A3: High Energy Physics Coursework Report

Alik Mondal (am3353)
Department of Physics, University of Cambridge

April 11, 2025

Word Count: 2932
Declaration of Autogeneration Tools: I declare that I have used ChatGPT and Perplexity AI as autogeneration tools for assistance with ideas, explanations, and drafting text for this coursework. These tools were utilized in accordance with the course guidelines to supplement my understanding and enhance the quality of my submission.

1 Introduction

This coursework explores the application of statistical analysis and machine learning in high energy physics, and is structured according to the order of the questions.

In Part A2, we construct a custom Keras layer that is equivariant under the discrete rotation group C_4 (i.e., 0°, 90°, 180°, 270°), and assess its performance on image classification. The layer is implemented using weight sharing across rotated versions of the same filter, thus enforcing discrete rotational symmetry in a principled way. We train and evaluate this layer on the MNIST dataset augmented with rotations, and compare its performance to a fully connected neural network and a regular CNN, quantifying the benefits of an equivariant design.

In Part B2, we focus on signal-background separation for the decay $B_s^0 \to D_s^{\pm}\pi^{\mp}$ using real and simulated data. We train a Boosted Decision Tree (BDT) using the `xgboost` package on a selection of variables derived from the event topology. After performing a preliminary mass fit to estimate the signal and background components, we optimize a cut on the BDT score by maximizing a figure of merit and evaluate the signal-to-background ratio in a $\pm 3\sigma$ window around the B_s^0 mass peak.

In Part C2, we build a multiclass neural network classifier to distinguish between $Z^0 \to q\bar{q}$ decays, where $q \in \{b, c, s\}$, using event-level, particle-level, and vertex-level features from simulated FCC-ee collisions. We implement the classifier using the `keras` API in TensorFlow, and explore FCNN, attention-based RNN, and DeepSets neural network architectures, which are suitable for set-structured and sequential data. We visualize the confusion matrix and classification probabilities for each class, and report the categorical accuracy on a validation dataset.

Each section includes code, plots, and quantitative analysis, with all implementations following good software development practices.

2 Part A2

We construct a convolutional layer that is equivariant under the discrete rotation group $C_4 = \{0°, 90°, 180°, 270°\}$ and demonstrate its efficacy on rotated MNIST classification. The goal is

to build a custom Keras layer that treats inputs equivariantly under these rotations by sharing weights across transformed versions of the kernel.

Theory

Let x be the input image and f a learnable kernel. For a rotation operator R_k that rotates by $90k^\circ$, we define the equivariant convolution output as:

$$y(x) = \frac{1}{4} \sum_{k=0}^{3} R_{-k} \left(R_k(x) * f \right),$$

where $*$ denotes standard convolution. This ensures that if the input is rotated, the output rotates in the same way. The operation is an example of a group convolution over C_4, a special case of equivariant neural networks where symmetry priors are baked into the architecture.

Implementation

We implement a custom Keras layer `RotationEquivariantLayer`, which applies the same convolution to the input at four rotated angles and then rotates the outputs back to their original orientation before averaging them. This ensures that the network learns features that are consistent under 90° rotations.

Listing 1: Custom Keras layer implementing C_4 rotation equivariance.

```
class RotationEquivariantLayer(layers.Layer):
    def __init__(self, filters, kernel_size, **kwargs):
        super(RotationEquivariantLayer, self).__init__(**kwargs)
        self.filters = filters
        self.kernel_size = kernel_size
        self.conv = layers.Conv2D(filters, kernel_size,
        padding='same')

    def call(self, inputs):
        x_0 = self.conv(inputs)
        x_90 = tf.image.rot90(inputs, k=1)
        x_90 = self.conv(x_90)
        x_90 = tf.image.rot90(x_90, k=3)
        x_180 = tf.image.rot90(inputs, k=2)
        x_180 = self.conv(x_180)
        x_180 = tf.image.rot90(x_180, k=2)
        x_270 = tf.image.rot90(inputs, k=3)
        x_270 = self.conv(x_270)
        x_270 = tf.image.rot90(x_270, k=1)
        return (x_0 + x_90 + x_180 + x_270) / 4

    def get_config(self):
        config = super(RotationEquivariantLayer, self).get_config()
        config.update({
            'filters': self.filters,
            'kernel_size': self.kernel_size
        })
        return config
```

Experiment: MNIST with Rotations

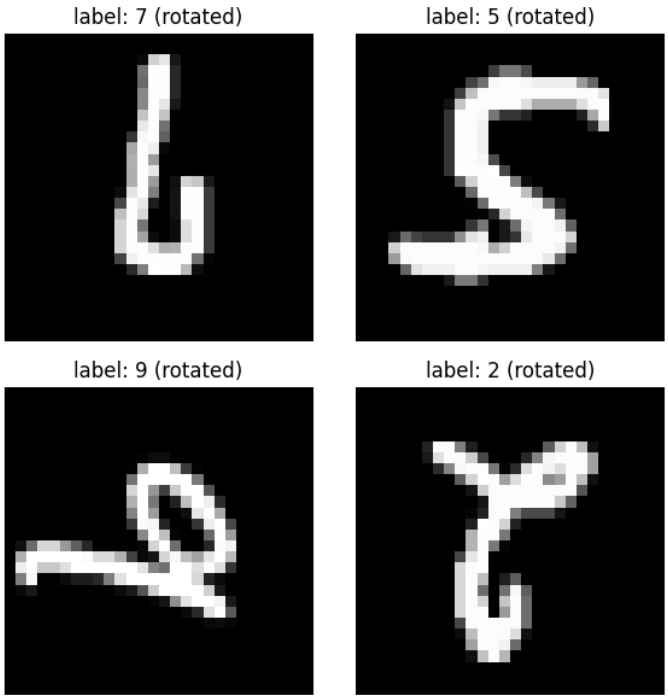


Figure 1: Rotated MNIST Dataset

We evaluate the model incorporating the equivariant layer using a modified MNIST dataset, where each training and test image is randomly rotated by 0°, 90°, 180°, or 270°, as shown in Figure 1. In the `test_equivariance()` function, we test the equivariance by taking a sample image, rotating it in four ways (0°, 90°, 180°, and 270°), then applying the `RotationEquivariantLayer(16, (3, 3))` and comparing the results (specifically, the 0th channel) with the original input image orientation, as shown in Figure 2.

Next, we perform a model performance comparison as follows:

- **Fully Connected Neural Network (FCNN):** A dense network trained on flattened inputs.
- **Regular CNN:** A standard convolutional neural network using typical 2D convolutions.
- **Equivariant CNN:** A network that uses the custom `RotationEquivariantLayer` as the first layer.

Each model is trained on approximately 40,000 rotated samples for 20 epochs, with a batch size of 64 using the Adam optimizer and sparse categorical cross-entropy loss.

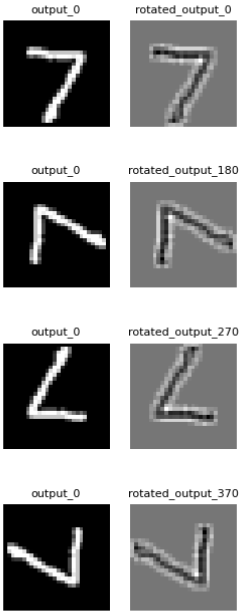
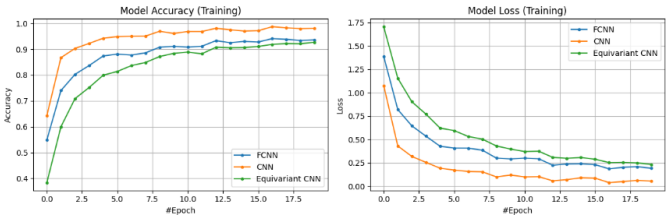


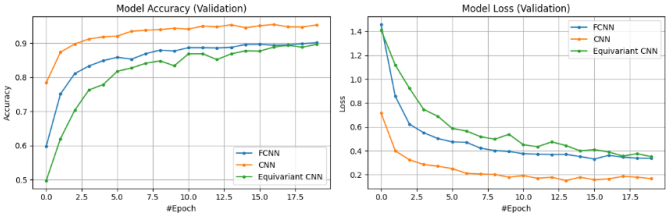
Figure 2: Rotation Comparison in $\{0°, 90°, 180°, 270°\}$.

Results

In accordance with Figure 3, the regular CNN trains significantly better in both loss reduction and accurate classification. The equivariant model starts the slowest but ultimately approaches and surpasses the fully connected model, ultimately yielding slightly better performance in the multiclass classification problem with significantly fewer parameters.



(a) Training Loss and Accuracy for Three Models



(b) Validation Loss and Accuracy for Three Models

Figure 3: Comparison of training and validation metrics across models.

‘ The FCNN achieved a test accuracy of approximately 89%, showing sensitivity to rotated

digits. The equivariant CNN improved performance to around 91%, demonstrating robustness across all rotation angles, while the regular CNN model reached a test accuracy of around 95%. These results validate the effectiveness of embedding rotational symmetry directly into the model's architecture.

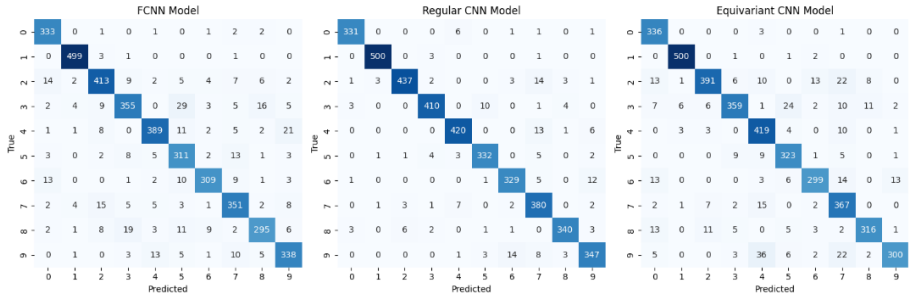


Figure 4: Confusion Matrix comparison of FCNN, CNN, Equivariant CNN

For the misclassification rate comparison, we have plotted the confusion matrix for the three models in Figure 4.

3 Part B2

In this section, we train a Boosted Decision Tree (BDT) to separate signal and background in $B_s^0 \to D_s^{\pm}\pi^{\mp}$ decays using a mix of MC simulated signal and real LHCb data. The goal is to construct a classifier that identifies signal-like events based on high-level kinematic and topological variables, and to optimize a cut on the BDT score to maximize sensitivity.

Data and Preprocessing

We use ROOT files provided in the assignment repository, containing trees of both signal and background events. The signal sample consists of Monte Carlo simulated $B_s^0 \to D_s^{\pm}\pi^{\mp}$ decays. The background sample is derived from real LHCb data by selecting candidates with invariant mass above the B_s^0 mass peak (i.e. $m > 5600$ MeV), where the signal contribution is negligible.

The following features are used to train the BDT: B_s^0 momentum and transverse momentum (`Bs_P`, `Bs_PT`), flight distance and its χ^2 significance, impact parameter and its χ^2, and the cosine of the angle between the B_s^0 momentum and the vector from the primary to secondary vertex. Similar variables are included for the $D_s^{\pm}$ and bachelor pion candidates: momentum, p_T, impact parameter χ^2, and cosine angle to the B_s^0. The D_s invariant mass (`Ds_M`) is also used as an input feature.

Events are loaded via `uproot` and preprocessed using `pandas` and `scikit-learn`. Standard scaling is applied to all input features.

Since we collected background samples from data with B_s^0 mass $> 5600\,\text{MeV}$, it is expected that we will have fewer background labels in comparison to the MC simulated signals. To address this imbalance, we use SMOTE (Synthetic Minority Over-sampling Technique) oversampling to obtain an equivalent sample size for both signal and background.

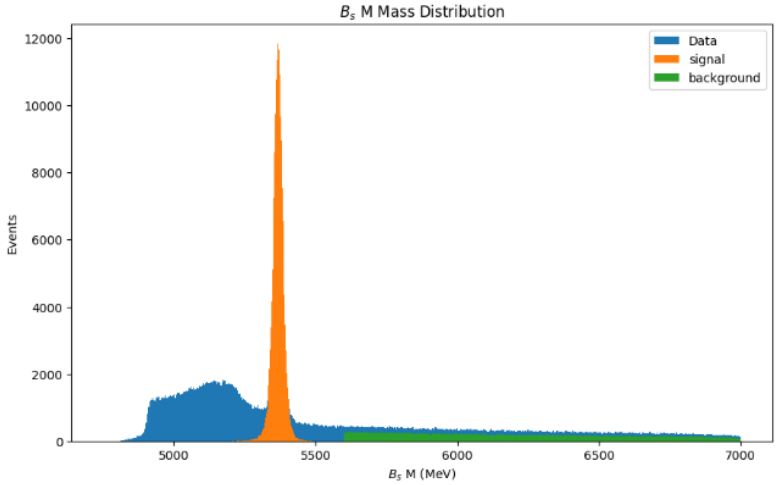


Figure 5: Invariant mass distribution of B_s^0 candidates for signal (blue), background (red), and data (black). The signal is visible as a narrow peak above a smooth combinatorial background.

Training and BDT Evaluation

We train a Boosted Decision Tree (BDT) using `xgboost`, with `n_estimators=500`, early stopping on a validation set, and hyperparameters tuned for optimal generalization. We then visualize the separation performance using the training loss and the ROC curve, which indicates near-perfect classification.

By examining the feature importance plot in Figure 6, we observe that the `Bs_CosMomDecayAngle` is the most influential feature, contributing significantly to the accurate classification of signal and background events. This variable likely captures key kinematic differences in the decay topology that help discriminate between the two classes. It is followed by `Ds_ImpactParameter` and `Pi_ImpactParameter_Chi2`, both of which are sensitive to the displacement of decay products from the primary vertex, a characteristic feature of long-lived particles in signal decays. The relative importance of these variables highlights the discriminative power of geometric and kinematic features in the BDT's decision-making process.

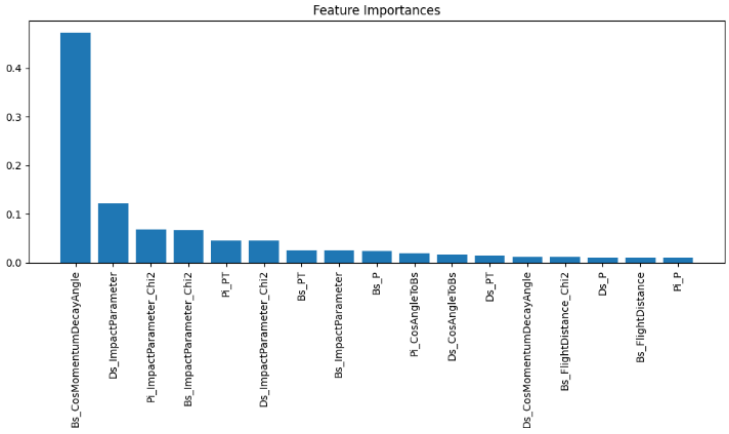
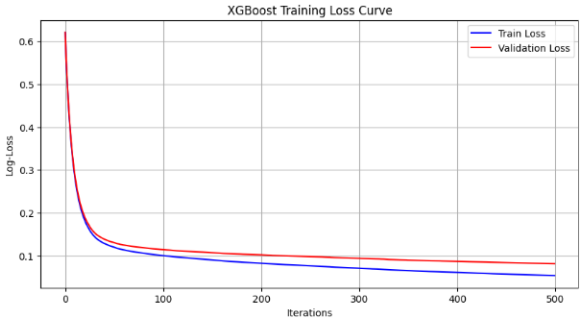
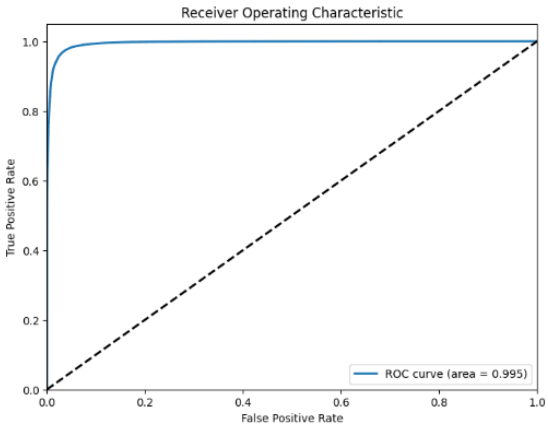


Figure 6: Feature importance plot highlighting the top discriminating variables used by the BDT.



(a) Training and Validation Loss vs. Iterations



(b) ROC Curve on Test Data

Figure 7: BDT training diagnostics. **Left:** Training and validation loss showing convergence. **Right:** ROC curve on test data indicating strong separation power.

Preliminary Fit

We assume a simple model consisting of a Gaussian distribution for the signal and an exponential distribution for the background. This model is fitted using the `scipy.optimize.curve_fit()` module, prior to any BDT-based selection. The resulting fit yields a χ^2 value of **1022.90**, indicating poor fit quality, and a signal-to-background ratio (S/B) of **0.12**, which is also very low. Additionally, the Gaussian fit returns a mean mass of 5370.28 ± 3.22 MeV and a width (σ) of 20.00 ± 3.26 MeV. These results highlight significant scope for improvement in the fitting methodology, particularly in achieving a more accurate representation of the signal component.

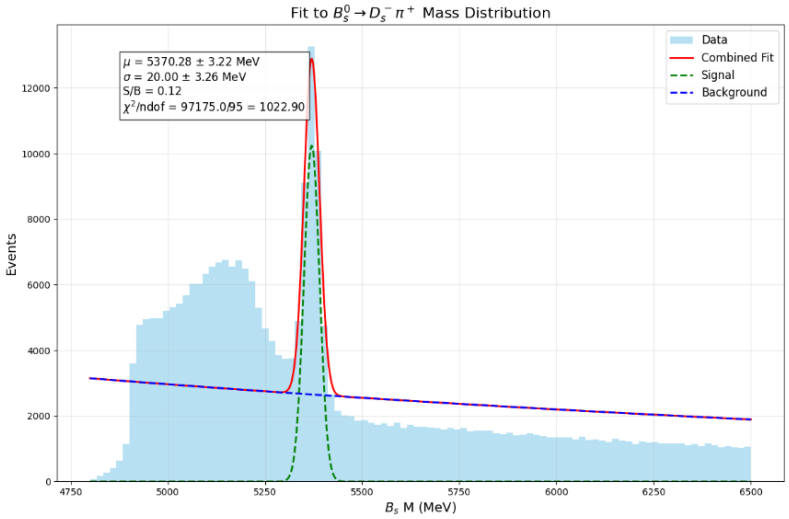


Figure 8: Preliminary boson mass fit using a Gaussian + exponential model before BDT selection.

Cut Optimization and Signal Extraction

We optimize the BDT cut using the figure of merit:

$$\text{FOM} = \frac{S}{\sqrt{S+B}},$$

Here, S and B represent the signal and background yields in the B_s^0 mass window after applying a score threshold. By scanning over cut values from 0.0 to 1.0, we determine the BDT threshold that maximizes the Figure of Merit (FOM).

We find an optimal cut at around 0.43 (with FOM of 484.89), as shown in Figure 9, with classifier probabilities for background and signal concentrated near 0 and 1, respectively—an expected outcome for a well-performing classifier. The fact that our optimal BDT threshold is close to 0.5 serves as a sanity check, since a perfect classifier would ideally separate signal and background such that the optimal cut lies near 0.5.

Final Selection and BDT Score Distribution

After applying the optimal BDT cut, we examine the filtered mass window around the B_s^0 peak ($\pm 3\sigma$). A fit to this post-cut region yields a much cleaner signal with significantly reduced background. The resulting fit gives a mass mean of 5370.32 ± 0.57 MeV and a width (sigma) of 20.00 ± 0.69 MeV.

The signal-to-background ratio (S/B) within the $\pm 3\sigma$ window improves dramatically to **158.46**, which is exponentially better than the naive pre-selection fit. Additionally, the χ^2 of the fit reduces substantially to **72.44**, indicating a significantly improved fit quality (see Figure 10).

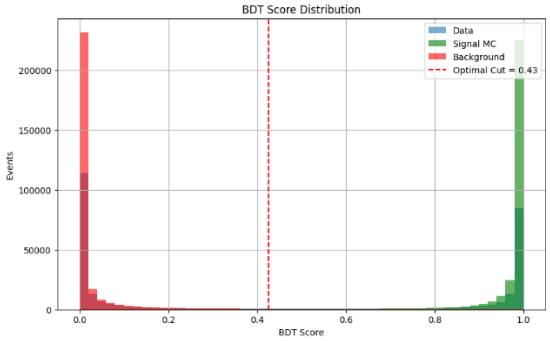


Figure 9: BDT score distribution for signal (green) and background (red) samples. The cut (red dashed line) is chosen to maximize the FOM while maintaining high signal purity.

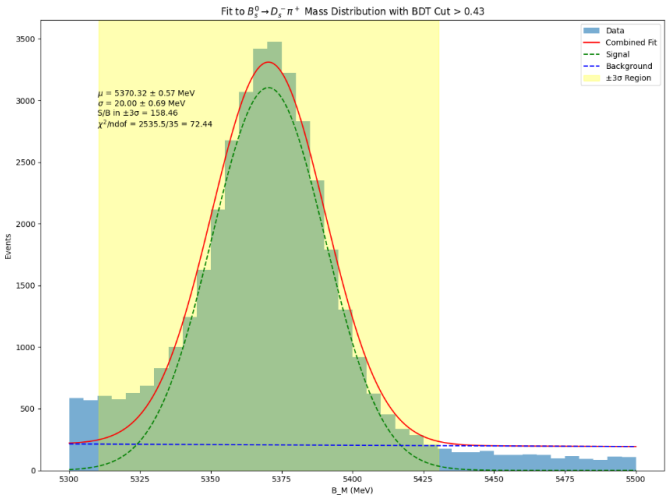


Figure 10: Mass fit after applying the BDT cut, showing improved separation between signal and background. The B_s^0 peak is now prominent and clean.

4 Part C2

In this section, we address the event classification of hadronic Z^0 decays into specific quark flavours. The task involves distinguishing between three classes: $Z^0 \to b\bar{b}$, $c\bar{c}$, and $s\bar{s}$, using features extracted from simulated e^+e^- collisions at the FCC-ee Z-threshold.

We approach this supervised multi-class classification problem by evaluating three different architectures: a Fully Connected Neural Network (FCNN), a Recurrent Neural Network (RNN) enhanced with an attention mechanism, and DeepSets, which leverages permutation invariance to effectively process unordered particle-level sets. Model performance is assessed through three evaluation strategies: confusion matrices, flavour-wise ROC curves, and flavour-wise classifier probability distributions.

Event Classification in HEP Detectors

In High Energy Physics (HEP), event classification refers to the process of distinguishing between different types of particle interactions or decays, such as signal events (which correspond to the desired physics process) and background events (which arise from other processes or detector noise). Event classification is typically done using a combination of kinematic, topological, and other event-level features that describe the properties of the particles involved in the event. Traditional methods like boosted decision trees (BDT) have been used effectively for segregation from engineered features (a lot of times), as we saw in part B2. But in this coursework, we use neural network architectures specifically – fully connected neural networks (FCNN), attention-based recurrent models (GRU), and DeepSets, which offer a more flexible and powerful approach capable of understanding underlying patterns and non-linear relationships in high-dimensional data, and are particularly good for large amounts of data. These can make more accurate predictions, improving the sensitivity of the event selection.

Data Description and Loading

The dataset consists of simulated $e^+e^- \to Z^0 \to q\bar{q}$ events at rest, generated using the IDEA detector model. Each event includes global event-level features (e.g., thrust), vertex-level features (e.g., number of tracks, vertex fit quality), and particle-level features (e.g., momentum components). These are stored in a ROOT file structure and organized event-wise, with each row corresponding to a different decay event of known flavour.

We load the data using `uproot` and construct a `pandas` dataframe, preparing a per-event structure suitable for model training. Since vertex-level and particle-level features vary in multiplicity across events, we aggregate them into fixed-size event-level features by computing the mean for the FCNN. For sequence- and set-based models, we preserve these features as sequences to exploit their underlying structure.

Useful Features for Tagging Quark Species

In particle physics, the features most useful for tagging quark species typically involve both kinematic and topological properties of the particles and vertices involved in an event. Key features for this task include `Thrust_x`, `Thrust_y`, and `Thrust_z`, which describe the event's momentum distribution and help differentiate quark species based on how the momentum is aligned within the detector.
Other important features are related to the event's vertex properties, such as `nVertex` (the number of reconstructed vertices), `Vertex_chi2` (the goodness-of-fit of the vertex), and `Vertex_ntracks`

(the number of tracks associated with the vertex). These provide insight into the topological complexity of the event. Additionally, `Vertex_chi2_mean` and `Vertex_chi2_std` offer a measure of the stability and consistency of the vertex reconstruction, which can also be indicative of the originating quark type.
Finally, the kinematic properties of individual particles, such as `Particle_p` (momentum) and `Particle_pt` (transverse momentum), provide critical information about how quarks are produced and interact. Collectively, these features are fed into the networks to perform the event classification task.

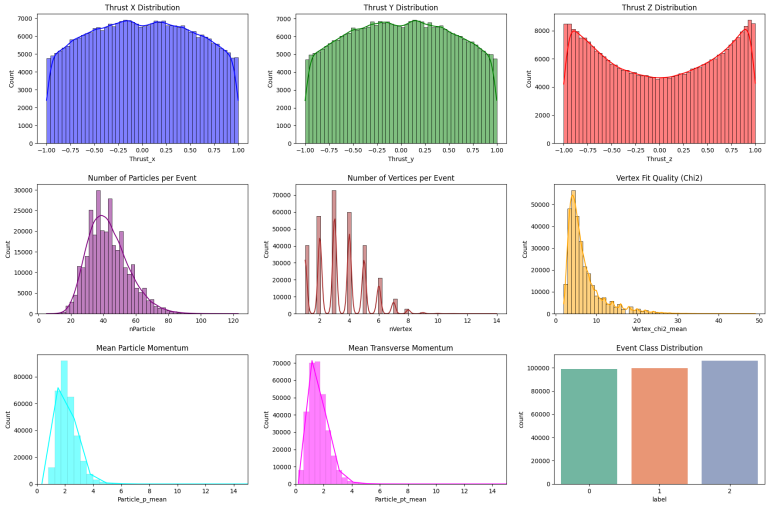


Figure 11: Selected Feature Distributions (Event-wise distribution).

The label, corresponding to one of the three decay classes ($b\bar{b}$, $c\bar{c}$, $s\bar{s}$), is given the values 0, 1, and 2, as shown in the last plot of Figure 11. This figure also depicts the distribution of all features on an event-wise basis.

Data Preprocessing

For preprocessing, we first apply outlier filtering using z-score thresholding: any event-wise feature value with a standard score magnitude greater than 4 is masked out. This is done using a per-feature mask, ensuring that extreme outliers do not skew training, as event-wise features are in the majority in our training set.

For the RNN and Deepsets models, we take an additional step in data preparation by converting all the event-wise features to particle-wise, sequential data by repeating them to match the maximum sequence length. This converts the data from a shape of $(n_{\text{events}}, n_{\text{features}})$ to $(n_{\text{events}}, \text{seq\_length}, n_{\text{features}})$ or $(n_{\text{samples}}, \text{set\_size}, n_{\text{features}})$, making it suitable for input into these models.

Following this, we apply normalization using the `keras.layers.Normalization()` layer. This ensures that the input distributions across features are centered and scaled appropriately, improving convergence speed and numerical stability during training. The normalizer is adapted to the training set only.

Model Architectures

To explore the effect of inductive biases on performance, we implement three different models using the `keras` API in TensorFlow.

1. Fully Connected Neural Network (FCNN)

As a baseline, we implement a simple Fully Connected Neural Network (FCNN) that takes only event-wise collapsed feature-level information.

2. Attention-Based GRU Network

To leverage the sequential nature of particle-level and vertex-level data, we implement a recurrent neural network based on GRU layers with an attention mechanism. Each particle feature vector is passed through a shared GRU cell, and attention weights are computed to emphasize the most informative particles.

3. Deepsets Model

Finally, we test a DeepSets model, which explicitly enforces permutation invariance over sets of particles. Each particle feature is passed through a shared Multi-Layer Perceptron (MLP) (the ϕ function), then aggregated using a sum and Global Max Pooling, both of which are permutation-invariant operations. The aggregated result is then passed through another MLP (the ρ function), which acts as the classifier head. This model is naturally suited for unordered set data and avoids the limitations of sequential models for unordered inputs.

Mathematical Foundation of DeepSets

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of particles, where each $x_i \in \mathbb{R}^d$ represents a feature vector for an individual particle. A permutation-invariant function $f(X)$, which operates on the set X, can be universally approximated by the following form:

$$f(X) = \rho\left(\sum_{x \in X} \phi(x)\right)$$

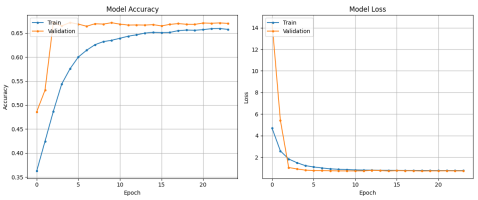
where:
-$\phi(x)$ is a transformation applied to each particle $x \in X$ through a shared function, mapping it to a latent space $\mathbb{R}^k$.
- The aggregation step $\sum_{x \in X} \phi(x)$ is a permutation-invariant operation, such as summation, averaging, or maximization.
ρ is another transformation applied to the aggregated result, producing the final output, which corresponds to the classification.

Training

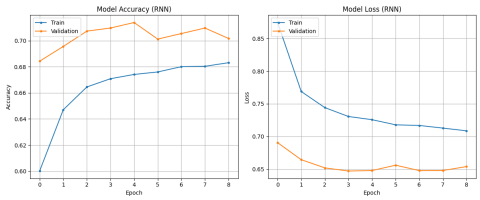
The training process was severely restricted due to the unreliability of the CSD3 nodes. We faced constant and irregular disconnections from the server, and there were extended periods during which we couldn't access it. As a result, we had to constrain the training to our local machine, which was limited to 50,000 events—the maximum our system could handle before running

out of memory or encountering kernel crashes. Data loading and preparation consumed high amounts of working memory, which we did not have access to, and there was no GPU available for training.

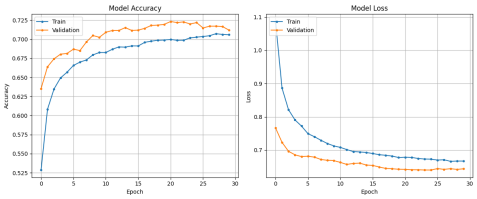
All models were trained using sparse categorical cross-entropy loss, directly taking the logits. The Adam optimizer was used, along with early stopping with a patience of 5, monitoring validation loss. Additionally, we applied learning rate decay on plateau with a factor of 0.3 and a patience of 3. The dataset was split into training, validation, and test sets with an 80/15/5 ratio, stratified by class. The training was conducted for 50 epochs with a batch size of 64 (see Figure 12).



(a) FCNN



(b) RNN



(c) Deepsets

Figure 12: Training/Validation Curves for FCNN, RNN, and Deepsets models.

Evaluation Metrics

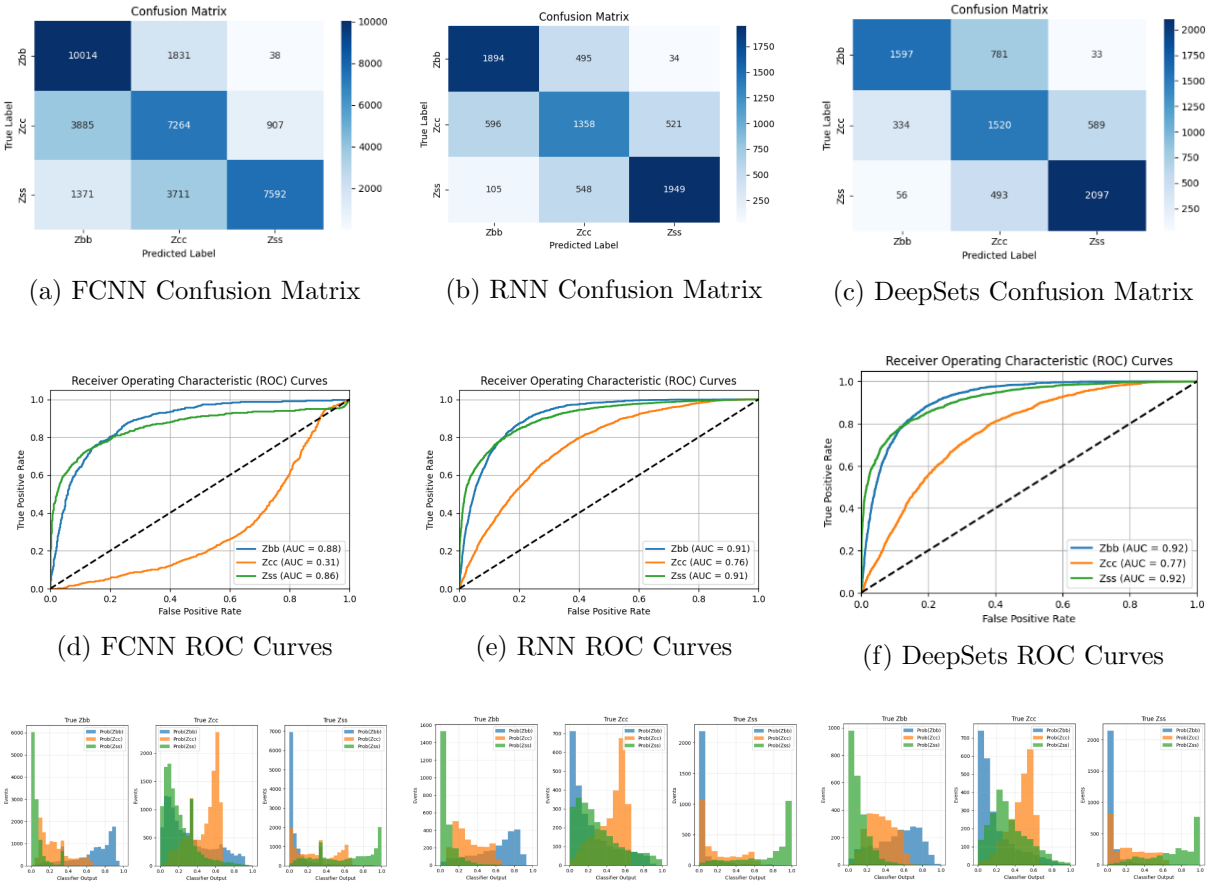
We evaluate model performance on a test set, using the following metrics:

- **Overall Test Accuracy** on the test set
- **Confusion Matrix**, visualized as a heatmap, to assess per-class performance
- **ROC Curves** for each class using a one-vs-rest approach
- **Classification Probability Distributions**, showing how confidently the network predicts each class

All the metrics are shown in Figure 13, except for the test accuracy, which is presented in Table 1.

Model	Test Accuracy
FCNN	66.36%
GRU + Attention	69.35%
Deepsets	72.19%

Table 1: Test Accuracy for the 3 models



(a) FCNN Confusion Matrix (b) RNN Confusion Matrix (c) DeepSets Confusion Matrix

(d) FCNN ROC Curves (e) RNN ROC Curves (f) DeepSets ROC Curves

(g) FCNN Class Probabilities (h) RNN Class Probabilities (i) DeepSets Class Probabilities

Figure 13: Comparison of model performance for FCNN, RNN, and DeepSets using (top row) confusion matrices, (middle row) ROC curves, and (bottom row) classification probability distributions.

Results and Discussion

One observation during our training process was that, even though the number of epochs was set to 50, we never really reached half of that. Both the loss and accuracy stagnated quickly, even with an adjustable learning rate, and the early stopping callback was triggered. This could indicate that the model was potentially lacking sufficient data for training and may not have had enough hidden units, both of which unfortunately require more memory and GPU resources, which we did not have access to.

The FCNN achieved a test accuracy of 66.3%, confirming that the model can learn rough class distinctions from global features. However, it performed poorly due to the loss of per-particle information and exhibited a notably high misclassification rate for $c\bar{c}$ quark decays.

The attention-based GRU model improved performance to 69.3%, demonstrating the value of treating particles as sequences. Attention weights indicated that focused learning was beneficial.

The best performance was achieved using the DeepSets architecture, with a test accuracy of 72.19%. This model effectively utilized all particle information in a permutation-invariant way and generalized well across all classes. ROC curves showed the strongest separation for $b\bar{b}$ events, followed by $s\bar{s}$.

5 Conclusion

This coursework has successfully explored multiple facets of machine learning applications in high-energy physics (HEP). In Section A2, we demonstrated the effectiveness of symmetry-aware neural network layers, specifically designed to respect discrete rotational symmetry in image classification tasks. The results showed that incorporating equivariant design principles can enhance model performance when applied to augmented datasets.

Section B2 provided insight into signal-background separation in the context of the $B_s^0 \to D_s^{\pm}\pi^{\mp}$ decay, where we used Boosted Decision Trees (BDT) to effectively discriminate between signal and background events. By optimizing a cut on the BDT score, we improved the signal-to-background ratio, offering a valuable technique for particle physics analyses in real-world datasets. This task exemplified the importance of machine learning in improving the precision of event selection, which is crucial for extracting meaningful physics from large, complex datasets.

In Section C2, we focused on the multi-class classification of Z^0 decays into three quark flavours: $b\bar{b}$, $c\bar{c}$, and $s\bar{s}$, using event-level, particle-level, and vertex-level features. We implemented and evaluated Fully Connected Neural Networks (FCNN), Attention-Based GRU, and permutation-invariant DeepSets models. Despite the constraints of the hardware environment, we observed promising results, demonstrating the potential of these models for classifying quark flavours in high-energy physics experiments.

References

[1] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alex Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, 2017. URL: `https://api.semanticscholar.org/CorpusID:4870287`.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.