An Open-Source Autoencoder Compression Tool for High Energy Physics

Axel Gallén

Supervised by Caterina Doglioni & Alexander Ekman

Co-supervised by Alma Oručević-Alagić

Thesis submitted for the degree Master of Science (60 hp)

LUND UNIVERSITY

Department of Physics

Division of Particle and Nuclear Physics

September 2022 - June 2023

Abstract

A common problem across scientific fields and industries is data storage. This thesis presents an open-source lossy data compression tool with its foundation in Machine Learning - *Baler*. Baler has been used to compress High Energy Physics (HEP) data, and initial compression tests on Computational Fluid Dynamics (CFD) toy data have been performed. For HEP, a compression ratio of $R = 1.6$ has generated reconstructions that can be deemed sufficiently accurate for physics analysis. In contrast, CFD data compression has successfully yielded sufficient results for a significantly lower compression ratio, $R = 88$. Baler's reconstruction accuracy at different compression ratios has been compared to a lossless compression method, `gzip`, and a lossy compression method, Principal Component Analysis (PCA), with case-wise larger compression ratios over `gzip`; and accuracy at the same compression ratio overall exceeding that of PCA.

Acknowledgments

First of all, a big thank you to Caterina for allowing me to do this Master's project with you and allowing me to travel and present the work at different stages of development! It has exceeded my expectations by miles, and I am ever grateful for everything! A big thank you to Alma as well for the great advice regarding the open sourcing of Baler. I strongly believe that the process of making Baler the tool it is today would have been infinitely harder without your expertise.

Secondly, I could not have done this project without you Alex! Always being available in the corridor or on Zoom has been crucial to all of the work, and you have pushed me to be at my best during the entire project. The traveling has also been a pure pleasure to have done with you, and all meetings with you and Pratik, whom I also would like to thank deeply for all input with regards to ML theory, ideas, and discussions, were always weekly highlights that I will cherish for a long time. Thank you!

Furthermore, I would like to thank the Baler team for a great few months. In my very early science career, this has been a highlight and something which I will bring with me for many years to come, and this work could not have been done without you. I would also like to say a big thanks to everyone in the particle physics corridor in Lund who have made the experience of being there a very pleasant and fun experience.

Last but not least, I want to thank my friends and family sincerely. I can't say thank you enough times to make you understand how much you've meant to me during this project. This work has been done for you.

Contents

1 Introduction

In fields of science that require large datasets, sufficient storage is essential. With many fields in science getting better detectors and better software to collect more data or make better simulations, the datasets have become ever-growing. The projected amount of data to be stored for future data collecting runs at big-data science experiments exceeds the storage assets. For the LHC, by the end of the next decade, the ATLAS experiment [1] alone will have recorded $\approx 5\,\text{EB}$ (5×10^{18} bytes) of data to disk, as seen in Figure 1. This amount of data is about ten times more than what is currently collected. Furthermore, the Square Kilometre Array (SKA) experiment [2] is expected to record 8.5 EB of data over the planned 15-year run-time and scientific fields heavily based on simulations, such as Computational Fluid Dynamics (CFD), rely on several terrabyte sized simulation samples, which needs to be stored. Only some scientific fields have been mentioned here, but numerous industries [3] suffer from the same problem; too much data but too little storage.

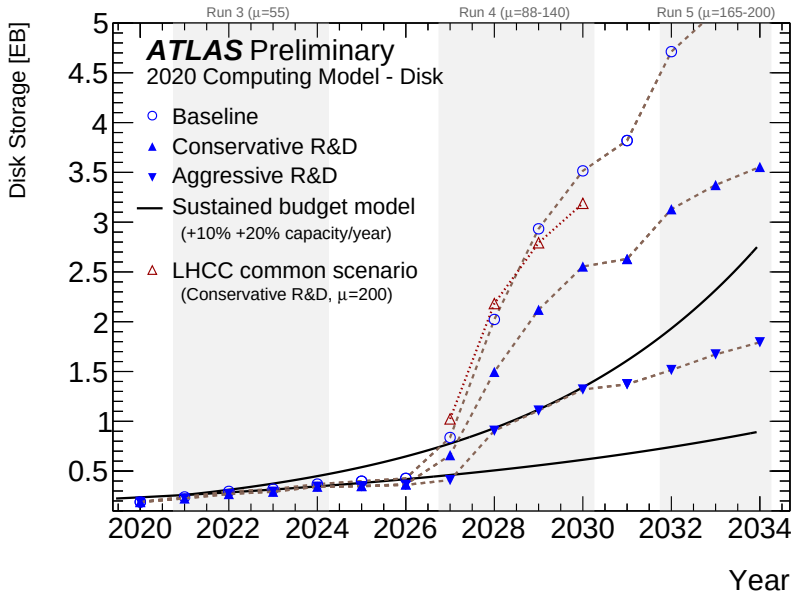


Figure 1: Figure showing a forecast of the ATLAS disk storage necessary in the near future. From Ref.[4].

In particle physics, the need for a compression method to reduce dataset size is undoubtedly a positive with regard to the storage issue, and with this, an added bonus of statistical importance will come. For example, if one manages to reduce the size of a recorded dataset by half, one could theoretically store double the amount of data. As a field, particle physics relies on counting events, and especially rare events, where events essentially are physical processes that are measurable in detectors. The Poisson distribution is a good model for the probability of rare events, and for any Poisson distribution, its mean is equal to its

variance, and it can be shown that the relative Poisson error scales as $1/\sqrt{N}$ [5], where N is the number of events, meaning that from a statistical point of view, minimizing the error demands more events. A large statistical sample is, therefore, vital because some crucial physical events are exceedingly rare and buried in the background, making substantial data necessary to find and analyze such events. Storing and analyzing twice the amount of data would therefore be highly impactful in these cases.

This thesis presents a compression method with a foundation in machine learning. We utilize a specific type of artificial neural network called an autoencoder, capable of reducing dimensionality by exploiting correlations in the data. Furthermore, the compression we are performing is for "offline compression", meaning that a single compression model is used for a particular dataset, and we do not generalize autoencoder models across datasets. Finally, different datasets have been compressed and reconstructed to investigate reconstruction performance during different circumstances, and a fully functional open-source compression tool, Baler, is presented.

2 Theory and Experimental Background

The necessary theoretical physics knowledge and the experimental background will be briefly introduced, conceptually covering the Standard Model with its different particles and forces. Further, the objects used in our data sample, hadronic jets, will be summarized together with their properties. Following that, two detectors at the Large Hadron Collider (LHC) capable of observing jets will be described. Finally, the different types of data compression are outlined.

2.1 The Standard Model

In the 1970s the Standard Model (SM) was developed, and its predictions have been confirmed repeatedly, with the latest being the discovery of the Higgs Boson in 2012 [6, 7]. All contents of the Standard Model known to date are presented in Figure 3.

The particles in the Standard Model can be split into two groups. These two groups separate the Standard Model particles based on their spin value. If a particle possesses a half-integer spin values, $\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \ldots$, it follows the so-called Fermi-Dirac statistics [8], and is therefore called a *fermion*. On the other hand, if a particle has an integer spin value, $0, 1, 2, \ldots$, it follows Bose-Einstein statistics [8] and is called a *boson*.

The fermions include particles known as *quarks* and *leptons*, with the difference being the force they interact with. Leptons interact via the electromagnetic (EM) and the weak force, mediated by the photon (γ) and $W^{\pm}/Z^0$ bosons, respectively, while the quarks interact via the EM, weak and strong force, mediated by the gluon (g). The mediator particles are known as *gauge bosons* and mediate three of the four fundamental forces. The final boson

is the Higgs boson and is responsible for most particles being massive, with that being a consequence of the Higgs mechanism [9].

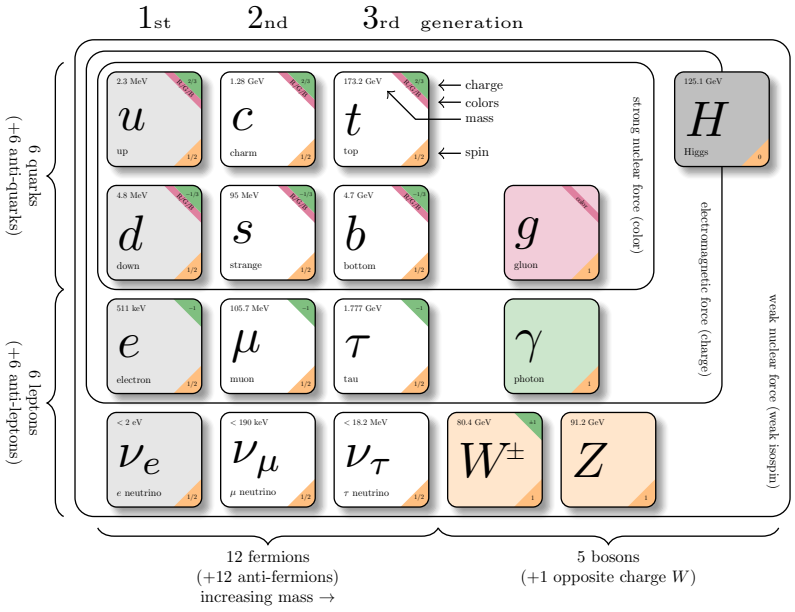


Figure 3: The contents of the Standard Model which contain the elementary particles, force carriers, and corresponding forces. Modified from Ref. [10].

The quarks interact via the strong force because they carry color charge, fundamentally described by Quantum Chromodynamics (QCD). If a fermion has a color, it can interact with a gluon and interact via the strong force. That is, in the simplest sense, the difference between a quark and a lepton, but there is one further difference, namely color confinement. Color confinement says that quarks can only exist in groups with neutral or white combined color charge. "Whiteness" can be obtained via color-doublets: color + anti-color; or via color-triplets: blue + red + green (or anti-blue + anti-red + anti-green). Depending on if the quark composition is a doublet or triplet, the composite particles are named *mesons* or *baryons*, respectively. A common name for these composite particles is *hadrons*. Figure 2 shows two illustrations: one of a baryon, the proton, and one of a meson, the pion.

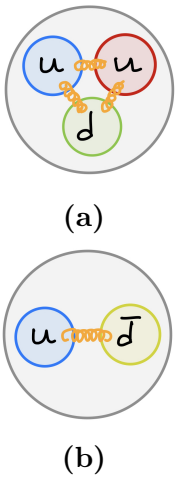


Figure 2: An illustration of (a) a proton, and (b) a pion.

Color confinement together with *asymptotic freedom* is critical in collider physics, as it allows for the process of *hadronization*, and with that, the creation of *hadronic jets*, explained further in Section 2.2. Asymptotic freedom is connected to the concept of how "hard" force mediators are connected to the particles. For QCD and gluons, this turns out to differ depending on how far apart the quarks are. If the quarks are spatially close, the force between them becomes negligible, and equivalently, the force becomes asymptotically larger as they move apart. This discussion is continued in the following section.

2.2 Jets

Most proton-proton collisions at colliders like the LHC create an abundance of quarks and gluons. As discussed previously, they carry color charge, meaning they can not exist separately due to color confinement. Given enough energy, the quarks and gluons created in proton-proton collisions will start to move apart, as seen in Figure 4, since $V(r) \propto r$ for color-fields [11]. At some point, this energy will overcome the threshold needed to create a quark-antiquark pair, creating two new hadrons instead of pulling the quarks further apart. The process of creating new hadrons is the hadronization process and, at its core, can be analogous to trying to pull apart a rubber band. If the quarks are the end of a stretched rubber band, it will eventually snap, creating two bands.

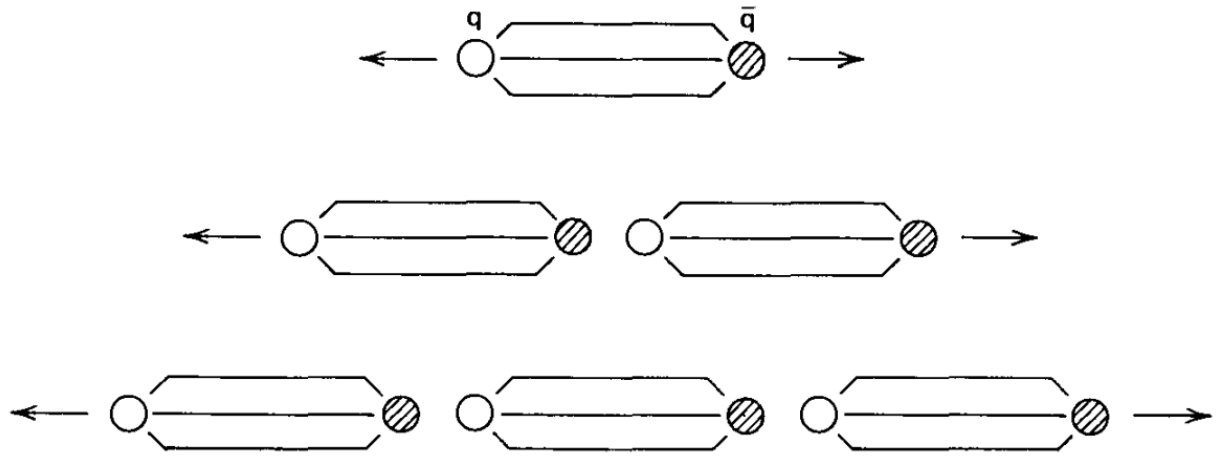


Figure 4: Illustration of how jets are created. Ref [11].

Since the LHC collisions occur with extreme energy, hadronization can occur several times in each collision and even in succession. With heavy particles decaying into lighter particles quite quickly on average, hadronization plus decay gives rise to particle showers known as a *hadronic jets*.

To be more precise, the jets observed at LHC collisions are final-state jets. These are the final stage of the hadronization process, and they reach and interact with the detector. For

analysis purposes, the jet detected is considered a single object, and the jet is then used as an intermediate step in analyzing the initial state of the collision.

2.2.1 Jet Kinematics

To analyze the jet as a single entity, all the different detector signals, or individual particles, creating the jet must be reconstructed. A reconstruction algorithm called the *anti-k_t algorithm* [12] does this reconstruction and is the main jet reconstruction algorithm used in ATLAS. This algorithm is a so-called inclusive jet finding algorithm that uses distances to group objects into jets, with the distances being defined as:

$$d_{ij} = \min(k_{ti}^{2p}, k_{tj}^{2p}) \frac{\Delta_{ij}^2}{R^2} \tag{2.1}$$

$$d_{iB} = k_{ti}^{2p}. \tag{2.2}$$

In these two equations, d_{ij} is the distance between two entities (either a particle or a pseudojet, meaning that it is not a particle or a jet) i and j, while d_{iB} is the distance between an entity i and the beam B. We also have k_{ti} which is the transverse momentum, and $\Delta_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2$ where y_i is the rapidity and ϕ_i is the azimuthal angle. R^2 is a common radius parameter defined as $R^2 = \eta^2 + \phi^2$, with η being the pseudorapidity defined as

$$\eta = -\ln\left(\tan\frac{\theta}{2}\right), \tag{2.3}$$

where θ is the polar angle and is connected to the rapidity, as discussed below. Finally, p is a parameter that "governs the relative power of the energy versus geometrical (Δ_{ij}) scales" - [12].

For any particle, four variables are needed to define the kinematics of the particle. These four variables are known as the *four-momentum*, and this quantity is also perfectly capable of describing the kinematics of a jet. This quantity incorporates the energy and momentum of the jet in two possible ways. Using natural units, $\hbar = c = 1$, they can be written either using energy and momentum components as

$$p^\mu = (E, p_x, p_y, p_z) = (E, \mathbf{p}), \tag{2.4}$$

or equivalently,

$$p^\mu = (p_T, \eta, \phi, m), \tag{2.5}$$

although now using the jet's mass and the momentum in the transverse plane together with the pseudorapidity and azimuthal angle. Pseudorapidity is a common convention due to $\eta \to \infty$ as $\theta \to 0$, and $\eta \to 0$ as $\theta \to \pi/2$. This means particles moving perpendicularly to the beamline will have a pseudorapidity of zero, while particles moving parallel to the beam axis will have a non-zero pseudorapidity.

Furthermore, in the ultra-relativistic limit, $|\mathbf{p}| \gg m \implies p_T \gg m$, an expression between mass, transverse momentum, and polar angle is

$$E = m_T \cosh y, \tag{2.6}$$

where $m_T = \sqrt{p_T^2 + m^2}$ and y is the *rapidity*. In terms of pseudorapidity, one can write the rapidity as [13]

$$y = \ln\left(\frac{\sqrt{m^2 + p_T^2 \cosh^2\eta} + p_T \sinh\eta}{\sqrt{m^2 + p_T^2}}\right), \tag{2.7}$$

and in order to convert Eq. 2.6 into an equation dependent on η instead of y, we do a second-order Taylor expansion of Eq. 2.7 in m/p_T around 0 to find

$$y \approx \eta - \frac{\cos\theta}{2}\left(\frac{m}{p_T}\right)^2. \tag{2.8}$$

Recalling that we are in the ultra-relativistic limit we get $y \approx \eta$, which implies that Eq. 2.6 becomes

$$E = \cosh\eta \cdot \sqrt{(p_T^2 + m^2)}. \tag{2.9}$$

Another important quantity is the *invariant mass*, defined for jets as

$$M_{JJ}^2 = \left(p_{J_1} + p_{J_2}\right)^2 \underset{E>>m}{=} 2\, p_{T,J_1} p_{T,J_2}(\cosh\Delta\eta - \cos\Delta\phi), \tag{2.10}$$

with J_1 referring to the leading jet, meaning the highest p_T jet, and J_2 to the sub-leading jet, i.e., the second highest p_T jet. Equivalently, for particles, one can write the invariant mass as

$$M^2 = E^2 - \mathbf{p}^2 = p^\mu p_\mu. \tag{2.11}$$

This quantity has an additional significant and valuable feature, which is that it is Lorentz invariant. A Lorentz invariant quantity means that this quantity is constant under Lorentz transformations and thereby equal in all reference frames.

When looking for unknown high-mass particles, *resonance particles* are often in the scope of search. For example, to search for resonances decaying into quarks, looking for events originating from quarks or gluons is an excellent place to start. At a hadron collider, resonances are very short-lived particles, like heavy bosons, frequently searched for via resonance searches or *bump hunts*. These searches use the invariant mass of the jet, see Eq. 2.10, since it resembles the mass of the decaying jet origin. This origin could be a particle produced via a resonance, hinting toward new physics.

Due to the smoothly falling invariant mass distribution of particle physics processes, most noticeably for QCD which make up the vast majority of all background events, if an excess is present, a bump would be visible in the invariant mass distribution. This lies at the very core of resonance searches. For example, the Higgs boson was discovered via a bump hunt in 2012, with the discovery plot shown in Figure 5.

When looking for resonances, it is worth noting that the width (resolution) of the bump will heavily depend on the significance, which depend on the number of events. As an example, again looking at Figure 5, this search looked for a Higgs decaying into a diphoton final state, $H \to \gamma\gamma$ with an intermediate W/top loop, and in practice, one counted the number of diphoton processes in different mass ranges and produced a histogram. If any diphoton resonance exists in these mass intervals, more diphotons would have been produced, and one would see a bump. Again, this method relies heavily on significance, and thus event numbers, so more events directly correlate to better resolutions.

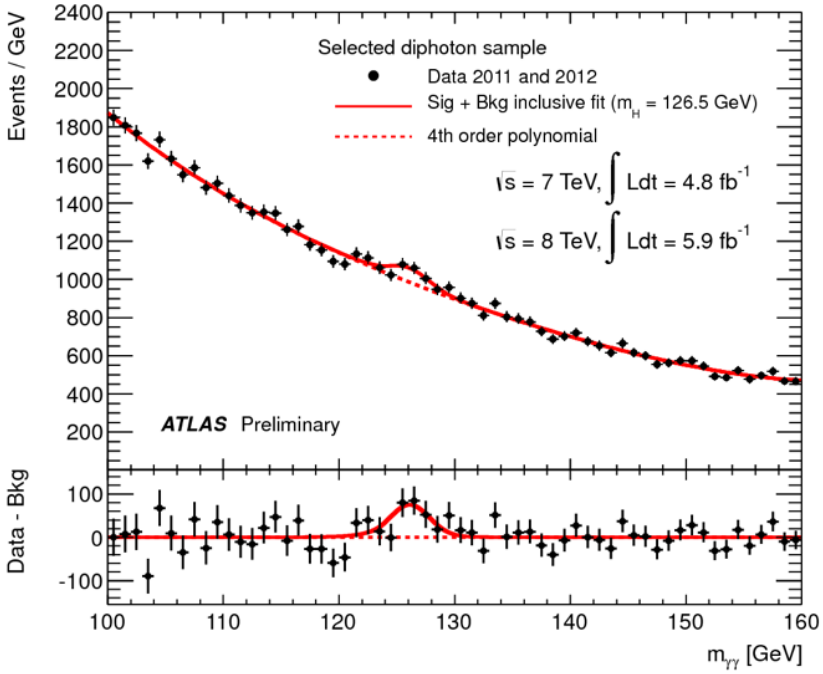


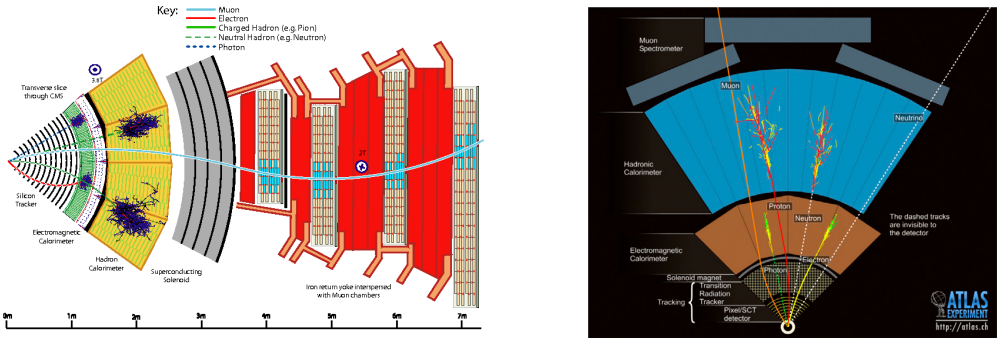
Figure 5: The invariant mass distribution from photonic pairs in the Higgs to di-photon analysis. The excess of events above the predicted background around 125 GeV is consistent with Standard Model predictions of the Higgs Boson [14].

2.2.2 General Purpose Detectors & the LHC

Enough energy to hadronize a quark or a gluon must be available to observe a jet. These energies can be obtained in high-energy collisions; the most prominent place to date, which is capable of high-energy collisions, is the LHC. Hosted by the European Organization for Nuclear Physics (CERN) on the border between Switzerland and France, the LHC is a two-ring circular collider with a total circumference of 27 km underground. It can accelerate protons or heavy ions to speeds extremely close to the speed of light and then collides them at four separate interaction points with a center-of-mass energy of 13 TeV. At each interaction point, a particle detector is located. These detectors are: *A Large Ion Collider Experiment*- (ALICE) [15], *A Toroidal LHC ApparatuS*- (ATLAS) [1], *Compact Muon*

Solenoid- (CMS) [16] and *Large Hadron Collider beauty* (LHCb) [17] experiments. The CMS and ATLAS experiments can be classified as general-purpose detectors and are the two most focused on observing jets. These two will now be explained briefly.

CMS and ATLAS have a structural layout consisting of layers and can be described as having an *Onion Layout*. Their components are configured cylindrically around the beamline, with the interaction point located in the detector's geometrical center. The innermost layer of both detectors is made of tracker systems, capable of identifying electrically charged particles for measurements of momenta, identification of charged particles, and observing the origin of jets, e.g., a jet coming from one specific kind of quark, and thereby tagging it. One layer outwards, we find the calorimeters in which close to all particles deposit all their energy and momentum, and on the outside, there are structures capable of detecting muons. The muons need their own detector modules due to their large mass and momenta, which cause them not to deposit significant energy in the calorimeters. Figure 6 figuratively shows the two layouts and how different particles behave in the different layers.



(a) **(b)**

Figure 6: **(a)**: Illustration of a slice of the CMS detector with all its layers and what happens to different particles in the detector. **(b)** Similar illustration, but now for the ATLAS detector. Figures obtained from [18] and [19] respectively.

At the LHC, collisions between bunches of protons occur at a rate of about 40 MHz, where each bunch contains numerous proton-proton interaction events, making it virtually impossible to process and store all events. Instead, determining which events are important enough to process and keep is taken care of by the ATLAS trigger system [20] and data acquisition system [21]. The ATLAS trigger system consists of two parts: a hardware part, the level-1 (L1) trigger [22], and a software part, the high-level trigger (HLT) [23].

The L1 trigger brings the initial event rate of 40 MHz down to 75 kHz by checking if each event fulfills a set of criteria. These criteria could for example be if the p_T exceeds a certain amount or if an event has enough electrons or photons to be considered compelling.

The HLT, on the other hand, uses the same methods as are to be used in the analysis to make more complex decisions regarding which events are needed for specific types of

analyses. This process decreases the event rate to $\mathcal{O}(100)$Hz.

After the HLT has determined what events are viable for analysis, these events are stored long-term and are the main events used for analysis. With an event taking up approximately 1 MB [21], this still results in an extraordinary amount of data being stored long-term, making it very viable to investigate data compression methods.

When examining what or where to apply compression methods, the first place which comes to mind as an application area is in offline storage. As reported in [24], during the latter stages of Run 2, approximately 150 PB of the 223 PB offline storage at ATLAS was filled with Analysis Object Data (AOD) or AOD derivations (DAOD). These are the primary data objects stored in `.ROOT` files, a file type mainly developed for the ROOT Data Analysis Framework [25].

Due to this, only one or two replicas can be stored because of the large sample size, but introducing compression here could lead to more replicas being able to be stored. From an economic point of view, being able to store more data in less space also reduces the cost of needing to purchase more storage. It is understood that before introducing any type of compression that changes the accuracy of the data, in-depth studies need to be undertaken so that the data can still be used for precision measurements and searches for new physics.

2.3 Data Compression

Data compression is, in short, the process of encoding information to reduce its size by using fewer bits than the original representation of the information. There are two distinct types of compression, namely *lossless* and *lossy* compression, where both will be described in detail below. First, however, how data is stored in the first place needs to be understood to explain the difference.

In computer memory, the most fundamental unit is the *bit*. A bit is a binary number with the value 0 or 1. Eight bits in sequence are called a *byte*, and in reality, this is what scientific data is; a sequence of binary numbers. There are, however, more efficient ways to store information in memory, with one of the most efficient ways to represent memory called *hexadecimal*. In this form, each digit represents a value between 0 and 16, with the conversion between *decimal*, hexadecimal and binary being shown in Table 1.

Table 1: Conversion between hexadecimal, decimal, and binary.

Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1011	1100	1101	1110	1111
Decimal	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f

Comparing hexadecimal and binary notation, one hexadecimal digit corresponds to 4 bits. Therefore, each byte of memory only requires two hexadecimal digits, compared to eight in binary.

Regarding scientific data, most data points stored are either *floating points* or *integers*. A floating point, or float, is simply a non-integer value, commonly stored as $m \cdot 2^e$. Here, m is the mantissa and denotes the number of significant digits of the float, while e is the exponent[1] [26]. Another bit is assigned, called the sign bit, which dictates the sign of the number ($\pm$). Generally speaking, there are two types of floating points used; 32-bit floating points (singles) and 64-bit floating points (doubles), which are stored in the following way

	Sign	Exponent (e)	Mantissa (m)
Float-32	31st bit	30th-23rd bit	22nd - 0th bit
Float-64	63rd bit	62nd - 52nd bit	51st - 0th bit

The exponent's job is to dictate the range of possible values. For doubles, eleven bits allows for a range of integers in the range [-127,127], and doing the computation: $2^{-127} \approx 6 \times 10^{-39}$ and $2^{127} \approx 2 \times 10^{38}$, meaning that the smallest possible value to store in a double is of the order of 10^{-39} and the largest is of the order of 10^{38}.

2.3.1 Lossless compression

In its essence, lossless compression is a fully reversible process. One can safely encode something, and upon decoding, there is no loss in information or quality. Lossless compression does this by, e.g., removing redundant data, such as duplicated numbers or bits. One prevalent lossless compression method for images is `PNG`, which uses run-length encoding to re-write the image information so that every bit value representing the same thing appears fewer times than in the original. For example, say one has a figure with a white background and a hollow red circle in the middle. Then, all one needs are one white "bit value" and one red "bit value" together with where they should be located, and you can successfully store the image at a reduced size. Another standard more general lossless compression method is `gzip`[2] [27], which also utilizes run-length encoding. This method will serve as a comparison later on.

2.3.2 Lossy compression

On the other hand, lossy compression is a process that is not fully reversible. As a result, one can only reconstruct an approximation of the original data, resulting in a quality loss. The lossy counterpart to `PNG` is `JPEG`. `JPEG` is a lossy compression method that employs size reduction by reducing brightness information for each pixel and averaging color information between neighboring pixels.

[1] Not to be confused with Euler's number.

[2] There is a difference between `gzip` and `zip`. The latter allows for the extraction of single compressed files from a `.zip` archive, while `gzip` demands the entire archive to be decompressed for the extraction of files.

Another feature worth mentioning when it comes to lossy compression is *generation loss*. Generation loss occurs after the chaining of lossy compressions on a file, making the file quality degrade with every generation, although the file size will shrink. Generation loss is also a counterpart to the lossless format, where the quality will not decrease, but the file size does not reduce further. It also points to a difference between lossy and lossless compression. Lossless compression is not useful if the files one wants to compress already are compressed, while lossy compression can still be used, although with reduced effectiveness.

2.3.3 Principal Component Analysis

Mathematically, there are ways to reduce the size of objects or datasets, with one of these ways being *dimensionality reduction*. Dimensionality reduction is a concept where one can, by mathematical means, find ways to represent a $n-$dimensional dataset as a $m-$dimensional object, with $\dim m < \dim n$. One of the most prominent ways to do this is via *Principal Component Analysis* (PCA). This mathematical analysis process identifies patterns in data and expresses these patterns to highlight differences and similarities. As described in Ref. [28], there are five steps needed to perform PCA for a given dataset:

1. For every data dimension, subtract the mean: $\overline{x} = x - \mu_x$
2. Compute the covariance matrix between all dimensions:

   $$C^{n\times n} = cov(\dim i, \dim j)$$

   where

   $$cov(X,Y) = \frac{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right)}{n-1}.$$

   Here, $C^{n\times n}$ is a square $n \times n$ matrix[3] and $\dim k$ is the kth dimension.
3. Find the eigenvalues and unit eigenvectors of the covariance matrix, by first finding the eigenvalues λ: $\det\{C - \lambda\mathbb{I}\} = 0$, and then find the eigenvectors, χ, by solving $(C - \lambda\mathbb{I}x) = 0 \;\forall\, \lambda$.
4. Choose components and form a feature vector, F, from the eigenvectors: $F = (\chi_1, \chi_2, \ldots, \chi_n)$. From this feature vector, shorten its length to p, where p will be the dimensionality of your dimensionality reduced dataset[4].
5. Create your new dataset, Z, by computing: $Z = F^T\overline{x}$.

[3] If your matrix isn't square, your matrix does not have eigenvalues. There are ways to move around this [29]. For this thesis, the implementation can be seen in Section 4.

[4] The choice of eigenvector to stop at matters. The largest eigenvalue is called the *principal component*, and this is important because it tells you the most significant relationship between data dimensions.

To revert this process, we compute $x = F^T Z + \mu_x$ with a more detailed explanation in Ref. [28].

Due to the dimensionality reduction involved in PCA, it is common to use PCA as a compression method for data [30]. Fundamentally, PCA would, in that case, be classified as a lossy compression method because approximations take place and you reduce the dimensionality via data features. PCA will be used as a comparison later on.

3 Artificial Neural Networks & Deep Learning

In recent times, the concept of machine learning and artificial intelligence has become well-known in most research fields, with physics being at the very forefront of those fields. Some of the main tools used under the large umbrella of machine learning are Artificial Neural Networks (ANNs). ANNs have become very popular in classification areas, such as image classification and language processing. The origin of the ANN can loosely be connected to the concept of a brain, where both consist of several neurons, called computational nodes in the ANN, which exchange information via the output of one node, becoming the input of another.

A commonly used family of ANNs is the perceptron, which consists of one computational node in the simplest case. Initially, the perceptron referred to the brain's memory storage [31] and a probability-based computational process. However, it has become a common name for many objects connected to probabilities and transformations.
Figure 7 shows that the perceptron input is an n-dimensional vector. It then uses weights w_i, $i = 1, \ldots, n$, to compute a weighted sum, which is then fed as input to a non-linear activation function, $f(x)$. The activation function is chosen to suit specific tasks. This can be to output a scalar quantity y, which is the perceptron's final output in this case. Mathematically, this can be written as

$$y = f\left(\sum_i^n w_i x_i\right). \tag{3.12}$$

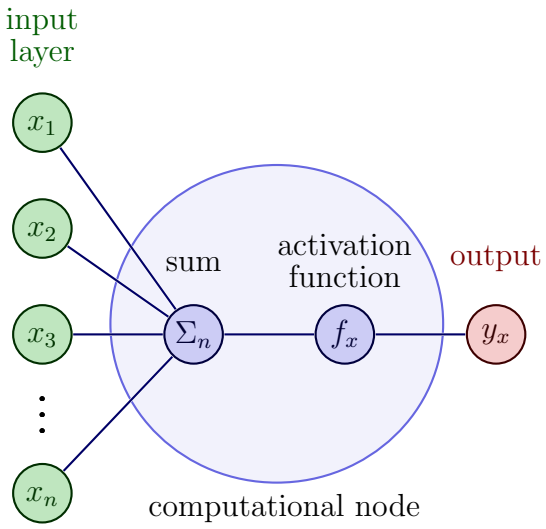


Figure 7: Example of a single computational node taking an $n-$dimensional vector as input and outputs a scalar. Modified from Ref [32].

3.1 Dense Neural Networks

However, a single-layer neural network cannot solve complex tasks like image classification. A potential solution is to increase model complexity by adding more nodes. Nodes, when stacked in parallel, then create so-called *hidden layers*, and if an ANN consists of more than one hidden layer, and the output of each node is an input of each node in the next hidden layer, it is deemed to be a *dense neural network* (DNN). Deep learning comes into the picture when one uses a DNN to perform machine learning. From here, we will only deal with DNNs, so ANNs and DNNs will be used interchangeably.

A more complicated ANN can look something like the one pictured in Fig 8, where the so-called *architecture* is described as $n-m-l-i-k$, with n and k representing the number of nodes in the input and output layers respectively, and m, l, i showing the number of nodes in the each hidden layer. An example of what will be used later on is a ANN with architecture

$$x - 200 - 100 - 50 - z - 50 - 100 - 200 - x,$$

with $x < 200$ and $z < 50$ (see Section 4.2).

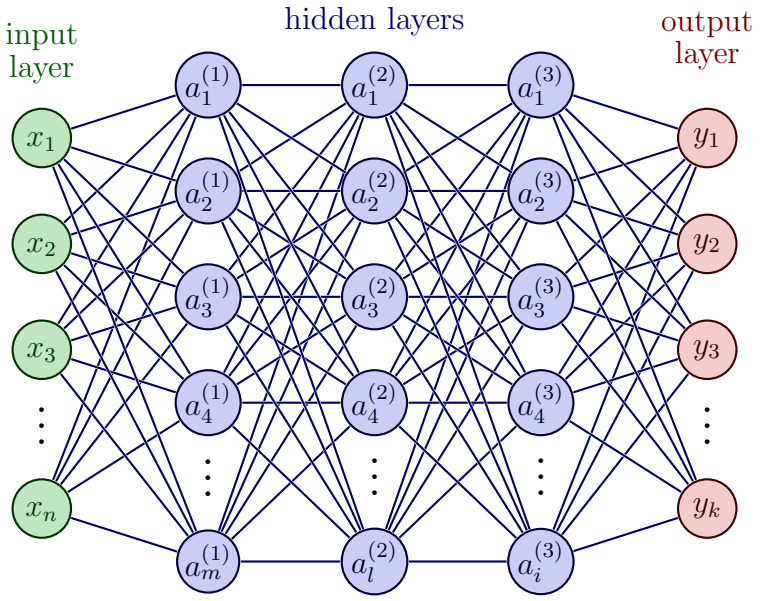


Figure 8: A dense neural network with one input layer of length n, three hidden layers of length m, l, i respectively, and an output layer of length k. a refers to the computational nodes, x represents the input and y the output. Modified from Ref. [32].

Now that the structure of a more complicated ANN has been described, the need for a non-linear activation function is easier to motivate. For a DNN, every node will perform a linear transformation on its inputs using corresponding weights and biases. Since the

composite of two linear functions is a linear function, what will be done between every layer is essentially a linear regression. For the network to learn more complex tasks than linear regression, non-linearity needs to be introduced in between layers to add complexity to the network.

Some of the most common activation functions to add non-linearity to an artificial neural network are

$$\text{ReLU:} \quad f(x) = \max(0, x) \tag{3.13}$$

$$\text{Leaky ReLU:} \quad f(x) = \max(ax, x), \ a < 0 \tag{3.14}$$

$$\text{Sigmoid:} \quad f(x) = \left(1 + e^{-x}\right)^{-1} \tag{3.15}$$

$$\tanh: \quad f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.16}$$

where the use case of the activation functions differs depending on what the end goal of the network is. For example, if the goal is to do binary classification, having an output in the range $[0, 1]$ is efficient, so the output can be interpreted as a probability. One of the functions which does this well is the Sigmoid function (Eq. 3.15), which takes input in the range $(-\infty, \infty)$ and outputs in $[0, 1]$.

However, a case where the Sigmoid activation function is not the best is where one has a DNN. There are some reasons for this, with the most obvious one being that, from a computational point of view, it is cumbersome to compute exponential functions and inverses. Secondly, as described in [33], a phenomenon known as the *vanishing gradient problem* can occur (also applicable to the hyperbolic tangent activation function, Eq. 3.16). To briefly explain this phenomenon the concept of training a neural network is important, and this will be discussed in more detail in Section 3.2. DNNs are trained by updating gradients of some loss function with respect to weights and do this in one layer at a time, starting from the back, thereby performing *backpropagation*. The vanishing gradient problem is caused by exceedingly small gradients for inputs larger than about 10, and during backpropagation, they will continue to decrease the further they move backward. Consequently, the nodes at the beginning of the network will learn slower than those in the latter parts, resulting in more significant training speeds and worse overall performance.

Two things stand out when talking about the two Rectified Linear Unit (ReLU) functions compared to the sigmoid and hyperbolic tangent. First, there is no need to compute exponential functions or inverses, making it less computationally heavy. Secondly, $\forall\, x \geq 0$, we have $f(x) = x$, so the possibility of a vanishing gradient is negligible.

Giving Eq. 3.13 & 3.14 and Fig. 9 a glance, Leaky ReLU, and ReLU are very similar. The only difference between the two is an arbitrary factor greater than zero. Although minimal, this factor solves a grave issue with the ReLU function, namely the *dying ReLU problem* [34]. If the node's input is purely negative, the output of the node will be zero due to the hard limit. If all outputs are zero, one can see the node as "dead" because there is no information to pass from the dead node to the next "alive" node. Instead, suppose

one does not have a hard limit at zero, like in the Leaky ReLU function. In that case, this issue is avoided because a small negative value is now passed through via backpropagation instead of a zero, and the possibility of dead nodes becomes inconsequential.

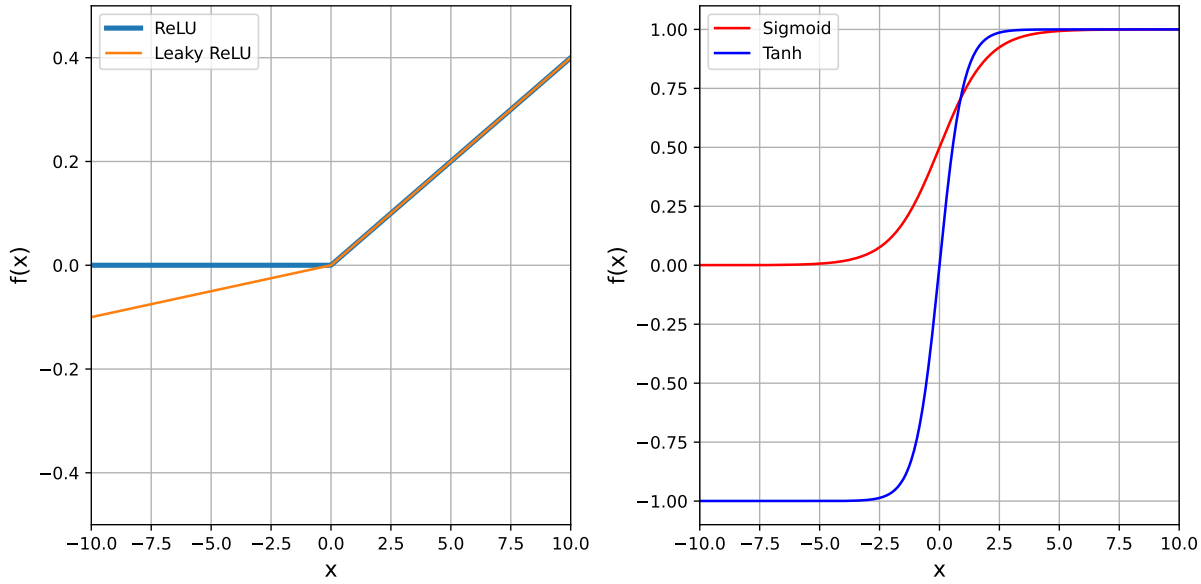


Figure 9: (Right) A plot of the Sigmoid activation function and the Tanh activation function. (Left) A plot of the ReLU and Leaky ReLU activation functions.

3.2 Training & Loss

With the conceptual understanding of an ANN out of the way, one can now start to talk about how the networks are used and how their performance is measured. The process we are talking about is called *training* and can be done by comparing Y, the values the model updates during training, to $\hat{Y}$, being the models' input values. A common performance measure of this comparison is the *mean squared error* (MSE) [35], defined as

$$\text{MSE} = \frac{1}{n}\frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)_j^2, \tag{3.17}$$

where m is the batch size and n is the number of samples one tests against. The concept of batch size will be defined in Section 3.3. These types of equations are commonly known as *loss functions*, *error functions*, or *cost functions*, where they are continuously evaluated in the network. Intuitively, obtaining a loss of zero means that $\hat{Y}_i = Y_i$, and this is what the network is trying to do, thus fundamentally making training the ANN and optimization problem. By evaluating this quantity continuously, the weights are somewhat shifted to minimize the loss quantity.

Although the most common, the MSE loss function is one of many. Another common one is the *earths movers distance* (EMD), or *Wasserstein distance* [36], defined as

$$\text{EMD} = \inf_{\pi \in \Gamma(u,v)} \int_{\mathbb{R}\times\mathbb{R}} |x - y| \, \mathrm{d}\pi(x, y). \tag{3.18}$$

Citing the author of [36], Cédric Villani, one can think of this quantity as:

> Assume that you are in charge of the transport of goods between producers and consumers, whose respective spatial distributions are modeled by probability measures. The farther producers and consumers are from each other, the more difficult will be your job, and you would like to summarize the degree of difficulty with just one quantity.

Although conceptually simple, Eq. 3.18 can be seen as quite complex. In practice, one takes the cumulative sum of the difference between the distributions, yielding a set of cumulative sum values, each in $(-\infty, \infty)$. Since we are looking for a "distance," the way to get this into an objective value is to take the sum of the absolute value of this set. For this thesis, Eq. 3.18 has been implemented via the `SciPy.stats` [37] python package with the function: `scipy.stats.wasserstein_distance`.

Another definition to make is that of training length. Defining how long one should train the model is most commonly defined as how many weight updates will be computed or even how many times the network is allowed to see the data. The name for this is an *epoch* and is defined as one forward and backward pass of the entire training set through the model. How this definition of an epoch is utilized and evaluated is explained below.

3.3 Machine Learning & Optimization Algorithms

Regardless of the choice of loss function, updating weights in the networks is most commonly done via the same method, namely *gradient descent* [38]. This algorithm computes the gradient of the loss function with respect to the model weights as following

$$\frac{\partial \mathcal{L}}{\partial w_i} = \sum_n \frac{\partial \mathcal{L}}{\partial y_n} \frac{\partial y_n}{\partial a_n} \frac{\partial a_n}{\partial w_i} = \sum_n \frac{\partial \mathcal{L}}{\partial y_n} f'(a_n) x_{ni}. \tag{3.19}$$

This computation is performed because we have an optimization problem that we want to solve: We want to find a set of weights to minimize the loss. The fact that the derivatives are computed constantly also puts an essential mathematical constraint on our loss and activation functions, which is that they are C^1-smooth. This is important because it ensures that the gradient can be computed anywhere and thereby update the weights no matter what.

To optimize the weights such that the loss minimizes, a common strategy is to start with all weights randomly initialized and then move in the negative gradient direction by taking small steps,

$$\Delta \mathbf{w} = -\gamma \cdot \nabla_{\mathbf{w}} \mathcal{L}, \tag{3.20}$$

and then updating the weights accordingly,

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \Delta \mathbf{w} = \mathbf{w}_i - \gamma \cdot \nabla_{\mathbf{w}} \mathcal{L}. \tag{3.21}$$

Here, γ is a positive parameter called the *learning rate*. The learning rate is an example of a *hyperparameter*, a parameter defined outside of training that is not learned during training and is implemented according to Algorithm 1. The negative sign in Equation 3.20 is there because we want to move in the opposite direction of the directional derivative, i.e., towards the minimum.

Algorithm 1 Gradient descent learning. Ref [39].

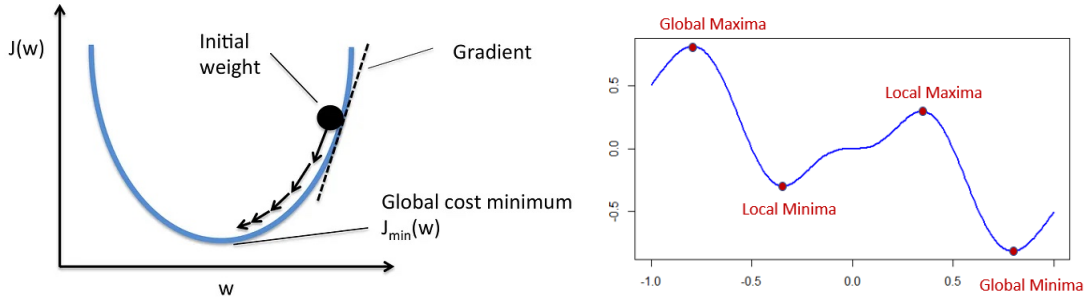
Require: w_i: Initialize the weights with small random numbers
Require: γ: Learning rate
Require: $\mathcal{L}$: Loss function
Require: $\mathbf{x}_n$: Input values
Require: $f()$: Activation function
while w_i not converged **do**
    $i \leftarrow i + 1$
    $y_n \leftarrow y(\mathbf{x}_n)$ ▷ For each pattern n compute the output
    $\delta_n \leftarrow -\partial \mathcal{L} / \partial a_n = -(\partial \mathcal{L} / \partial y_n) f_0'(a_n)$ ▷ Compute the derivative
    $w_i \leftarrow w_{i-1} + \gamma \sum_n \delta_n x_{ni}$ ▷ Update weights accordingly
end while
return w

An illustration of this process can be observed in Figure 10a. However, this figure is quite deceiving because it is a highly trivial case. In reality, the "loss space" is very complicated, as there are as many weights as connections between nodes. Therefore, the gradient of the cost function is non-linear due to the non-linear connections between the nodes and the cost function itself. What can be an issue, though, is the randomization factor of the initialization of the weights. For example, looking at Figure 10b, let us say that the randomized weight initialization leads to the first gradient computation taking place at the slope between the global maximum and the local minimum. There is a large possibility that the gradient descent algorithm converges towards the local minimum and then is done with the training.

In some cases, this is acceptable, and one can argue that this is optimal[5], but if one lands slightly left of the local maxima, then the gradient descent algorithm will converge towards

[5]Since dimensionality of the loss landscapes is directly proportional to the number of weights, the dimensions can quickly enter the millions. Finding the global minimum is, therefore, nothing one expects, so a local minimum is usually more than enough

the saddle point because the gradient in a minimum and at a saddle point are the same; zero. In this way, there is still quite a lot left to be learned from the network, and one has not reached a point where the loss is minimized, but a point where the network thinks that the loss has been minimized and will not, therefore, update the weights anymore.



(a) **(b)**

Figure 10: **(a)**: An illustration of the problem gradient descent is trying to solve. The initial weight is randomized, and the steps towards the bottom are negative gradient steps, together with the learning rate. The learning rate tells one how large steps to take in the direction of the negative gradient. Reaching the "global cost minimum" would mean a successfully optimized cost with respect to the models' weights, $J(w)$. Ref. [40] **(b)** a slightly more complicated landscape with several minima and saddle points [41].

This problem, among others, is solved by the most common optimization algorithm in deep learning, namely the *Adaptive moment estimation* (Adam)[42]. This optimizer is described in detail in Algorithm 2, but very shortly, the main difference between gradient descent and Adam is that Adam keeps track of the decaying averages of the gradient as well as the squared gradients,

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where $\beta_{1,2}$ are two decay factors separate for the two averages (see Algorithm 2 for the variable definitions). More specifically, m_t is an estimator of the mean (first moment) of the gradient g_t, and v_t (second moment) is an estimator of the variance of g_t.

The Adam optimizer is used throughout this thesis, and several reasons exist for why this is the case. The main reason is that, as stated previously, it is one of the most used in deep learning, and it has also proved to be very effective in previous studies [43, 44].

Algorithm 2 The Adam Optimizer [42] algorithm for stochastic optimization.

Require: γ: Learning Rate
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moments estimates. Suggested defaults: 0.9 and 0.999 respectively.
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector

$m_0 \leftarrow 0$ ▷ Initialize 1^{st} moment vector
$v_0 \leftarrow 0$ ▷ Initialize 2^{st} moment vector
$t \leftarrow 0$ ▷ Initialize time step
while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Compute gradients
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ ▷ Keep track of the mean
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ ▷ Keep track of the variance
    $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
    $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
    $\theta_t \leftarrow \theta_{t-1} - \gamma \hat{m}_t \,/\, \left(\sqrt{\hat{v}_t} + \epsilon\right)$ ▷ Update parameters accordingly ($\epsilon = 10^{-8}$)
end while
return θ_t ▷ Resulting parameters

Referring back to Section 3.2, as stated, training the entire data set for one epoch would correspond to the network seeing the entire data set once. In terms of the gradient descent algorithm, after every epoch, a total loss for that epoch will be computed by averaging every change in the gradient descent algorithm. An obvious problem with doing this is that the amount of connections between the input layer and the first hidden layer increases drastically with the network architecture and data set size so this process will be very computationally heavy. An easy fix to this problem is to split the training dataset into smaller training sets, accordingly,

$$\text{training dataset} = \text{batch\_size} \times \text{iteration},$$

where each iteration is defined as the number of batches, and the batch size is how many sets one splits the training dataset in.

3.4 Normalization

Apart from the different optimizers, cost functions, and activation functions, there are advantageous steps the user can take to make the ANN behave and train better. For example, having input data of the same magnitude range is quite important. For the scope of this thesis, one can immediately look at particle physics data. Referring back to Equation 2.4, in the extremely essential four-vector, p_T covers an approximate range of

$(1, 1500)$ GeV, while the azimuthal angle ϕ only covers $[-\pi, \pi]$ radians. By e.g., normalizing your input data, one can solve this problem.

In the thesis, the so-called MinMax normalization or feature scaling is used. This normalization performs a linear transformation on the input data such that scaled data becomes in the $[0, 1]$ range. This is done according to,

$$\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{3.22}$$

and the transformation back is done by taking the inverse

$$x = \hat{x} \cdot (x_{max} - x_{min}) + x_{min}. \tag{3.23}$$

Here, x is the set of values normalized, x_{max} is the maximum value in the set, and x_{min} is the minimum. For the scaling to be reversible and to return to physical quantities, one needs to save x_{max} and x_{min} for every scaled set. This extra set of values saved will play a role in Section 3.8.

The choice of activation function also plays a role in this, as it ultimately converts the node input to an output. Again, using the hyperbolic tangent as an example (Eq. 3.16), the larger the nodal input, the more said input will dominate the output, thereby suppressing the smaller inputs completely.

3.5 Under- and overfitting

One central challenge in machine learning is that the training needs to be done on data that the network has not seen previously. A common way to do this is by separating the entire input data into two sets, a training set and a test set, and then sampling the training set to choose parameters to reduce the error of the training set. This is followed by a test set sample, where one expects the test error to be greater or equal to the expected training set error. One can therefore classify two factors that determine how well the machine learning algorithm will perform. These two factors are [35] (1): how well the network minimizes the training error, and (2): how well it minimizes the gap between the training and test errors. These two factors correspond to two central challenges in machine learning, namely *under-* and *overfitting*. As seen in Figure 11, underfitting occurs when the loss values are not sufficiently small, and overfitting is when the gap between the two is too large.

For a trained network, there are some clues whether the model has been over or underfitted. An overfitted model is a model which is very capable of reconstructing the training data but not capable of performing well on another equivalent dataset. In other words, it fails to generalize well, as it learns patterns from potential noise in the data to a point where it negatively impacts performance. On the other hand, an underfitted model can neither fit the training data nor the test data.

Referring back to the MSE loss function, Equation 3.17, a common re-write of this equation in statistics is the following [45]

$$\text{MSE}(\hat{\theta}) = \text{Var}_\theta(\hat{\theta}) + \text{Bias}_\theta(\hat{\theta}, \theta)^2, \tag{3.24}$$

where $\hat{\theta}$ is an estimator with respect to some unknown parameter θ. In terms of over and underfitting, an underfitted model has a high bias but not necessarily a low variance. In contrast, an overfitted model has a low bias but a high variance.

One might now think that under and overfitting are bad things, but Section 3.7.1 discusses a significant difference and caveat to the overfitting problem.

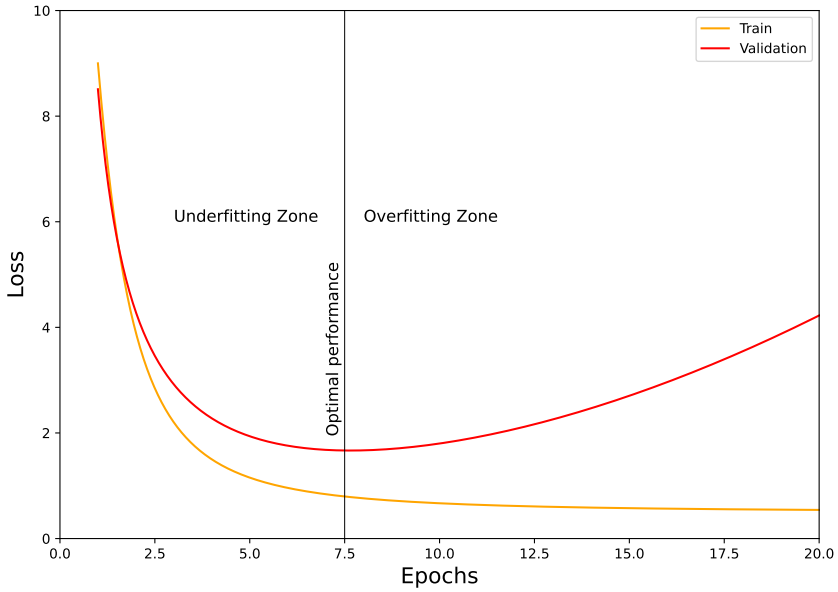


Figure 11: A typical "loss plot" shows that training and validation loss behaves differently. In the "underfitting zone," the two losses are high. As the number of epochs increases, and thereby the number of weight updates, the training error decreases while the validation error eventually increases. This is when the "overfitting zone" is entered, and one has passed the optimal performance.

3.6 Regularization

There are also ways to drastically improve model training by introducing either intermediate layers, which aim to improve training, or by introducing algorithms to detect overfitting. These are called regularization methods, some of which will be summarized below.

One of the most common and straightforward regularization methods is *early stopping*. Early stopping keeps track of the loss value after every epoch, and if the loss is much greater

than the lowest loss value, it starts a counter. If the loss has not decreased below the lowest value for p epochs, the training ends. This is implemented according to Algorithm 3. This is advantageous to include because there is no point in continuing to train a model that has reached its potential. If the loss values are not decreasing, the network is not learning anything more about the data.

Algorithm 3 Early stopping algorithm to cancel training when the overfitting region has been entered. Modified from [35].

Require: Patience value, p ▷ Value chosen to 100
Require: Loss value, L
Require: Best loss, L_B
Require: Improvement value δ ▷ Value chosen to 0
$i \leftarrow 0$ ▷ Initialize counter
$L_B \leftarrow L$ ▷ Set the first loss value to the initial best loss
while $i < p$ **do**
    Train the network and update the loss, L, via e.g. GD (see Alg. 1 or Alg. 2)
    if $L_B - L > \delta$ **then** ▷ Check if the loss has improved
        $i \leftarrow 0$ ▷ Reset counter if the loss is still improving
        $L_B \leftarrow L$ ▷ Update the best loss value
    else ▷ If $L_B - L < \delta$
        $i \leftarrow i + 1$ ▷ If the best loss hasn't improved, add to the counter
    end if
end while

Furthermore, there are also methods one can introduce to escape plateaus, as mentioned in Section 3.3. One way to do this is to reduce the learning rate when the training has hit a plateau. This can be done with the `PyTorch` [46] function `ReduceLROnPlateau`[47], which checks whether specific optimizer parameters have decreased or not. For example, suppose they have stagnated within some range after a patience value decided by the user. In that case, the optimizer's learning rate will reduce by a factor also decided upon by the user.

A prime example of a case where we can benefit from the Learning Rate scheduler and the early stopping is seen in Figure 12. Here, one can see that the model is not learning anything without the scheduler due to it being stuck in some shallow local minimum or similar. With the scheduler, one can reach lower losses, but even in this case, there is a large possibility of no room for improvement. In this case, it would have been optimal to use an early stopping algorithm to cancel the training after ~ 300 epochs for the blue/orange training and ~ 1200 epochs for the red/green training. The reason why there are no further Learning Rate decays after epoch ~ 1100 is due to there being a lower bound available on the scheduler. See [47] for a complete list of user options.

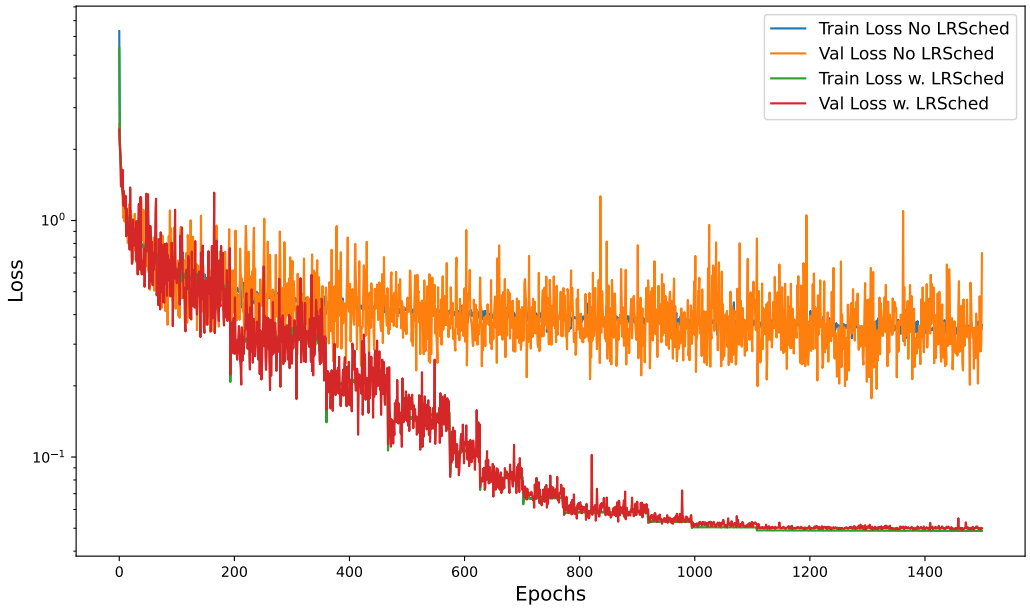


Figure 12: Two different training runs difference between using regularization methods (red/green) and not (orange/blue).

3.7 Autoencoders

Looking more specifically at an ANN capable of doing lossy compression, one usually talks about an autoencoder [48]. An autoencoder is an artificial neural network capable of mapping each input, x, to some latent space z, with $\dim z \leq \dim x$, and then mapping the latent space back to an output y with $\dim y = \dim x$. The mapping between the input and the latent space is done via an encoder, and the mapping between the latent space and the output is performed with a decoder. This is a regular, fully connected feed-forward artificial neural network, figuratively described in Figure 13.

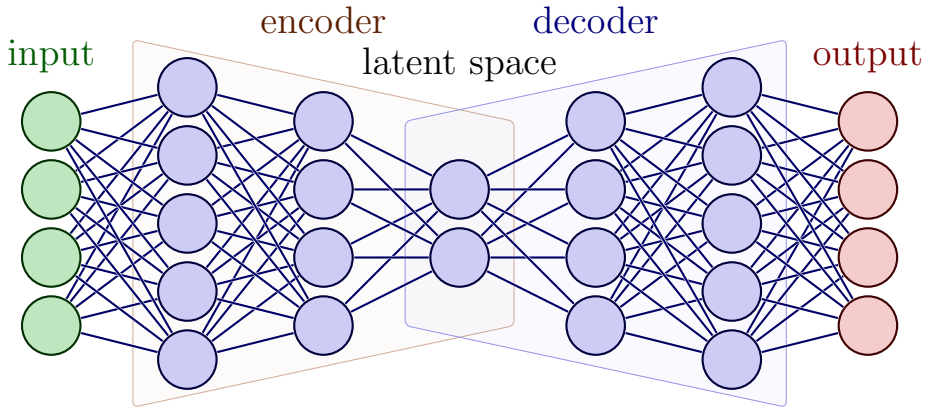


Figure 13: Illustration of an autoencoder consisting of an input and output layer. In between the input and output, hidden layers and a latent space are present, where the dimensionality of the latent space is, in this case, less than the input and output dimensions. Modified from Ref. [32].

With the autoencoder being described generally, there are several types of autoencoders. Some of these are *Convolutional autoencoders* (ConvAEs), *Variational autoencoders* (VAEs) and *Sparse autoencoders* (SAEs). This thesis will use SAEs based on previous work [49]. An SAE is defined by adding a regularization term to the total cost function, and in our case, we have chosen to use the L1 regularization term,

$$\mathrm{L1} = \lambda \sum_i |w_i|, \tag{3.25}$$

with λ being a regularization parameter. The L1 term is seen as a term that introduces a sparsity penalty to the training criterion. This sparsity penalty is proportional to the absolute value of the magnitude of the weight coefficients, meaning that the effect of this term leads to a shrinkage of the penalty coefficients towards zero. This is advantageous when one aims to do, e.g., feature extraction because it forces weak features in the network to have coefficients equal to zero. Introducing this term to the cost function has an additional advantage, being that of making the autoencoder model smaller. This will be important in Section 5.

3.7.1 Offline and Online compression

Autoencoder compression can be split into two fields, *online* and *offline* compression, with the main difference between the two being *generation*. A generational algorithm in the sense of compression methods would use a dataset, A, to train an autoencoder model, M_A, and then use M_A to compress a different dataset, B. In this thesis, we designate this use case, the name online compression, due to no standard nomenclature being available for this use case. Offline compression would then be where one uses M_A to *exclusively* compress A and not any other dataset.

An example of a potential online compression case would be in the, e.g., ATLAS trigger system. One could in theory train an autoencoder model on a large amount of very general particle physics data and then compress new data directly at trigger level.

The use case for offline compression use case is very general. For any given dataset, one can train a model capable of compressing it, and then only save the model and the compressed state (plus additional auxiliary data, defined in Section 3.8) to decompress it when it fits. Given time and resources, there is also a possibility to tailor the model and test what works for the specific case. For this case, we want to overfit the model to the data it has been trained on. Upon achieving this, it would mean that the model works exceptionally well for the data it has been trained on but poorly for any other dataset. This is a significant caveat to the general understanding of under/overfitting and is a core theoretical obstacle to clear when trying to understand *how* one needs to optimize the training to achieve optimal conditions for compression.

3.8 Evaluation Metrics

In order to evaluate the performance of the autoencoder for compression, two metrics are most prominently used. These two are the *residual* and the *response*, defined as following

$$\text{residual} = \hat{x} - x, \tag{3.26}$$

$$\text{response} = \frac{\hat{x} - x}{x}, \tag{3.27}$$

with x being the original data and $\hat{x}$ being the reconstructed data.

For data compression, another important metric is the compression ratio. The compression ratio measures how much percent one has decreased/increased the original file size by. Theoretically, the compression ratio is found as

$$R = \frac{size(\text{original})}{size(\text{compressed})},$$

but for autoencoder compression, this is not the case. Since the compression is based on a mapping via weights, the un-mapping is done in the same way. Therefore, to compress and decompress, one needs the autoencoder's weights which act like a key capable of telling the encoder/decoder how to scramble/unscramble the data. One will then need a model file that stores these weights. Additionally, we will also make a definition of *auxiliary data*. Auxiliary data is all data necessary for the full decompressed output to be structurally the same as the original. For example, if one has scaled the input data (with, e.g., MinMax), you will also need the scaling features to perform the inverse transformation to your output data. In total, one can summarize the actual compression ratio as

$$R^* = \frac{size(\text{original})}{size(\text{output})} = \frac{size(\text{original})}{size(\text{compressed + model + auxiliary data})}.$$

One might see this as an extreme disadvantage, but for an SAE, the model size is more or less constant compared to the input data size. For constant auxiliary data and model sizes, Figure 14 represents well how this effect becomes negligible for large input dataset sizes for different compression ratios.

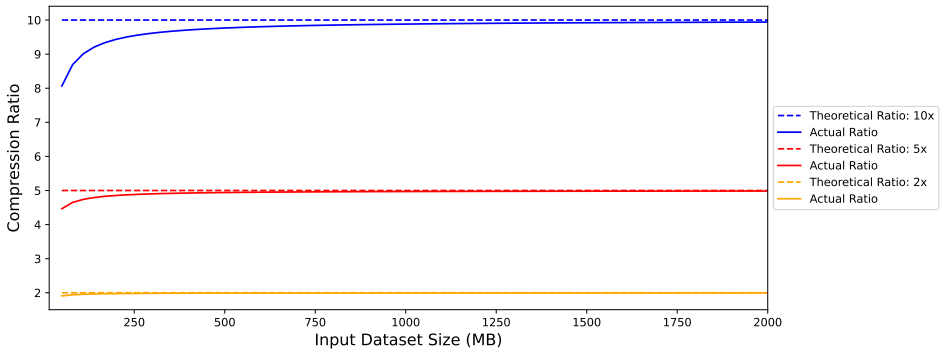


Figure 14: Dashed lines show the theoretical compression ratios for three cases, while the solid lines show the actual compression ratio including auxiliary data and model size.

Additionally, we will deal with many variables, so an easy way to present distribution features is via *box-and-whisker plots*, or *box-plots*. These are common statistical plots used to present the following attributes of a distribution concisely:

1. Median (second quartile)
2. 1st and 3rd quartiles
3. The Minimum and maximum the distribution
4. Eventual outliers,

where the quartiles are separations in the distribution with equal amounts of observations in them. The different box shapes and how to correlate them to the distribution shape are shown in Figure 15.

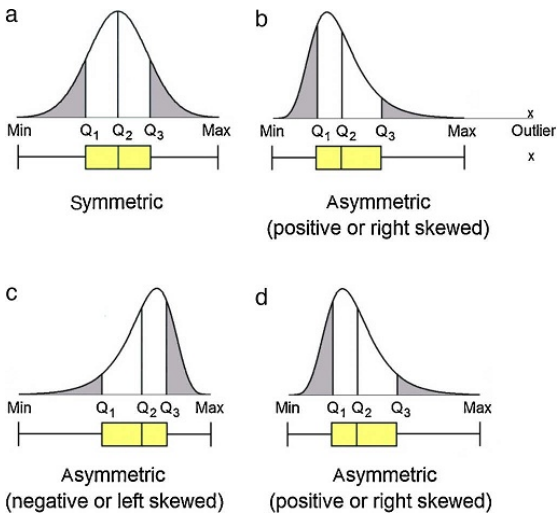


Figure 15: Figurative description of how to correlate box-plot shapes to distributions. $Q_{1,2,3}$ represents the distribution's first, second, and third quartiles. From Ref. [50].

One big advantage with using box-plots is that they show skewnewss very well. This is not shown very well in other common visualization methods, such as the standard deviation.

4 Methodology and Implementation

During and as part of this thesis, an open-source tool named *Baler* [51] was developed with the support of a Turing-Manchester Feasibility Project grant [52]. Baler's main objective is to create an easy-to-use, open-source platform for people to use, which provides the fundamentals for using autoencoders as a compression tool for scientific data. Below, we will discuss our methodological implementation of how Baler has been used for this thesis, the motivation of data handling, different use cases, and the main model setup for the results presented in Section 5. Attention is also directed towards Ref. [53], which presents a short and concise description of Baler's development stage and early preliminary results.

To compare our compression method to PCA, described in Section 2.3.3, we have used the `sklearn.decomposition.PCA` function from the `scikit-learn` library [54], which allows for a highly efficient implementation of the PCA method concerning both time and computational resources.

4.1 The Data

For this thesis, the data used to develop this compression tool are open datasets from the CMS collaboration released under the Creative Commons CC0 waiver [55]. The datasets in question are presented briefly in Table 2, with them all having different features and importance to the final result. Originally, the data is stored in `.root` files. These files contain many datasets in so-called *branches*. One of these branches has been selected, which contains data based on specific pre-selections, such that we only deal with jet data.

Before training, the data goes through a pre-processing step. This step is required to ensure that the data is in the correct form, which `PyTorch` and Baler support. The first step in the pre-processing is flattening the hierarchic data structure initially present to comply with the requirements of `PyTorch`. Secondly, this step also allows for selection cuts to be introduced. Selection cuts are introduced to remove potential noise from the data or values so small that they play no role in analyses. In this thesis, cuts on mass and transverse momentum are present, such that $p_T \in [1, 8000]\,\text{GeV}$ and $m \in [10^{-3}, 800]\,\text{GeV}$. Table 2 shows the amount of events pre and post-cuts. We also remove all variables only containing zeroes. This is because they will not benefit the network, and reconstructing them is trivially done without an autoencoder. Finally, the pre-processed file is saved as a `.npz` file.

Table 2: Summary of the datasets used.

	HEP1 [56]	HEP2 [57]
Name	JetHT primary dataset	TTbarDMJets
Format	AOD	AODSIM
Simulated or Real Data	Real	Simulated
Total dataset size (`.root`)	3.26 GB	1.95 GB
Subset of dataset size (`.npz`)	137.6 MB	31.6 MB
Number of events pre-cuts	716446	494197
Number of events post-cuts	716104	493080

Structurally, the two datasets are similar, but there is one big difference. HEP1 contains 24 variables after pre-processing, while HEP2 only contains 8 variables. The variables contained in the two datasets and a short description are present in Appendix A.1. One thing to note regarding the data is that all data is not of the same data type. In HEP1, 9 of the 24 variables (every multiplicity variable) are integers, while the rest are floats. Due to `PyTorch` needing to convert the data into tensors, it can only handle *one* datatype at the time, so integers are treated as floats throughout. This is not optimal for the final evaluation since it will introduce a big error factor. Upon reconstruction, if an integer variable now is a float with a Gaussian profile, there will be a 50% chance of incorrectly reconstructing this variable. Figure 16 shows the distribution of the HEP1 dataset, while Figure 17 shows the HEP2 distributions.

Further, we scale all data using the MinMax scaling method described in Section 3.4. This scaling is done column-wise, so for particle physics data, we get the following structure, with n representing the event number and i the number of variables contained in the dataset:

$$
\begin{pmatrix}
p_T \\ {\scriptstyle \in[\min p_T, \max p_T]} & \eta \\ {\scriptstyle \in[\min \eta, \max \eta]} & \phi \\ {\scriptstyle \in[\min \phi, \max \phi]} & m \\ {\scriptstyle \in[\min m, \max m]} & \cdots & i \\ {\scriptstyle \in[\min i, \max i]} \\
\hline
p_{T_0} & \eta_0 & \phi_0 & m_0 & \cdots & i_0 \\
p_{T_1} & \eta_1 & \phi_1 & m_1 & \cdots & i_1 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
p_{T_n} & \eta_n & \phi_n & m_n & \cdots & i_n
\end{pmatrix}
$$

$$
\xrightarrow[\text{MinMax}]{}
\underbrace{\begin{pmatrix}
p_T \\ {\scriptstyle \in[0,1]} & \eta \\ {\scriptstyle \in[0,1]} & \phi \\ {\scriptstyle \in[0,1]} & m \\ {\scriptstyle \in[0,1]} & \cdots & i \\ {\scriptstyle \in[0,1]} \\
\hline
\overline{p_T}_0 & \overline{\eta}_0 & \overline{\phi}_0 & \overline{m}_0 & \cdots & \overline{i}_0 \\
\overline{p_T}_1 & \overline{\eta}_1 & \overline{\phi}_1 & \overline{m}_1 & \cdots & \overline{i}_1 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\overline{p_T}_n & \overline{\eta}_n & \overline{\phi}_n & \overline{m}_n & \cdots & \overline{i}_n
\end{pmatrix}}_{\text{Normalized data}}
+
\underbrace{\begin{pmatrix}
\min p_T & \max p_T \\
\min \eta & \max \eta \\
\min \phi & \max \phi \\
\min m & \max m \\
\vdots & \vdots \\
\min i & \max i
\end{pmatrix}}_{\text{For reverse transformation}}
$$

Since this normalization is a scaling transformation that shifts the data to lie in another range, the shape of the distribution before and after normalization will be identical. This can be seen in Appendix A.2.

As stated, the four-momentum is the most crucial part of the datasets. In reality, this quantity will likely be left alone to avoid poor reconstructions of these variables, but for this thesis, the entire dataset will be compressed as an example.

Lastly, one thing to note in Figure 17 is that the jet mass distribution prominently shows two peaks at $\sim 82\,\text{GeV}$ and $\sim 170\,\text{GeV}$. These two are the $W^{\pm}$ and top quark mass peaks, respectively. Since the data we are investigating are single jets and not di-jets, obtaining the masses might not be trivially understood. Briefly, the data utilizes the four-momentum invariant mass to obtain well-defined expectations for the decays. For this to be reliable, one uses that the mass of a hadronic decay should correspond to the mass of the parent particle, and one also assumes that the complete decay is contained in the jet [58].

To understand how well the compressed data reproduces irregular shapes, such as peaks, an analysis of the $W^{\pm}$ and top-quark mass peaks will be performed.

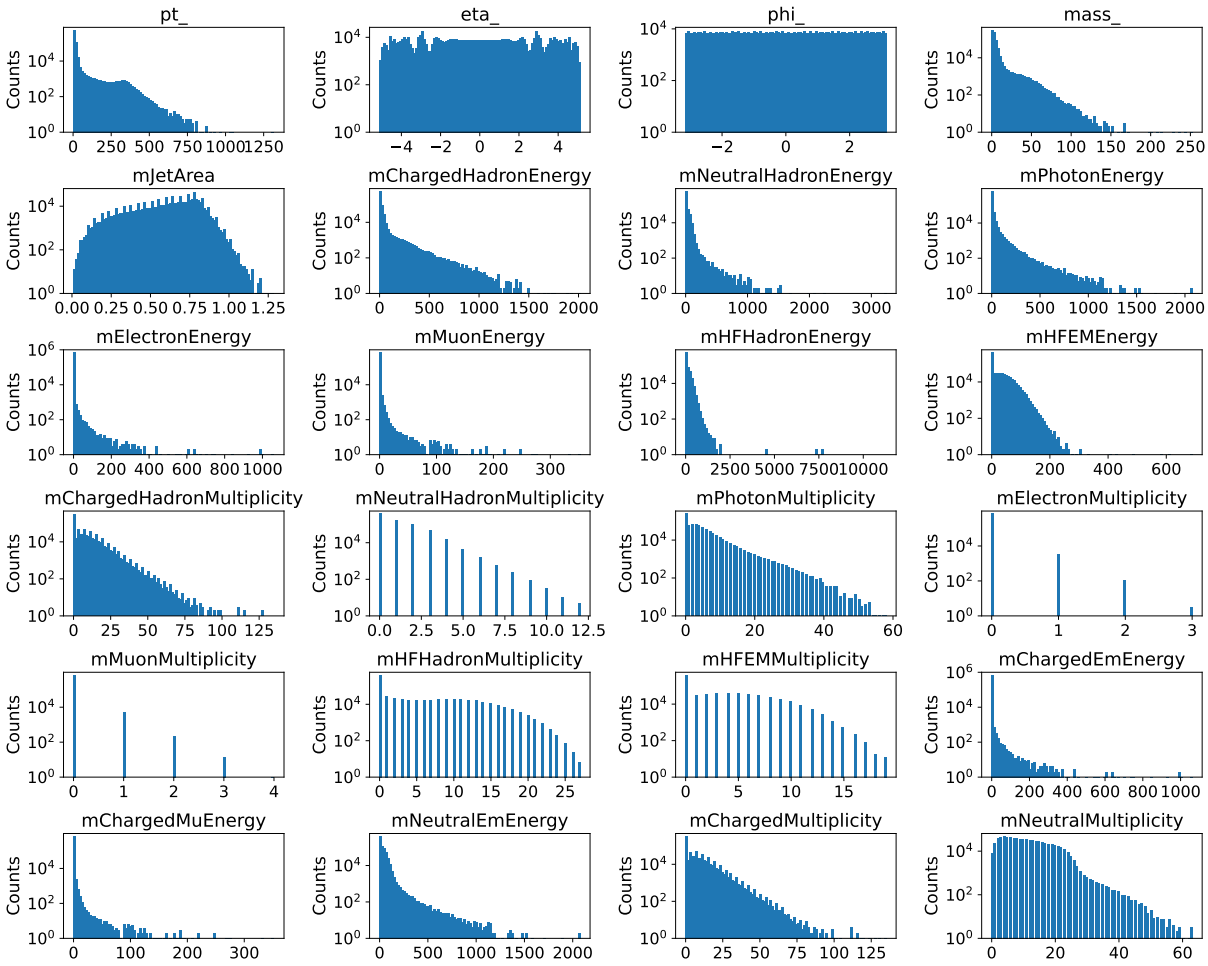


Figure 16: Distributions of all 24 variables in the HEP1 dataset.

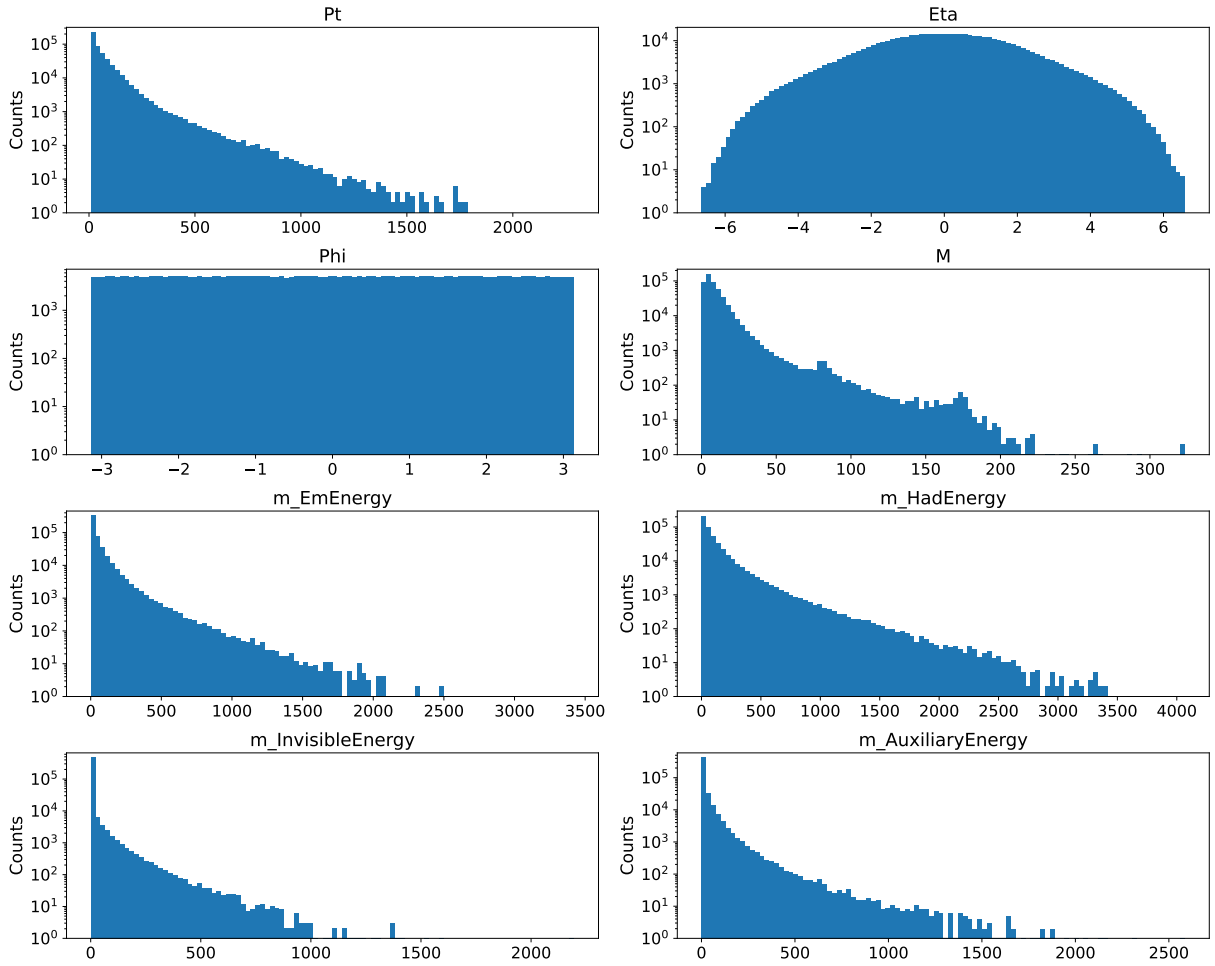


Figure 17: Distributions of all 8 variables in the HEP2 dataset.

4.2 Model & hyperparameter selection

Based on previous studies [43, 44, 49, 59–61], a model architecture that provides good results for four-momentum compression, and selective compression for higher dimensions have been established. Together with this, a set of base model parameters was also decided upon, which are:

1. Batch Size: 512 **2.** Learning Rate (γ): 10^{-3} **3.** Reg. Parameter (λ): 10^{-3}

4. Early Stopping Parameter: 100 **5.** LR Scheduler Parameter: 50

The model architecture used throughout is

$$x - 200 - 100 - 50 - z - 50 - 100 - 200 - x,$$

with $x \in \{8, 24\}$ and $z \in \{2, 4, 5, 15\}$ such that $R \in \{1.6, 4, 6\}$. These values are motivated in Section 5.1. For every node, the LeakyReLU activation function (Equation 3.14) has been chosen and will be used throughout the thesis.

We will also consider two different loss functions, namely:

$$\mathcal{L}_1 = \text{MSE\_SUM} + \text{L1}, \tag{4.28}$$

$$\mathcal{L}_2 = \text{MSE\_SUM} + \text{EMD} + \text{L1}, \tag{4.29}$$

where MSE_SUM is a modification of Eq. 3.17. This modification is motivated by an edge case. Suppose one have $\hat{Y} = \{1, 0, 0, 0, 1\}$ and $Y = \{0, 0, 0, 0, 0\}$, one then gets an MSE value of 0.4. This is a relatively low loss despite two of five values being reconstructed 100% incorrectly. To solve this, we can remove the $1/m$ scaling to get an MSE of 0.8. Removing this factor penalizes the training more, but the final model should show better reconstructions, albeit with the loss converging to a higher number. Hence, we define MSE_SUM as

$$\text{MSE\_SUM} = \frac{1}{n} \sum_{j=1}^{m} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i \right)_j^2.$$

As discussed previously, we have clarified that there are two types of autoencoder compression; online and offline. For the results, we will perform offline compression. Recalling this, we will train a model M_A to *exclusively* compress dataset A, and not any other dataset. We will exclude the standard train-test split to no introduce any generation in this, as mentioned in Section 3.5.

4.3 Baler workflow

Throughout this thesis, and thereby Baler, the `PyTorch` [46] library made for machine learning and neural networks in `Python` was used in junction with `NumPy` [62] and `SciPy` [37].

Baler has been developed as a tool to support file formats that are commonly used, namely `NumPy` arrays. There are several reasons for this, but the main reasons are that scientists across many different fields know how to handle `NumPy` array structures, both in the sense of how to convert data into that format, but also how to handle the format; in the case of the dataset outliving the `NumPy` library, reverse engineering the format will be a simple task; and finally, `PyTorch` requires the conversion to tensors, so a data conversion is needed in some stage no matter what.

Furthermore, all necessary ML steps need to be outlined and easily accessed to create a user-friendly tool for ML. For this to be the case, a Baler flowchart has been created, visible in Figure 18, which clearly outlines the steps available for the product we have created.

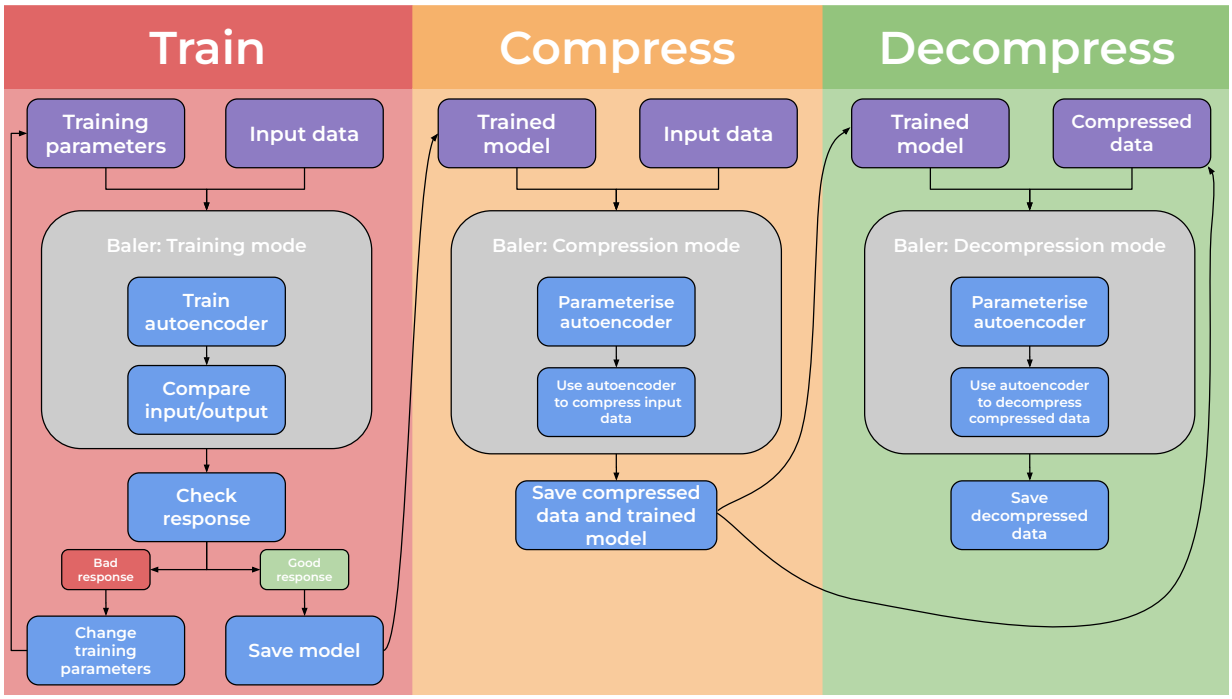


Figure 18: A flowchart showcasing the different steps available in the Baler tool. The three steps are necessary for every form of autoencoder compression.

To explain the flowchart further, the steps can briefly be summarized:

1. Training: During training, the network trains on the already pre-processed user-inputted dataset, which is due to be compressed. This process is the most time-consuming step, as it demands the encoder and decoder to be trained on either the whole dataset or a set of if, depending on whether one aims to do online or offline compression.

   After training, one can quickly compare the input and output to see if the response is within the frame of what is considered to be a tolerable reconstruction distortion. If it is, one can move on; otherwise, a tweaking of the training parameters can improve the result.

2. Compression: The compression step is when one has obtained the trained model and wishes to compress the original data to a smaller size. When this is done, the model file and potential auxiliary data are all needed to decompress the compressed data.

3. Decompression: When one wishes to use the compressed dataset, all needed to do is to decompress the now compressed dataset. This utilizes the decoder part, resulting in a file the same size as the original one and ready for analysis.

4.4 Open Source

For several reasons, one could make a project open source. First, a project being open source means that the code necessary to reproduce a product from the ground up is entirely available to the public. Open-sourcing projects can be a potent tool because it gives people a much more accessible platform to adopt and contribute to a project, which can significantly impact project improvement. Apart from this, an essential feature of an open-source project is transparency. In the field of science, this is especially important because being able to control fellow peers' work by reproducing results and ensuring that the results have not been modified to look good plays a major role in the scientific context of trust.

In reality, it is straightforward to open source a software project, but if one wants to create an excellent open-source project, much work needs to be done. This has been further investigated in a Master's thesis done in parallel and part-wise in junction with this thesis by Fritjof Bengtsson at Lunds Tekniska Högskola (LTH) **(not yet published)**, which goes into much greater detail concerning this subject, but here I will outline some essential points beyond having the code fully public.

First of all, every open-source project needs to have a license. This license guarantees that others know how to modify, copy, contribute, and use the project without any surprise repercussions or potential legal issues. There are several popular licenses to choose from, with different restrictions on details concerning contributions, copies, and modifications. Some of the most popular licenses are `Apache 2.0`, `MIT`, `GPL` and `CC-BY-4.0`.

Secondly, open-sourced software needs clear information regarding how one proceeds to use the software and how contributions will be formed. This can be done in the form of a `README` file, which carefully explains and guides the user through the steps of how to use the software, such as introducing the necessary commands needed to run the software and even how to install the required prerequisites on different operative systems. Forming a guideline for contributions does not have an apparent spot in the `README`, but a good open-source project shall have a separate contributions guidelines document for this. This document should have the necessary information to make suggesting a new feature or reporting a bug pleasant and equal to everyone who wants to contribute with a suggestion.

Finally, a code of conduct has to be established. This sets ground rules for the behavior of the participants and contributors and helps to build a pleasant community around the project. In some regards, this is the most important part because if one has a large community, the standards concerning how to behave and improve collaborations between users will most likely correlate to how much the project develops.

Making software open source is easiest done on a platform like `GitHub` [63]. `GitHub` is a platform that allows code to be developed and hosted on an online platform with easy access to user input. However, although `GitHub` yields a simple way to fetch code to a local machine, it does not generate a way for local machines to ensure that the specific packages necessary for particular code to run are, e.g., the same versions. This can be crucial for

software to run correctly and can be solved by using, e.g., `Docker` [64] or `Poetry` [65]. Both software packaging programs are quickly introduced in a software project, allowing package versions to be consistent no matter where or when one uses software locally.

5 Results

To begin with, compression ratios have been investigated, and it has been determined how much Baler needs to compress the given dataset to outperform methods such as `gzip`. Secondly, a study on what loss function performs the best is presented. Finally, reconstructions and fits of two peaks were then performed from the HEP2 dataset to tell us how well the autoencoder can reconstruct unique features.

We then discuss offline compression, followed by a brief discussion of online compression, and finalize the section with a short preview of an application in a different field of physics, Computational Fluid Dynamics (CFD), together with results regarding the open-sourcing of the project.

All results presented below have been produced using Baler v.1.0.0 [51].

5.1 Compression Ratios

Since we need to convert our input data into `NumPy` arrays, thus becoming `.npz` files, a fair compression method to compare to would be `gzip` since it is commonly available. By zipping the files HEP1 and HEP2 files, go from

$$\begin{aligned} &\text{HEP1} : size(\texttt{.npz}) = 137.5\,\text{MB} \rightarrow size(\texttt{.zip}) = 29.2\,\text{MB} \implies R = 4.7, \\ &\text{HEP2} : size(\texttt{.npz}) = 31.6\,\text{MB} \rightarrow size(\texttt{.zip}) = 14\,\text{MB} \implies R = 2.3. \end{aligned}$$

To have an advantage over `gzip`, we aim to compress HEP1 with $R = 6$ and HEP2 with $R = 4$. These are also the compression ratios that we will use for the PCA method.

Furthermore, as discussed in Section 3.8, the difference between R and R^* is negligible for HEP data. This can be motivated by looking at the size of the auxiliary data. For HEP data, both HEP1 and HEP2, the model size does not go above 550 KB. Counting all auxiliary data, which includes, e.g., normalization features, headers, and even all values for the loss plots, the auxiliary file size does not reach past 600 KB. Previous studies show that a compression ratio of $R = 1.6$ has given good performance for multi-variable HEP datasets, so this compression ratio will also be looked at and used as a baseline compression ratio. Here on forth, the theoretical compression ratios will be used in plots and tables. Taking all of this into account, we get the following differences

$$\begin{aligned} &R^{\text{HEP2}} = 4 \rightarrow R^* = 3.72, && R^{\text{HEP1}} = 6 \rightarrow R^* = 5.85, \\ &R^{\text{HEP2}} = 1.6 \rightarrow R^* = 1.55 && R^{\text{HEP1}} = 1.6 \rightarrow R^* = 1.57 \end{aligned}$$

5.2 Determination of Loss Function & Training times

First of all, the two loss functions $\mathcal{L}_1$ and $\mathcal{L}_2$ has been investigated as defined in Equations 4.28 & 4.29. After initial training of 1000 epochs with the model parameters defined in Section 4.2, Figure 19 presents the loss plots obtained with $R = 1.6$.

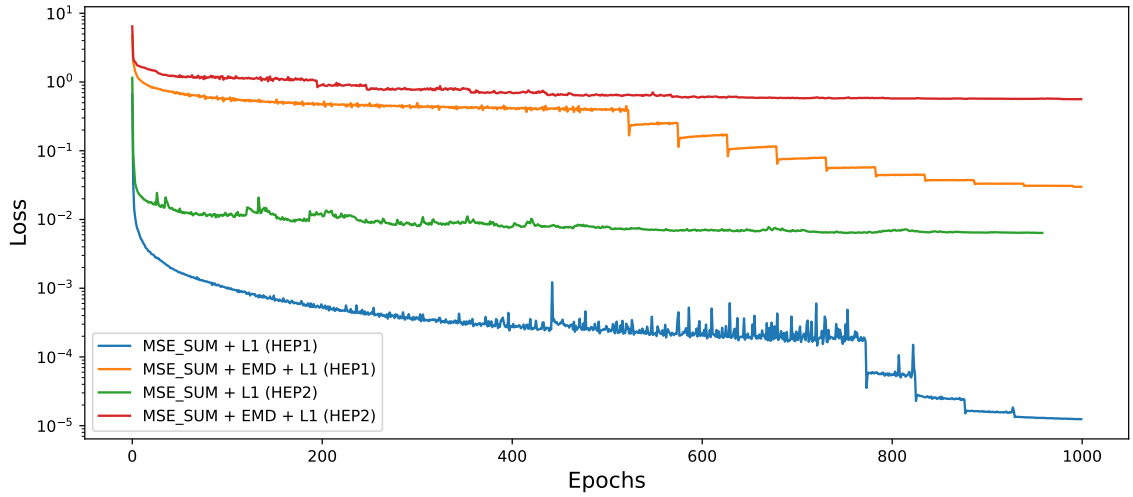


Figure 19: Training loss for two different loss functions using a Learning Rate Scheduler.

In Appendix B.1, the loss plots for the HEP1 and HEP2 at $R = 4$ and $R = 6$ respectively are presented, for both $\mathcal{L}_1$ and $\mathcal{L}_2$. For the respective cases, the final loss values reached are shown in Table 3.

Table 3: Final loss values for three compression ratios using two different loss functions.

Loss Function	$\mathcal{L}_1$ = MSE_SUM + L1			$\mathcal{L}_2$ = MSE_SUM + EMD + L1		
Comp. Ratio	1.6	4	6	1.6	4	6
Min. Loss HEP1	1.24×10^{-5}	-	2.36×10^{-2}	2.99×10^{-2}	-	1.23
Min. Loss HEP2	6.34×10^{-3}	1.35×10^{-1}	-	5.62×10^{-1}	1.35×10^{-1}	-

The training was done on the AURORA cluster hosted by LUNARC [66] at Lund University. On the cluster, the nodes are running CentOS 7.2 x86_64 and consist of 2 Intel Xeon E5-2650 v3 (2.3 GHz, 10-core) with 64 GB of memory (3.2 GB/core). `PyTorch`s CUDA implementation also allows for training on GPUs, and this has been utilized on the AURORA cluster with NVIDIA TESLA K80 GPUs. Most of the training was done on the GPU clusters due to the decreased training times, with Table 4 summarizing and comparing the training lengths. In this table, we also include the training of a CFD dataset, which will be discussed in Section 5.6.

Table 4: Summary of different training runs with different datasets and epochs on either a CPU or GPU cluster. All training has been performed on the AURORA cluster hosted by LUNARC at Lund University.

Dataset	Epochs	Training Time	
		CPU	GPU
HEP1	1000	10h 10min 10s	03h 01min 04s
HEP2	1000	06h 36min 40s	01h 35min 24s
CFD	2000	03h 35min	8min 54s

5.3 Balers performance at different compression ratios

Figure 20 presents the four-vector variable histograms plotted before compression and three cases after compression. After PCA, AE using $\mathcal{L}_1$ and AE using $\mathcal{L}_2$, all for $R = 1.6$. In Appendix B.2, all reconstructed variable distributions for different compression ratios are present. PCA will be discussed more in Section 5.4.

Figure 21a compares the performance of Baler's compression of the HEP1 dataset at $R = 1.6$ and $R = 6$. For HEP2, a comparison between $R = 1.6$ and $R = 4$ is shown in Figure 21b. These plots are for the $\mathcal{L}_1$ loss function, and the corresponding plots for $\mathcal{L}_2$ are shown in Figure 22.

There are no outliers shown in the box-plots. They will be dealt with in Section 5.5.

For all plots below, the boxes extend between the first and third quartiles of the distribution, and the whiskers are located at $Q_1 - 1.5(Q_3 - Q_1)$ and $Q_3 + 1.5(Q_3 - Q_1)$ respectively. The orange line displays the median, while the green line indicates the mean. The reason for the energy variables mostly being shown as residuals is that they can have a much more dynamic range than the rest.

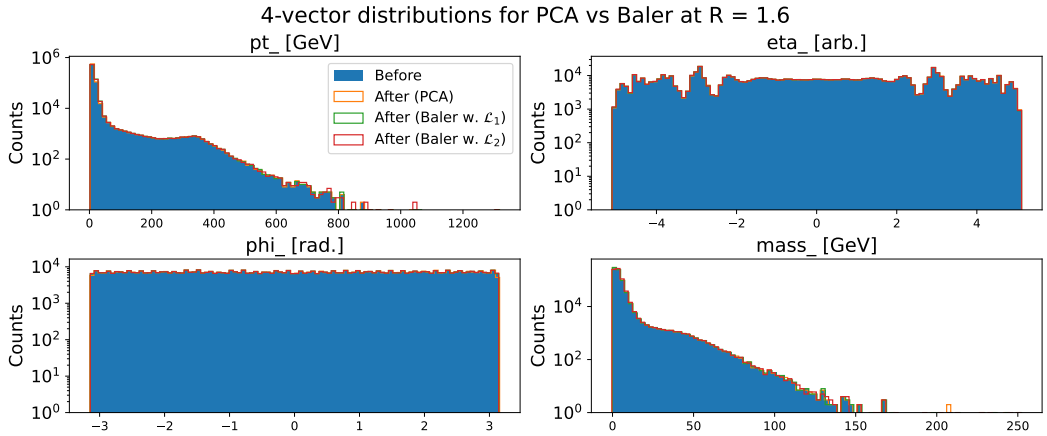
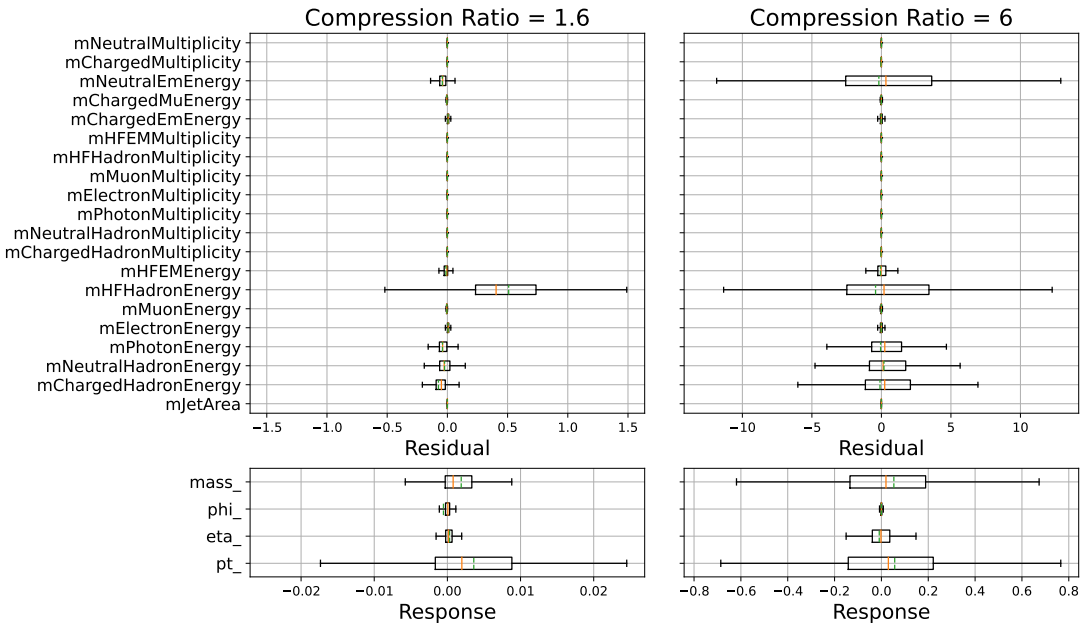
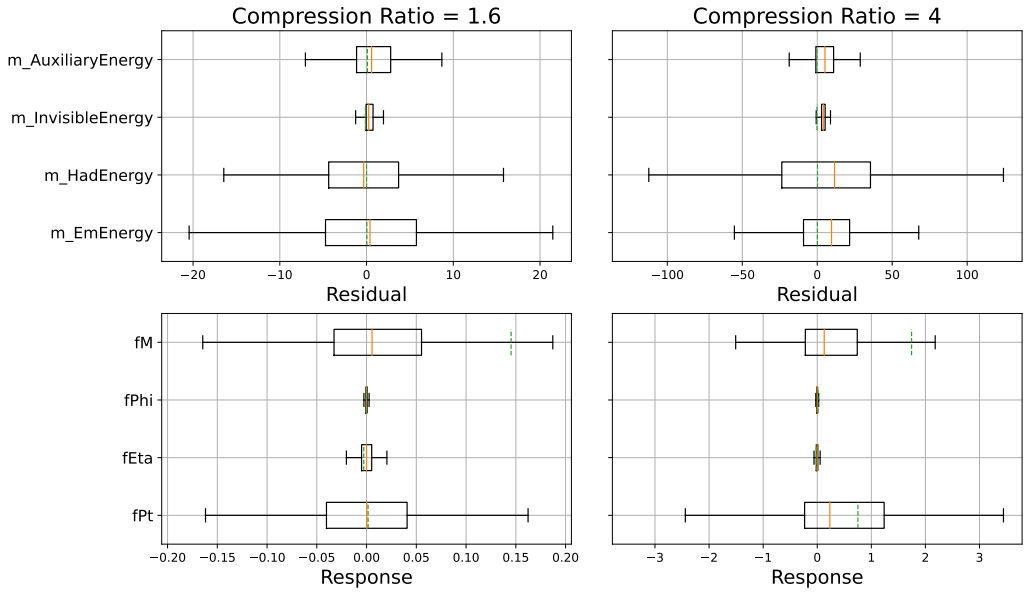


Figure 20: The original (blue) four-vector variable distributions, after PCA (orange), after AE compression using $\mathcal{L}_1$ (green) and after $\mathcal{L}_2$ (red) at $R = 1.6$.

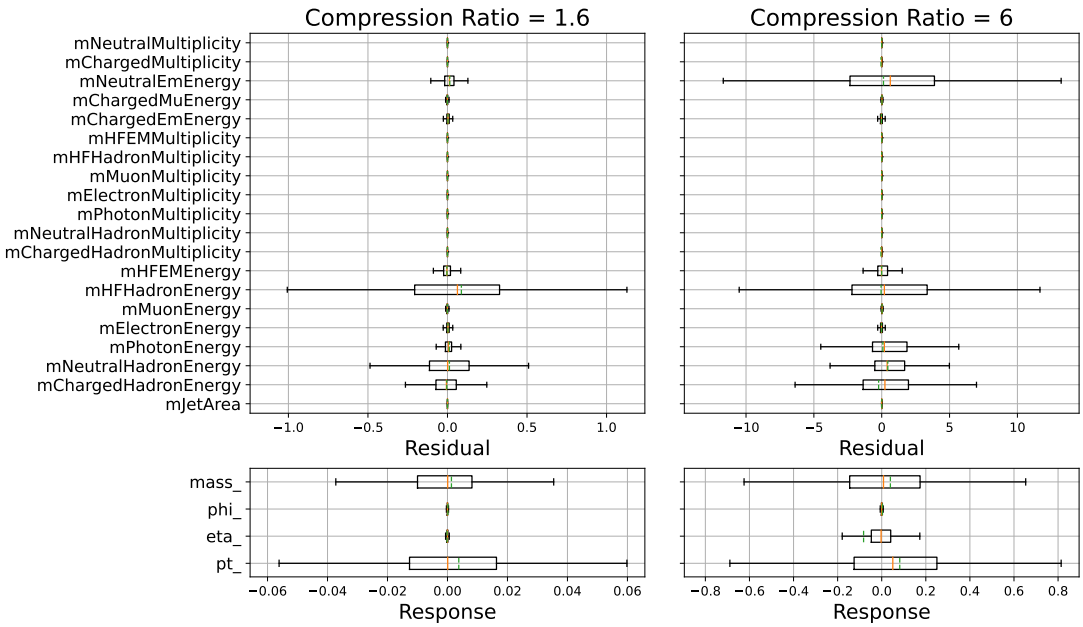


(a)

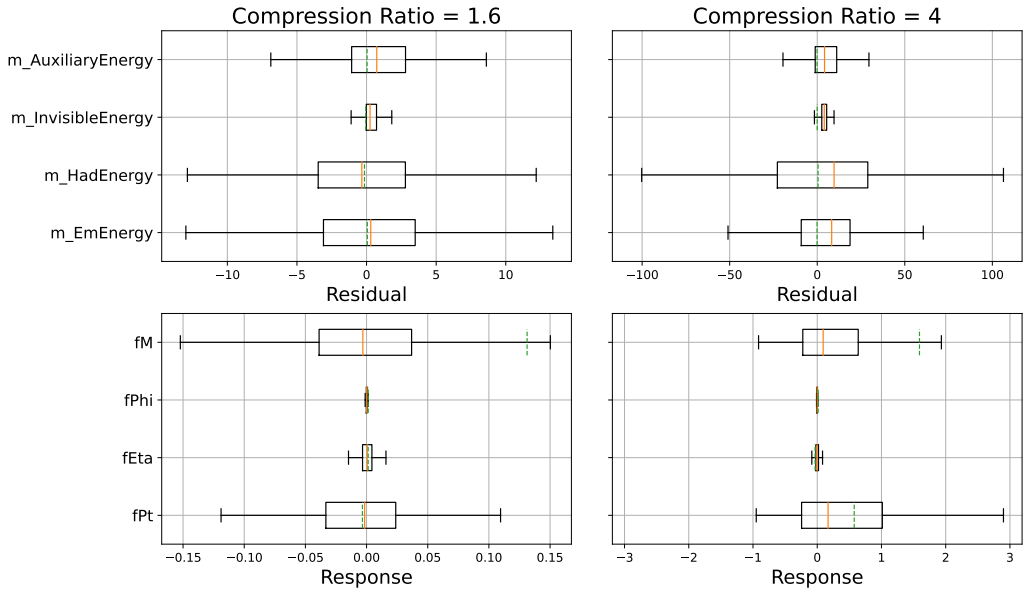


(b)

Figure 21: Response distribution for the four-momentum variables and residual distributions for the other variables presented as boxplots at two compression ratios for **(a)** HEP1, and **(b)** HEP2 after training the autoencoder model with $\mathcal{L}_1$.



(a)

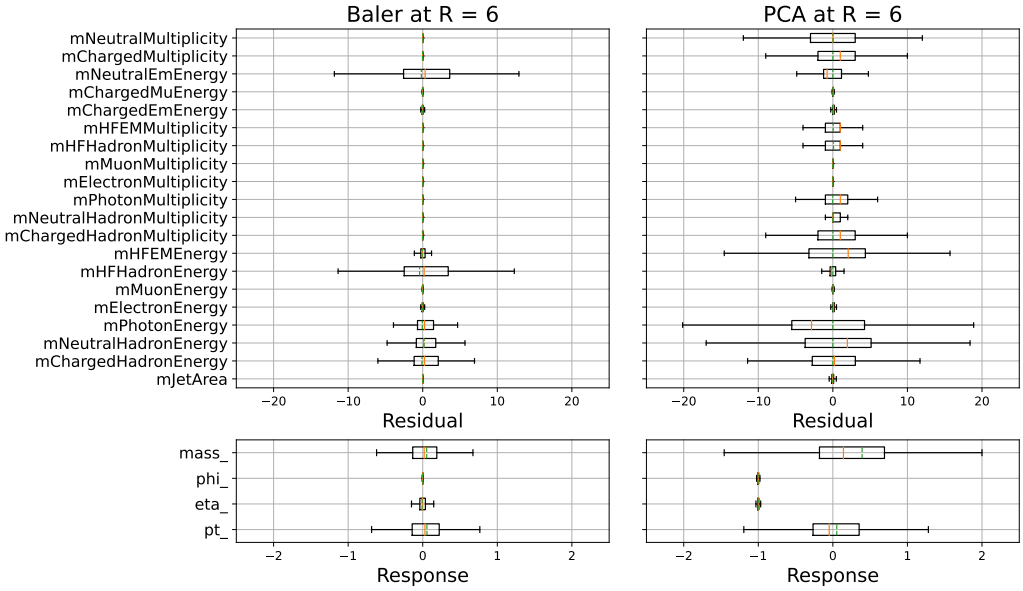


(b)

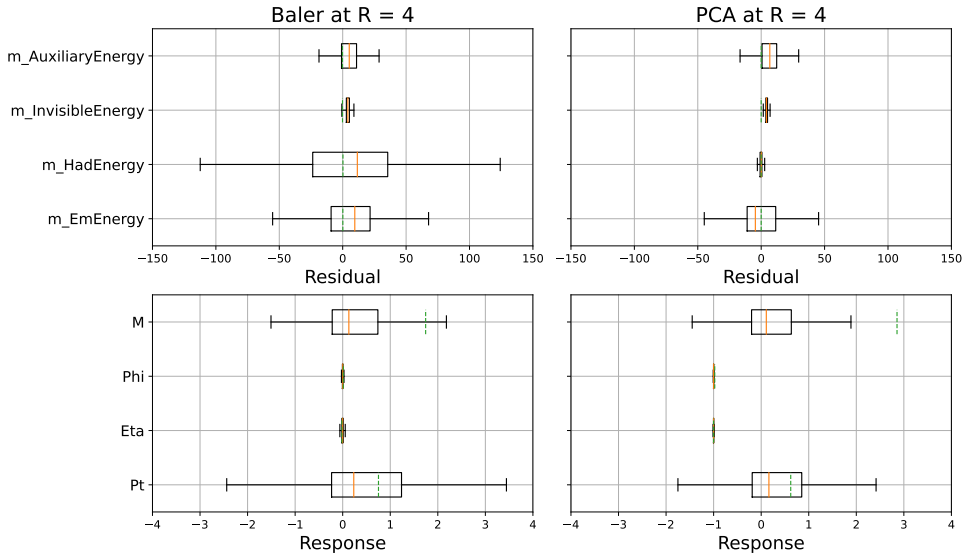
Figure 22: Response distribution for the four-momentum variables and residual distributions for the other variables presented as boxplots at two compression ratios for **(a)** HEP1, and **(b)** HEP2 after training the autoencoder model with $\mathcal{L}_2$.

5.4 Baler vs PCA

We show the comparison between Baler and PCA in a similar fashion. Figure 23a shows a comparison between HEP1 and PCA at $R = 6$, while Figure 23b compares HEP2 and PCA at $R = 4$.



(a)



(b)

Figure 23: Response distribution for the four-momentum variables and residual distributions for the other variables presented after Baler compression (left) and PCA (right) for **(a)** HEP1 and **(b)** HEP2.

5.4.1 Reconstruction of peaks

Figure 24 shows a reconstruction of the two mass peaks seen in Figure 17 at $R = 1.6$, while Figure 25 shows it at $R = 4$. This has been done by fitting a Gaussian to the corresponding locations and extracting the local maximum to obtain a mass value. The full reconstructed distributions can be seen in Appendix B.2. Obtained from [67], the actual mass values are [6]

$$m(W^{\pm}) = 80.377 \pm 0.012\,\text{GeV} \qquad m(t) = 172.69 \pm 0.30\,\text{GeV}.$$

Furthermore, the analysis of the peaks has been made with $\mathcal{L}_1$, motivated by the shapes of the reconstructed peaks in Appendix B.2. The physical interpretations of the fits are further discussed in Section 6.

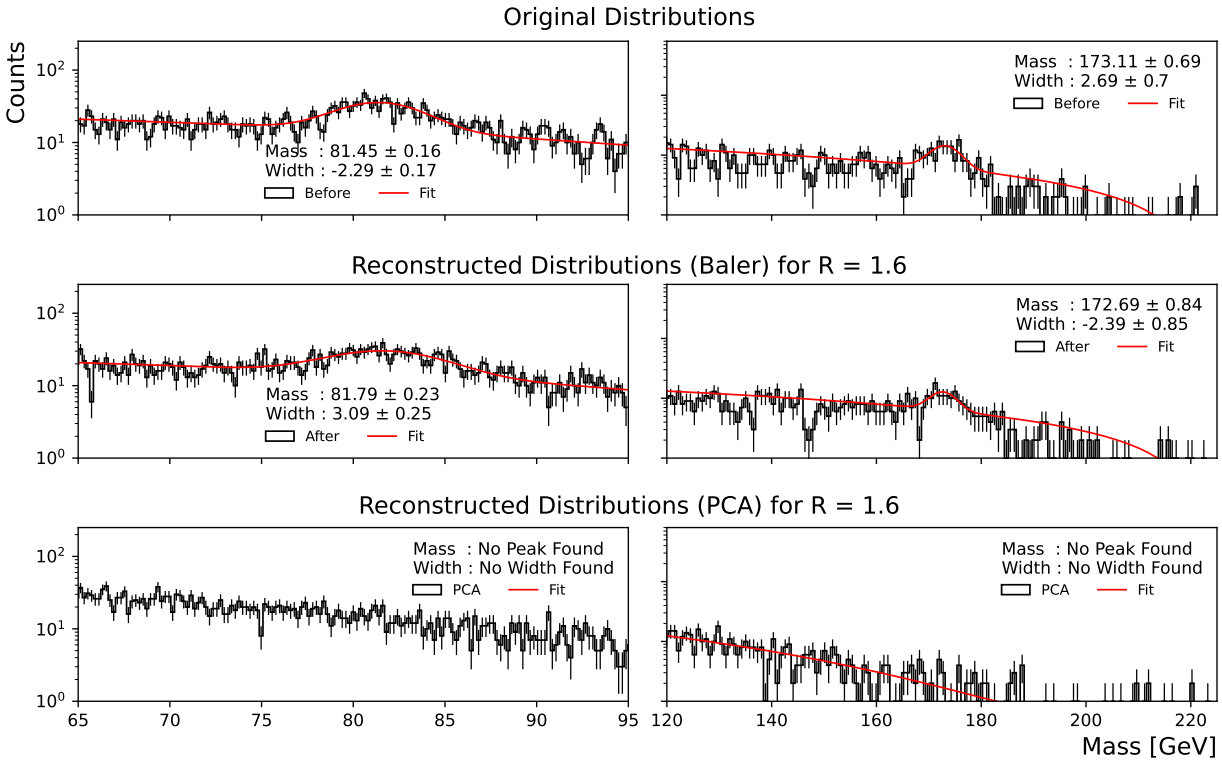


Figure 24: Fitted Gaussians to the $W^{\pm}$ (left) and top-quark (right) mass peaks for Baler versus PCA after compression with $R = 1.6$.

[6] The top mass is derived from direct measurements. Cross-section measurements yield a mass of $m(t) \approx 162.5^{+2.1}_{-1.5}$GeV [67]

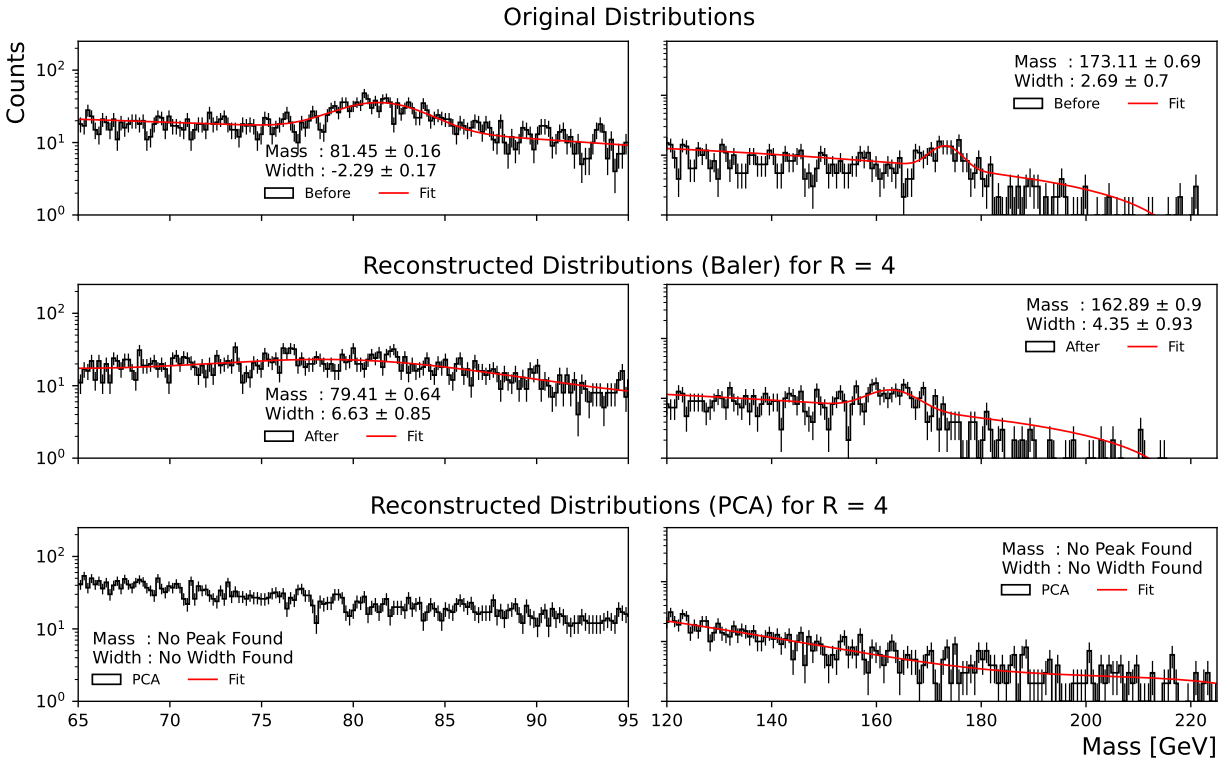


Figure 25: Fitted Gaussians to the $W^{\pm}$ (left) and top-quark (right) mass peaks for Baler versus PCA after compression with $R = 4$.

5.5 Outliers

For both the residual and response distributions, a common theme presents itself. This theme is that all distributions are extremely broad, leading to many outliers being present. Dealing with outliers has not been a focus in this thesis, so we will present the number of outliers per distribution variable and then discuss ways to deal with outliers in Section 6.

Outliers are defined as values that lie outside of the whiskers, i.e., outside of

$$[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)].$$

We present the number of outliers for $R = 1.6$ using $\mathcal{L}_1$ in Tables 5 & 6. For all outliers using $\mathcal{L}_2$ and $R = 6$ and $R = 4$, see Appendix B.3.

Table 5: The number of outliers in the residual and response distributions for all variables in the HEP1 dataset after compression with $R = 1.6$ using $\mathcal{L}_1$. These are values that lie outside of $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$

Variable ($R = 1.6$)	Response		Residual	
	Nr. of outliers	Percentage [%]	Nr. of outliers	Percentage [%]
p_T	70040	9.781	64565	9.016
η	81531	11.385	10280	1.436
ϕ	92014	12.849	25024	3.494
mass	88691	12.385	57699	8.057
mJetArea	37273	5.205	24882	3.475
mChargedHadronEnergy	0	0	70727	9.877
mNeutralHadronEnergy	0	0	41250	5.76
mPhotonEnergy	269723	37.665	22750	3.177
mElectronEnergy	0	0	15244	2.129
mMuonEnergy	0	0	11642	1.626
mHFHadronEnergy	0	0	66372	9.268
mHFEMEnergy	0	0	19627	2.741
mChargedHadronMultiplicity	0	0	5	0.001
mNeutralHadronMultiplicity	0	0	0	0
mPhotonMultiplicity	0	0	0	0
mElectronMultiplicity	0	0	0	0
mMuonMultiplicity	0	0	0	0
mHFHadronMultiplicity	0	0	0	0
mHFEMMultiplicity	0	0	1	0.0
mChargedEmEnergy	0	0	15288	2.135
mChargedMuEnergy	0	0	11659	1.628
mNeutralEmEnergy	97913	13.673	45157	6.306
mChargedMultiplicity	0	0	5	0.001
mNeutralMultiplicity	0	0	0	0

Table 6: The number of outliers in the residual and response distributions for all variables in the HEP2 dataset after compression with $R = 1.6$ using $\mathcal{L}_1$. These are values that lie outside of $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$

Variable ($R = 1.6$)	Response		Residual	
	Nr. of outliers	Percentage [%]	Nr. of outliers	Percentage [%]
p_T	34537	7.004	51541	10.453
η	77681	15.754	57474	11.656
ϕ	80037	16.232	52775	10.703
mass	78890	15.999	75618	15.336
m_EmEnergy	74160	15.04	58028	11.768
m_HadEnergy	64795	13.141	54272	11.007
m_InvisibleEnergy	0	0	53806	10.912
m_AuxiliaryEnergy	0	0	81402	16.509

5.6 Applications in other fields

As previously stated, Baler's performance has also been tested in different scientific fields. Computational Fluid Dynamics (CFD) is the other field used as a testing point. This field relies heavily on simulations, and the file sizes can quickly get out of hand. With the help of a Convolutional Autoencoder, a simple simulation of air flowing over a wall-mounted cube has been compressed and decompressed. To make this case as general and straightforward as possible, we have only considered a slice of a 3D simulation, making the simulation compressed essentially a figure of 2 dimensions.

In Figure 26, the convolutional model has been trained to compress the original file with $R = 88$ and has resulted in a maximum/minimum velocity difference between the original and reconstruction of approximately $\pm 4 \times 10^{-5}$ mm/s, as seen in Figure 26c.

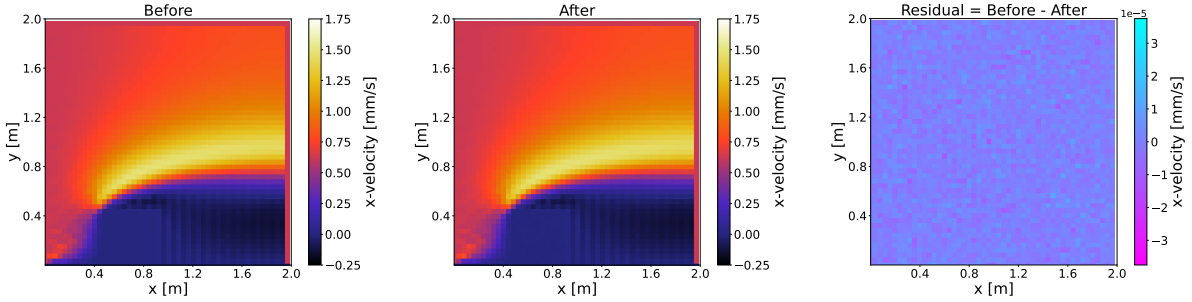


Figure 26: A Computational Fluid Dynamics simulation showing x-component of air velocity before compression (a), after decompression (b), and the difference between the two (c).

This compression ratio and file size reduction are considered adequate for CFD, but this method has one major drawback. Convolutional models are much larger than SAE models, so although this file has been compressed with $R = 88$, $R^* \ll R$, meaning that the model size is much greater than the input file size.

Despite this, compression of CFD files has brought enlightenment to a different feature, namely that of comparison to `gzip`. From our previous results, `gzip` is currently beating Baler's compression if we want the evaluation metric to lie within what is considered acceptable. However, this is not true for CFD data. Using `gzip` on the CFD dataset from Figure 26, a compression ratio of $R = 2.2$ is achieved. This is significantly worse than what Baler can compress, but again, concerning the auxiliary files (mainly the Convolutional autoencoder model), we obtain $R^* < 1$. The reason for this is discussed in Section 6.

6 Discussion

To begin with, it is worth discussing the training times and how computationally heavy the process of training an autoencoder is. Referring back to Table 4, there is an immediate difference between training on a CPU versus a GPU. Today, a modern GPU is not uncommon, so training a neural network on a GPU is not unheard of. However, matching CUDA with the correct hardware can be tricky. CUDA is software developed by NVIDIA, with no availability to run on other GPU types, such as AMD or M1. This limits CUDA and GPU training to be *mostly* viable on Windows machines, which commonly is not the most compatible with UNIX-developed open software. This issue can however be solved with an implementation via `Docker` and `Apptainer/Singularity`. Given the above points, a cluster is the best place to train any neural network. A GPU or CPU cluster is a highly viable solution, allowing for "non-local" training.

Discussing the compression ratios and comparing to `gzip`, we notice that Balers performance with $R = 1.6$ is significantly better than with $R = 6$ for HEP1, and the same goes for HEP2 with $R = 4$. The reason for this is believed to be the number of repeated values in HEP1 and HEP2, making `gzip`'s compression much more effective. We can see this effect further from two perspectives. The first one is that applying `gzip` on an already compressed file barely reduces the file size, and secondly, CFD Baler outperforms `gzip`. By investigating the compressed files, one finds no duplicate values, which is an effect of the autoencoder's ability to maximize information stored in a selected space. Looking at the CFD dataset, it, first of all, contains a lot fewer values than both HEP files and rarely any duplicates. This drastically decreases `gzip`'s effectiveness, making an autoencoder much more viable for compression. Despite this, we also highlight the downside of using Convolutional Autoencoders: the model sizes are much larger. This is an effect highlighted when compressing small datasets. However, a positive is that the model size should not increase linearly with the input file size due to the fixed amount of weights, making this problem relatively easily solved.

Moving on to Baler's performance compared to PCA, we first see some similarities. For HEP1 with $R = 6$ in Figure 23a, apart from the integer reconstruction and the two angular distributions, none of the reconstructions are acceptable. This effect is much more prominent in PCA, where both η and ϕ are entirely incorrectly reconstructed, but both mass and p_T are better rebuilt than compared to Baler. Looking at Figure 23b, we see that PCA performs much better in every Baler variable, except for the two angular distributions. We also direct attention to Appendix B.2, where a full view of the reconstructions is present. One of the main points worth noting is that for $R = 1.6$, all variables are overall reconstructed well. However, what is especially notable for PCA in HEP2 is that $R = 1.6$ compresses eight columns down to five, and by looking at Figure 31, we see five variables reconstructed well and three severely flawed reconstructed variables. What happens here is that PCA correctly approximates the five variables in the latent space using the identity matrix, making the reconstruction near-perfect. This feature does not hold up for

compression to lower dimensions, as seen in the rest of the figures in Appendix B.2.

Continuing the Baler versus PCA discussion, the mass peak fits in Figure 24 and 25 tell the same story. PCA has a hard time reconstructing irregular shapes in distributions, while autoencoder compression can reconstruct irregularities for smaller compression ratios.

The many zero values available in the distributions are a common theme that presents itself and is a critical factor to why the four-momentum is presented as responses. For close to every variable, the number of jets with a variable equal to zero is often one or two orders of magnitude larger than the rest of the values. This leads to very large response values which shifts the variable means, and can be seen most prominently for the mass variable in Figures 21, 22 and 23.

Another feature that PCA is considerably worse at is reconstructing irregularities. It is clear from both the mass distributions in Section B.2 and the fits in Figures 24 25 that autoencoder compression can reconstruct irregular shapes much better than PCA. It is also worth noting that the masses obtained from the fits originally were inaccurate. However, this should not affect the comparison since we aim to reconstruct the values of the original distribution, incorrect or not. We can also not safely draw any conclusions regarding the $W^{\pm}$ peak for Baler's $R = 4$ reconstruction in Figure 25, as it visually looks to be non-existent. Finally, the case might be that of $R = 4$ being too large of a compression ratio for Baler to accurately reconstruct such minor irregularities as these peaks are (very few events make up these peaks), but nonetheless, they need to be reconstructed well for particle physics applications.

Furthermore, discussing the outliers, any statistical data analyses will be skewed due to the large number of values in each distribution and overall asymmetrical distributions. In practice, a very selective part of the distributions is looked at, and most of the distribution is neglected. However, removing the majority of the data not generally analyzed is not an option since different analyses use different parts of the distributions. Since we have only considered the cases of compressing, to some extent, *general* HEP data, we have decided not to look further into how outliers are treated but notice them as existent and illuminate that the box-plot distributions miss some information. The plots in Appendix B.2 are referred to for a more qualitative view of the reconstruction results.

Previously the difference between offline and online compression was outlined. However, we have not discussed the online case further as we see it as an extension of offline compression, and the differences in architecture setup and training goals differ substantially. With this in mind, testing online compression is easy, as one only has to compress another dataset with the exact input dimensions. This has been tested, and the results are far from what is considered sufficiently good, but this is not unexpected and more studies are needed.

From a neural network point of view, optimal reconstruction conditions would theoretically take place if no data batching took place. As discussed in the theory section, to decrease training time, the data is batched, and the network is trained batch-wise. Training the network on the full dataset every iteration would be optimal since the network would see

the entire dataset simultaneously, but this was considered too computationally heavy for our computing nodes on the cluster.

7 Conclusions & Outlook

7.1 Conclusions

In conclusion, an open-source tool has been developed that is capable of training autoencoders to compress and decompress data from two fields; HEP data containing individual jets and CFD toy data displaying the flow velocity of a fluid over a stationary cube. During this thesis, the HEP case has been in focus. Two HEP datasets with different numbers of variables and dissimilar features have been compressed and decompressed. One of the datasets contained 24 variables and was compressed to a four or 15-dimensional latent space. The other dataset consisted of eight variables and was compressed to five or two latent dimensions. These values were motivated by how much `gzip` could compress the datasets, and further, the compression performance was compared to PCA.

The reconstruction performance was evaluated by looking at the difference and relative difference between the original and reconstructed distributions on an event level and a distributional level. In addition, the loss values of the trained autoencoder models were considered.

At a compression ratio of 1.6, after approximately three hours of training on a GPU, we successfully trained an autoencoder model to achieve reconstructed distributions considered sufficiently good for HEP data. Furthermore, the auxiliary data needed is determined to be negligible. This compression ratio is, however, not enough to beat, e.g., `gzip`, but the results for a compression ratio capable of competing with `gzip` are not distant. On the other hand, for CFD compression ratios, we beat `gzip` substantially, but the auxiliary data file size issue becomes a major issue.

7.2 Outlook

Several extension studies are possible based on this work's conclusions and findings. First, an exploration of metrics determining whether or not different HEP datasets are appropriate for autoencoder compression would be interesting. From Ref. [68], the implementation of so-called *Coefficient of Variation* has been looked into. Based on the same paper, the compression ratios achieved for datasets in diverse scientific fields have been intriguing. Finally, the same study has implemented error-bounded compression, which is necessary for using this tool in HEP beyond the prototype stage and ought to be studied further.

Concerning auxiliary file sizes, the HEP models are sufficiently lightweight, but further improvements must be made to the convolutional models to make the applications to datasets

of "higher dimension" viable. Based on Ref [68], sufficiently high actual compression ratios have been developed to compress "higher dimensional data," such as images.

Online compression is also a subject of future work, with plenty of potentials available within the area.

References

[1]The ATLAS Collaboration, **3**, S08003 (2008).

[2]A. M. M. Scaife, "Big telescope, big data: towards exascale with the square kilometre array", Phil. Trans. R. Soc. A. **378** (2020).

[3]M. Khan, X. Wu, X. Xu, and W. Dou, "Big data challenges and opportunities in the hype of industry 4.0", in 2017 ieee international conference on communications (icc) (2017), pp. 1–6.

[4]P. Calafiura, J. Catmore, D. Costanzo, and A. Di Girolamo, *ATLAS HL-LHC Computing Conceptual Design Report*, tech. rep. (CERN, Geneva, 2020).

[5]M. Ghosh, T. Ghosh, and M. Y. Hirose, "Poisson counts, square root transformation and small area estimation", Sankhya B **84**, 449–471 (2022).

[6]CMS collaboration, "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC", Physics Letters B **716**, 30–61 (2012).

[7]ATLAS collaboration, "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC", Physics Letters B **716**, 1–29 (2012).

[8]Y. Kim, "Spin and statistics of elementary particles", in *Mathematical foundations of quantum theory*, edited by A. Marlow (Academic Press, 1978), pp. 347–349.

[9]P. W. Higgs, "Broken symmetries and the masses of gauge bosons", Phys. Rev. Lett. **13**, 508–509 (1964).

[10]C. Burgard, *Example: standard model physics*, (2016) `https://texample.net/tikz/examples/model-physics/` (visited on 03/22/2021).

[11]F. Halzen and A. D. Martin, *QUARKS AND LEPTONS: AN INTRODUCTORY COURSE IN MODERN PARTICLE PHYSICS* (1984).

[12]M. Cacciari, G. P. Salam, and G. Soyez, "The anti-kt jet clustering algorithm", Journal of High Energy Physics **2008**, 063–063 (2008).

[13]C. Y. Wong, *Introduction to high-energy heavy ion collisions* (1995), pp. 17–25.

[14]G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. A. Khalek, A. Abdelalim, O. Abdinov, and R. A. et al, "Observation of a new particle in the search for the standard model higgs boson with the ATLAS detector at the LHC", Physics Letters B **716**, 1–29 (2012).

[15]The ALICE Collaboration, **3**, S08002 (2008).

[16]The CMS Collaboration, **3**, S08004 (2008).

[17]The LHCb Collaboration, **3**, S08005 (2008).

[18]A. Sirunyan, A. Tumasyan, W. Adam, E. Asilar, T. Bergauer, J. Brandstetter, E. Brondolin, and M. Dragicevic, "Particle-flow reconstruction and global event description with the CMS detector", Journal of Instrumentation **12**, P10003–P10003 (2017).

[19]C. Kourkoumelis and S. Vourakis, "Hypatia—an online tool for atlas event visualization", Physics Education **49**, 21 (2014).

[20]The ATLAS Collaboration, *Trigger and data acquisition system*, [Online; accessed 08-April-2023].

[21]Z. Jinlong, *Atlas data aqcusition*, `http://cdsweb.cern.ch/record/1239011/files/ATL-DAQ-PROC-2010-005.pdf`, [Accessed 24-Apr-2023], 2010.

[22]W. Buttinger, "The ATLAS Level-1 Trigger System", Journal of Physics: Conference Series **396**, 012010 (2012).

[23]J. Stelzer and (. behalf ofthe ATLAS collaboration), "The atlas high level trigger configuration and steering: experience with the first 7 tev collision data", Journal of Physics: Conference Series **331**, 022026 (2011).

[24]J. Elmsheuser, C. Anastopoulos, J. Boyd, J. Catmore, H. Gray, J. A. Mcfayden, C. Meyer, A. Sfyrla, J. Strandberg, K. Suruliz, and T. Theveneaux-Pelzer (ATLAS), "Evolution of the ATLAS analysis model for Run-3 and prospects for HL-LHC", (2019).

[25]I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, P. Canal, D. Casadei, O. Couet, V. Fine, L. Franco, G. Ganis, A. Gheata, D. G. Maline, M. Goto, J. Iwaszkiewicz, A. Kreshuk, D. M. Segura, R. Maunder, L. Moneta, A. Naumann, E. Offermann, V. Onuchin, S. Panacek, F. Rademakers, P. Russo, and M. Tadel, "Root — a c++ framework for petabyte data storage, statistical analysis and visualization", Computer Physics Communications **180**, 40 YEARS OF CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures, 2499–2512 (2009).

[26]"Ieee standard for floating-point arithmetic", IEEE Std 754-2019 (Revision of IEEE 754-2008), 1–84 (2019).

[27]J.-l. Gailly, *gzip: The data compression program*, Apr. 2022.

[28]Smith, Lindsay I, *A tutorial on principal components analysis (computer science technical report no. oucs-2002-12)*, [Retrieved from `http://hdl.handle.net/10523/7534`].

[29]B. Shapiro and M. Shapiro, "On eigenvalues of rectangular matrices", Proceedings of the Steklov Institute of Mathematics **267**, 248–255 (2009).

[30]Chen, Yaxiong, Huang, Zhangcan, Sun, Hao, Chen, Mengying, and Tan, Hua, "Lossy image compression using pca and contourlet transform", MATEC Web of Conferences **54**, 08002 (2016).

[31]F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.", Psychological review **65 6**, 386–408 (1958).

[32]Izaak Neutelings, *Neural networks*, [Online; accessed 27-February-2023; Last edited 11 September 2022], 2021.

[33]F. Informatik, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies", A Field Guide to Dynamical Recurrent Neural Networks (2003).

[34]L. Lu, "Dying ReLU and Initialization: Theory and Numerical Examples", Communications in Computational Physics **28**, 1671–1706 (2020).

[35]I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, `http://www.deeplearningbook.org` (MIT Press, 2016), pp. 106–111.

[36]C. Villani, *Optimal transport - old and new*, `https://doi.org/10.1007/978-3-540-71050-9` (Springer Berlin, 2009), p. 105.

[37]P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", Nature Methods **17**, 261–272 (2020).

[38]S. Ruder, *An overview of gradient descent optimization algorithms*, 2016.

[39]M. Ohlsson and P. Edén, *Introduction to Artificial Neural Networks and Deep Learning*, Lecture Notes to the course FYTN14/EXTQ40/NTF005F (2021), p. 17.

[40]Kurtik Pykes, *Gradient descent*, [Online; accessed 11-March-2023; Last edited 13 August 2020], 2020.

[41]A. Patel and R. K. Rama, "An overview of boltzmann machine and its special class", 3 (2020).

[42]D. P. Kingma and J. Ba, *Adam: a method for stochastic optimization*, 2014.

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

[44]Åstrand, Sten, *Autoencoder Compression in High Energy Physics*, eng, Student Paper, 2022.

[45]Wikipedia contributors, *Mean squared error*, [Online; accessed 28-March-2023], 2022.

[46]A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Pytorch: an imperative style, high-performance deep learning library*, 2019.

[47]*ReduceLROnPlateau 2014; PyTorch 2.0 documentation — pytorch.org*, `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html`, [Accessed 09-Apr-2023].

[48]M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks", AIChE Journal **37**, 233–243 (1991).

[49]D. George, *Deep Autoencoders for ATLAS Data Compression - George Dialektakis - Google Summer of Code 2021 Project*, version 1, Sept. 2021.

[50]J. E. V. Ferreira, M. T. S. Pinheiro, W. R. S. dos Santos, and R. da Silva Maia, "Graphical representation of chemical periodicity of main elements through boxplot", https://doi.org/10.1016/j.eq.2016.04.007 (2016).

[51]A. Ekman, F. Bengtsson, O. Woolland, A. Gallén, M. C. Santasmasas, P. Jawahar, C. Doglioni, and S. Xu, *Baler-collaboration/baler: v1.0.0*, version v1.0.0, Apr. 2023.

[52]*Turing-UoM Sandpit: Data Science and AI for Translational Digital Health — events.manchester.ac.uk* https://events.manchester.ac.uk/event/event:s2m-l70ax9u5-6bblsa/turinguom-sandpit-data-science-and-ai-for-translational-digital-health, [Accessed 24-Apr-2023], 2022.

[53]F. Bengtsson, C. Doglioni, P. A. Ekman, A. Gallén, P. Jawahar, A. Orucevic-Alagic, M. C. Santasmasas, N. Skidmore, and O. Woolland, *Baler – machine learning based compression of scientific data*, 2023.

[54]F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: machine learning in Python", Journal of Machine Learning Research **12**, 2825–2830 (2011).

[55]*Creative Commons 2014; CC0 1.0 Universal — creativecommons.org*, https://creativecommons.org/publicdomain/zero/1.0/, [Accessed 10-Apr-2023].

[56]CMS collaboration (2017), "JetHT primary dataset in AOD format from Run of 2012 (/JetHT/Run2012B-22Jan2013-v1/AOD). CERN Open Data Portal.", [Dataset: 00992A80-DF70-E211-9872-0026189437FE.root], 10.7483/OPENDATA.CMS.KL8H.HFVH.

[57]CMS Collaboration (2021), "Simulated dataset TTbarDMJets_EFT_M-50_TuneCUETP8M1_13TeV-madgraphMLM-pythia8 in MINIAODSIM format for 2015 collision data. CERN Open Data Portal.", [Dataset: 00CE5A28-41B8-E511-9700-38EAA78E2C94.root], 10.7483/OPENDATA.CMS.FOP7.PI6B.

[58]F. A. Dreyer and S. Schramm, "Introduction to Jet Substructure https://indi.to/53NSG", in (2018).

[59]Kildetoft, Love, *Evaluation of float-truncation based compression techniques for the ATLAS jet trigger*, eng, Student Paper, 2021.

[60]Wallin, Erik, *Tests of Autoencoder Compression of Trigger Jets in the ATLAS Experiment*, eng, Student Paper, 2020.

[61]H. Gupta, C. Doglioni, B. Ravina, A. Boveia, L. Heinrich, E. Wallin, and E. Wulff, *Deep-compression for High Energy Physics data - Honey Gupta - Google Summer of Code 2020 Project*, Sept. 2020.

[62]C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy", Nature **585**, 357–362 (2020).

[63]github, *Github*, `https://github.com/`, 2008.

[64]D. Merkel, "Docker: lightweight linux containers for consistent development and deployment", Linux journal **2014**, 2 (2014).

[65]Poetry Collaboration, *Poetry - python packaging and dependency management made easy*, `https://python-poetry.org/`, 2018.

[66]*Aurora — LUNARC*, [Accessed: 2023-04-11], 2019.

[67]R. L. Workman et al. (Particle Data Group), "Review of Particle Physics", PTEP **2022**, 083C01 (2022).

[68]T. Liu, J. Wang, Q. Liu, S. Alibhai, T. Lu, and X. He, "High-ratio lossy compression: exploring the autoencoder to compress scientific data", IEEE Transactions on Big Data **9**, 22–36 (2023).

A Data processing and Variables

A.1 Variable Descriptions

Short description of all the variables in the two datasets together with the unit of the variable. All variables without a footnote are only available in the HEP1 dataset.

- p_T [GeV] - Jet momentum in the transverse place[7]
- η [arb.] - Pseudorapidity[4]
- ϕ [rad] - Azimuthal angle[4]
- m [GeV] - Total mass of the Jet[4]
- $mJetArea$ [length^2] - The space occupied by the Jet
- $mChargedHadronEnergy$ [GeV] - Total amount of energy from charged hadrons
- $mNeutralHadronEnergy$ [GeV] - Total amount of energy from neutral hadrons
- $mPhotonEnergy$ [GeV] - Total amount of energy from photons only
- $mElectronEnergy$ [GeV] - Total amount of energy from electrons only
- $mMuonEnergy$ [GeV] - Total amount of energy from muons only
- $mHFHadronEnergy$ [GeV] - Total amount of energy from high frequency hadrons
- $mHFEMEnergy$ [GeV] - Total amount of energy from high frequency electromagnetic processes
- $mChargedHadronMultiplicity$ [counts] - Average number of charged hadrons produced
- $mNeutralHadronMultiplicity$ [counts] - Average number of neutral hadrons produced
- $mPhotonMultiplicity$ [counts] - Average number of photons produced
- $mElectronMultiplicity$ [counts] - Average number of electrons produced
- $mMuonMultiplicity$ [counts] - Average number of muons produced
- $mHFHadronMultiplicity$ [counts] - Average number of High Frequency hadrons produced

[7]Variable available in both HEP1 & HEP2

- $mHFEMMultiplicity$ [counts] - Average number of High Frequency Electromagnetic particles produced
- $mChargedEmEnergy$ [GeV] - Total amount of energy from charged electromagnetic processes
- $mChargedMuEnergy$ [GeV] - Total amount of energy from charged muons
- $mNeutralEmEnergy$ [GeV] - Total amount of energy from Neutral electromagnetic processes
- $mChargedMultiplicity$ [counts] - Average number of charged particles produced
- $mNeutralMultiplicity$ [counts] - Average number of neutral particles produced
- $mInvisibleEnergy$ [GeV] - Total amount of energy by particles not depositing all of its energy in the detector[8]
- $mAuxiliaryEnergy$ [GeV] - Total amount of energy by particles not depositing all of its energy in the detector[5]
- $EmEnergy$ [GeV] - Total amount of energy from electromagnetic processes[5]
- $HadEnergy$ [GeV] - Total amount of energy from hadronic processes[5]

[8]Variable only available in HEP2

A.2 Normalized distributions

Figure 27 & 28 shows both the normalzied and un-normalized variable distributions before compression.

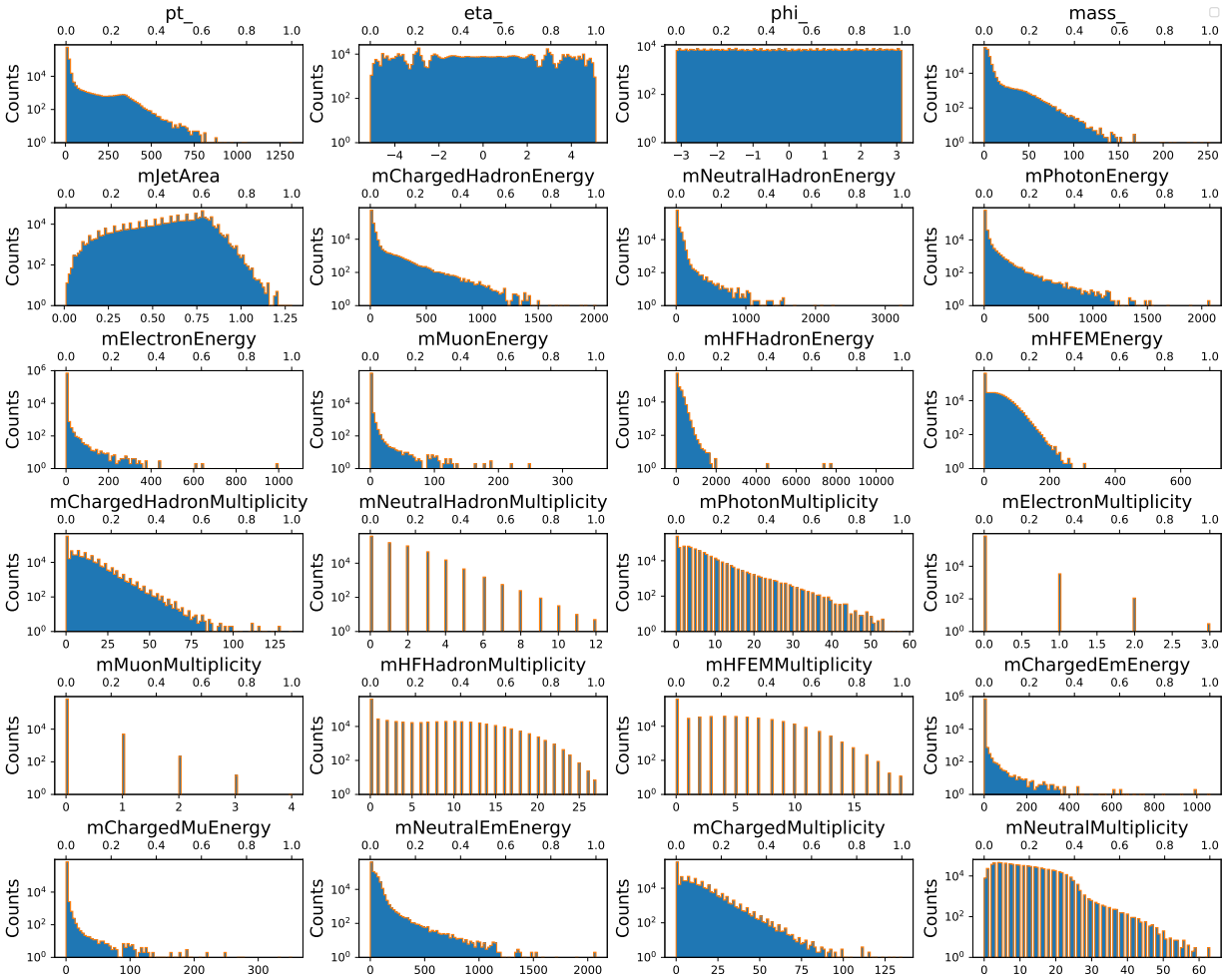


Figure 27: Distributions of all 24 variables in the HEP1 dataset, both un-normalzied (blue, bottom scale) and normalized (orange, top scale).

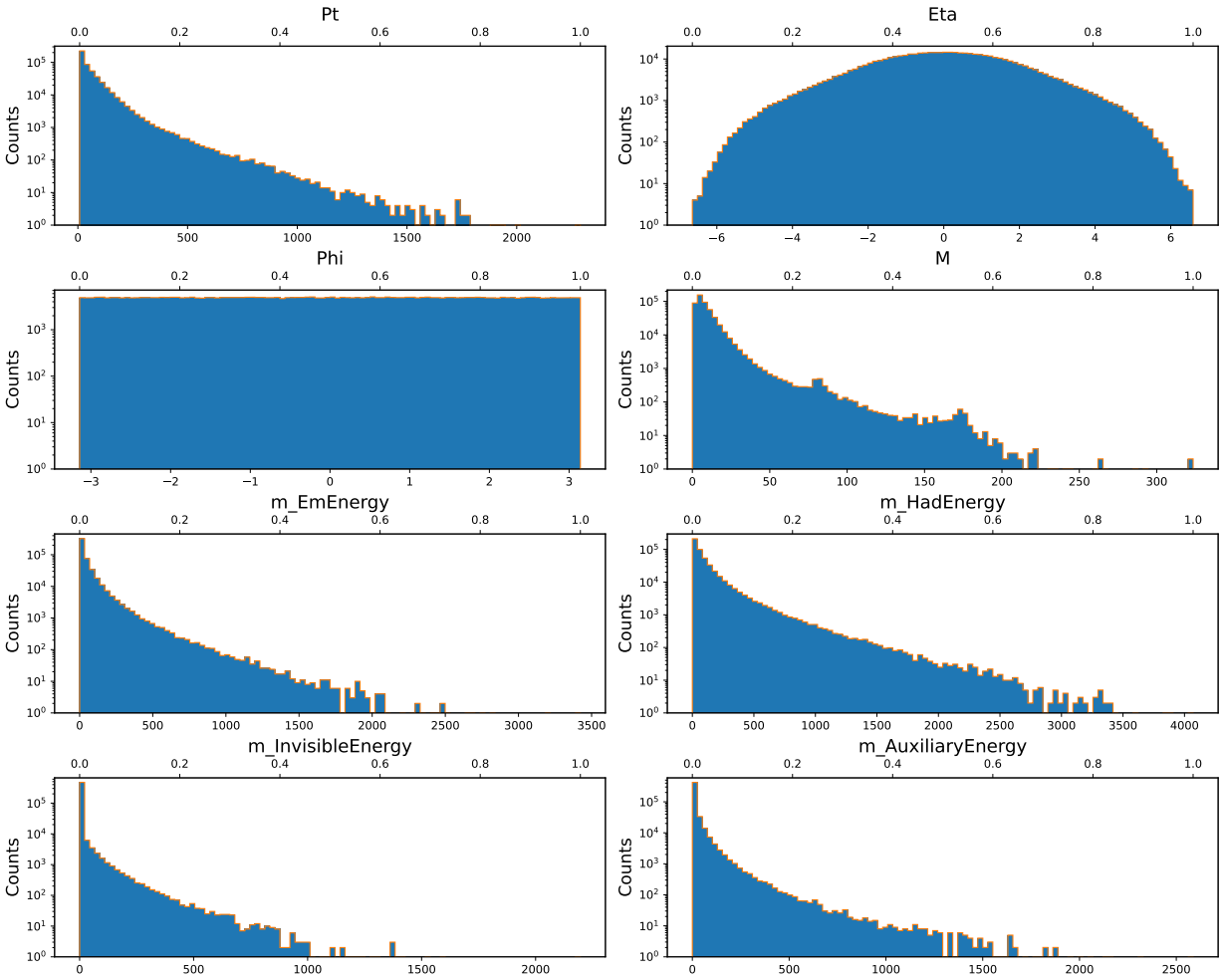


Figure 28: Distributions of all 8 variables in the HEP2 dataset, both un-normalzied (blue, bottom scale) and normalized (orange, top scale).

B Complimentary Results

B.1 Loss Plots

Here, further loss plots are presented at different compression ratios. Figure 29 shows the loss plots for HEP1 and HEP2 at $R = 6$ and $R = 4$ respectively. For both HEP2 training runs, early stopping canceled the training.

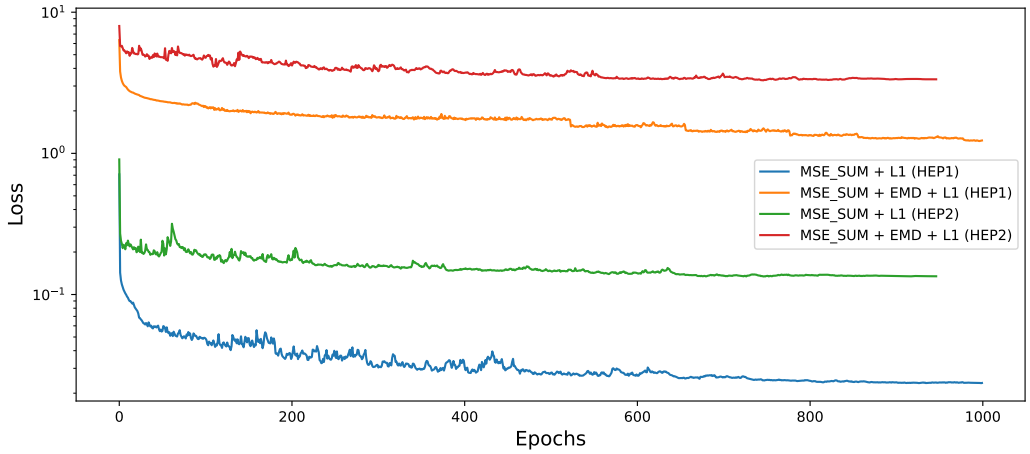


Figure 29: Loss plots for two different loss functions and two different datasets after training to compress HEP1 to $R = 6$ and HEP2 to $R = 4$.

B.2 Reconstructed Distributions

In Figure 30 and Figure 31, a comparison between all HEP1 reconstructions for $R = 1.6$ is compared to the original distribution. Figure 32 and Figure 33 similarly present all reconstructions but for HEP1 and HEP2 at $R = 6$ and $R = 4$ respectively..

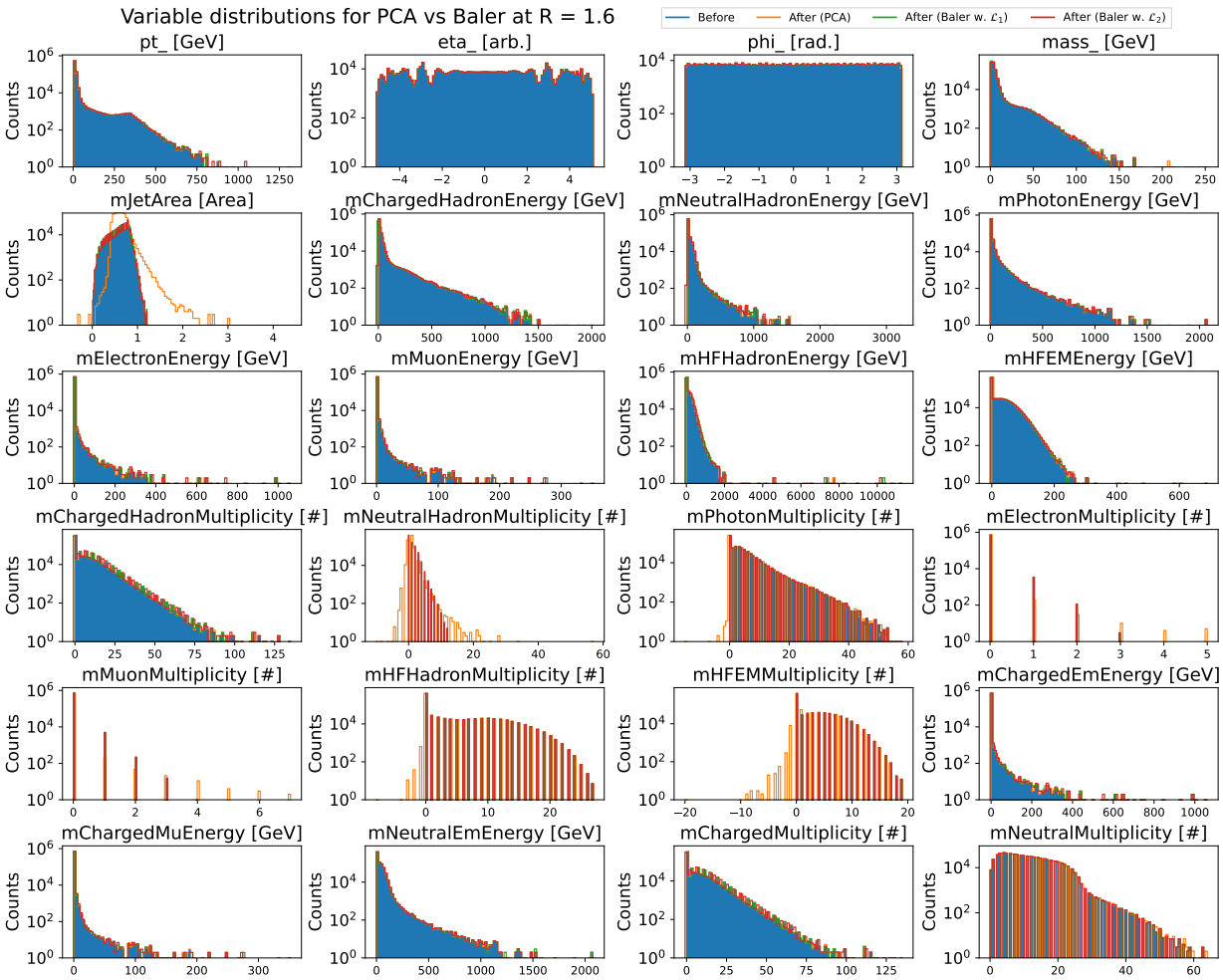


Figure 30: Variable distributions for the HEP1 dataset. In the plot, the original distribution (blue) is present together with PCA reconstruction (orange), $\mathcal{L}_1$ reconstruction (green), and $\mathcal{L}_2$ reconstruction (red); all after compressing to $R = 1.6$.

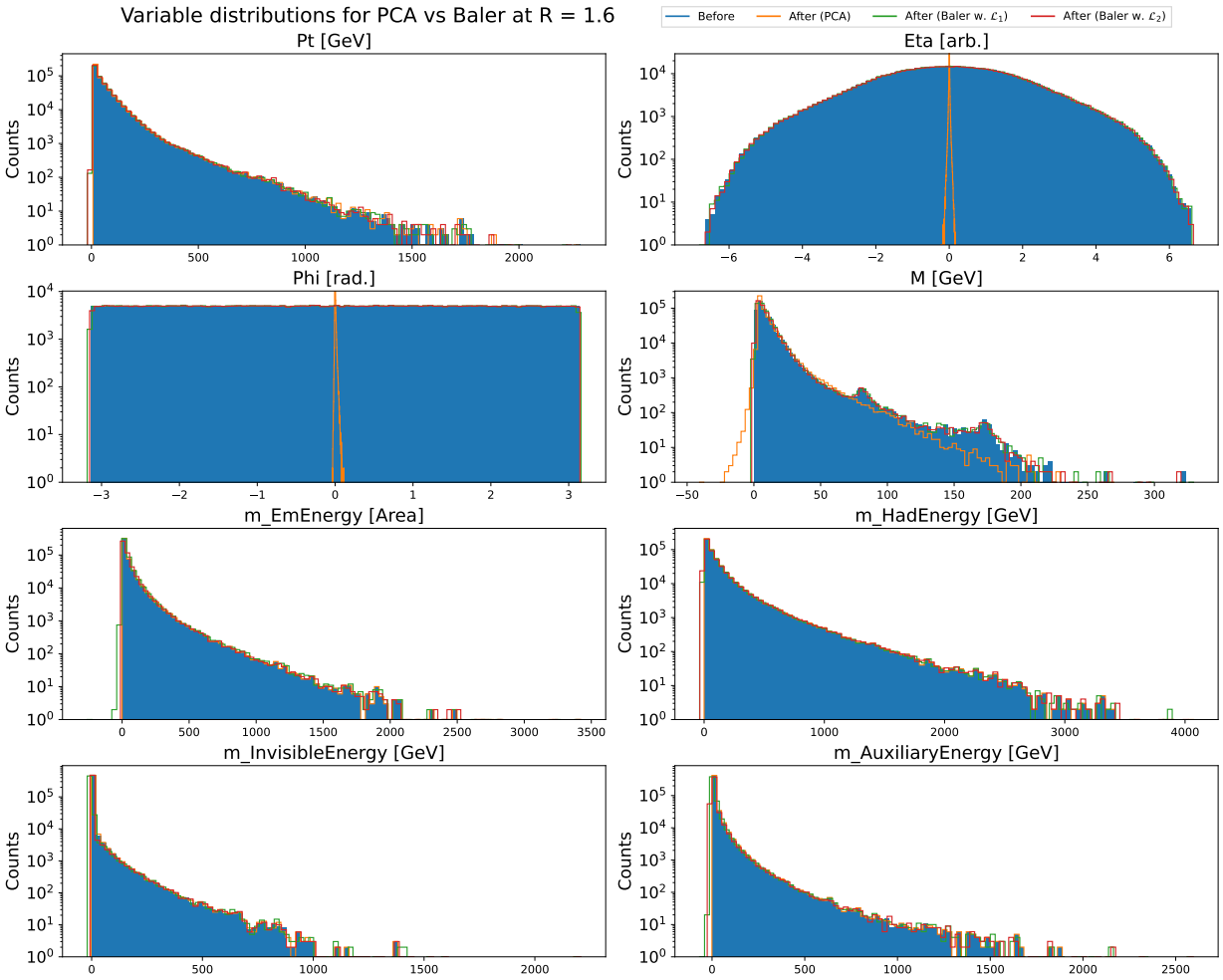


Figure 31: Variable distributions for the HEP2 dataset. In the plot, the original distribution (blue) is present together with PCA reconstruction (orange), $\mathcal{L}_1$ reconstruction (green), and $\mathcal{L}_2$ reconstruction (red); all after compressing to $R = 1.6$.

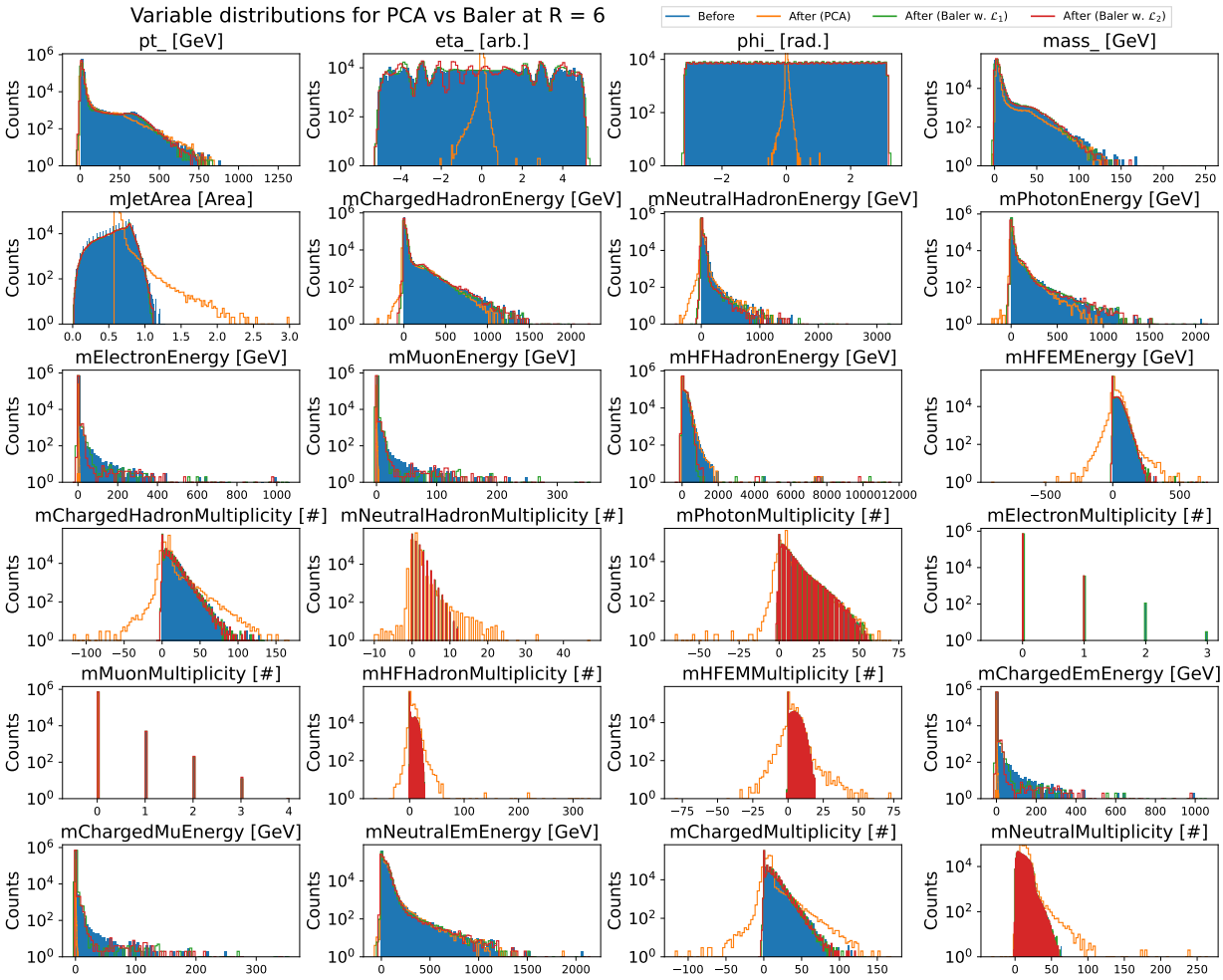


Figure 32: Variable distributions for the HEP1 dataset. In the plot, the original distribution (blue) is present together with PCA reconstruction (orange), $\mathcal{L}_1$ reconstruction (green), and $\mathcal{L}_2$ reconstruction (red); all after compressing to $R = 6$.

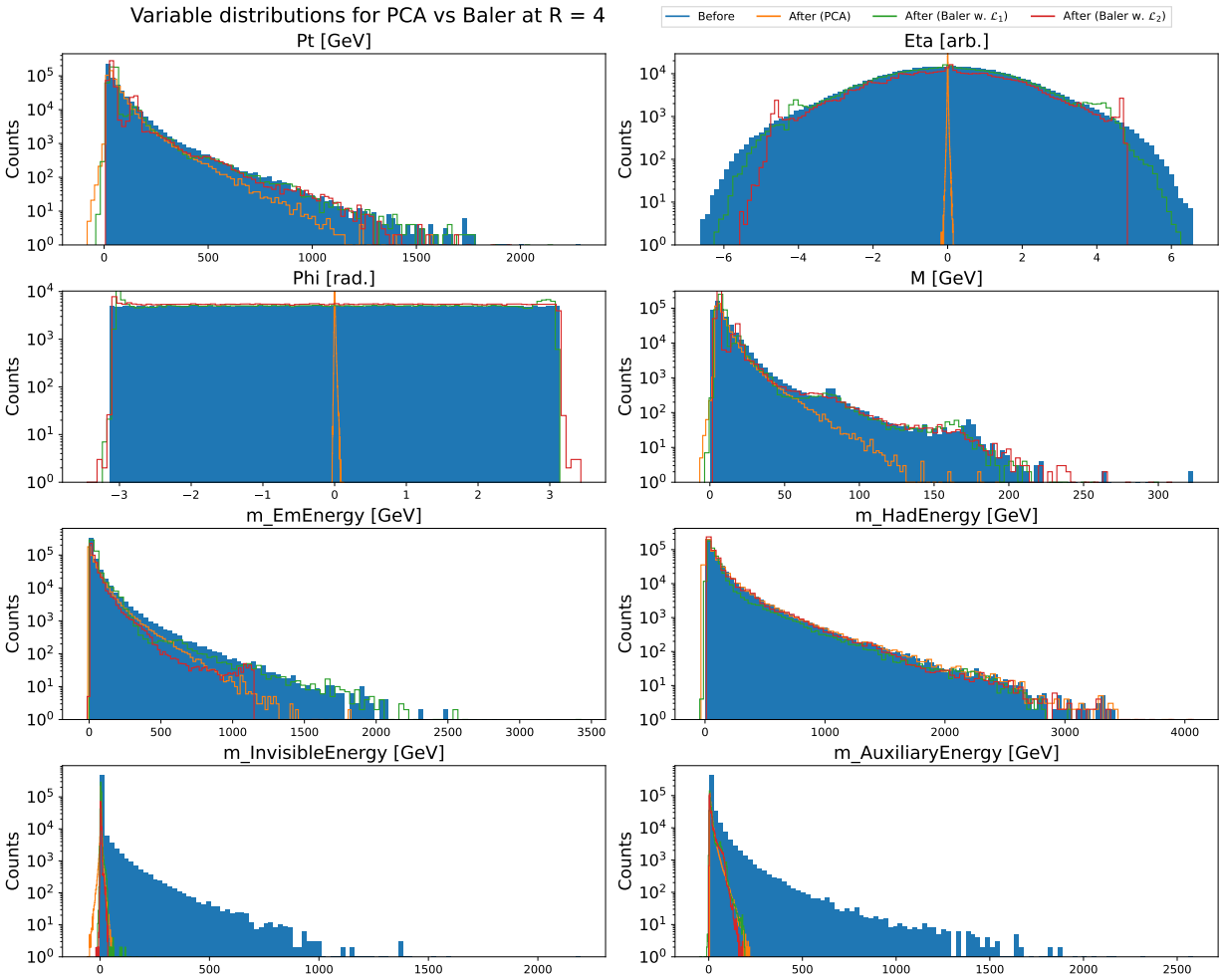


Figure 33: Variable distributions for the HEP2 dataset. In the plot, the original distribution (blue) is present together with PCA reconstruction (orange), $\mathcal{L}_1$ reconstruction (green), and $\mathcal{L}_2$ reconstruction (red); all after compressing to $R = 4$.

B.3 Outliers

For the remaining $\mathcal{L}_1$ outliers, see Table 7 for HEP1 with $R = 6$ and Table 8 for HEP2 with $R = 4$.

Moving on to $\mathcal{L}_2$, Table 9 contains the HEP1 outliers for $R = 1.6$, while Table 10 presents the outliers for HEP2 with $R = 1.6$. See Tables 11 and 12 for $\mathcal{L}_2$, $R = 6$ and $\mathcal{L}_2$,$R = 4$ respectively.

Table 7: The number of outliers in the residual and response distributions for all variables in the HEP1 dataset after compression with $R = 6$ using $\mathcal{L}_1$. These are values that lie outside of $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$

Variable ($R = 6$)	Response		Residual	
	Nr. of outliers	Percentage [%]	Nr. of outliers	Percentage [%]
p_T	17567	2.453	61970	8.654
η	101610	14.189	33042	4.614
ϕ	106095	14.816	47575	6.644
mass	25869	3.612	57870	8.081
mJetArea	92547	12.924	90573	12.648
mChargedHadronEnergy	0	0	126996	17.734
mNeutralHadronEnergy	0	0	158812	22.177
mPhotonEnergy	248063	34.641	148036	20.672
mElectronEnergy	0	0	44619	6.231
mMuonEnergy	0	0	40738	5.689
mHFHadronEnergy	0	0	203269	28.385
mHFEMEnergy	0	0	289079	40.368
mChargedHadronMultiplicity	0	0	293699	41.013
mNeutralHadronMultiplicity	0	0	14056	1.963
mPhotonMultiplicity	0	0	92390	12.902
mElectronMultiplicity	0	0	21	0.003
mMuonMultiplicity	0	0	48	0.007
mHFHadronMultiplicity	0	0	5362	0.749
mHFEMMultiplicity	0	0	522	0.073
mChargedEmEnergy	0	0	44672	6.238
mChargedMuEnergy	0	0	40596	5.669
mNeutralEmEnergy	69705	9.734	115238	16.092
mChargedMultiplicity	0	0	293696	41.013
mNeutralMultiplicity	0	0	86373	12.062

Table 8: The number of outliers in the residual and response distributions for all variables in the HEP2 dataset after compression with $R = 4$ using $\mathcal{L}_1$. These are values that lie outside of $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$

Variable ($R = 4$)	Response		Residual	
	Nr. of outliers	Percentage [%]	Nr. of outliers	Percentage [%]
p_T	34745	7.047	30114	6.107
η	76042	15.422	82546	16.741
ϕ	74013	15.01	83885	17.012
mass	32577	6.607	25476	5.167
m_EmEnergy	71528	14.506	63931	12.966
m_HadEnergy	51505	10.446	37945	7.696
m_InvisibleEnergy	0	0	41812	8.48
m_AuxiliaryEnergy	0	0	90542	18.363

Table 9: The number of outliers in the residual and response distributions for all variables in the HEP1 dataset after compression with $R = 1.6$ using $\mathcal{L}_2$. These are values that lie outside of $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$

Variable ($R = 1.6$)	Response		Residual	
	Nr. of outliers	Percentage [%]	Nr. of outliers	Percentage [%]
p_T	100559	14.043	85543	11.946
η	97529	13.619	48269	6.741
ϕ	109236	15.254	41215	5.755
mass	131718	18.394	102011	14.245
mJetArea	70750	9.88	60442	8.44
mChargedHadronEnergy	324796	45.356	80979	11.308
mNeutralHadronEnergy	0	0	128292	17.915
mPhotonEnergy	265850	37.124	60135	8.398
mElectronEnergy	0	0	42584	5.947
mMuonEnergy	0	0	56905	7.946
mHFHadronEnergy	0	0	127988	17.873
mHFEMEnergy	0	0	111772	15.608
mChargedHadronMultiplicity	0	0	241	0.034
mNeutralHadronMultiplicity	0	0	0	0
mPhotonMultiplicity	0	0	2	0.0
mElectronMultiplicity	0	0	0	0
mMuonMultiplicity	0	0	0	0
mHFHadronMultiplicity	0	0	44	0.006
mHFEMMultiplicity	0	0	5	0.001
mChargedEmEnergy	0	0	43767	6.112
mChargedMuEnergy	0	0	55766	7.787
mNeutralEmEnergy	129144	18.034	102015	14.246
mChargedMultiplicity	0	0	238	0.033
mNeutralMultiplicity	0	0	44	0.006

Table 10: The number of outliers in the residual and response distributions for all variables in the HEP2 dataset after compression with $R = 1.6$ using $\mathcal{L}_2$. These are values that lie outside of $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$

Variable ($R = 1.6$)	Response		Residual	
	Nr. of outliers	Percentage [%]	Nr. of outliers	Percentage [%]
p_T	44427	9.01	58801	11.925
η	81394	16.507	60166	12.202
ϕ	81275	16.483	40461	8.206
mass	92417	18.743	79237	16.07
m_EmEnergy	75940	15.401	71171	14.434
m_HadEnergy	70479	14.294	58956	11.957
m_InvisibleEnergy	0	0	52462	10.64
m_AuxiliaryEnergy	0	0	87588	17.763

Table 11: The number of outliers in the residual and response distributions for all variables in the HEP1 dataset after compression with $R = 6$ using $\mathcal{L}_2$. These are values that lie outside of $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$

Variable ($R = 6$)	Response		Residual	
	Nr. of outliers	Percentage [%]	Nr. of outliers	Percentage [%]
p_T	19413	2.711	67344	9.404
η	113224	15.811	28952	4.043
ϕ	95970	13.402	35957	5.021
mass	27071	3.78	59805	8.351
mJetArea	76272	10.651	78555	10.97
mChargedHadronEnergy	0	0	119752	16.723
mNeutralHadronEnergy	0	0	166411	23.238
mPhotonEnergy	254114	35.486	130149	18.175
mElectronEnergy	0	0	31246	4.363
mMuonEnergy	0	0	40972	5.722
mHFHadronEnergy	0	0	205291	28.668
mHFEMEnergy	0	0	278521	38.894
mChargedHadronMultiplicity	0	0	213788	29.854
mNeutralHadronMultiplicity	0	0	555	0.078
mPhotonMultiplicity	0	0	85658	11.962
mElectronMultiplicity	0	0	121	0.017
mMuonMultiplicity	0	0	10	0.001
mHFHadronMultiplicity	0	0	24616	3.437
mHFEMMultiplicity	0	0	2007	0.28
mChargedEmEnergy	0	0	31119	4.346
mChargedMuEnergy	0	0	40767	5.693
mNeutralEmEnergy	71520	9.987	105247	14.697
mChargedMultiplicity	0	0	213478	29.811
mNeutralMultiplicity	0	0	78585	10.974

Table 12: The number of outliers in the residual and response distributions for all variables in the HEP2 dataset after compression with $R = 4$ using $\mathcal{L}_2$. These are values that lie outside of $[Q_1 - 1.5(Q_3 - Q_1), Q_3 + 1.5(Q_3 - Q_1)]$

Variable ($R = 4$)	Response		Residual	
	Nr. of outliers	Percentage [%]	Nr. of outliers	Percentage [%]
p_T	32662	6.624	18802	3.813
η	97932	19.861	85748	17.39
ϕ	80587	16.344	52844	10.717
mass	31669	6.423	18534	3.759
m_EmEnergy	71880	14.578	70199	14.237
m_HadEnergy	49903	10.121	43170	8.755
m_InvisibleEnergy	0	0	41039	8.323
m_AuxiliaryEnergy	0	0	94404	19.146