Honors Report
Monsoon 2024

Introduction to Collision Simulations and Brief Review on Foundation Models

Abhiram Tilak - 2022113011

[*]*International Instituite of Information Technology, Hyderabad*

Abstract: This report documents efforts to acquire proficiency in tools commonly used for collision simulations, specifcally Pythia8, and ROOT. The application of these tools is demonstrated through the execution of sample simulations.
Additionally, the report provides a concise review of foundational models in the field of high-energy physics, with a particular focus on a paper discussing Masked Particle Modeling.

Contents

1 Collision Simulations **3**
1.1 A simple proton collision . . . 3
1.2 Creating a Histogram: . . . 4
1.3 Root Trees and Histogram . . . 7
1.3.1 Configuring TTree . . . 7
1.3.2 Analysing the ROOT File . . . 8
1.3.3 Creating a custom Histogram for Analysis . . . 9

2 Foundation Models **10**
2.1 Foundation Models in Other Fields . . . 11
2.2 Contrastive Learning . . . 12
2.2.1 Popular Models . . . 14
2.2.2 Adaptations of Contrastive learning Models . . . 15
2.3 Generative Models . . . 16
2.4 Adaptations of Generative Models . . . 17
2.5 Dynamic Masking . . . 17

3 Masked Particle Modelling (July 2024) **19**
3.1 Challenges Faced during Adaptation . . . 19
3.2 Experimenting different Tokenization Strategies and Ordering Strategies . . . 20
3.3 High Level Overview: . . . 21
3.3.1 Calculating Loss function . . . 21
3.3.2 Overview Diagram . . . 21
3.4 Future Direction and MPMv2 . . . 22
3.5 Efforts in Running and Auditing the Model . . . 23

4 Conclusion and Outlook **23**

List of Figures

1 ASCII histogram of momentum distribution output by Pythia's Hist tool . . . 5
2 PyPlot version of Momentum Distribution Histogram . . . 6
3 Tabulated values recorded in simulation into the TTree . . . 9
4 Histogram for size (number of particles in an event) in TBrowser . . . 10
5 Histogram for Pz (Momentum in Z-direction) in TBrowser . . . 11

6 Histogram for Mass for particle id == 2224 (Δ^{++}), created using the Draw() tool . 12
7 Foundation Models Evolution Timeline 13
8 Overview model of contrastive learning 14
9 A graphical representation of a probabilistic model. 16
10 MAE architecture 18
11 Overview Diagram for Masked Particle Modelling 22

Listings

1 Pythia configuration for Sample Simulation 4
2 Initializing Histogram object in Pythia 4
3 Configuring Pyplot for Histogram using HistPlot 6
4 Configuration of Root TTree in Pythia 7

1. Collision Simulations

A standard method for studying collider physics is to simulate the collisions of two particles using the computer program PYTHIA and assume that their interaction is governed by a particular physics model, which can be inserted into PYTHIA as an input file and generated by yet another program called Mad- Graph. Performing this simulation repeatedly to achieve a desired statistical certainty is called a Monte Carlo simulation.

For the sake of sample simulations, we won't be using Mad-Graph and perform multiple simulations, but perform individual simulations usign Pythia8 and explore tools like ROOT which help in analyzing the results.

The Pythia program is a standard tool for the generation of high-energy collisions, comprising a coherent set of physics models for the evolution from a few-body hard process to a complex multihadronic final state. While previous versions were written in Fortran, Pythia 8 represents a complete rewrite in C++.

1.1. A simple proton collision

The simulation of the proton-proton collision is a complex procedure that takes many steps in order to best reproduce the detector output generated by different physics processes.

The simulation chain can be summarized in three main steps: event generation, detector simulation and digitisation. We will be concentrating in the first step here.

The first step in the simulation chain is the generation of all the physics processes of interest. At this level the beam collision is treated only as the interaction between two partons of the protons, described using specific PDFs.

Below is the Pythia configuration used for the simulation:

```
    // proton pgp is 2212
    pythia.readString("Beams: idA = 2212");
    pythia.readString("Beams: idB = 2212");
    pythia.readString("SoftQCD:all = on");
    pythia.readString("HardQCD:all = on");
    pythia.init();
```

Listing 1: Pythia configuration for Sample Simulation

Configuration: First, the program initializes the Pythia object and configures it to simulate collisions between two protons, identified by their Particle Data Group (PDG) code, 2212. It enables both soft and hard QCD (Quantum Chromodynamics) processes, which govern how quarks and gluons interact, ensuring a wide range of collision dynamics. After the setup, the simulation is initialized.

Note that the full source code can be found on my github.

Code Explanation: The main loop runs for a predefined number of events (n). For each event, Pythia generates the particles resulting from the collision. If the generation is successful, the program retrieves and prints the total number of particles in the event. For each particle, it extracts and displays properties like its particle ID (a unique identifier for the particle type), its mass, and its momentum (calculated from its momentum components in the x, y, and z directions). By analyzing these properties, users can study the physics of the collisions and the types of particles produced.

The output will show basic information that we have printed out using the `cout` commands, like the entry number and other values like momentum and mass in each iteration. But this output is not the most readable, we have to use different tools to plot them in graphs or find other programs to analyse and infer for us.

1.2. Creating a Histogram:

Using the `Pythia8::Hist` object we can create a histogram and plot any parameter of choice.

To initalize a simple Histogram which plots z-component of momentum (Pz) we add the following command in the configuration phase:

```
    Pythia8::Hist hpz("Momentum (Pz) Distribution", 100, -10,
  10);
```

Listing 2: Initializing Histogram object in Pythia

The arguments are 100; which refers to the number of buckets in the histogram. The -10 and 10 refer to the range of values the histogram should cover. In this case since we chose 100 buckets from -10 to 10, which have 20 values, each bucket will have a width of 0.5.

Once the code is compiled and run, the histogram is directly printed out in the terminal STDOUT shown in figure 1:

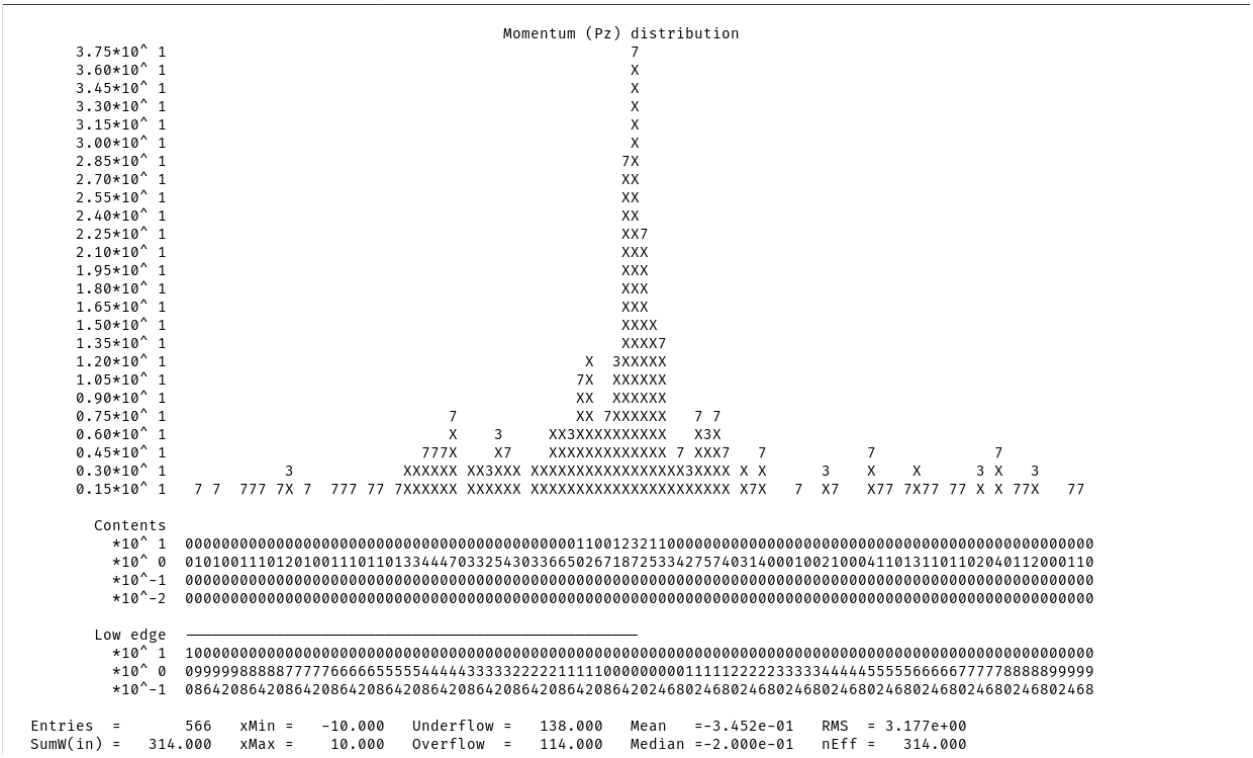

Fig. 1: ASCII histogram of momentum distribution output by Pythia's Hist tool

From the graph we can make the following inferences:

- We have 566 entries, that means 566 particles have been generated in the simulation.
- The SumW is the Weighted sum of the values in the given graph.
- Underflow and Overflow refer to the mininum and maximum values of the momentum which are not captured in the graph because of the xMin and xMax values, set during configuration.

- Both the mean and median are very small values, suggesting more or less it is near to zero and is evident because the initial momentum of the beams in z-direction is 0.
- The RMS shows the root mean square value of the Pz, which should be similar to that of the mean of `P_abs` which we also print.

Using HistPlot and Pyplot: We can also use pyplot to get a better smoother graph than the ascii one in the terminal. Note that, using this tool we get a python file along with an output file, and might need python installed and tools like matplotlib setup for the script to work.

We can configure the HistPlot at the very end after we print out the histogram as follows:

```
Pythia8::HistPlot hpl("myhistplot");
hpl.frame("histout", "Momentum (Pz) Distribution", "Momentum
  (Pz)", "Entries");
hpl.add(hpz);
hpl.plot();
```

Listing 3: Configuring Pyplot for Histogram using HistPlot

Now after compiling and running, we see an additional data file and a python file, which will generate the actual plot in pdf format. The plot can be shown in the figure 2

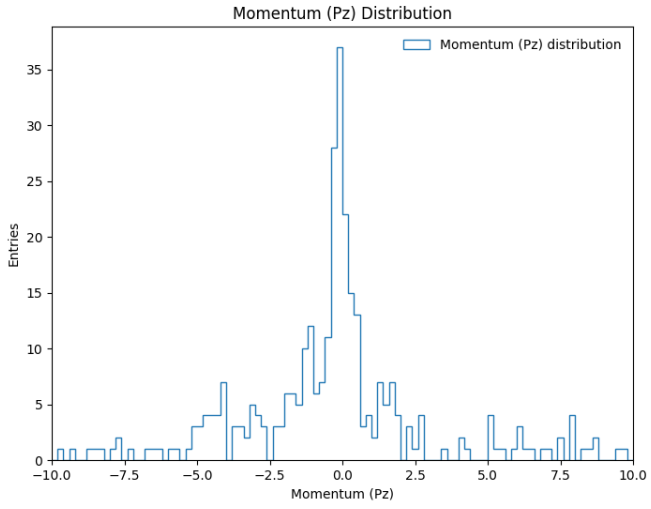

Fig. 2: PyPlot version of Momentum Distribution Histogram

1.3. Root Trees and Histogram

ROOT is a software framework for data analysis and I/O: a powerful tool to cope with the demanding tasks typical of state of the art scientific data analysis. Among its prominent features are an advanced graphical user interface, ideal for interactive analysis, an interpreter for the C++ programming language, for rapid and efficient prototyping and a persistency mechanism for C++ objects, used also to write every year petabytes of data recorded by the Large Hadron Collider experiments. This introductory guide illustrates the main features of ROOT which are relevant for the typical problems of data analysis: input and plotting of data from measurements and fitting of analytical functions.

ROOT provides myriad of tools for data analysis, but In this sample analysis we will be focussing in root trees using the `Ttree` tool.

A TTree behaves like an array of a data structure that resides on storage - except for one entry (or row, in database language). That entry is accessible in memory: you can load any tree entry, ideally sequentially. You can provide your own storage for the values of the columns of the current entry, in the form of variables. In this case you have to tell the TTree about the addresses of these variables; either by calling TTree::SetBranchAddress(), or by passing the variable when creating the branch for writing. When "filling" (writing) the TTree, it will read the values out of these variables; when reading back a TTree entry, it will write the values it read from storage into your variables

1.3.1. Configuring TTree

Configuration of the root tree, involves initializing the TFile file-system, and initializing the TTree object. We create different variables which are to be written, and create a new branch corresponding to each of the variables:

```
1     // TFile setup - First argument is the name of the file/
    tree
2     TFile *output = new TFile("root_tree.root", "recreate");
3     TTree *tree = new TTree("tree", "tree");
4
5     int id, event, size, no;
6     double m, px, py, pz;
7
8     // Initialize tree branches
9     tree->Branch("event", &event, "event/I");
10    tree->Branch("size", &size, "size/I");
```

```
11      tree->Branch("no", &no, "no/I");
12      tree->Branch("id", &id, "id/I");
13      tree->Branch("m", &m, "m/D");
14      tree->Branch("px", &px, "px/D");
15      tree->Branch("py", &py, "py/D");
16      tree->Branch("pz", &pz, "pz/D");
```

Listing 4: Configuration of Root TTree in Pythia

Again please consult the full source code before I try to explain some components:

Explanation In the main loop for every successful event, the total number of particles (n_particles) produced in the collision is retrieved. The program iterates through all these particles, extracting their properties such as particle ID (id), mass (m), momentum components (px, py, pz), and the particle's index (no) within the event. Each particle's data is then stored in a TTree—a ROOT data structure designed for efficient storage and retrieval of structured data. The tree->Fill() command writes these properties for each particle into the TTree.

Once all events are processed, the output file is written to disk using output->Write(), and the file is closed. The resulting ROOT file can then be analyzed further, allowing physicists to study particle properties, kinematics, and distributions resulting from the simulated collisions.

1.3.2. Analysing the ROOT File

After running and compiling the above cpp file, a new root file is created. To analyse it we open it with the root command (`root output.root`).

The first step is to check if our root tree is filled with all the required information. We can get access to an ascii-table with all the recorded info using the Scan() function, the output can be found in the figure 3

We can then open the root file-system corresponding to the file and browse through it by spawning a TBrowser instance. (`new TBrowser`).

```
[I] ~/s/r/s/introduction > root root_tree.root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.34.00                        https://root.cern |
  | (c) 1995-2024, The ROOT Team; conception: R. Brun, F. Rademakers |
  | Built for linuxx8664gcc on Nov 14 2024, 06:14:00                 |
  | From tags/v6-34-00-rc1@v6-34-00-rc1                              |
  | With c++ (Debian 12.2.0-14) 12.2.0                               |
  | Try '.help'/'.?', '.demo', '.license', '.credits', '.quit'/'.q'  |
   ------------------------------------------------------------------

root [0]
Attaching file root_tree.root as _file0...
(TFile *) 0x556391f3a910
root [1] tree->Scan()
************************************************************************************************************
*    Row   * event.eve * size.size *     no.no *     id.id *       m.m *     px.px *     py.py *     pz.pz *
************************************************************************************************************
*        0 *         0 *       137 *         0 *        90 *     14000 *         0 *         0 *         0 *
*        1 *         0 *       137 *         1 *      2212 *   0.93827 *         0 *         0 * 6999.9999 *
*        2 *         0 *       137 *         2 *      2212 *   0.93827 *         0 *         0 * -6999.999 *
*        3 *         0 *       137 *         3 *        -3 *         0 *         0 *         0 * 0.4528171 *
*        4 *         0 *       137 *         4 *        21 *         0 *         0 *         0 * -2430.286 *
*        5 *         0 *       137 *         5 *        -3 *       0.5 * 0.6151833 * -0.894463 * -0.336197 *
*        6 *         0 *       137 *         6 *        21 *         0 * -0.615183 * 0.8944630 * -2429.497 *
*        7 *         0 *       137 *         7 *        -3 *       0.5 * 0.0391046 * -0.314172 * 0.1314249 *
*        8 *         0 *       137 *         8 *        21 *         0 * 0.5757145 * -0.579760 * -1.905975 *
*        9 *         0 *       137 *         9 *        21 *         0 * -0.614819 * 0.8939334 * -2428.058 *
*       10 *         0 *       137 *        10 *        21 *         0 * -0.909009 * 1.0715139 * -2400.334 *
*       11 *         0 *       137 *        11 *        21 *         0 * 0.3025703 * -0.186019 * -27.75172 *
*       12 *         0 *       137 *        12 *        21 *         0 * 0.5673345 * -0.571321 * -1.878232 *
*       13 *         0 *       137 *        13 *        -3 *         0 * -1.048497 * 0.5877927 * -0.344552 *
*       14 *         0 *       137 *        14 *        21 *         0 * -0.293625 * 0.5472304 * -2430.341 *
*       15 *         0 *       137 *        15 *        -3 *       0.5 * -0.814904 * 0.1645875 * -0.268959 *
*       16 *         0 *       137 *        16 *        21 *         0 * -1.199706 * 1.6126585 * -2402.137 *
```

Fig. 3: Tabulated values recorded in simulation into the TTree

The file directory should contain the tree that we plotted and different branches are the paramaters. Just by double clicking, we can obtain a histogram of that paramter. Some limits in the histogram need adjusting because the defaults are usuallly high, but can be adjusted by mouse scroll.

Figure 4 and 5 show some screenshots of the TBrowser in action.

1.3.3. Creating a custom Histogram for Analysis

We can use the Draw() tool to draw, a specific Histogram with a given condition. The condition can be passed as a second paramter to the Draw command along with the original paramter on which condition is placed. Example:

```
root [1] tree->Draw("m", "id == 2224")
```

This will create a histogram for mass (m) for the particle with id (pdb id) as 2224. The resultant histogram can be found in the figure 6. The output from this histogram can also be stored and passed around to other tools like Geant4.

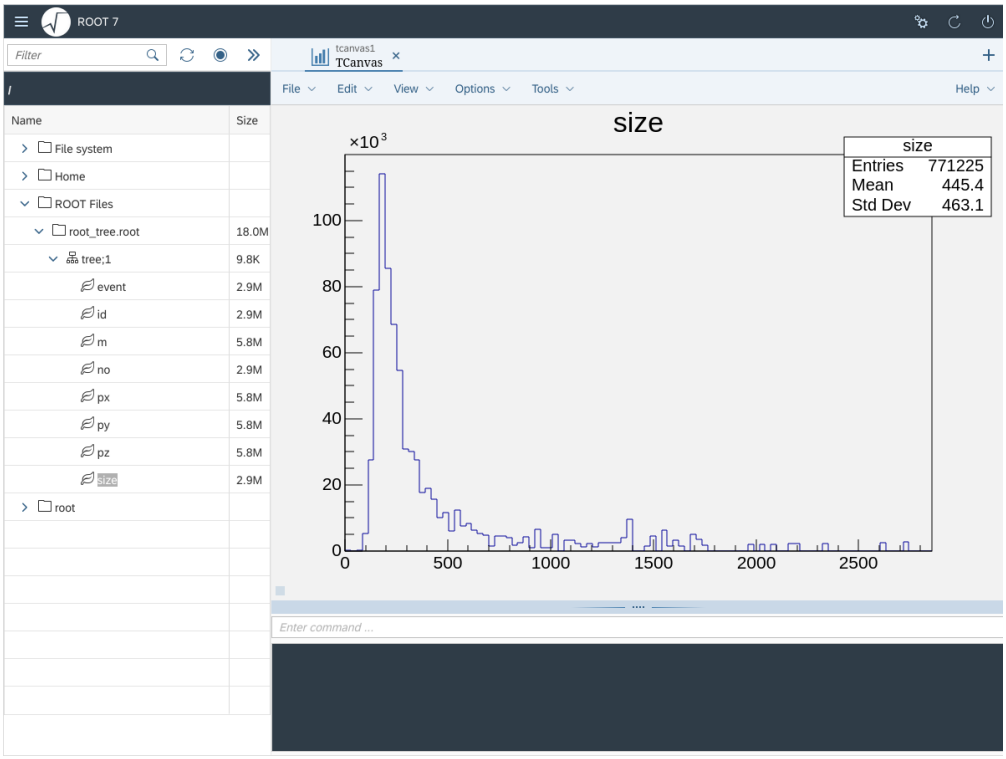

Fig. 4: Histogram for size (number of particles in an event) in TBrowser

2. Foundation Models

[1] Foundation models are multi-dataset and multi-task machine learning methods that once pre-trained can be fine-tuned for a large variety of downstream applications.

The aim of this report is to briefly summarize the state of the art of Foundation models and rapidly increasing amount of research coming out in the field of particles physics. There are maily three popular methods in building Foundation models which are popular in other fields:

- Contrastive Learning Methods
- Generative Methods
- Masking Strategies

This report addresses all major adaptations of above models that are found to date.

[1] **Note:** Most of the contents of this part are a subset of ones given in the presentation. Do excuse the fact that this part is presented less as a report but more like presentation, because the contents are derived from there.

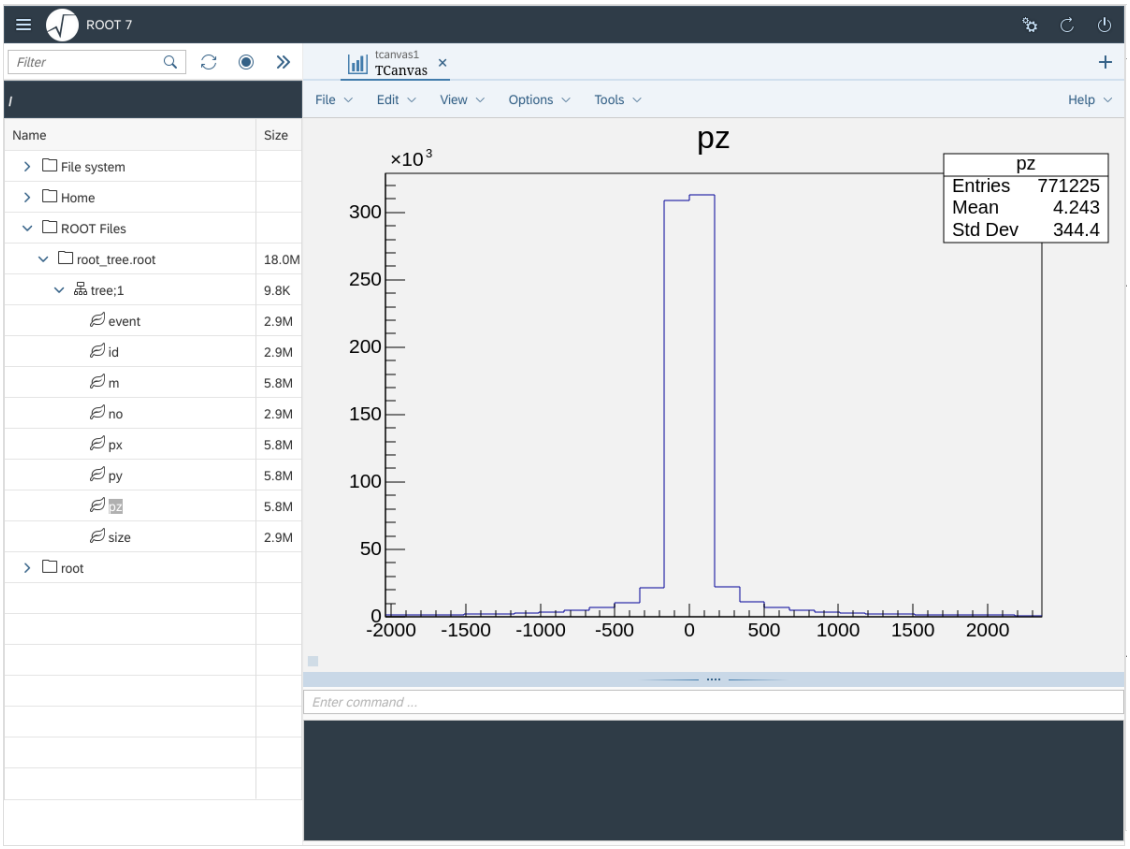

Fig. 5: Histogram for Pz (Momentum in Z-direction) in TBrowser

2.1. Foundation Models in Other Fields

Foundation models are recent developments in artificial intelligence (AI). Models like GPT 4, BERT, DALL-E 3, CLIP, Sora, etc., are at the forefront of the AI revolution. While these models may seem diverse in their capabilities – from generating human-like text to producing stunning visual art – they share a common thread: They are all pioneering examples of foundation models.

The figure 7 shows a timeline chart on popular foundation models, divided based-on various fields in which they are used.

These models are trained on vast amounts of unlabeled or self-supervised data to acquire a broad knowledge base and language understanding capabilities. They extract meaningful features and patterns from the training data and fine-tune them for specific tasks.

Why Self-Supervised? Self-supervised learning enables foundation models to be trained on vast amounts of unlabeled data without requiring expensive human-annotated labels

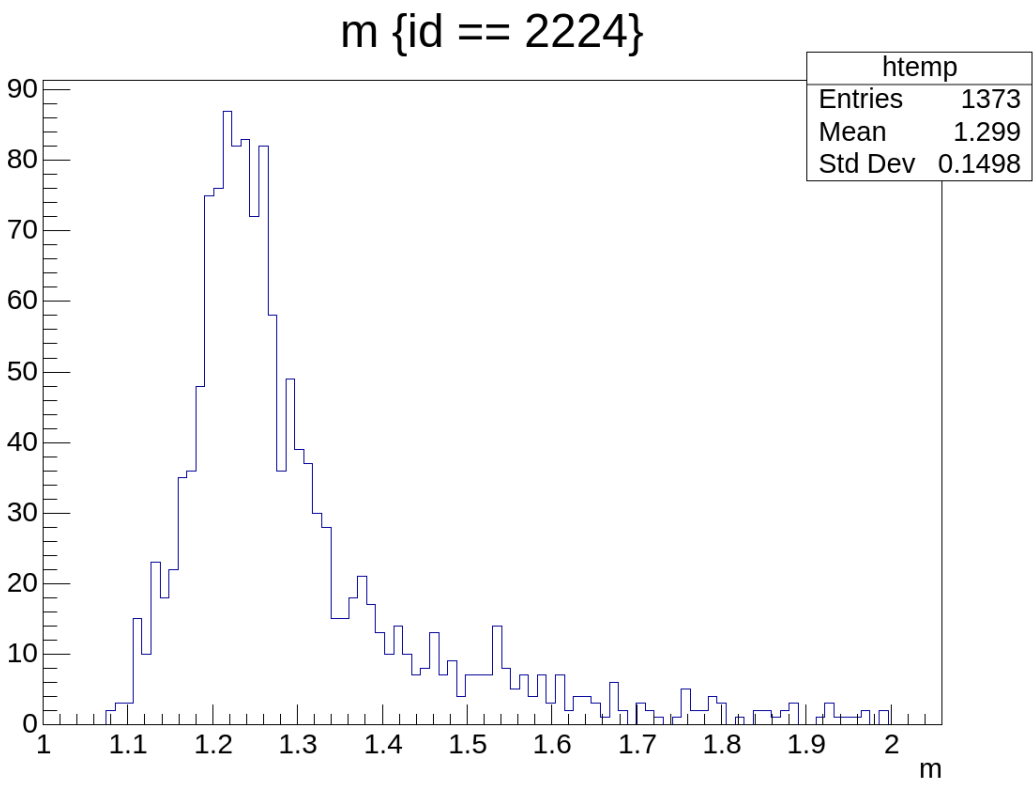

Fig. 6: Histogram for Mass for particle id == 2224 (Δ^{++}), created using the Draw() tool

Foundation models aim to learn generalized representations that capture intrinsic semantic structures in data. This is more effectively achieved through self-supervised learning. Self-supervised objectives create labels directly from the input data, allowing the model to learn meaningful representations without explicit human supervision.

2.2. Contrastive Learning

Contrastive Learning is a machine learning technique that teaches computers to understand similarities and differences by comparing pairs of data points. Similar things should be close together in the computer's understanding, while different should be apart

The process involves three main steps:

- **Data Augmentation:** Data augmentation involves applying various transformations to generate diverse versions of the input data. Techniques like cropping, flipping, rotation, and color adjustments are used to create augmented views of the same instance. This increases

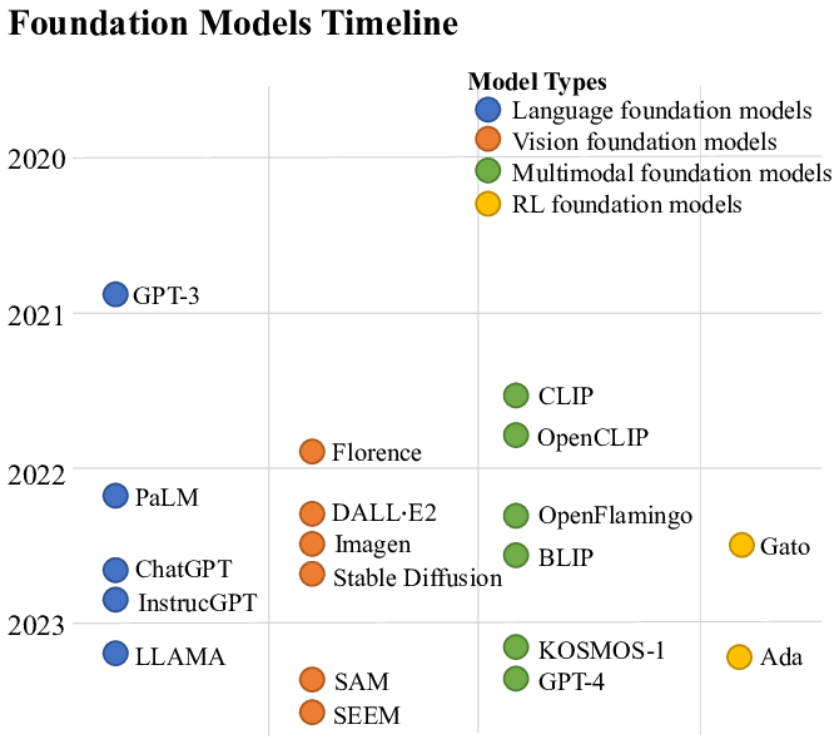

Fig. 7: Foundation Models Evolution Timeline

the dataset's variability and exposes the model to different perspectives of the same data, helping it learn more robust representations.

- **Feature Extraction:** The encoder network takes the augmented inputs and maps them to a latent representation space. This network learns to extract high-level features and similarities from the data, enabling discrimination between similar and dissimilar instances. Typically implemented as a deep neural network like a CNN for images or RNN for sequences, the encoder captures abstract representations that are useful for downstream tasks.

- **Projection:** The projection network transforms the encoder outputs into a lower-dimensional embedding space. This step helps reduce data complexity and redundancy by mapping the representations to a more compact space. The projection enhances the model's ability to distinguish between similar and dissimilar instances, improving overall discriminative power of the learned representations.

Figure 8 gives a good overview of SIMClr model specifically, but can be generalized to any contrastive learning process.

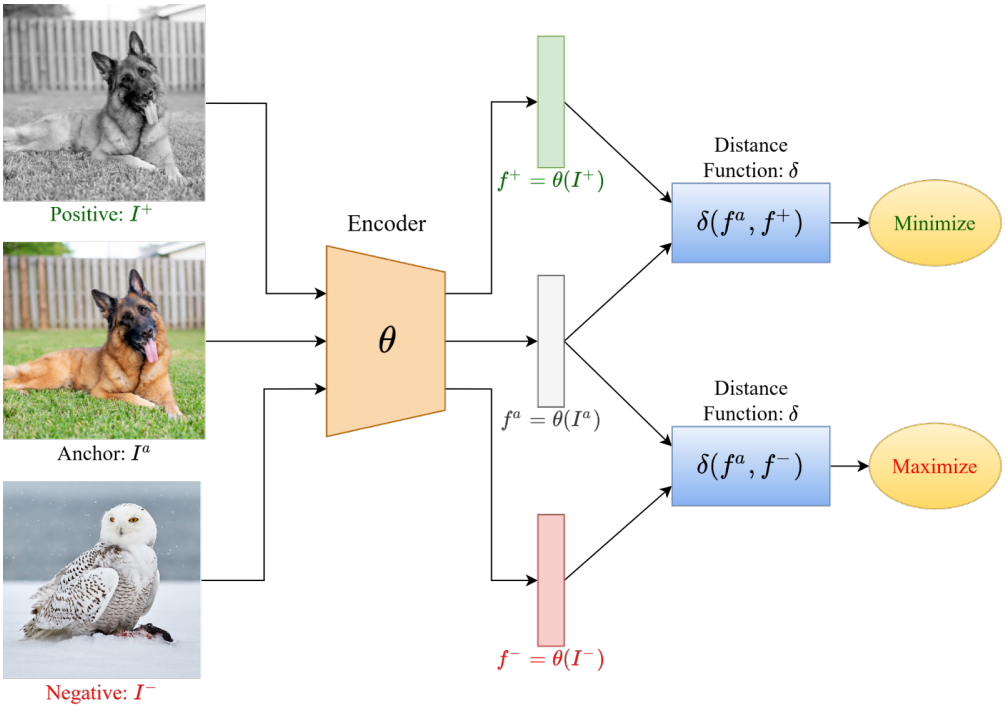

Fig. 8: Overview model of contrastive learning

2.2.1. Popular Models

Some popular models based on contrastive learning include:

- MoCo (Nov 2019) (Momentum Contrast for unsupervised Visual Representation Learning):

  MoCo treats contrastive learning as a dynamic dictionary lookup process using two encoders - a query encoder and a momentum encoder - achieving 60.6% accuracy with ResNet-50 and scaling up to 68.6% with R50w4×, making it competitive with larger models while using standard architectures.

- SimCLR (Feb 2020) (A simple framework for contrastive learning of visual representation):

  SimCLR is a simplified framework for self-supervised visual learning that outperforms previous methods by data augmentations and non-linear transformations, achieving 76.5% accuracy with linear classifier and 85.8% on fine-tuned on ResNet-50.

- CLIP (Feb 2021) (Contrastive Language-Image Pre-training)

  CLIP (Contrastive Language-Image Pre-training) pairs images with their text captions using dual encoders (ViT for images, text encoder for captions), matching images to their

descriptions in a shared embedding space, achieving notable zero-shot capabilities in image classification tasks when trained on 400M image-text pairs

2.2.2. Adaptations of Contrastive learning Models

Three notable adaptations have emerged in recent years:

- JetCLR (August 2022)

  JetCLR represents a significant innovation in particle physics applications of contrastive learning. This adaptation focuses on creating symmetry-aware representations for jets, emphasizing built-in physical symmetries such as rotation, translation, and permutation. Additionally, JetCLR incorporates soft/collinear safety considerations, addressing potential issues in jet reconstruction. The approach utilizes a transformer-encoder network architecture and evaluates performance through linear classifier tests, demonstrating its effectiveness in distinguishing between different jet configurations.

- Dark CLR (October 2024)

  Dark CLR marks a crucial advancement in the quest to detect semi-visible jets from dark sector models. This adaptation leverages contrastive learning with "anomalous enhancements" specifically designed for dark sector physics. By combining representation learning with a normalized autoencoder, Dark CLR enhances its ability to identify subtle anomalies indicative of dark matter signatures. This approach represents a significant step forward in the pursuit of understanding beyond-standard-model physics at the Large Hadron Collider.

- AnomalyCLR (August 2024)

  AnomalyCLR presents a more generalized approach to model-agnostic anomaly detection at the Large Hadron Collider. This adaptation introduces "anomaly-augmentations" that mimic generic features of anomalous events, such as high multiplicity, large missing transverse energy (MET), and large transverse momentum (pT). The method employs a two-step process: contrastive learning followed by an autoencoder applied to the learned representations. This approach demonstrates versatility in detecting a wide range of potential anomalies across various physics scenarios.

Another honorable mention is RS3L (March 2024) which employs a method of re-simulation to drive data augmentation for contrastive learning

2.3. Generative Models

A generative model is a type of machine learning model that aims to learn underlying patterns or distributions of data to generate new, similar data. This is often used in the context of enabling computers to understand the real world.

Generative models can be both probabilistic or Neural Network based. Figure 9 shows the overview of processes in training a probabilistic generative model.

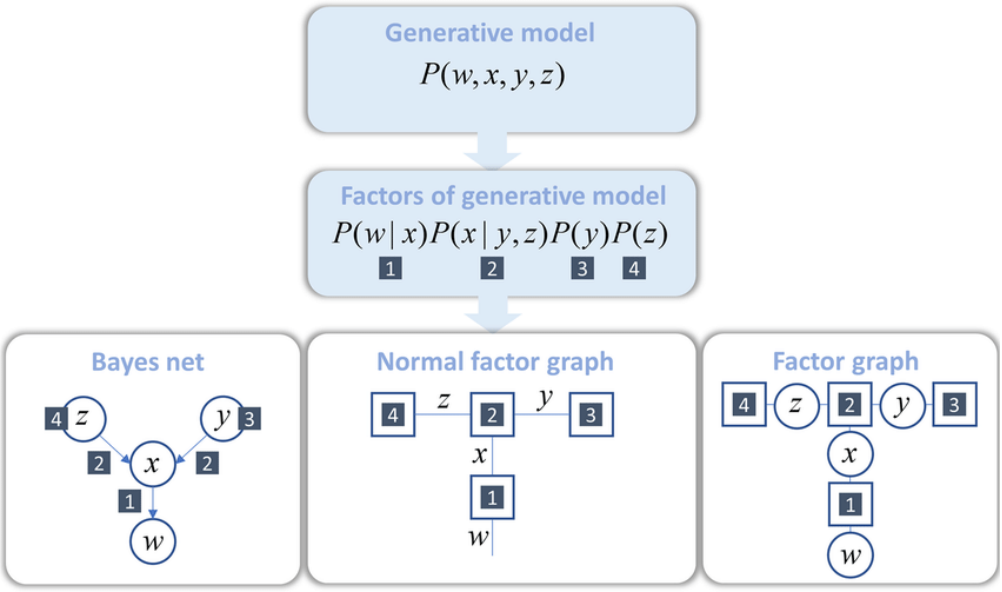

Fig. 9: A graphical representation of a probabilistic model.

Currently generative models can generate:

- Images (Dalle, Stable Diffusion, MidJourney)
- Text Generation (LLaMA, GPT, Mistral)
- Audio and Music (WaveNet, BachBOt)

There are still efforts to generate Video content through models like SORA (OpenAI), but the current results are questionable and not promising.

Generative models are too broad of a topic. We can cover some of the popular approaches and efforts in adapting them to physics

2.4. Adaptations of Generative Models

- ParT (Particle Transformer) (July 2023) :

  ParT is based on the traditional Transformer architecture and a successor to ParticleNet, but introduces a modified attention mechanism specifically for jet tagging. Its major innovation is adding a new term to capture pairwise particle interactions within the attention mechanism. The model processes jets as particle clouds (unordered, variable-sized sets of outgoing particles) and was trained on the massive JETCLASS dataset, which is significantly larger than previous datasets. ParT's training approach is straightforward supervised learning for jet classification, contrasting with more complex pre-training strategies.

  Note: There is a new model MiParT which is works and in version 3, which accommodates the More-Interaction Attention (MIA) mechanism.

- OmniJet (Mar 2024): OmniJet is based on the GPT but adapted for continuous physics data. The model's key innovation is its tokenization strategy for converting continuous particle physics data into discrete tokens, expanding the codebook size to 8192 tokens (up from 512 in previous approaches). Unlike ParT's direct classification approach, OmniJet first trains in an unsupervised manner to generate jets as tokens, following the successful pre-training paradigm of language models. The model demonstrates transfer learning capabilities, where the knowledge gained during generative pre-training can be applied to downstream classification tasks. Finally, OmniJet aims to be a foundation model for multiple jet physics tasks, contrasting with ParT's focus on classification specifically

2.5. Dynamic Masking

Dynamic masking means that the process of selecting and masking words within a sentence is not the same for every epoch during the pretraining phase. This introduces randomness into the training process, and different subsets of words are masked in each epoch. The idea behind dynamic masking is to encourage the model to learn more robust and contextual representations of words.

Such strategies are found in fields like Next Word Predictors which are used in autocorrects in google keyboards, etc. They use the framework called Masked Language Model (or general Masked Token Prediction). Popular models like BERT and RoBERTa use this method.

There is also a popular computer vision adaptation, which use Masked Auto-Encoding strategy (MAE). MAE is a self-supervised learning technique that masks (removes) portions of an image and learns to reconstruct them.

- Randomly masks a very high portion of image patches (typically 75%)

- Uses much higher masking ratio than NLP because images have more redundant information
- This forces the model to understand the overall image context rather than just copy nearby pixels

The figure 10 goes over a high level overview on the masking strategy used.

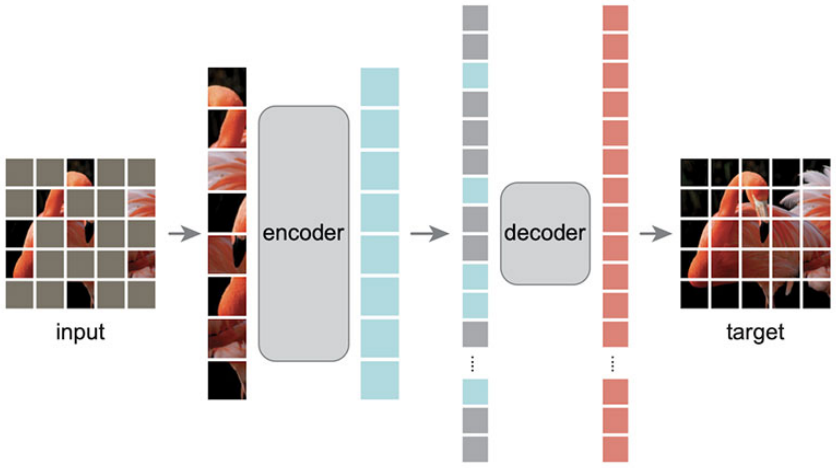

Fig. 10: MAE architecture

In next section discusses a brief overview particular paper which can be seen as an effort to adapt MLM into High energy physics.

3. Masked Particle Modelling (July 2024)

[2]The paper Masked Particle Modelling introduces a self-supervised model designed to process unordered sets of particles, characterized by specific parameters, as input. The primary objective of this research is to develop a versatile foundational model that can subsequently be fine-tuned for a variety of downstream tasks, thereby demonstrating its adaptability and broad applicability.

Inspired by masking strategies employed in models like BERT and BEiT, the authors propose a novel approach for masking a subset of particles within each jet. This technique challenges the model to predict the masked particles based on the contextual information derived from the remaining particles. The resulting model is thus designed to infer the original particle set effectively, leveraging the inherent relationships within the data.

Why to mask the tokens? In BERT the MLM model gave it a dramatic improvement over previous state-of-the-art models and served as on of the earliest examples of large language model. Traditional language models process text sequentially, either from left to right or right to left. This method limits the model's awareness to the immediate context preceding the target word. By masking random tokens, the model is forced to use both left and right context to predict the masked word. This enables the model to learn bidirectional representations, unlike traditional left-to-right language mode

3.1. Challenges Faced during Adaptation

The results from models in NLP seem promising, but there are a few fundamental problems in dataset in the context of HEP which differ the training scheme used, which need to be addressed:

- **Tokenization:** Unlike language models, which operate on a finite and discrete vocabulary of words, many of the features one that describe particles are continuous, such as momentum, direction, distance of closest approach to the primary collision, etc. This changes the way we pass the input or parse the output when pre-training.
- **Permuation of datasets:** The particles in a jet don't have a specific order, so it makes sense to use a model that treats them all equally. However, if we replace all masked particles with the same placeholder, the model can't tell them apart. This means it would produce the same output for all masked particles, which isn't very useful. To avoid this problem, we need to either give the particles an order or add some way for them to interact with each other uniquely

[2]Some parts of these report are from this presentation

3.2. Experimenting different Tokenization Strategies and Ordering Strategies

The paper explores several methods for tokenizing the input particle sets, each with distinct characteristics and limitations.

- The first method, Simple Binning, involves dividing each feature into a finite set of discrete ranges, or bins. This straightforward approach assigns particles to bins based on their feature values. However, it suffers from a significant limitation: the lack of contextual information about the relationships between particles, which may undermine its effectiveness in capturing complex interactions.
- The second method employs a Vector Quantized Variational AutoEncoder (VQ-VAE), this encoder is really popular in the field of computer vision. This approach utilizes an encoder to map input features to latent vectors, which are then projected onto the nearest element in a finite codebook. By incorporating context from all input elements, this method ensures that each particle is encoded with consideration of the other particles in the jet. While VQ-VAE provides a more context-aware representation, it is more complex to implement and potentially computationally expensive.
- Lastly, the paper considers K-means Clustering as a tokenization strategy. Using the K-means++ algorithm, particles are grouped into clusters, with each cluster assigned a unique index as its label. Unlike VQ-VAE, this method is context-independent and does not account for the relationships between particles. As a result, it may fail to capture important interactions within the particle set, despite being simpler to implement

The following ordering strategies have been experimented with:

- **Ordering the input to the backbone:** Order particles by decreasing transverse momentum (pT) at the input stage of the model.

  Limitation: Breaks permutation invariance for all downstream tasks, which may adversely affect predictive performance.

- **Ordering only at the input to the pre-training prediction:** This method helps break symmetry for masked predictions without affecting the backbone's permutation invariance. Apply pT ordering just before the prediction head, using learned positional embeddings.

  Limitation: Adds complexity to the model architecture and may increase computational cost, though less severe than method 2

Figure 1 shows the the, accuracy found with different combination of ordering and Tokenization strategies used

Ordering	**Inputs**	**Loss**	**Accuracy**
no ordering	continuous VQ-VAE	classification	54.1%
order head	continuous VQ-VAE	classification	**56.8%**
order backbone	continuous VQ-VAE	classification	53.4%
order head	quantized VQ-VAE	classification	51.1%
order head	quantized K-means	classification	49.3%
order head	continuous K-means	classification	56.2%
order head	continuous	regression	48.9%
order backbone	continuous	regression	46.3%

Table 1: Results for different input orderings, loss functions, and tokenization strategies.

3.3. High Level Overview:

3.3.1. Calculating Loss function

Let's say a jet is given by:

$$X = \{x_i\}_{i=1}^{N} \qquad (x_i \text{ represents a particle})$$

MPM partitions the set of particles in each jet in the dataset into masked and unmasked set:

$$\mathcal{M}_x = \{x_i\}_{i\in\mathcal{M}}, \quad \mathcal{U}_x = \{x_i\}_{i\in\mathcal{U}}$$

The goal of pretraining is to find out a parametric function $f_\theta : X \to \mathbb{R}^{Nxd}$ (assume d-dimentional paramter set for each particle) such that the expectation of loss $\mathcal{L}$ can be minimized.

$$\mathbb{E}_x \left[\frac{1}{|\mathcal{M}_m|} \sum_{i\in\mathcal{M}_m} \mathcal{L}(x_i, f_{\theta,i}(\mathcal{M}_m, \mathcal{U}_x)) \right]$$

3.3.2. Overview Diagram

The figure 11 illustrates the proposed model and training scheme for MPM. In this model representation, a jet is represented as a set of particles, where each particle is described by a list of features. The input set undergoes a masking process, where some particles are replaced by learnable vectors. The modified set is then processed by a transformer encoder, referred to as the MPM Backbone in this framework.

The training objective is to predict the discrete token identity of the masked particles. These token identities are defined by the encoder of a pre-trained VQ-VAE. The VQ-VAE Encoder dis-

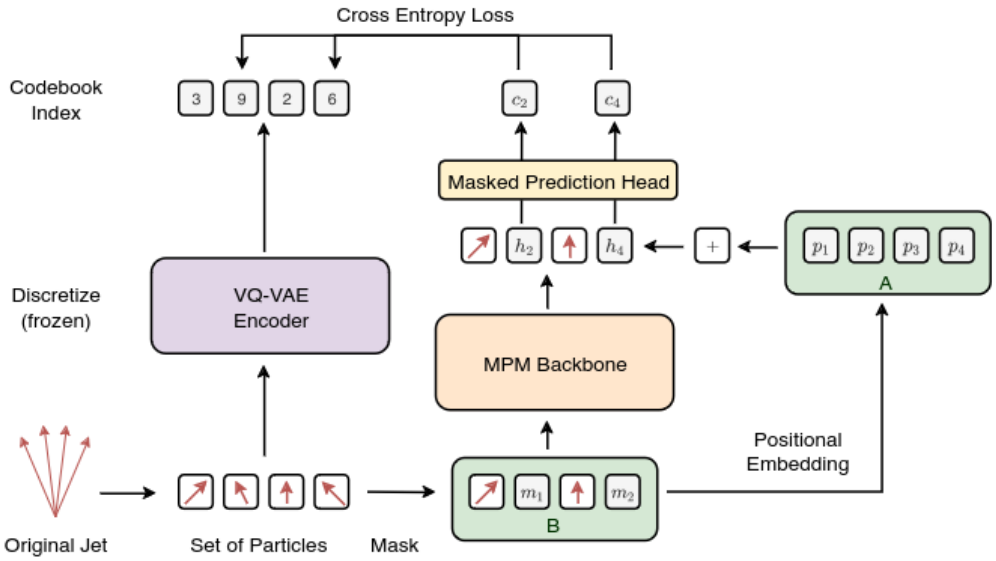

Fig. 11: Overview Diagram for Masked Particle Modelling

cretizes the input features into discrete codebook indices, which serve as the ground truth for the training process. The cross-entropy loss is used to evaluate the model's performance by comparing the predicted and actual token identities.

To ensure permutation invariance in the backbone, positional embeddings are not applied within the MPM Backbone itself. Instead, they are utilized only in the Masked Prediction Head, depicted in the figure (box A), to break degeneracies that can arise when the masked tokens (e.g., m_1 and m_2) have identical values. This strategy allows the model to preserve the symmetry of unordered sets while still leveraging positional information during token prediction.

The complete training pipeline thus integrates masking, transformation via the backbone, and prediction via the masked head, forming an end-to-end framework to develop a self-supervised foundation model for jet analysis.

3.4. Future Direction and MPMv2

Currently a new version of MPM paper is up for proposal. This uses a more powerful decoder and tries to use conditional generative models without the tokenization.

This paper presents several additional key contributions.

(1) An improved MPM training paradigm, referred to as MPMv2, is proposed by enhancing the model architecture and addressing existing inefficiencies. Additionally, the particle attributes are expanded to provide a more detailed representation.

(2) A comprehensive study is conducted on alternative reconstruction tasks for MPMv2 pre-training, replacing the computationally expensive VQ-VAE-derived targets.

(3) A new test bed is introduced for pre-trained models, encompassing a broader range of downstream tasks commonly encountered in jet physics

3.5. Efforts in Running and Auditing the Model

I was originally supposed to run this model and try to replicate the results found in this paper. But after setting the model up in the ADA, I have faced various issues with particularly with modules and libraries not loading and getting stuck in loops. I have reached out for help but not was unable to locate through and single out what the issue could be. I definitely plan to work on it and figure it out so that we have some findings to show and compare in future.

4. Conclusion and Outlook

In this report, I have discussed the basic components of an event-generation pipeline. To make this report brief, I have take the example of a proton-proton collision production to discuss the various aspects and application of these softwares. I have glossed over some of details, for example the various uses of ROOT Analyser and myriad of subtools within it.

In terms of looking through various foundation models, they might seem promising at first but I am afraid they require huge amount of resources to train and test compared to any other approach known to date. It will definitely be a struggle navigating through the subject with the limited resources we have in our college.