on to particle features, and starting with the energy of the first jet, we can find that the ratio histograms show a great match in the left tail of the distribution. Then, in central values the ratio rises a bit, but the biggest difference is found in the right tail, where we find the biggest discrepancies when comparing the generated events with the original Monte Carlo data.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.60. Histograms of 1^{st} jet p_T particle feature with different β values in experiment III.

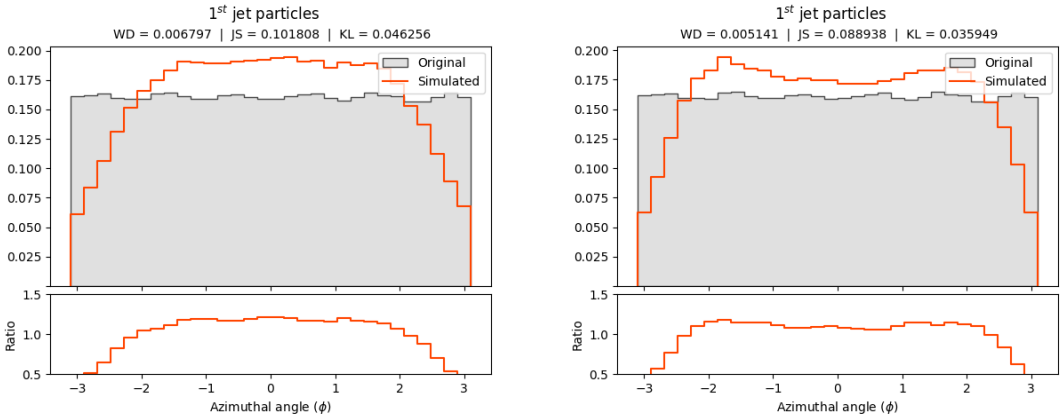
Observing the p_T feature of the first jet, we can see that in general the distribution is well adjusted with only some higher differences in the rightmost part of the generated data, where we see the highest ratio overall. The differences between both distributions decrease as the value of β increases, as we observe in the values of the metrics.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.61. Histograms of 1^{st} jet η particle feature with different β values in experiment III.

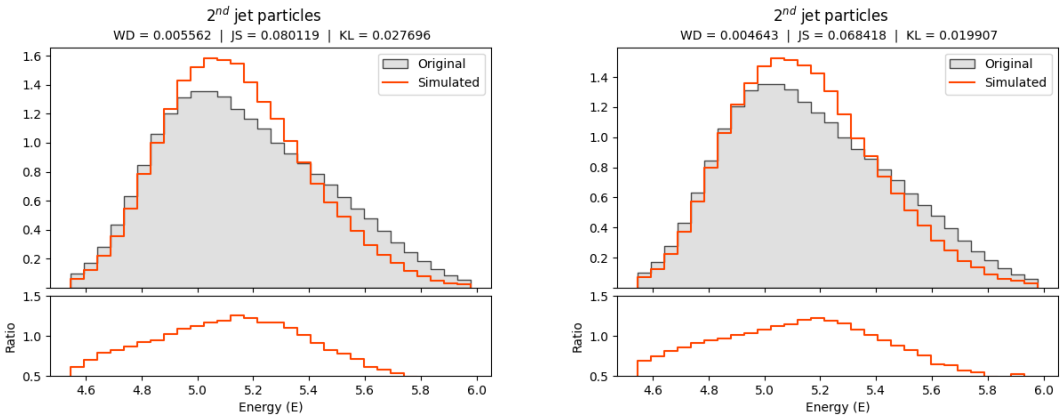
Looking at the η particle feature values of the first jet, it is clear that with low values of β the model seems to learn something similar to two normal distributions together. However, as β increases, the distribution looks more similar to the original data.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.62. Histograms of 1^{st} jet ϕ particle feature with different β values in experiment III.

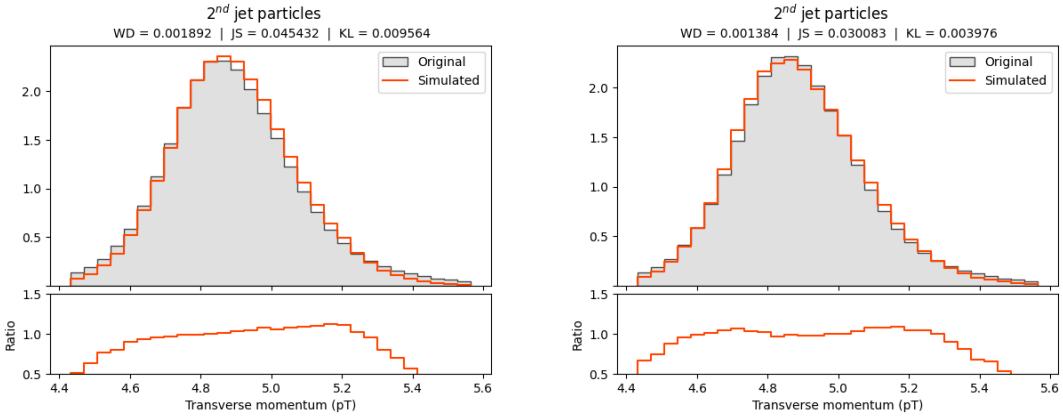
Analyzing the values of the ϕ particle feature of the first jet, we see once again that the distribution is well adjusted except in the lowest and highest values, where we find a ratio of almost 0.5. However, most of the values present a ratio around 1.0.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.63. Histograms of 2^{nd} jet E particle feature with different β values in experiment III.

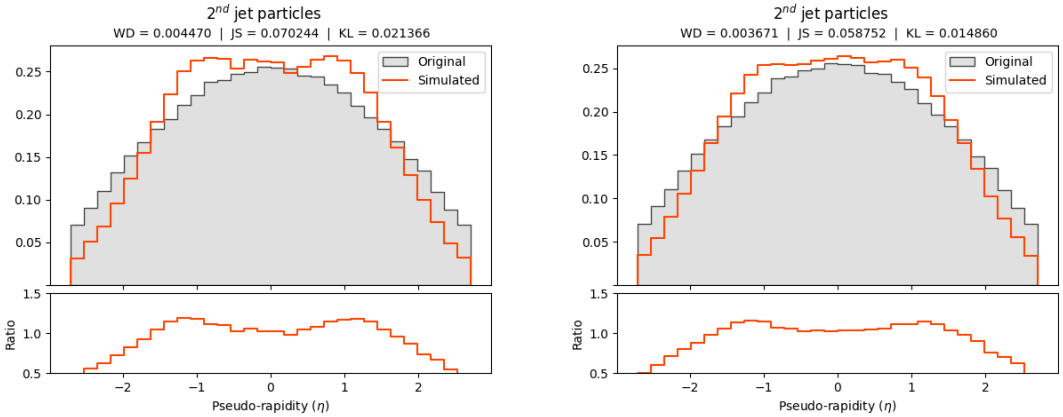
Regarding the E feature of the second jet (as there are no leptons in this subset of the data), we find a similar effect than what happened in the first jet, with some values in the rightmost part of the distribution appearing to have been shifted to the central part. This effect is reduced slightly when $\beta > 1.0$, as we can distinguish by comparing the values of the metrics.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.64. Histograms of 2^{nd} jet p_T particle feature with different β values in experiment III.

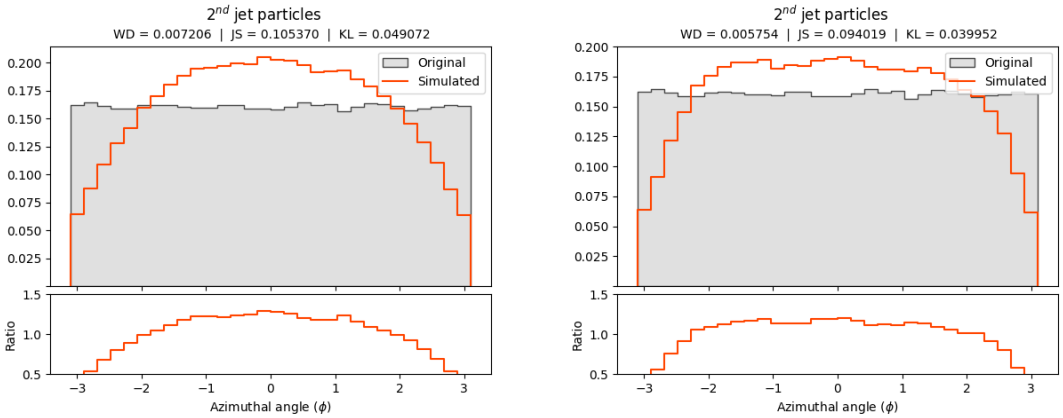
Observing the p_T particle feature of the second jet, we find that there is an almost perfect match between the original data and the generated events distribution. While there are some minor differences when $\beta = 0.7$, those seem to be mostly corrected as this hyperparameter increases.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.65. Histograms of 2^{nd} jet η particle feature with different β values in experiment III.

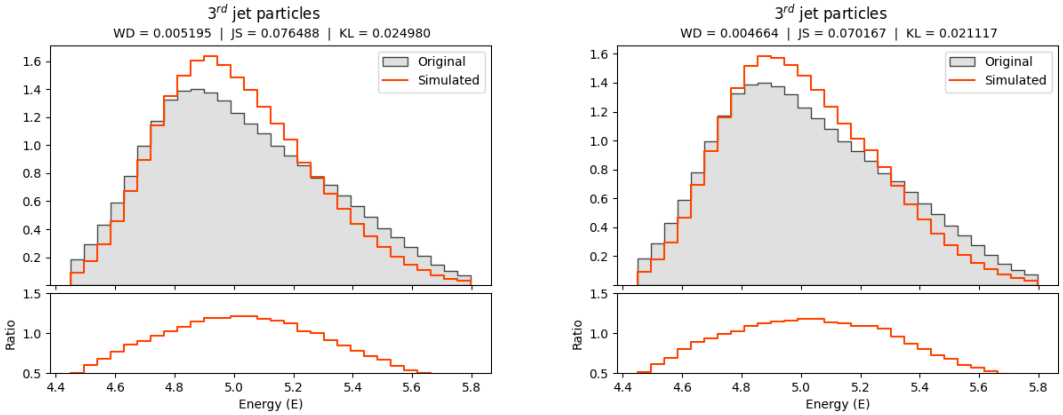
The η particle feature of the second jet presents a similar effect than what happened with the same feature of the first jet but in this case the adjustment to the original data seems more precise and the effect of two normal distributions together almost disappears when β is in the highest tested value.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.66. Histograms of 2^{nd} jet ϕ particle feature with different β values in experiment III.

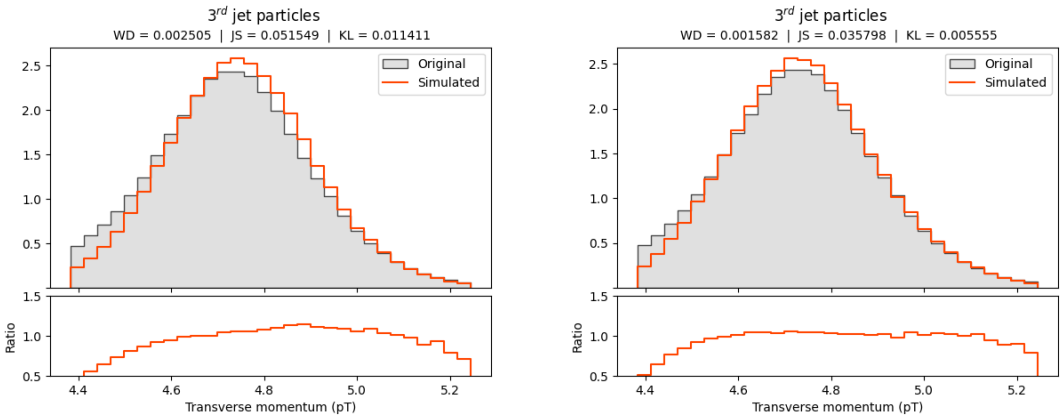
Looking at the ϕ feature of the second jet, we observe that this particle feature has been the most difficult to approximate with the model of this experiment so far, altough the ratio is around 1.0 in most values.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.67. Histograms of 3^{rd} jet E particle feature with different β values in experiment III.

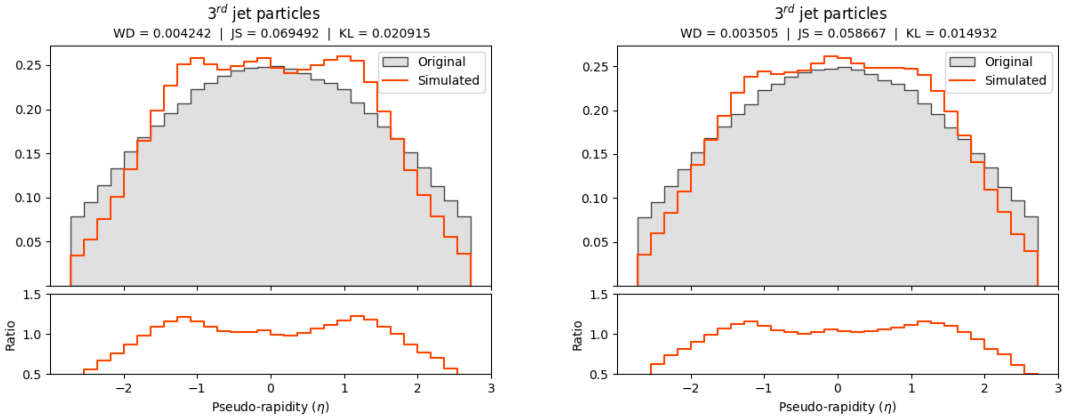
Observing the last jet in our analysis, and observing its energy feature, we can see that most of the values of the generated data distribution are similar than those of the original *ground truth* data. However, there are some differences in the central and rightmost part.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.68. Histograms of 3^{rd} jet p_T particle feature with different β values in experiment III.

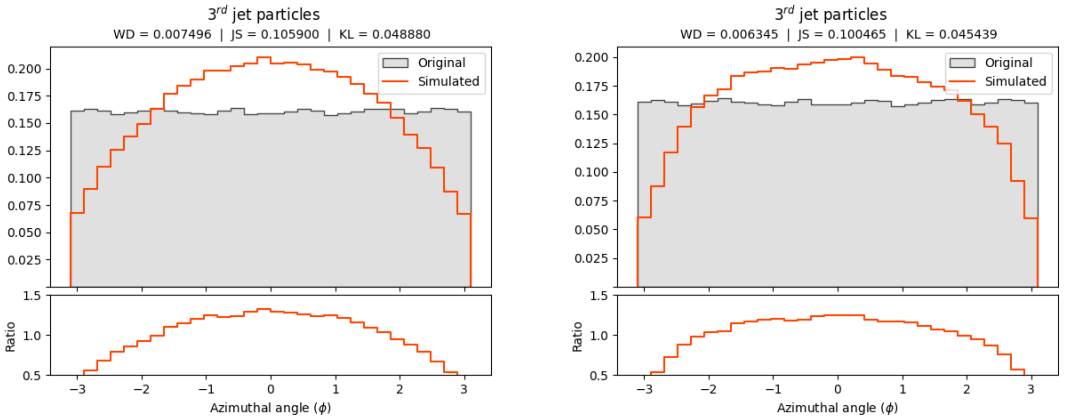
Analyzing the p_T particle feature of the third jet, we see that, except in the left tail of the distribution where the ratio is around 0.5, most of the values have an almost perfect fit with ratios around 1.0 and a distribution shape that matches the original data.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.69. Histograms of 3^{rd} jet η particle feature with different β values in experiment III.

Regarding the η feature of the third jet, we observe that with the highest β value that we tested, the distribution of the generated data is close to that of the original Monte Carlo generated events. The ratio histograms shows a closer match around the central part.



(a) $\beta = 0.7$ **(b)** $\beta = 1.5$

Figure 7.70. Histograms of 3^{rd} jet ϕ particle feature with different β values in experiment III.

To conclude, we observe in the ϕ particle feature of the third jet that the uniform distribution has not been correctly modelled, and looks like a normal distribution. However, the ratio histograms show values around one in the central part.

Observing the results of this last experiment in Table 7.4, we see that the amount of generated invalid events decreases when compared with the previous experiment, and also the generation speed is drastically reduced again, due to the use of only one model for the whole event instead of a model per particle. In fact, this model is the fastest one generating events, taking less than a millisecond per event.

However, we find as a drawback that some distributions, such as the uniform distribution of the ϕ particle feature or the distribution of the η particle feature are not modelled as good as before. Those discrepancies make the data distribution metrics become worse than the results of the previous experiment. In the case of ϕ, this fact is not very

concerning as that feature is present for validation purposes but its physics meaning is not of high relevance.

For this last experiment, we do not appreciate a high difference in the results when the value of β changes. There is a similar amount of generated invalid events, and the data distribution metrics do not vary drastically.

All things considered, we observe that the best value of the β hyperparameter in this case is $\beta = 1.5$, as it obtains the best value in all the data distribution metrics and the second best in the invalid event ratio quality metric.

β	**Data distribution metrics**			**Event generation quality**	
	WD	*JS*	*KL*	*IER*	*Gen. speed*
0.5	0.005011	0.080478	0.030602	4.99%	**0.92 ms**
0.7	0.005045	0.079034	0.029801	5.33%	0.94 ms
1.0	0.004477	0.072933	0.025383	4.95%	0.95 ms
1.2	0.004469	0.070651	0.023814	4.84%	0.96 ms
1.5	**0.004172**	**0.067629**	**0.022565**	4.40%	0.95 ms
2.0	0.004218	0.070423	0.024934	**4.33%**	0.94 ms

Table 7.4. Metrics of the generated events in experiment III.

Finally, analyzing the analysis of the correlation in Figure 7.71, which was the key aspect to improve in this experiment, we find that it has improved with respect to both of the previous approaches. In this model, we find that the small correlation that existed between the `MET` data and some particle features has been maintained, and that the correlation among particles themselves has also increased to levels that are close to those of the original data.

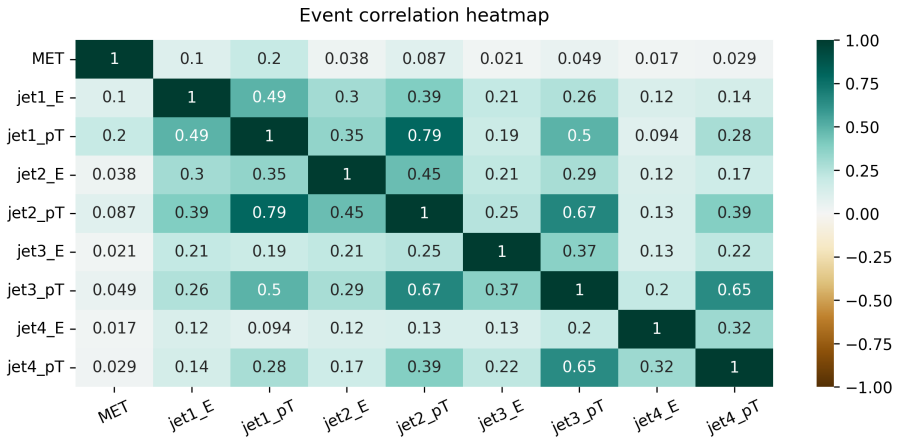


Figure 7.71. Correlation heatmap of several features on events generated in experiment III. Some features have been omitted due to them not being present in this subset of the original data.

However, it must be taken into account that this experiment has been designed only using a subset of the original dataset and with the particular properties of this kind of process (i.e., using up to 4 jets and 2 bjets). This means that, for other kind of processes, their most frequent particle combinations should be analyzed and used as inputs for a potential alternative model.

7.7 Model application in BSM processes

In experiment I, we trained a model using the dataset of a particular kind of event, obtaining some results that adjusted properly to the original (*ground truth*) data distributions, but those events were generated starting from already existing data. Therefore, we decided to try some variants with the objective of being able to generate events just from a random normal distribution, without any previous data.

Our objective was to use the models with other kinds of events once we found a model that was good enough. We proposed other model architectures during experiments II and III, obtaining two promising approaches that fit the distributions of *ttbar* events accurately, with some room for improvement.

To conclude our work, we decided to train the two β-VAE models that were able to obtain good results with random generation among all the executed experiments with a different type of event: *stop_02*, from the Beyond the Standard Model event dataset (described in Table 3.2).

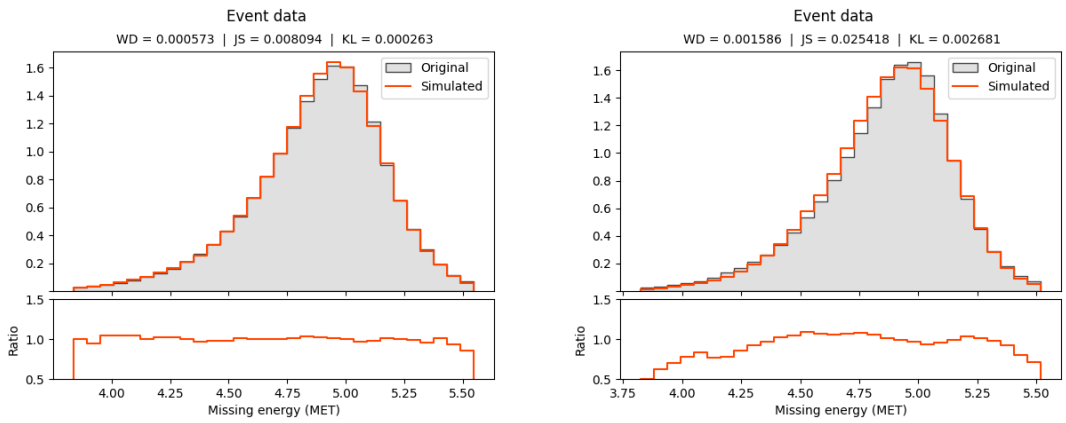
The chosen models that we highlight here were the following ones:

- **Experiment II** architecture: using a β-VAE with $\beta = 1.5$
- **Experiment III** single model: also using a β-VAE with $\beta = 1.5$

In the case of experiment III, we also decided to go with a model with up to 4 jets and up to 2 bjets, because we observed that 42.72% of the samples of that event type met those constraints.

With this final experiment, we can see how well these models perform for any kind of event, and we see whether we can generate BSM events that help proving if the theoretical foundations of these events can be observed in practice or not.

7.7.1. Training results

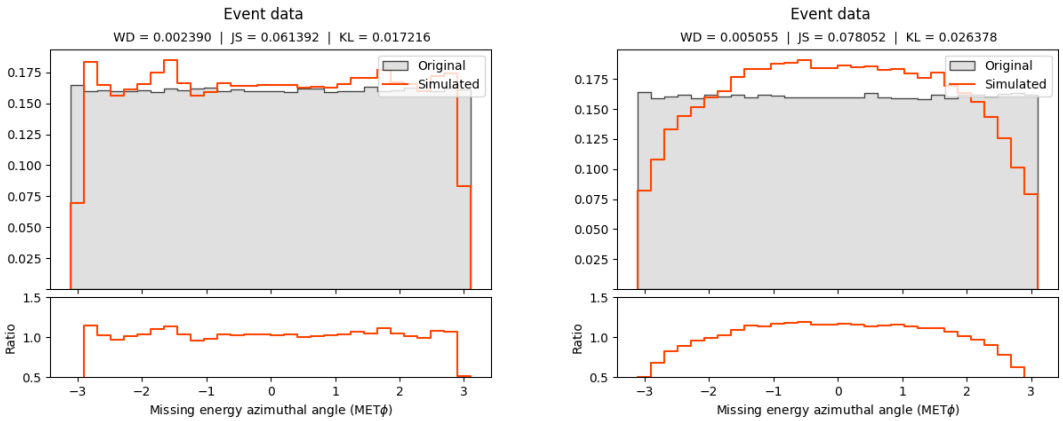


(a) Experiment II: β-VAE ($\beta = 1.5$)

(b) Experiment III: β-VAE ($\beta = 1.5$)

Figure 7.72. Histograms of `MET` event feature with different models in the BSM dataset.

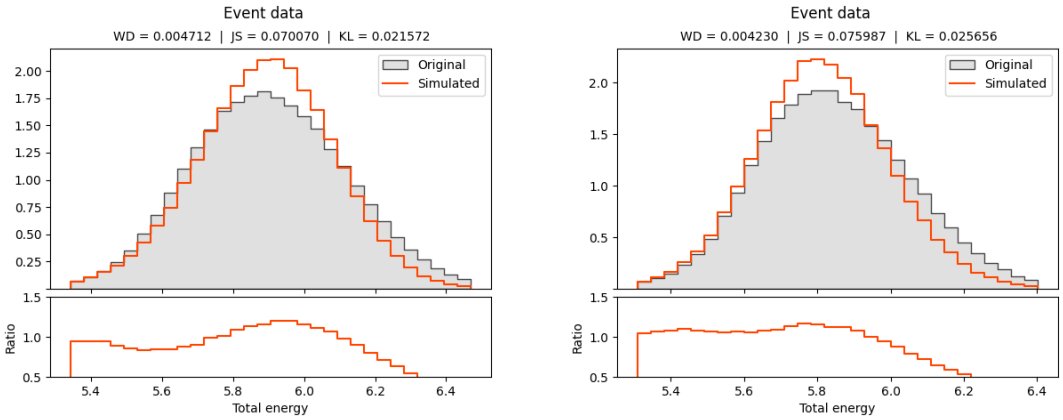
Looking at the `MET` event feature, we can see a great adjustment in the models of both experiments, being the model of experiment II the best result. However, the distribution in experiment III is also very close to the original *ground truth* data.



(a) Experiment II: β-VAE ($\beta = 1.5$) **(b)** Experiment III: β-VAE ($\beta = 1.5$)

Figure 7.73. Histograms of $\texttt{MET}\phi$ event feature with different models in the BSM dataset.

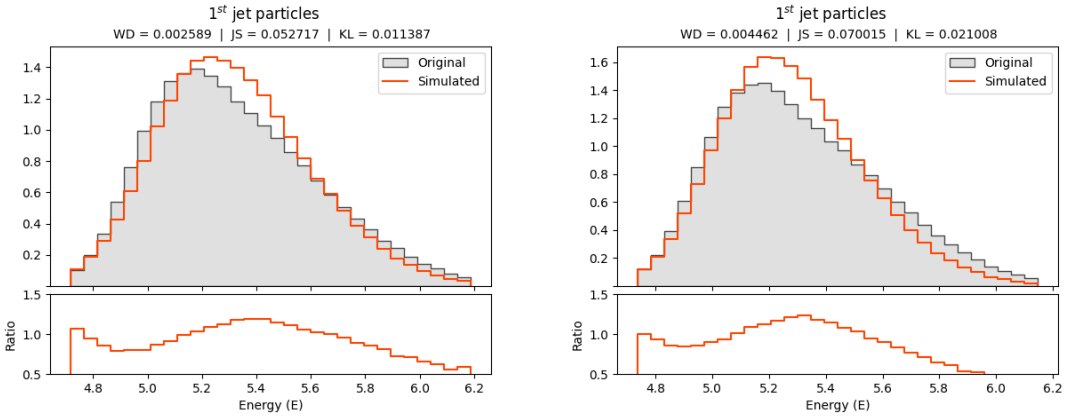
Regarding the $\texttt{MET}\phi$ event feature, we can see that the model of experiment II is able to model the uniform distribution in a more precise way than experiment III. However, in both experiments the ratio is close to one except in the left and rightmost values.



(a) Experiment II: β-VAE ($\beta = 1.5$) **(b)** Experiment III: β-VAE ($\beta = 1.5$)

Figure 7.74. Histograms of total energy event feature with different models in the BSM dataset.

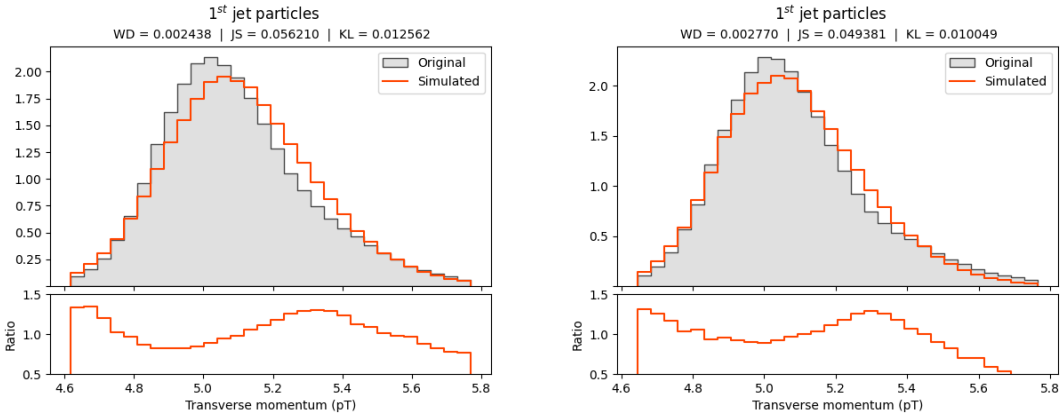
Observing the total energy of the events in both experiments, we can see a similar result. The main difference in both cases is the higher amount of values in the central part of the generated events distribution.



(a) Experiment II: β-VAE ($\beta = 1.5$) **(b)** Experiment III: β-VAE ($\beta = 1.5$)

Figure 7.75. Histograms of 1^{st} jet E particle feature with different β values in the BSM dataset.

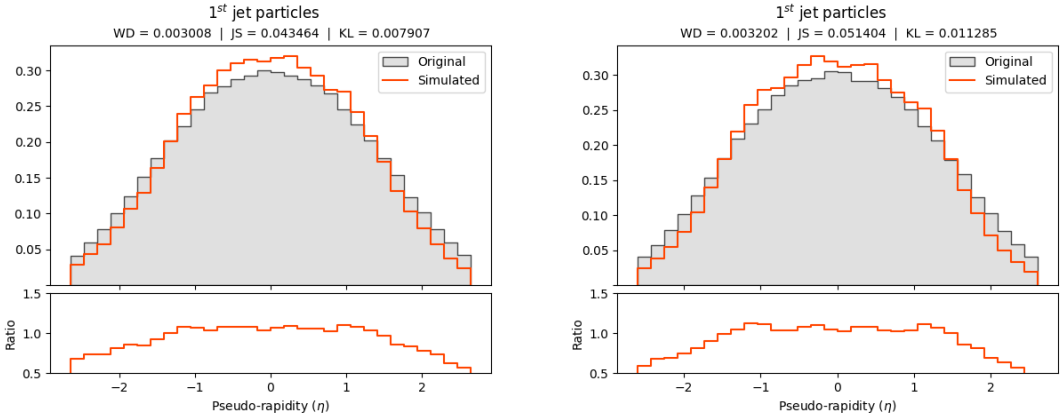
Moving on to particle features, and looking at the energy of the first jet, we see a similar distribution in both experiments, being the result in the second experiment a bit more adjusted to the original data than the one on the third experiment. It should be taken into account that the architecture of experiment II was composed of a model per particle type, whereas experiment III is only composed of one model for all kinds of particles and event features.



(a) Experiment II: β-VAE ($\beta = 1.5$) **(b)** Experiment III: β-VAE ($\beta = 1.5$)

Figure 7.76. Histograms of 1^{st} jet p_T particle feature with different β values in the BSM dataset.

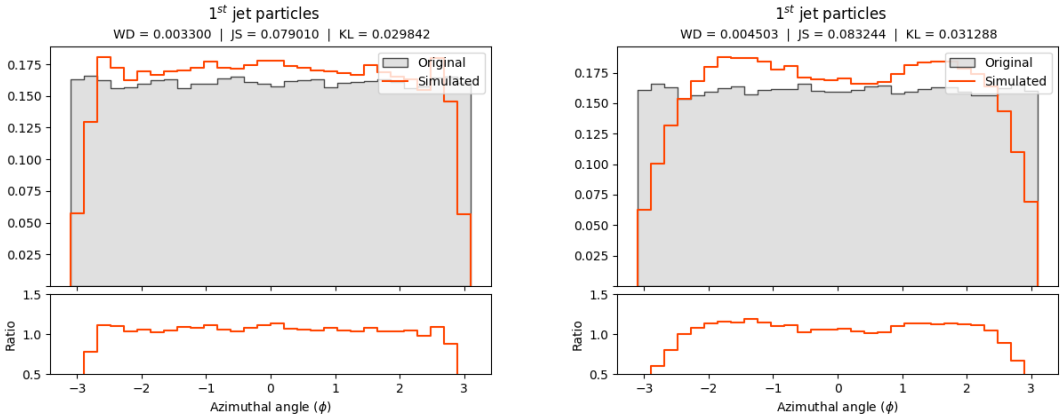
Analyzing the p_T feature of the first jet we observe that in this case, although both results fit the original data distribution very well, the one obtained in the data generated from the third experiment fits slightly better, as we can see in two of the three data distribution metrics.



(a) Experiment II: β-VAE ($\beta = 1.5$) **(b)** Experiment III: β-VAE ($\beta = 1.5$)

Figure 7.77. Histograms of 1^{st} jet η particle feature with different β values in the BSM dataset.

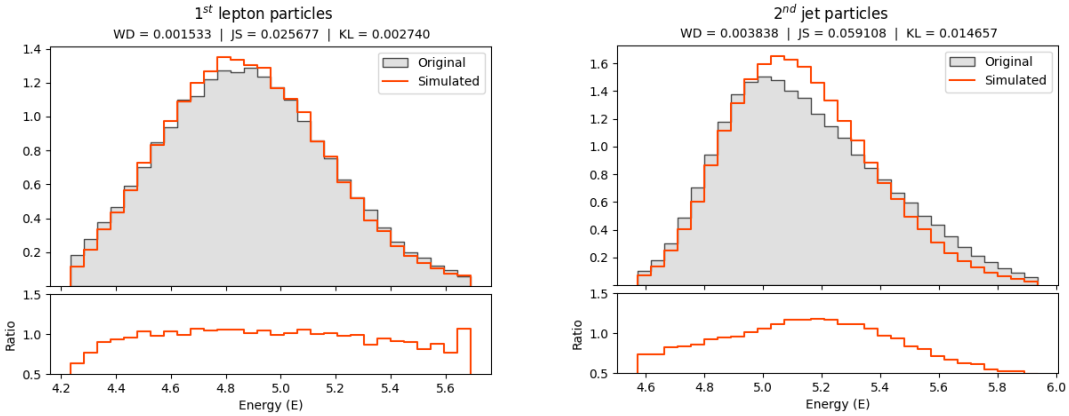
The η particle feature of the first jet shows two distributions that resemble the original *ground truth* data in a great way, being the biggest divergence the one observed in the central values of the distribution of both experiments.



(a) Experiment II: β-VAE ($\beta = 1.5$) **(b)** Experiment III: β-VAE ($\beta = 1.5$)

Figure 7.78. Histograms of 1^{st} jet ϕ particle feature with different β values in the BSM dataset.

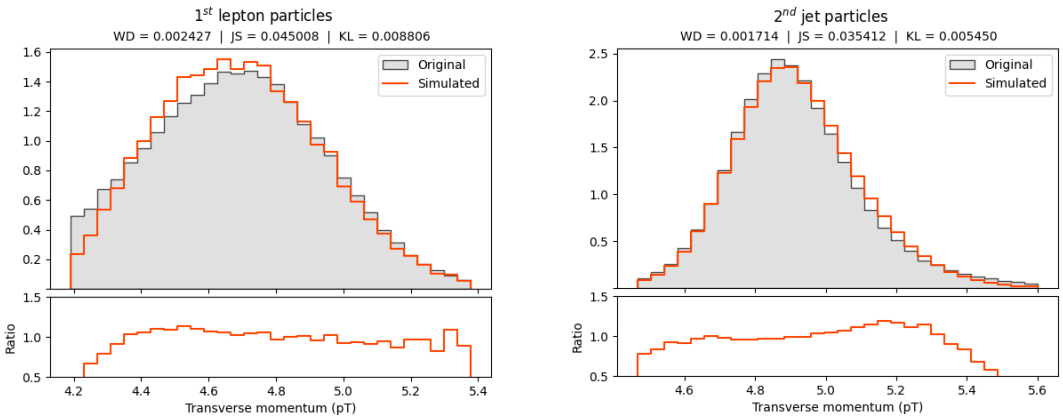
Regarding the ϕ feature of the first jet, we find in this case a better fit in the case of experiment II than what is generated by the model of experiment III. In the latter, we observe that the model tries to merge normal distributions to generate a uniform one.



(a) Exp. II: β-VAE ($\beta = 1.5$) - 1^{st} lepton **(b)** Exp. III: β-VAE ($\beta = 1.5$) - 2^{nd} jet

Figure 7.79. Histograms of E particle feature with different β values in the BSM dataset.

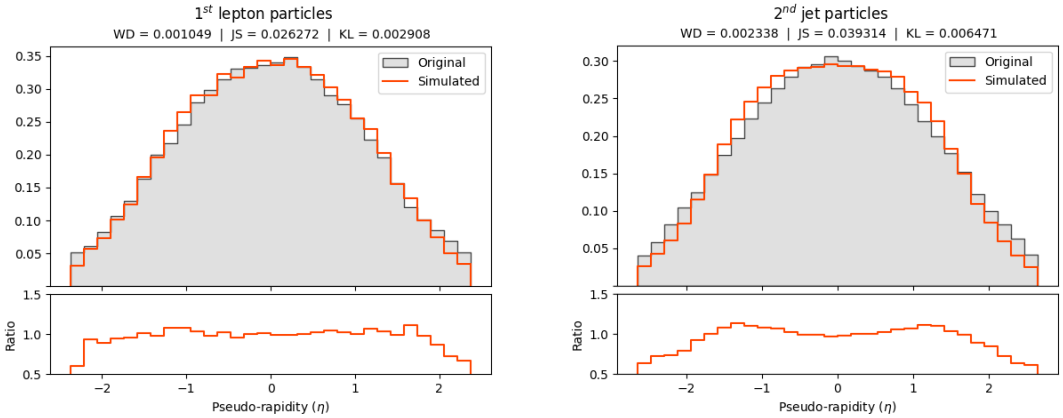
Observing the E feature of the first lepton in experiment II, see a great match between the original and generated data, with only a bigger ratio in the rightmost values of the distribution. The same happens in that feature of the second jet in experiment III.



(a) Exp. II: β-VAE ($\beta = 1.5$) - 1^{st} lepton **(b)** Exp. III: β-VAE ($\beta = 1.5$) - 2^{nd} jet

Figure 7.80. Histograms of p_T particle feature with different β values in the BSM dataset.

Looking at the p_T feature of the first lepton in experiment II, we find some differences in the leftmost part of the distribution, although most of the distribution has a great fit with most ratio values being around one. In the same feature of the second jet in experiment III, we see a greatly adjusted distributions with the biggest ratios in the right tail.

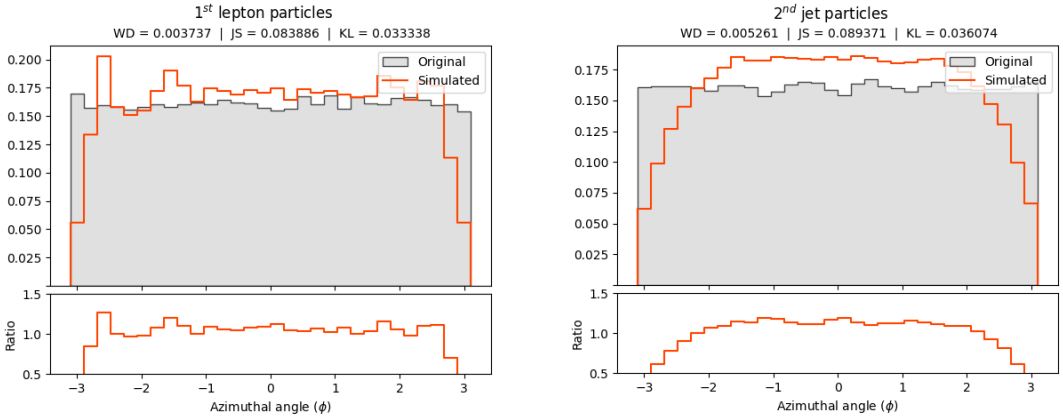


(a) Exp. II: β-VAE ($\beta = 1.5$) - 1^{st} lepton

(b) Exp. III: β-VAE ($\beta = 1.5$) - 2^{nd} jet

Figure 7.81. Histograms of η particle feature with different β values in the BSM dataset.

In the η feature of the first lepton in experiment II, we observe again that the model is able to learn the shape of the *ground truth* data in a very accurate way. Looking at the same feature of the second jet in experiment III, the adjustment of the generated data distribution is again very close to the original data, with some minor differences.

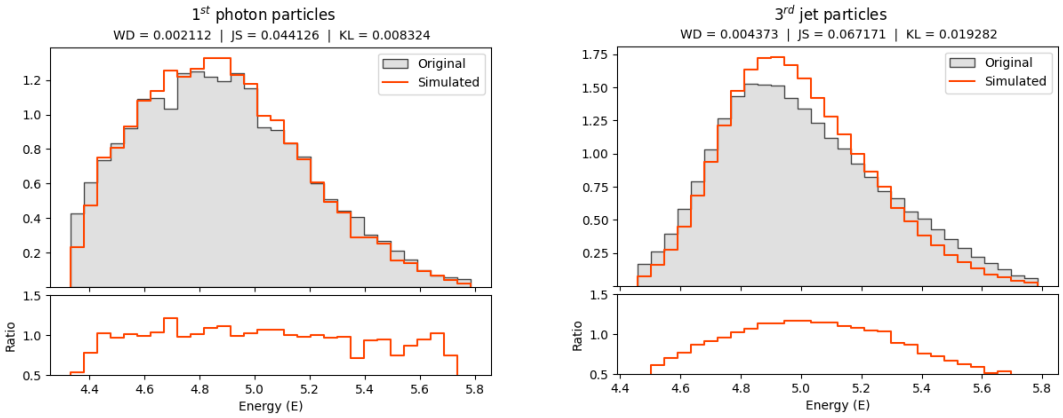


(a) Exp. II: β-VAE ($\beta = 1.5$) - 1^{st} lepton

(b) Exp. III: β-VAE ($\beta = 1.5$) - 2^{nd} jet

Figure 7.82. Histograms of ϕ particle feature with different β values in the BSM dataset.

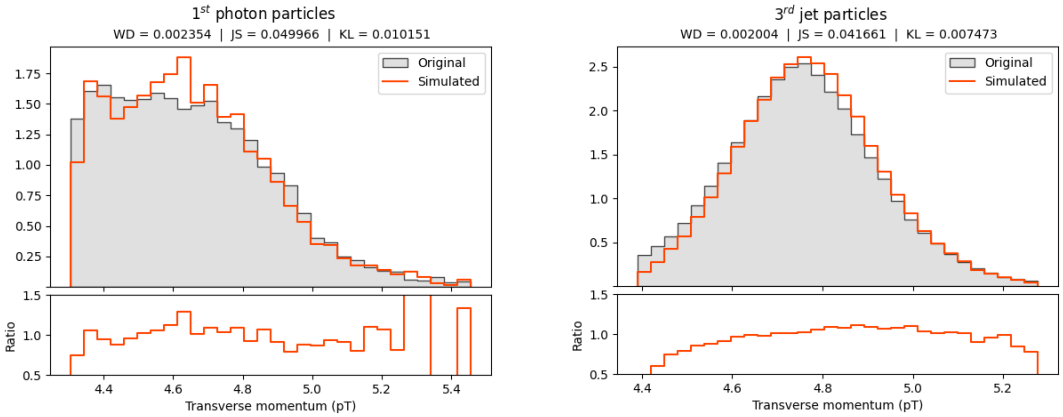
Analyzing the ϕ particle feature of both the first lepton in experiment II and the second jet in experiment III, we can see that the ratio values are all around one. The generated data of the second experiment resembles more to the uniform distribution than the data of the third one, which is more similar to a normal distribution.



(a) Exp. II: β-VAE ($\beta = 1.5$) - 1^{st} photon **(b)** Exp. III: β-VAE ($\beta = 1.5$) - 3^{rd} jet

Figure 7.83. Histograms of E particle feature with different β values in the BSM dataset.

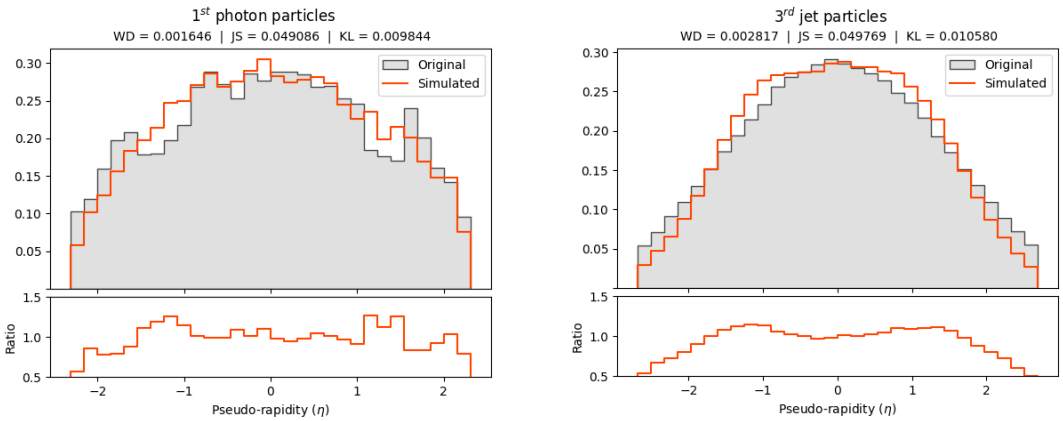
The E feature of the first photon in experiment II follows a very similar distribution than the original events. In the same feature of the third jet in experiment III, we find the biggest differences in the central part of the distribution.



(a) Exp. II: β-VAE ($\beta = 1.5$) - 1^{st} photon **(b)** Exp. III: β-VAE ($\beta = 1.5$) - 3^{rd} jet

Figure 7.84. Histograms of p_T particle feature with different β values in the BSM dataset.

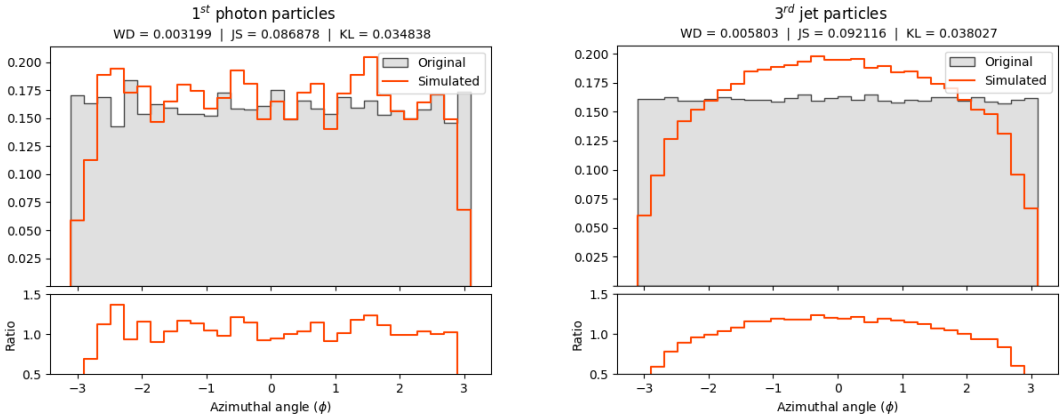
Regarding the p_T particle feature of the first photon in experiment II, we can see that there are some differences in the highest values, with some ratios ≥ 1.5. The generated data of the 3^{rd} jet in the third experiment presents a very close match to the original data.



(a) Exp. II: β-VAE ($\beta = 1.5$) - 1^{st} photon **(b)** Exp. III: β-VAE ($\beta = 1.5$) - 3^{rd} jet

Figure 7.85. Histograms of η particle feature with different β values in the BSM dataset.

Regarding the η feature of the first photon in experiment II, we can see that the ratios vary greatly, but are always greater than 0.5 and less than 1.5. The generated data of the third jet in the third experiment fits the original data and the ratios are also inside the previously mentioned interval.



(a) Exp. II: β-VAE ($\beta = 1.5$) - 1^{st} photon

(b) Exp. III: β-VAE ($\beta = 1.5$) - 3^{rd} jet

Figure 7.86. Histograms of ϕ particle feature with different β values in the BSM dataset.

To conclude, the ϕ feature of the first photon in experiment II presents the same behaviour as this same feature on previous analyzed particles on this experiment: a great adjustment with some variations in the ratio. That feature in the third jet of the third experiment resembles more to a normal distribution but it also maintains ratios around the 0.5 to 1.5 interval.

After observing the results of the final executions over the BSM dataset, we find that in both cases the results are similar, even slightly better than the ones obtained with the same model architectures trained and used to generate events with the *ttbar* dataset. In general, as we can see in Table 7.5, the data distribution metrics improve, and the proportion of invalid generated events increases in the case of experiment II and is reduced to almost half in experiment III.

The generation speed stays the same as the model architectures are the same independently of the trained weights for each physical process.

Experiment	β	**Data distribution metrics**			**Event generation quality**	
		WD	*JS*	*KL*	*IER*	*Gen. speed*
II	1.5	0.002388	0.050689	0.013760	5.35%	75.07 ms
III	1.5	0.003779	0.064287	0.020012	2.80%	0.95 ms

Table 7.5. Metrics of the generated BSM events in experiments II and III.

Regarding the event features correlation, we find a similar situation in this kind of process when compared to the SM dataset: in the original data (Figure 7.87), we find a strong correlation between the energy (E) and transverse momentum (p_T) of the first four jets in each particle order and also between different orders. The first lepton's E and p_T are also correlated among them but not with respect to other particles (jets). There is also a slight correlation between the missing energy (`MET`) and some particle features.

The previously described correlation of the particle features decreases slightly but is mostly kept in experiment II (Figure 7.88), but the correlation among the `MET` and particle features is completely lost. However, in experiment III (Figure 7.89) that correlation is restored at the same time that the particle feature correlation is maintained.

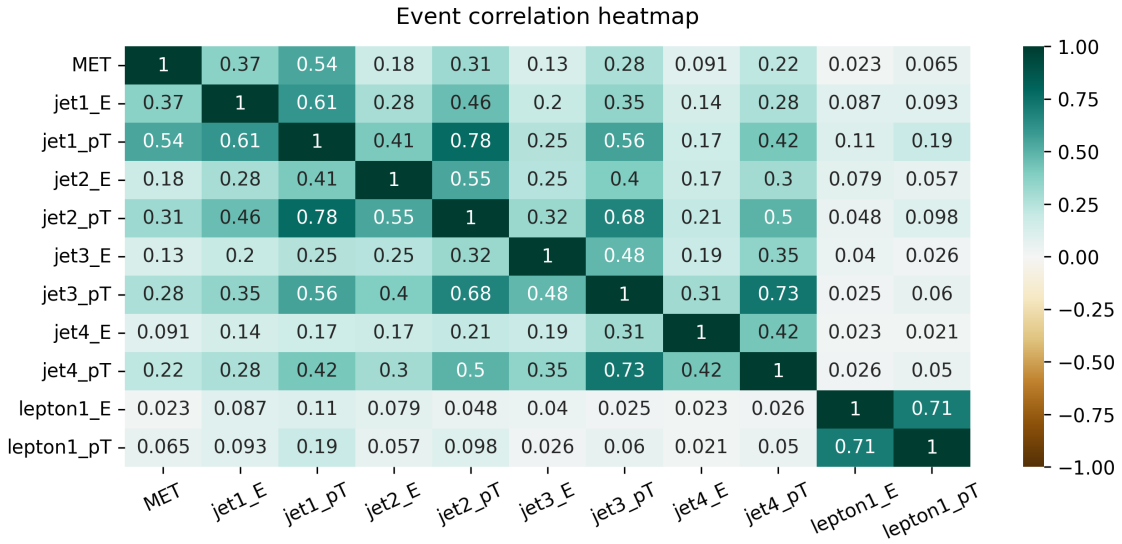


Figure 7.87. Correlation heatmap of several features on *ground truth* BSM events.

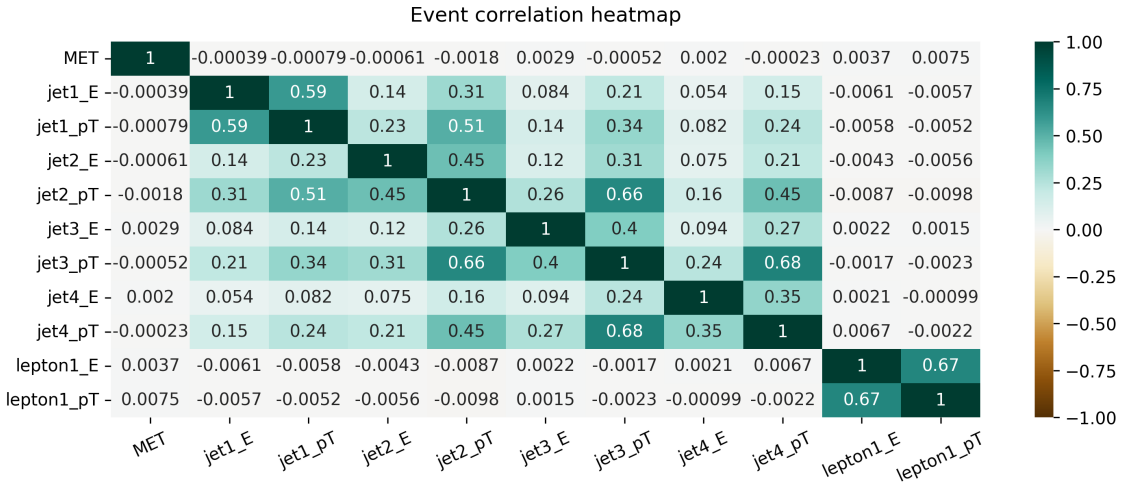


Figure 7.88. Correlation heatmap of several features on BSM events in experiment II.

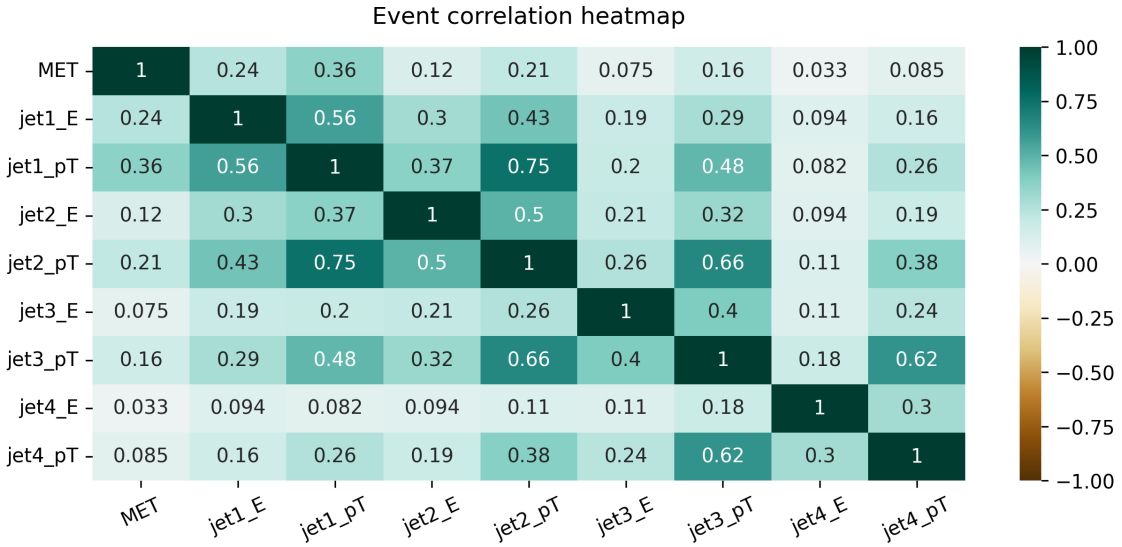


Figure 7.89. Correlation heatmap of several features on BSM events in experiment III.
Some features have been omitted due to them not being present in this subset of the original data.

CHAPTER 8
Conclusions

During the development of this work, we have experimented with several approaches to generate events in a more efficient way when compared to traditional, Monte Carlo-based methods. We explored a wide range of model variants than can be created using Variational Autoencoders.

Through those experiments, we obtained many interesting results in our work. We generated events from the Standard Model (Table 8.1) and Beyond the Standard Model (Table 7.5) datasets, and the generation process was done in a notably faster way than traditional Monte Carlo methods.

We presented a model that used α-VAEs to overcome the shortcomings of the best model of our previous work using $\alpha \geq 0.2$, obtaining as a result a solid model to generate events taking as a starting point *ground truth* data. Then, we explored some β-VAE models to generate events from random numbers, that performed better when using high values of β (≥ 1.0).

Despite the great results achieved in this work, there is a continuing need to tweak the internal architecture and parameters of the event generation models that use random numbers. This refinement is necessary to enhance the generation of specific particle features that have not aligned closely enough with their original distributions, as we would have liked.

However, considering the initially defined objectives, we can confidently assert that they have been successfully met. We have developed a framework for event generation that represents a significant improvement over conventional methods in terms of time and computational efficiency. Moreover, this framework allows for further enhancements to be made in the future, leaving some room for future improvements.

Experiment	α	β	**Data distribution metrics**			**Event generation quality**	
			WD	*JS*	*KL*	*IER*	*Gen. speed*
Baseline	0.2	-	0.002077	0.037879	0.013133	1.68%	1.23 ms
I	0.3	-	0.000973	0.023365	0.004781	0.46%	339.90 ms
II	-	1.5	0.002771	0.055761	0.016530	4.55%	74.64 ms
III	-	1.5	0.004172	0.067629	0.022565	4.40%	0.95 ms

Table 8.1. Metrics of the generated events in the best model of each experiment.

8.1 Future work

Given the enormous potential and critical importance of the topic at hand, and due to the future need of generating a large volume of events, we propose here some future work that could be done to continue improving the event generation techniques further, taking as a starting point our work.

On the one hand, it would be interesting to explore other kinds of generative models in the research. As we commented in our study of the state of the art, flow-based models are a promising approach that have already been used for the same purpose of our investigation.

Perhaps, using Autoencoders as components of more complex Normalizing Flow models, in such a way that the density of points in the latent space is learned by a flow model trained end-to-end with an AE, would produce good results.

On the other hand, because we believe that VAEs still have some potential to improve the results, different model architectures than the ones we have experimented with could be tested, so that it can be either confirmed that these models can generate events up to a certain quality, or its results can improve and become even better.

Finally, it would be interesting to create some architectures that are based on multiple models, where each model takes as input more information to have more context in order to keep the correlation among different particle features and with the missing energy event feature. The correlation itself could also be considered as an input to be provided to the model during training.

Bibliography

[1] Salvatore Rappoccio. The experimental status of direct searches for exotic physics beyond the standard model at the Large Hadron Collider. *Reviews in Physics*, 4:100027, 2019. doi:10.1016/j.revip.2018.100027.

[2] Riccardo Maria Bianchi and ATLAS Collaboration. ATLAS experiment schematic illustration. Available at: https://cds.cern.ch/record/2837191.

[3] Raúl Balanzá García. Use of Deep Learning generative models for Monte Carlo event simulation in the context of LHC experiments. Bachelor's thesis, Universitat Politècnica de València, 2022. Available at: http://hdl.handle.net/10251/185480.

[4] Sea Agostinelli, John Allison, K al Amako, John Apostolakis, H Araujo, Pedro Arce, Makoto Asai, D Axen, Swagato Banerjee, GJNI Barrand, et al. GEANT4—a simulation toolkit. *Nuclear instruments and methods in physics research section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003. doi:10.1016/S0168-9002(03)01368-8.

[5] Thea Aarrestad, Melissa van Beekveld, Marcella Bona, Antonio Boveia, Sascha Caron, Joe Davies, Andrea de Simone, Caterina Doglioni, Javier Duarte, Amir Farbin, et al. The Dark Machines Anomaly Score Challenge: Benchmark Data and Model Independent Event Classification for the Large Hadron Collider. *SciPost Physics*, 12(1), jan 2022, doi:10.21468/scipostphys.12.1.043.

[6] Sydney Otten, Sascha Caron, Wieske de Swart, Melissa van Beekveld, Luc Hendriks, Caspar van Leeuwen, Damian Podareanu, Roberto Ruiz de Austri, and Rob Verheyen. Event Generation and Statistical Sampling for Physics with Deep Generative Models and a Density Information Buffer, 2019, doi:10.48550/ARXIV.1901.00875.

[7] ATLAS Collaboration. A strategy for a general search for new phenomena using data-derived signal regions and its application within the ATLAS experiment. *The European Physical Journal C*, 79(2), feb 2019, doi:10.1140/epjc/s10052-019-6540-y.

[8] Hosein Hashemi, Nikolai Hartmann, Sahand Sharifzadeh, James Kahn, and Thomas Kuhr. Ultra-High-Resolution Detector Simulation with Intra-Event Aware GAN and Self-Supervised Relational Reasoning, 2023. doi:10.48550/arXiv.2303.08046.

[9] Riccardo Di Sipio, Michele Faucci Giannelli, Sana Ketabchi Haghighat, and Serena Palazzo. DijetGAN: a generative-adversarial network approach for the simulation of QCD dijet events at the LHC. *Journal of high energy physics*, 2019(8), 2019. doi:10.48550/arXiv.1903.02433.

[10] Louis Vaslin, Vincent Barra, and Julien Donini. GAN-AE : An anomaly detection algorithm for New Physics search in LHC data, 2023. doi:10.48550/arXiv.2305.15179.

[11] Gregor Kasieczka, Benjamin Nachman, David Shih, Oz Amram, Anders Andreassen, Kees Benkendorfer, Blaz Bortolato, Gustaaf Brooijmans, Florencia Canelli, Jack H Collins, et al. The LHC Olympics 2020 a community challenge for anomaly detection in high energy physics. *Reports on progress in physics*, 84(12):124201, 2021. doi:10.1088/2632-2153/ac7c56.

[12] Mary Touranakou, Nadezda Chernyavskaya, Javier Duarte, Dimitrios Gunopulos, Raghav Kansal, Breno Orzari, Maurizio Pierini, Thiago Tomei, and Jean-Roch Vlimant. Particle-based fast jet simulation at the LHC with variational autoencoders. *Machine Learning: Science and Technology*, 3(3):035003, 2022. doi:10.1007/JHEP09(2021)024.

[13] Melissa van Beekveld, Sascha Caron, Luc Hendriks, Paul Jackson, Adam Leinweber, Sydney Otten, Riley Patrick, Roberto Ruiz De Austri, Marco Santoni, and Martin White. Combining outlier analysis algorithms to identify new physics at the LHC. *Journal of High Energy Physics*, 2021(9):1–33, 2021. doi:10.1088/1361-6633/ac36b9.

[14] Anja Butter, Tilman Plehn, Steffen Schumann, Simon Badger, Sascha Caron, Kyle Cranmer, Francesco Armando Di Bello, Etienne Dreyer, Stefano Forte, Sanmay Ganguly, et al. Machine learning and LHC event generation. *SciPost Phys.*, 14:079, 2023. doi:10.21468/SciPostPhys.14.4.079.

[15] Cheng Chen, Olmo Cerri, Thong Q. Nguyen, Jean-Roch Vlimant, and Maurizio Pierini. Data Augmentation at the LHC through Analysis-specific Fast Simulation with Deep Learning, 2020. doi:10.48550/arXiv.2010.01835.

[16] Jesus Arjona Martínez, Thong Q Nguyen, Maurizio Pierini, Maria Spiropulu, and Jean-Roch Vlimant. Particle Generative Adversarial Networks for full-event simulation at the LHC and their application to pileup description. In *Journal of Physics: Conference Series*, volume 1525, page 012081. IOP Publishing, 2020. doi:10.1088/1742-6596/1525/1/012081.

[17] Corentin Salaün, Adrien Gruson, Binh-Son Hua, Toshiya Hachisuka, and Gurprit Singh. Regression-based Monte Carlo integration. *ACM Transactions on Graphics (TOG)*, 41(4):1–14, 2022. doi:10.1145/3528223.3530095.

[18] R Cranley and TNL Patterson. A regression method for the Monte Carlo evaluation of multidimensional integrals. *Numerische Mathematik*, 16:58–72, 1970. doi:10.1007/BF02162407.

[19] Christina Gao, Joshua Isaacson, and Claudius Krause. i-flow: High-dimensional integration and sampling with normalizing flows. *Machine Learning: Science and Technology*, 1(4):045023, nov 2020. doi:10.1088/2632-2153/abab62.

[20] Rob Verheyen. Event Generation and Density Estimation with Surjective Normalizing Flows. *SciPost Physics*, 13(3), sep 2022, doi:10.21468/scipostphys.13.3.047.

[21] Pratik Jawahar, Thea Aarrestad, Nadezda Chernyavskaya, Maurizio Pierini, Kinga A. Wozniak, Jennifer Ngadiuba, Javier Duarte, and Steven Tsan. Improving Variational Autoencoders for New Physics Detection at the LHC With Normalizing Flows. *Frontiers in Big Data*, 5, 2022, doi:10.3389/fdata.2022.803685.

[22] DarkMachines community. LHCsimulationProject, Feb 2020, doi:10.5281/zenodo.3685861. Available at: https://zenodo.org/record/3685861.

[23] G. Brooijmans, A. Buckley, S. Caron, A. Falkowski, B. Fuks, A. Gilbert, W. J. Murray, M. Nardecchia, J. M. No, R. Torre, et al. Les Houches 2019 Physics at TeV Colliders: New Physics Working Group Report, 2020, doi:10.48550/ARXIV.2002.12220.

[24] T. Aarrestad, M. van Beekveld, M. Bona, A. Boveia, S. Caron, J. Davies, A. De Simone, C. Doglioni, J. M. Duarte, A. Farbin, et al. The Dark Machines Anomaly Score Challenge: Benchmark Data and Model Independent Event Classification for the Large Hadron Collider. *SciPost Phys.*, 12:43, 2022, doi:10.21468/SciPostPhys.12.1.043.

[25] Wulff, Eric. Deep Autoencoders for Compression in High Energy Physics, 2020. Student Paper.

[26] Aditya Sharma. Variational Autoencoder in TensorFlow, Apr 2021. Available at: https://learnopencv.com/variational-autoencoder-in-tensorflow/.

[27] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes, 2022. arXiv:1312.6114.

[28] Geektopia/NVIDIA. NVIDIA GeForce RTX 4090, 2022. [Online; accessed June 16, 2023]. Available at: https://www.geektopia.es/es/product/nvidia/geforce-rtx-4090/.

[29] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning, 2016. arXiv:1605.08695.

[30] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[31] Shawn Hymel. Getting Started with Machine Learning Using TensorFlow and Keras. Available at: https://www.digikey.com/en/maker/projects/getting-started-with-machine-learning-using-tensorflow-and-keras/0746640deea84313998f5f95c8206e5b.

[32] Aldrin Yim, Claire Chung, and Allen Yu. *Matplotlib for Python Developers: Effective techniques for data visualization with Python*. Packt Publishing Ltd, 2018.

[33] Ekaba Bisong and Ekaba Bisong. Introduction to Scikit-learn. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, pages 215–229, 2019.

[34] Ángela García Mínguez. Machine learning en el estudio ttbar del experimento ATLAS. Master's thesis, Universitat de València, 2020.

[35] Alastair J. Walker. An Efficient Method for Generating Discrete Random Variables with General Distributions. *ACM Trans. Math. Softw.*, 3(3):253–256, sep 1977. doi:10.1145/355744.355749.

[36] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.

List of Figures in Appendices

APPENDIX A
Model components

In this appendix we describe the main components that we have used to build our models during the experimentation phase of this work, in order to clarify its definition and purpose.

A.1 Activation functions

In neural networks, an activation function is a mathematical function that can introduce non-linearity into the output of a network. It is applied to the weighted sum of the inputs at each neuron or node of the network to determine the output or activation of that neuron. Neurons perform a computation similar to the one described in A.1.

$$out = \sum(weight * input) + bias \tag{A.1}$$

The activation function plays a crucial role in the ability of the neural network to learn complex patterns and make non-linear predictions. Without activation functions, its expressive power would be limited.

Linear

The linear activation function, also known as the identity function, is a simple activation function that computes the output of a neuron as the input value without any modification.

This function does not introduce non-linearity into the network's output and is often used in the output layer of a neural network when the task requires predicting continuous values. It allows the network to learn linear relationships between the input features and the target variable.

Rectified Linear Unit

The Rectified Linear Unit (ReLU) activation function is a non-linear activation function commonly used in neural networks. It introduces non-linearity by mapping the input to the output in a piecewise linear manner.

In other words, the ReLU function "activates" or outputs the input value as long as it is positive, and otherwise, it produces zero. It has gained popularity in deep learning due to its simplicity and computational efficiency. It helps neural networks learn com-

plex patterns and representations by allowing them to model non-linear relationships between the inputs and the output.

Sigmoid

The sigmoid activation function is a common non-linear activation function used in neural networks. It maps the input to a range between 0 and 1, providing a smooth and continuous output.

This function has an S-shaped curve, with the output close to 0 for large negative inputs, close to 1 for large positive inputs, and approximately 0.5 at the origin (x = 0). It is often used in classification problems where a sample can have more than one label, so that each value can be interpreted as the probability of belonging to a certain class.

A comparison of this each activation function can be seen in Figures A.1 and A.2. Moreover, their equations can be observed in Table A.1.

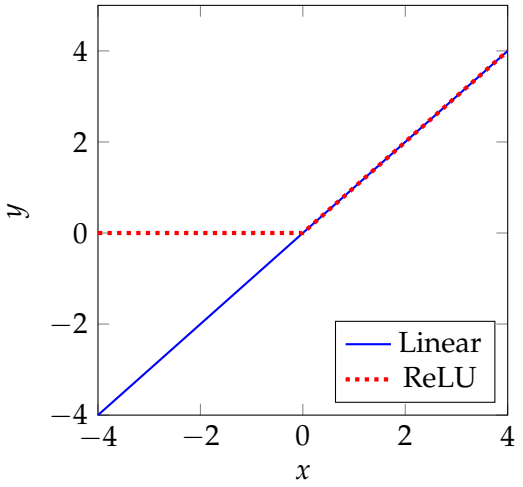


Figure A.1. Linear and ReLU functions.

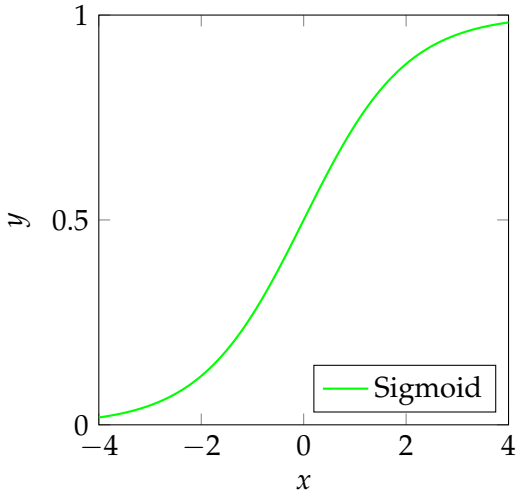


Figure A.2. Sigmoid function.

Linear	ReLU	Sigmoid
$f(x) = x$	$f(x) = max(0, x)$	$f(x) = 1/(1 + e^{-x})$

Table A.1. Equations of each described activation function.

A.2 Loss functions

A loss function in the context of neural networks is a measure of how well the network's predictions match the desired or true values. It quantifies the discrepancy between the predicted output and the actual output, providing a measure of the network's performance.

Each model attempts to learn the probability distribution of the input data, and evaluates the quality of the estimation at each step of the training process using these kind of functions.

The choice of a specific function depends on the specific task or problem the neural network is designed to solve. In this work, we used the *Mean Squared Error*, *Binary Crossentropy*, and *KL loss* (already described in Chapter 4).

Mean Squared Error

The Mean Squared Error (MSE) loss function is a commonly used loss function in regression problems. It measures the average squared difference between the predicted values and the true values. We use it together with the *Linear* activation function.

The formula for the MSE can be expressed as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{pred}}^{(i)} - y_{\text{true}}^{(i)})^2 \tag{A.2}$$

where y_{pred} represents the predicted values, y_{true} represents the true values or targets for each sample, and n is the number of samples in the dataset.

It calculates the squared difference between each predicted value and its corresponding true value. It then averages these squared differences over the entire dataset to compute the final loss value. By squaring the differences, the loss function penalizes larger deviations between the predicted and true values more heavily.

In our case, this function is used to measure the error in the estimation of the different particle features of each event.

Binary Crossentropy

The Binary Crossentropy loss, also known as log loss, is a commonly used loss function in binary classification problems. It measures the dissimilarity between the predicted probabilities and the true binary labels. It is usually combined it with a *Sigmoid* activation function (see A.1).

The formula for this loss function can be expressed as:

$$BCE = -\sum_{i=1}^{n}\left(y_{\text{true}}^{(i)} * \log(y_{\text{pred}}^{(i)}) + (1 - y_{\text{true}}^{(i)}) * \log(1 - y_{\text{pred}}^{(i)})\right) \tag{A.3}$$

where y_{pred} represents the predicted probabilities, y_{true} represents the true binary labels (0 or 1), and n is the number of samples in the dataset.

The loss is calculated for each sample and then averaged over the entire dataset. It penalizes large differences between the predicted probabilities and the true labels. When the predicted probability is close to the true label, the loss is smaller, indicating a better match.

A.3 Layers in Keras

As mentioned in Section 5.2, **Keras** is one of the primary Python libraries used to implement all the models in the conducted experiments. By choosing this library, we were able to concentrate on designing the models without concerning ourselves with implementation details. Keras offers pre-built implementations for the commonly used layers in ANN architectures, enabling us to avoid potential mistakes in that aspect.

Dense

Dense layers are fundamental building blocks of neural network models. They are fully connected layers where each neuron in the layer is connected to every neuron in the

previous layer, creating a dense network of connections. This layer is responsible for performing a linear transformation on the input data, followed by an activation function that can introduce non-linearity.

BatchNormalization

Batch Normalization [36] implements a transformation that ensures the mean output is centered around zero and the output standard deviation is close to one. It operates differently depending on whether the neural network is in the training or inference phase:

- During the **training** phase, the layer normalizes its output by utilizing the mean and standard deviation calculated from the current batch of inputs.
- During **inference**, it ayer normalizes its output using a moving average of the mean and standard deviation, which are computed from the batches it encountered during the training process.

Consequently, the layer will only perform input normalization during inference after having been trained on data that has comparable statistical characteristics to the inference data.

Activation

In Keras, the *Activation* layer is used to apply an activation function to the outputs of the preceding layer in a neural network. It is a simple wrapper around an activation function and is primarily used to introduce non-linearity into the network.

Lambda, *Multiply* and *Add*

Keras also provides some utility layers that provide custom operations or element-wise mathematical operations within a neural network model.

The *Lambda* layer allows to define and apply custom operations to the inputs or outputs of the preceding layer. It is useful for implementing custom transformations to the data. The *Multiply* and *Add* layers perform element-wise multiplication or addition, respectively, on its tensor inputs.

These layers allow to perform specific mathematical operations within the neural network model, providing flexibility in designing complex architectures, such as Variational Autoencoders.

Flatten and *Reshape*

The *Flatten* layer converts the multidimensional input tensor into a 1D array, which is sometimes convenient. Conversely, if the data in our model requires to be converted to a multidimensional format at some point, the *Reshape* layer transforms the data into the desired shape.

In some of our experiments, the data is originally represented in a multidimensional format and we use a *Flatten* layer to transform it into a 1D vector, then it goes through the model, and finally it is transformed back into the original format using the *Reshape* layer.

APPENDIX B
Ground truth data distributions

Before beginning with the experimentation a data analysis process, described in Section 7.2, was carried out. As part of that process, the statistical distributions of the source data of both the used Standard Model and Beyond the Standard Model datasets were calculated. In this appendix, we show the distribution of the *ground truth* data of both datasets.

Note that, as mentioned in Section 7.2, we noticed that the E and p_T features followed a log-normal distribution, and therefore the logarithm has been applied to the histograms of those event and particle features.

B.1 Event features

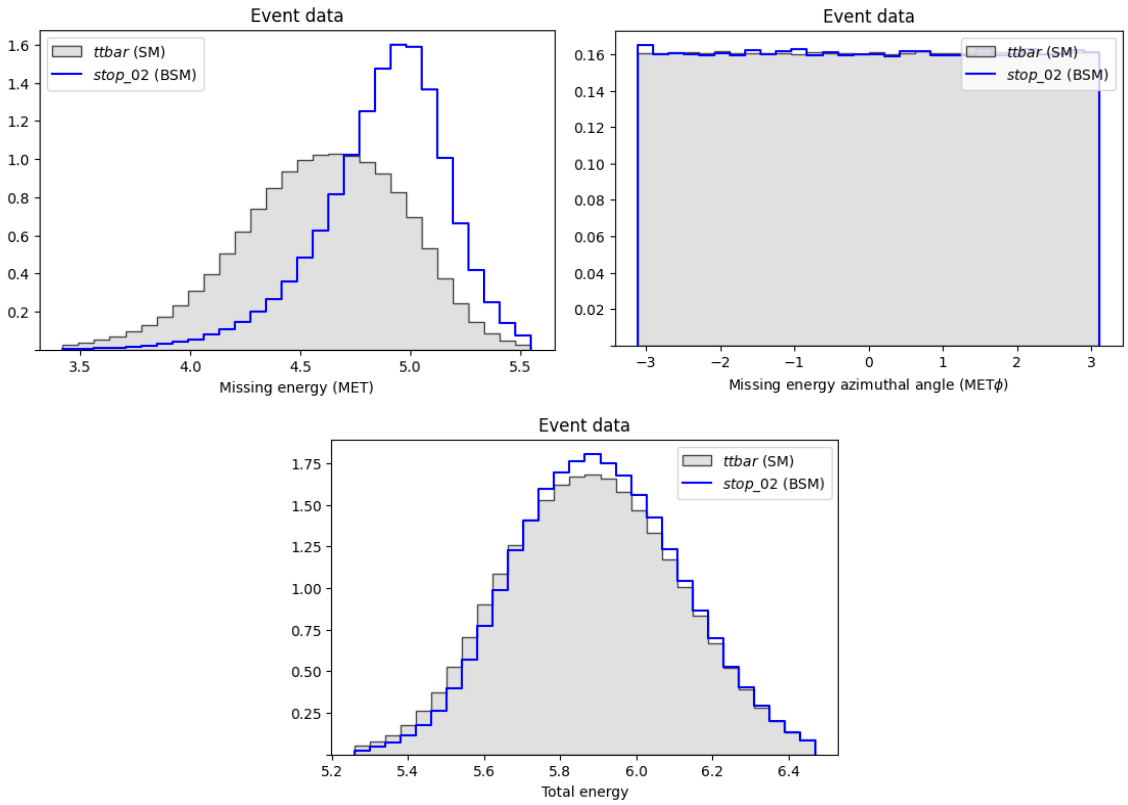


Figure B.1. Histograms of missing energy (`MET` and `METϕ`) and total energy in each event.

B.2 Particle data

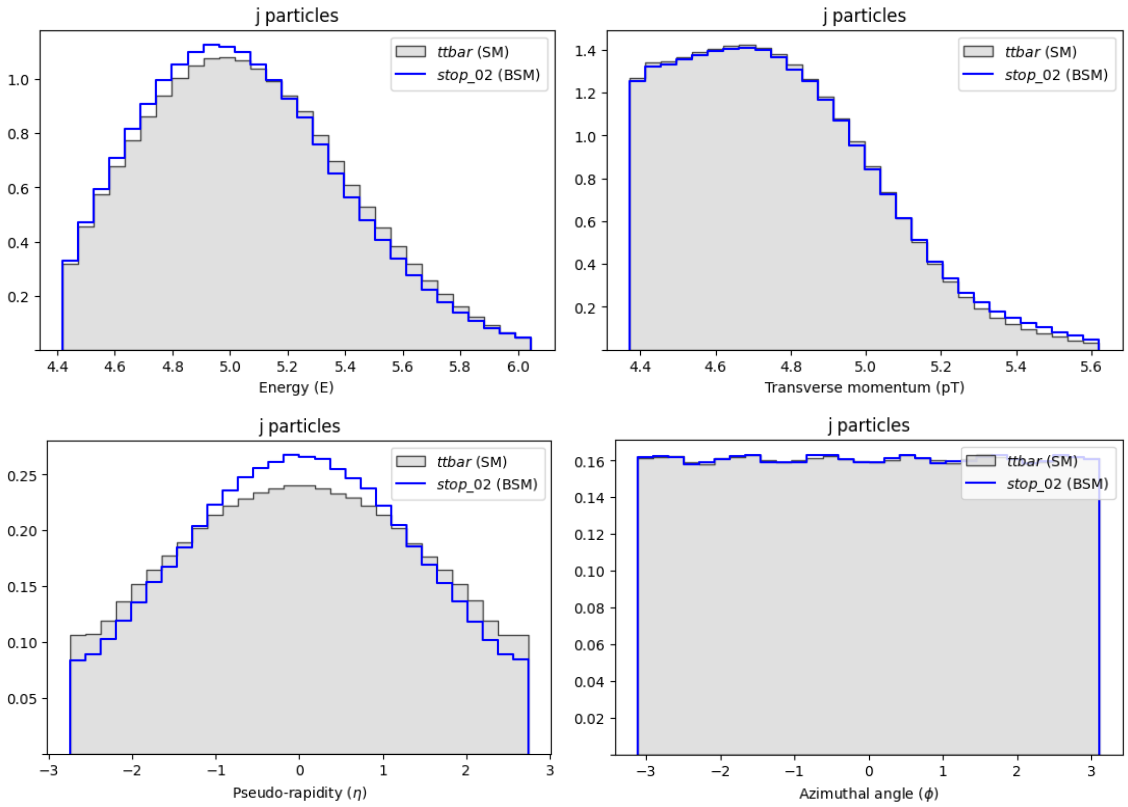


Figure B.2. Histograms of jet (j) parameters in the *ground truth* data.

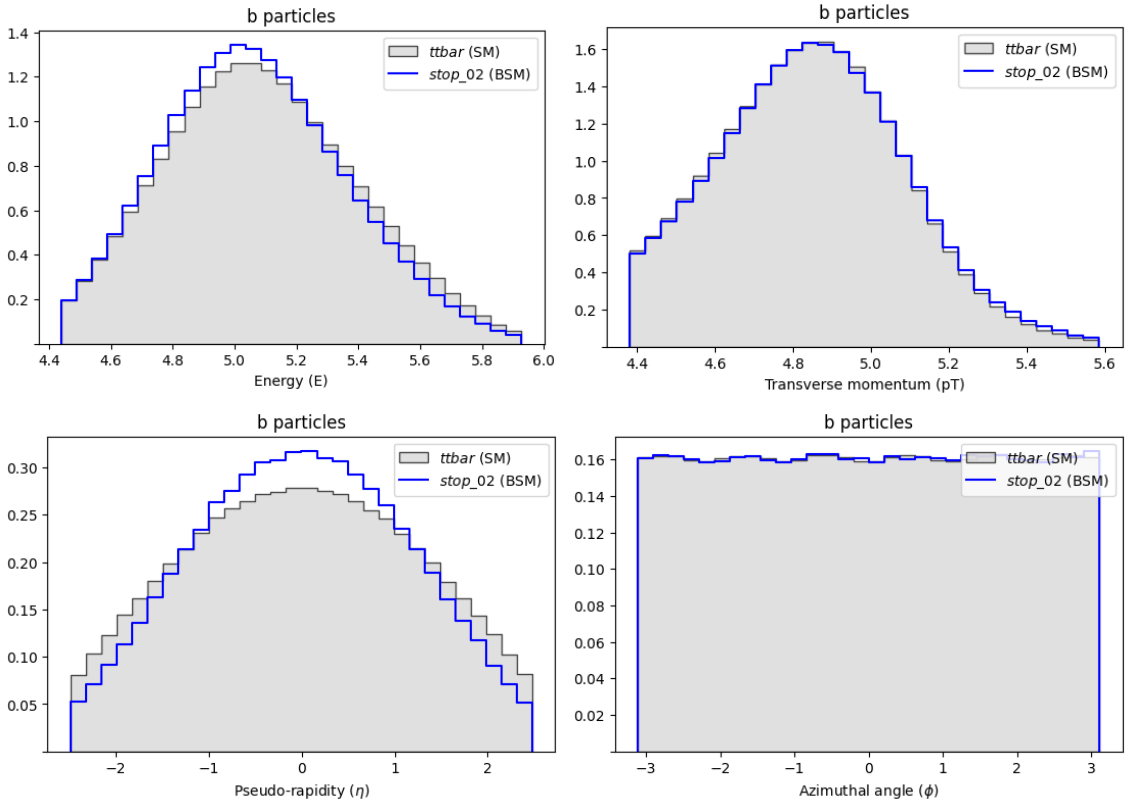


Figure B.3. Histograms of bjet (b) parameters in the *ground truth* data.

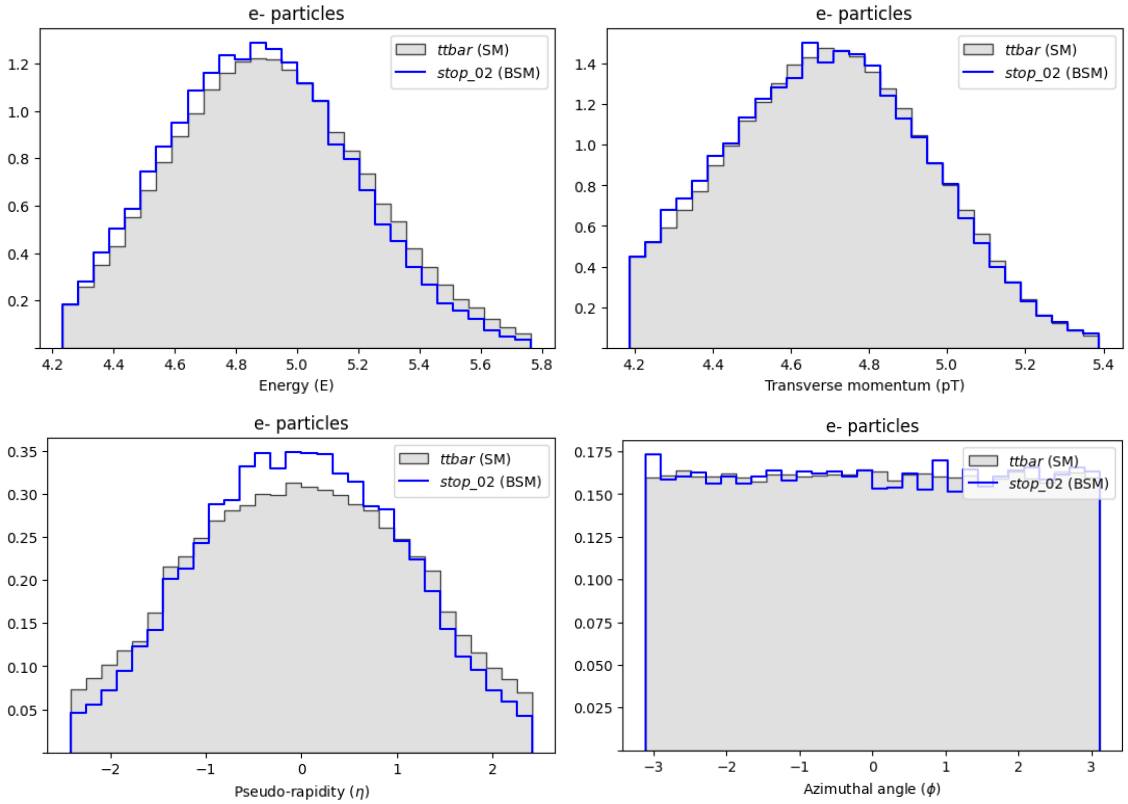


Figure B.4. Histograms of electron (e-) parameters in the *ground truth* data.

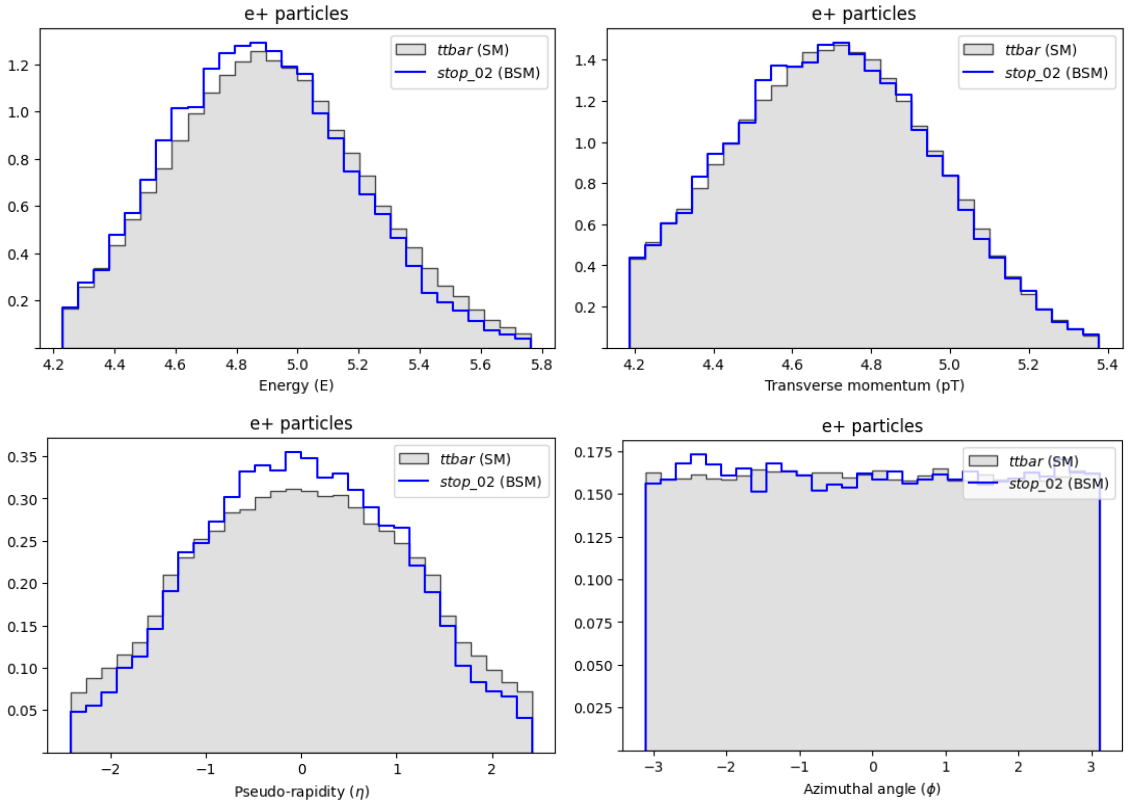


Figure B.5. Histograms of positron (e+) parameters in the *ground truth* data.

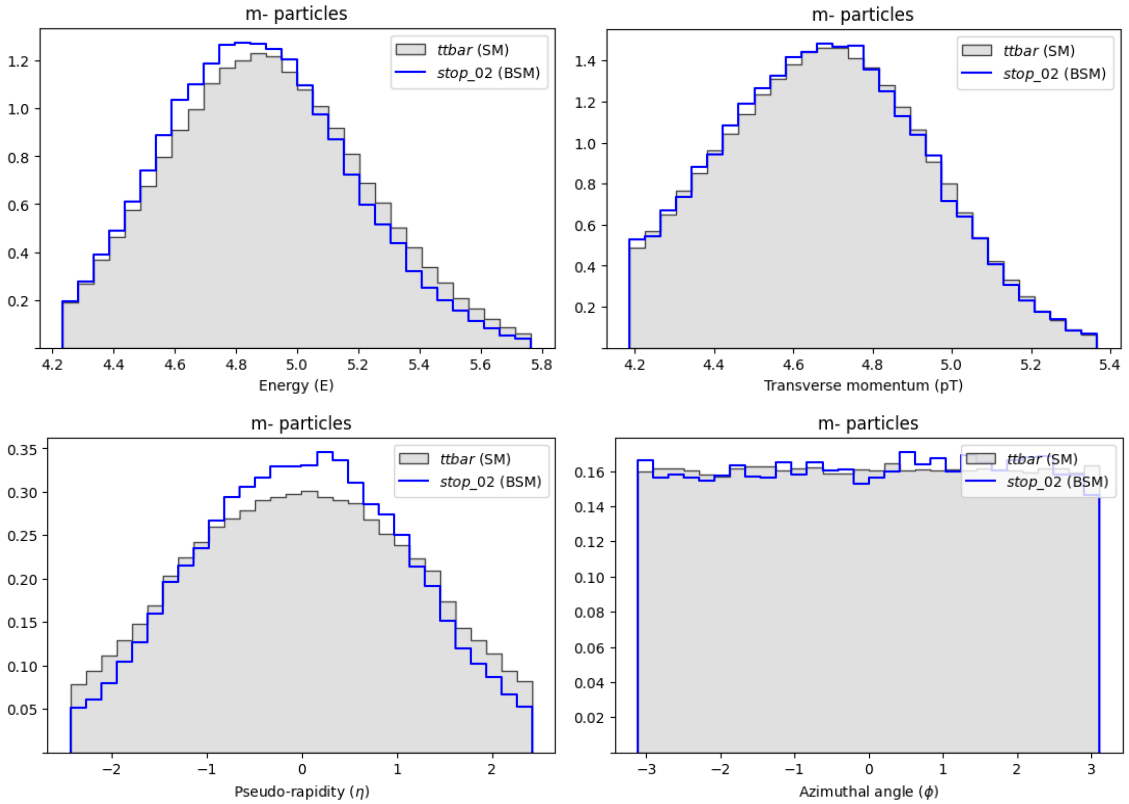


Figure B.6. Histograms of muon (`m-`) parameters in the *ground truth* data.

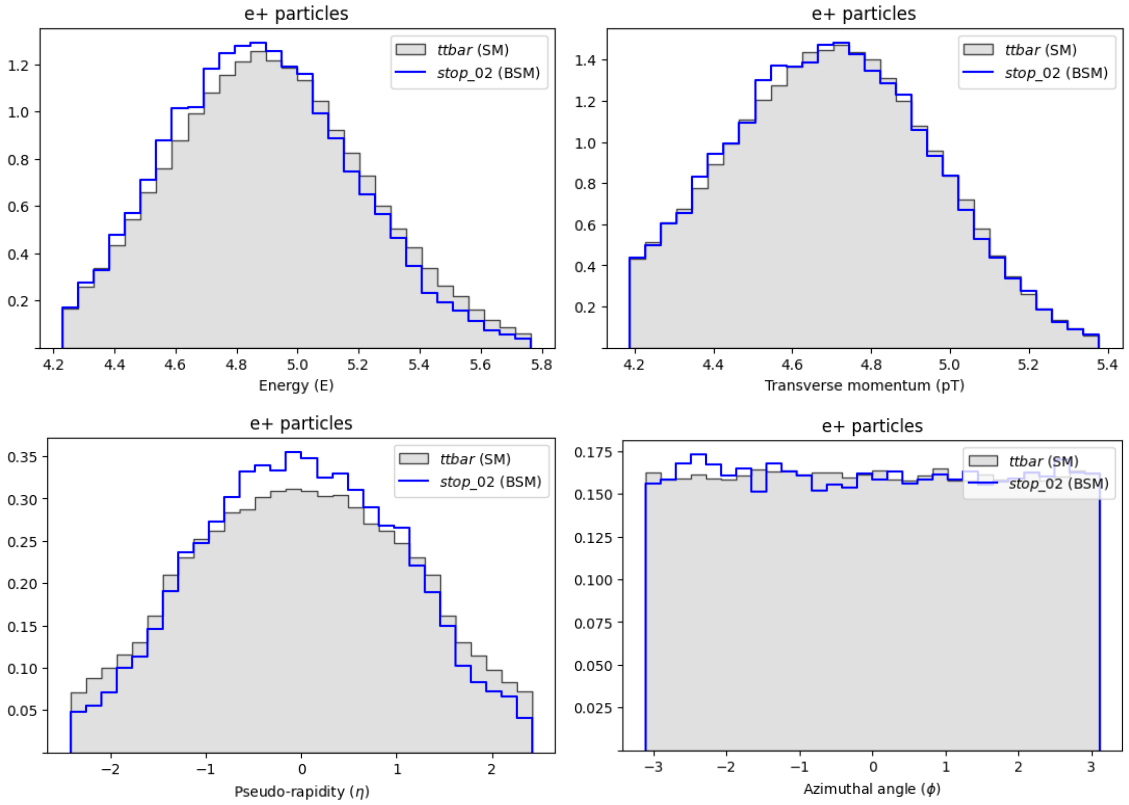


Figure B.7. Histograms of antimuon (`m+`) parameters in the *ground truth* data.

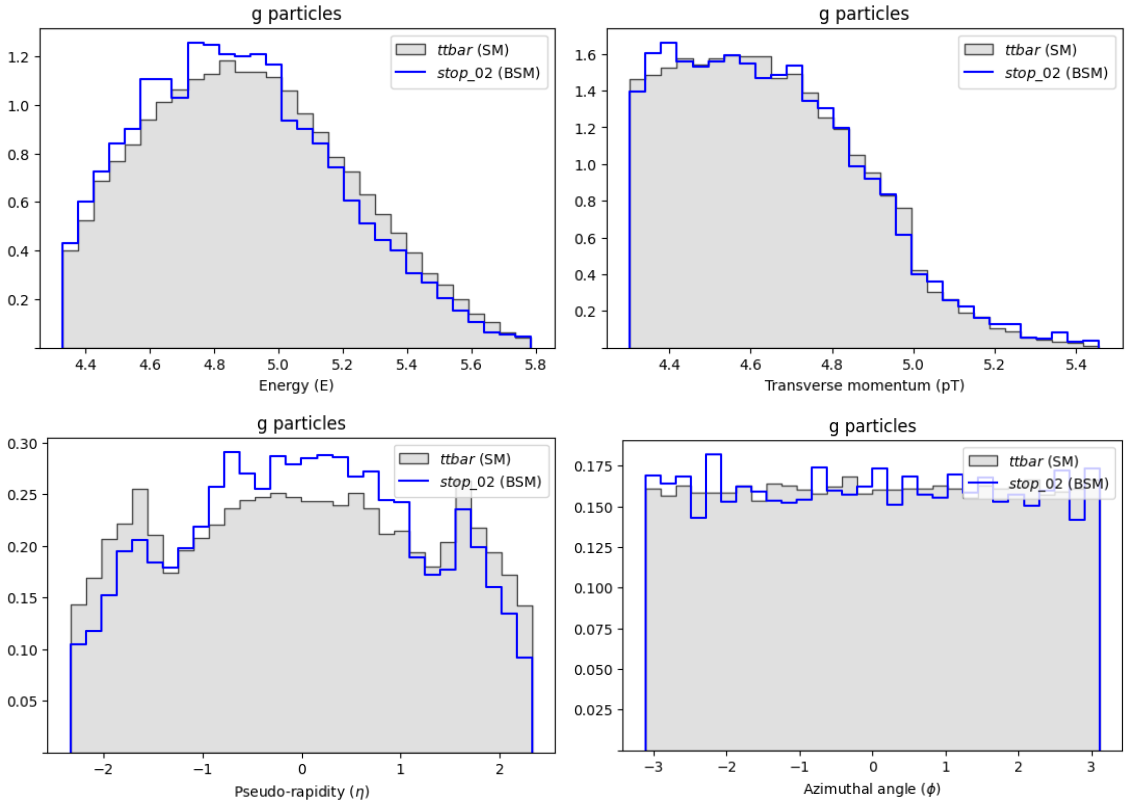


Figure B.8. Histograms of photon (g) parameters in the *ground truth* data.

APPENDIX C
Sustainable Development Goals

In this appendix, we explore the relationship of our work with the Sustainable Development Goals (SDG) established by the United Nations.

Sustainable Development Goals	High	Medium	Low	Unrelated
SDG 1. **No poverty.**				X
SDG 2. **Zero hunger.**				X
SDG 3. **Good health and well-being.**				X
SDG 4. **Quality education.**				X
SDG 5. **Gender equality.**				X
SDG 6. **Clean water and sanitation.**				X
SDG 7. **Affordable and clean energy.**		X		
SDG 8. **Decent work and economic growth.**				X
SDG 9. **Industry, innovation and infrastructure.**			X	
SDG 10. **Reduced inequalities.**				X
SDG 11. **Sustainable cities and communities.**				X
SDG 12. **Responsible consumption and production.**				X
SDG 13. **Climate action.**	X			
SDG 14. **Life below water.**				X
SDG 15. **Life on land.**				X
SDG 16. **Peace, justice and strong institutions.**				X
SDG 17. **Partnerships for the goals.**		X		

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

Dep. de Sistemes Informàtics i Computació
Camí de Vera, s/n, 46022, València
T +34 963 877 350
F +34 963 877 359
depsic@upvnet.upv.es - www.dsic.upv.es

Discussion on the relationship of this work with the selected SDGs

Out of the aforementioned Sustainable Development Goals, this work primarily aligns with four of them.

First and foremost, this work exhibits a significant association with the **Climate action** objective. One of the primary goals of this project was to achieve the fastest method of generating physics events compared to conventional approaches while maintaining a high level of accuracy. By accomplishing this objective, we effectively reduced the energy consumption associated with computational time, as the required computers for event generation would be operational for a shorter duration. This reduction in energy consumption is especially advantageous for future scenarios where a substantial number of events need to be generated, leading to significant energy savings.

Furthermore, this work is also closely linked to the **Partnerships for the goals** objective, as it stems from a collaborative effort with the *Instituto de Física Corpuscular* (*Centro Mixto CSIC-Universitat de València*). The mission of this partnership is to merge the experiments conducted in the field of High Energy Physics with the accurate and efficient data simulation capabilities of Machine Learning. By combining these two domains, the aforementioned objectives can be effectively pursued.

Moreover, this work maintains a significant connection with the **Industry, innovation, and infrastructure** objective by contributing to the potential validation of new physics models. This has the potential to revolutionize certain processes utilized in various industries. The experiments conducted at the ATLAS detector, located at CERN, play a crucial role in achieving a high degree of innovation. Our work assists in developing these experiments more efficiently, facilitating advancements in the field.

Lastly, in the fourth position, this work establishes a connection with the **Affordable and clean energy** objective. Despite the high energy consumption involved, it is noteworthy that CERN primarily sources its electricity from France, where 88% of energy production is carbon-free. Additionally, CERN has committed to reducing its direct greenhouse emissions by 28% by 2024, achieved through the replacement of fluorinated gases with carbon dioxide, which possesses significantly lower global-warming potential[1]. Moreover, our models' significant reduction in energy consumption compared to existing methods contributes significantly to a lower energy consumption, further reinforcing the impact on energy savings.

Hence, we conclude that the project is dedicated to various aspects of the SDGs outlined in the United Nations' 2030 Agenda for Sustainable Development, established in 2015. These goals present a significant opportunity for collective progress towards enhancing quality of life while being mindful of the impact of every project throughout its execution. Consequently, the project embraces this transformative journey, aiming to contribute to sustainable development and improve the well-being of all.

[1] https://hse.cern/environment-report-2019-2020

Dep. de Sistemes Informàtics i Computació
Camí de Vera, s/n, 46022, València
T +34 963 877 350
F +34 963 877 359
depsic@upvnet.upv.es - www.dsic.upv.es