Ludwig-Maximilians-Universität München

Faculty of Physics
AI in Physics Research Group

PP

Group A

Vyron Arvanitis
Emanuele Poggi
Paul Hofmann
Moritz Heidtmann
Angelos Aslanidis

Supervisor

Nikolai Krug

July 28, 2025

Contents

1 Introduction

In this lab course, we work with simulated data from the Belle II experiment to develop a fast and efficient simulation pipeline. The goal is to train and compare machine learning models that can predict early in the simulation chain whether an event is likely to survive later selection stages. This allows us to discard unimportant background events early, saving computational resources typically spent on full detector simulation and reconstruction (processess which are far more comptutaionaly expensive). The strategy is illustrated in Fig. 1, where a neural network acts as a pre-filter to speed up the overall simulation process.

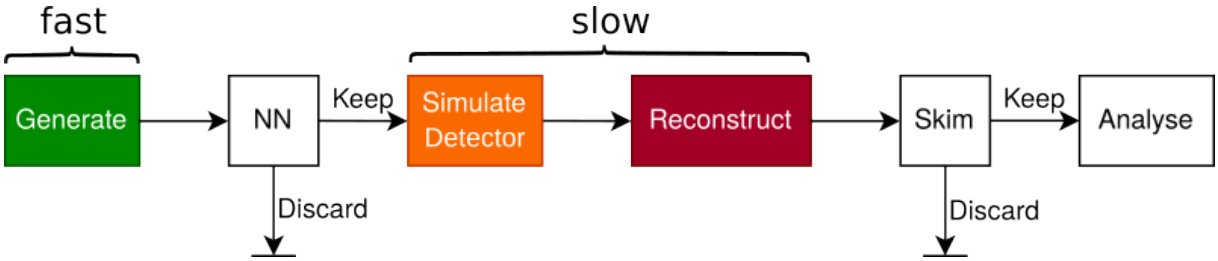

Figure 1: Smart simulation strategy: a neural network filters generated events before expensive detector simulation. Figure adapted from the official LMU AI Lab repository [3].

2 The Belle experiment

The Belle experiment is a so-called *B-Factory*, a type of particle accelerator specifically designed to produce large numbers of B *mesons*. These facilities operate with a center-of-mass energy tuned to the $\Upsilon(4S)$ *resonance*, a state that decays almost exclusively into pairs of B and $\bar{B}$ mesons. This selective production mechanism enables detailed studies of B meson decays, allowing physicists to investigate phenomena such as *CP violation*, rare decays, and physics beyond the Standard Model. The Belle II experiment, located at the *SuperKEKB collider in Tsukuba, Japan*, is the successor to the original Belle experiment and significantly improves on its precision and data collection capabilities.

Unlike proton-proton colliders (such as the LHC), Belle II uses electron-positron (e^+e^-) collisions. Electrons and positrons are elementary particles with no internal structure, which offers several advantages. First, their interactions are much cleaner, and the background is easier to interpret—there is a notable absence of complex hadronic activity such as jets. Moreover, the absence of internal substructure allows us to precisely know the initial momenta of the colliding particles, enabling accurate reconstruction of the event kinematics. This precision is crucial for tuning the aforementioned center-of-mass energy to specific resonances, such as the $\Upsilon(4S)$.

B mesons are unstable particles that decay via the weak interaction, and their decays can be classified into three categories: hadronic, semileptonic, and rare decays. In *hadronic decays*, the B meson decays into only hadrons, such as combinations of kaons (K), pions (π), and other mesons. *Semileptonic decays* involve both hadrons and leptons, typically resulting in a final state containing a charged lepton ($e^{\pm}$ or $\mu^{\pm}$), a neutrino, and hadrons. *Rare decays* may involve photons, multiple leptons, or other suppressed processes that are especially sensitive to

effects beyond the Standard Model. These decay modes provide a rich ground for studying CP violation, flavor physics, and potential new physics.

The Belle II detector is structured as a layered system of subdetectors (like an onion) built around the interaction point. These subdetectors are designed to detect and identify the various particles produced in the collisions. Closest to the interaction region is the Vertex Detector (VXD), composed of the PiXel Detector (PXD) and Silicon Vertex Detector (SVD), which provide precise tracking information for reconstructing decay vertices. Surrounding the VXD is the Central Drift Chamber (CDC), responsible for tracking charged particles and measuring their momenta and energy loss. Particle identification is handled by the Time Of Propagation (TOP) detector in the barrel region and the Aerogel Ring Imaging Cherenkov (ARICH) detector in the forward region.Photons and electrons are measured in the Electromagnetic Calorimeter (ECL) via the production of showers, while muons and long-lived neutral kaons (K_L^0) are identified in the outermost KLong and Muon (KLM) detector. The entire setup is enclosed in a 1.5 T magnetic field provided by a superconducting solenoid, which not only alligns the beams but also enables accurate momentum measurements. An overview of the collider (figure 2) and the particles that can be identified in each subdetector can be seen in table 1

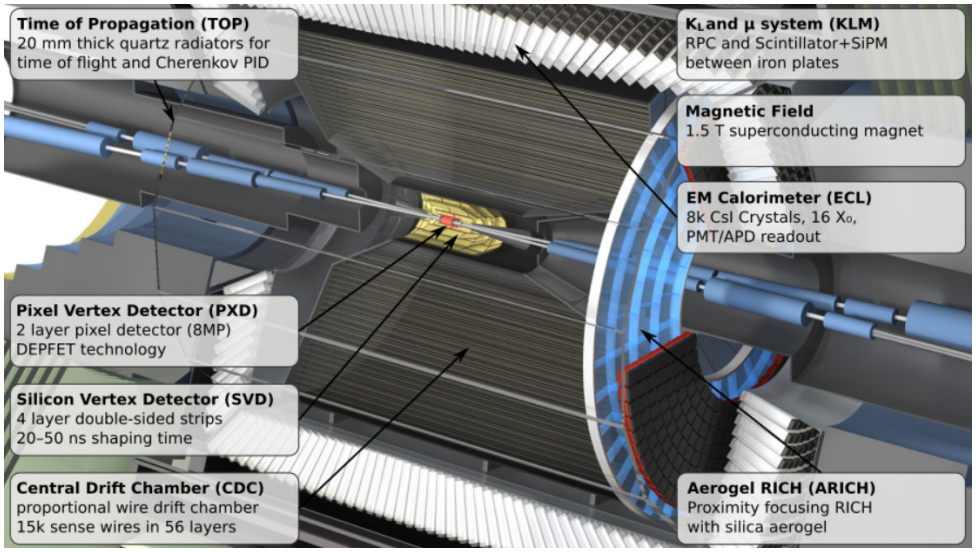

Figure 2: Closeup of the Belle II detector indicating all the different subdetectors [1].

Subdetector	**Main Purpose**
PXD + SVD (VXD)	Track reconstruction near the IP, measures short-lived particles
CDC	Tracking of charged particles
TOP / ARICH	Particle identification (e.g. distinguish π, K, p) via Cherenkov radiation
ECL	Detection of electrons and photons (electromagnetic showers)
KLM	μ identification and detection of K_L^0

Table 1: Belle II subdetectors and the particles they primarily detect.

2.1 Problem Description

The Belle II experiment collects an enormous amount of data, with its center-of-mass energy tuned to the $\Upsilon(4S)$ resonance *corresponding to a center-of-mass energy of* $\sqrt{s} = 10.58$ *GeV*. While these "resonant" events are the primary source for B physics, Belle II also records other types of data.

To achieve this center-of-mass energy the two beams colliding have respective energies of 7 GeV and 4 GeV. This assymetric beam configuration introduces a forward boost to the CoM system, forcing the produced B (and $\bar{B}$) mesons to travel measurables distances in the detector before they decay.

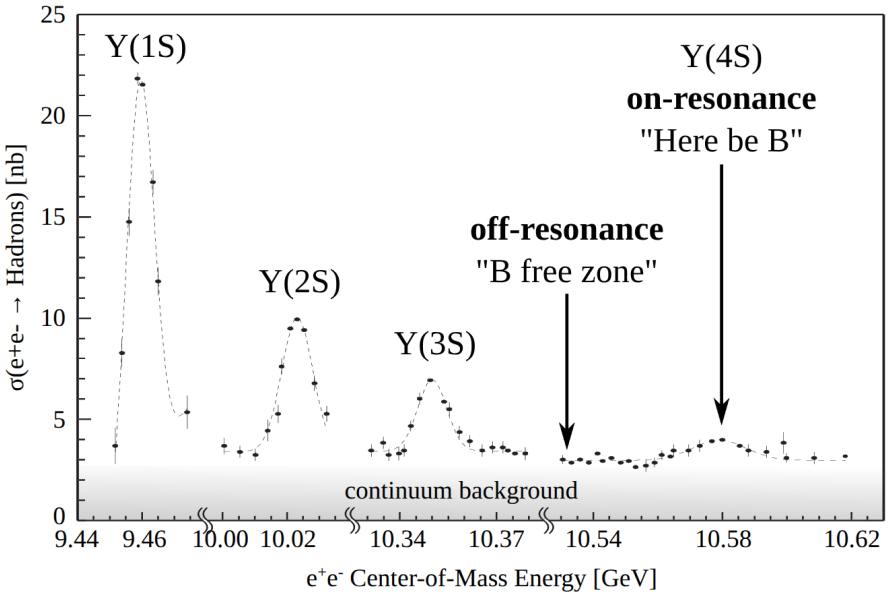

Figure 3: Cross section of e^+e^- with respect to the center-of-mass energy [GeV] [2].

To reduce the data volume and focus on interesting events, Belle II employs a two-level filtering system: the Level-1 hardware trigger (TRG) and the High Level Trigger (HLT), which together reduce the event rate before data is stored. Even after this, the number of events retained is still too large for direct analysis, so further selection — known as skimming — is performed to extract smaller datasets.

In parallel, Monte Carlo (MC) simulations are used to model the detector response and to train and validate analysis strategies. These simulated events must pass through all stages of detector simulation, digitization, reconstruction, and selection, mirroring the real data processing pipeline. However,this process is computationally expensive, especially when simulating large background samples, most of which are eventually discarded by skims.

To address this issue, the goal of this lab course is to explore a technique known as *Smart Background Simulation*. Instead of fully simulating every event, we aim to train a neural network that can predict early on whether a given event will survive a later skim. This skim selects events in which at least one B^0 meson can be reconstructed through hadronic decays. After building several neural networks we evaluate their performance on an independent test set and calculate

the speedup offered by each network.

3 Theory

3.1 Deep Sets

In many physical systems, such as collections of particles or events in high-energy physics, the data can be naturally represented as unordered sets. Traditional neural networks, however, are not inherently permutation-invariant. This means that they assume an order in their input. To address this, Deep Sets provide a principled way to process such set-structured data while maintaining invariance to the order of elements. Figure 4 visualizes the structure of a Deep Set model.

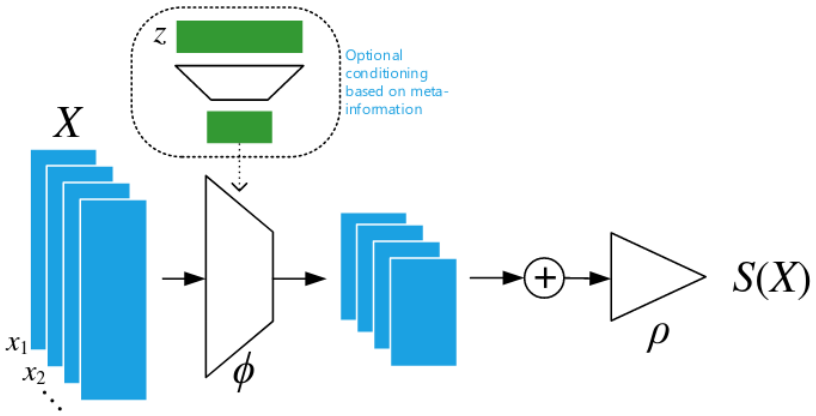

Figure 4: Schematic illustration of a Deep Set model.

A Deep Set model operates by applying a shared function ϕ independently to each element in the set. This produces a new set of embeddings, which are then aggregated using a permutation-invariant operation such as sum, mean, or max. The aggregated result is passed through a second function ρ, typically implemented as a multilayer perceptron (MLP), to produce the final output:

$$f(X) = \rho\left(\sum_{x \in X} \phi(x)\right) \tag{1}$$

This structure ensures that the model respects the symmetry of sets. The theoretical foundation of Deep Sets is discussed in [4], where it is shown that any function on a set that is invariant to permutations can be decomposed in this form.

In our project, we use this architecture to model collections of particles, where the order of particles is physically irrelevant. The Deep Set model thus provides a natural and efficient baseline for learning from such data.

3.2 Graph Convolutional Neural Networks

In contrast to Deep Set models, which treat the input as an unordered collection, Graph Neural Networks (GNNs) explicitly leverage relational information between data points. This is particularly important in our context, where particles are connected through a decay tree structure that naturally forms a graph. Each node in the graph represents a particle, and edges represent physical relationships (e.g., parent-child connections in the decay chain).

In terms of Deep Sets, graph convolutions can be seen as permutation equivariant operations. As long as the aggregation function (e.g., sum, mean, or max) is symmetric, the update remains invariant to the ordering of neighbors.

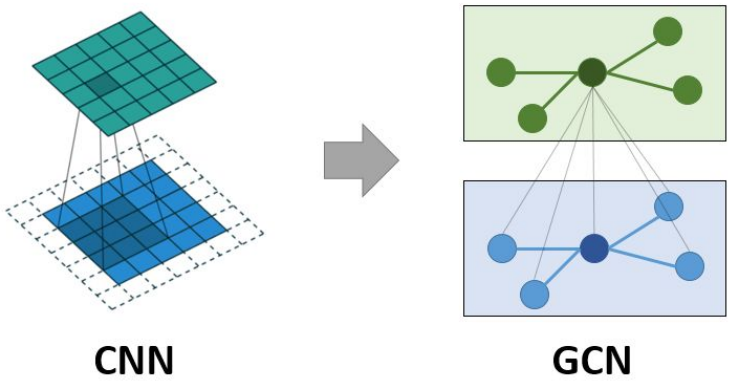

Figure 5: Analogy between CNNs and GCNs. In CNNs, pixels are updated based on spatial neighborhoods; in GCNs, nodes are updated based on graph neighborhoods.

Figure 5 illustrates the analogy between Convolutional Neural Networks (CNNs) and Graph Convolutional Networks (GCNs). In CNNs, pixels are updated based on their spatial neighborhoods, while in GCNs, nodes are updated based on their graph neighborhoods. This allows GNNs to capture complex relationships and dependencies in the data, making them particularly suitable for tasks involving structured data like particle decays.

3.3 Multi Layer Perceptron

The Multi Layer architecture is as

3.4 Transformer

The transformer architecture is as

4 Methodology

To identify the optimal model architecture, each group member conducted small-scale investigations targeting specific aspects of the model design. These investigations included the following:

- Does it help to increase the number of linear layes?

- Which sequence of GCN and Linear layers works best?
- Is it possible to use a standard MLP as well? If yes how?
- How important is the masking in this case?
- Methods to combat overfitting
- Do we need to transform the values for the particle features?
- Are there any redundant features in the dataset that could be removed?
- Could one make use of some (approximate) symmetries in the data?
- Ideas for different model architectures

4.1 The Dataset

The dataset of the experiment consists of simulated particle collision events, with each event containing multiple decay particles. These events contain a variable number of final state and intermediate particles. Hence, they can be structured as a graph, where each particle is a node and the edges represent the relationships between particles (e.g., parent-child relationships in decays). The graph can be seen in figure 6.

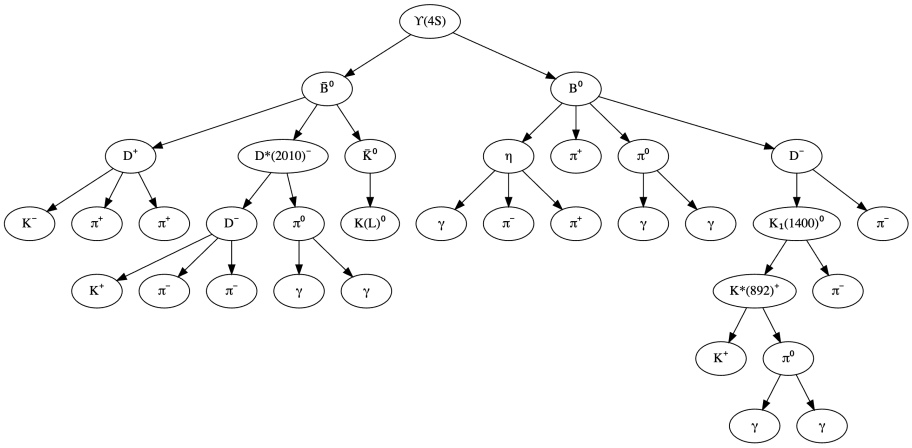

Figure 6: Example of a particle decay graph. The nodes represent particles, and the edges represent the decay relationships between them.

Each particle is described by a fixed number of features, which include:

- `prodTime` — the production time of the particle,
- `energy` — the energy of the particle,
- `x`, `y`, `z` — the spatial coordinates of the particle at production,
- `px`, `py`, `pz` — the momentum components of the particle,
- `pdg` — the PDG identifier of the particle, indicating its type,
- `index`, `mother_index` — the particle's index and the index of its mother particle (if any).

- `label` — this indicates weather or not the particle has passed the downstream event selection or not, it takes binary values 0 or 1

Each event is represented as a tensor of shape `(num_particles, num_features)`. Since the number of particles can vary between events, padding is applied to ensure consistent input dimensions across batches. These padded values are properly masked during training to prevent them from affecting the model.

4.2 Transforming the feature values

For this investigation, we used our best model obtained after the first lab day — that is **CombinedModel with GCN** model — and focused on normalizing the inputs. The normalization was done according to the following function

```
def normalize_inputs(inputs):
    x = inputs["feat"] # with "feat" we mean the position, momentum
        vectors, the production time and the Energy
    # x.shape = (batch_size, num_particles, num_features)
    mean = x.mean(dim=(0,1), keepdim=True)  # Collapse batch and
        particle dimensions; end up with (num_features) -> one mean per
        feature!
    std = x.std(dim=(0,1), keepdim=True) + 1e-8  # avoid divide-by-zero
    x_norm = (x - mean) / std
    return {**inputs, "feat": x_norm}
```

Essentially, we subtracted the mean and divided by the standard deviation of each feature, calculated across both the batch and particle dimensions.

4.3 Unecessary Features

To investigate whether or not some features are unnecessary in our analysis, we used again the same model (**CombinedModel with GCN**) and evaluated it, subtracting features via physical arguments. We performed six tests.

First of all, we removed the momentum vector ($\vec{p}$). The argument for this was that the energy and momenta of a particle are associated with the well-known relationship:

$$E^2 = p^2 + m^2$$

so, in principle, the model could reconstruct the momenta from the energy (or vice versa), assuming the mass is constant or implicitly learned.

For the same reason mentioned above, we explored removing the energy of the particles.

Afterwards, we removed the position vectors x, y, and z. This started as a simple test, however the following argument was made: in our problem, there exists a cylindrical symmetry — the x and y axes can be arbitrarily chosen (as the beams are considered to be colliding along the z axis). Therefore, we considered removing only the x and y coordinates.

However, after consideration we came to the conclusion that this is physically wrong because removing both x and y would eliminate any information about the transverse plane. Therefore, we opted to remove only the x position coordinate, as a compromise to test sensitivity to symmetry arguments.

In the end, we removed the `prodTime` feature just for further analysis, since its physical importance was unclear and we wanted to check if it had any significant impact on performance.

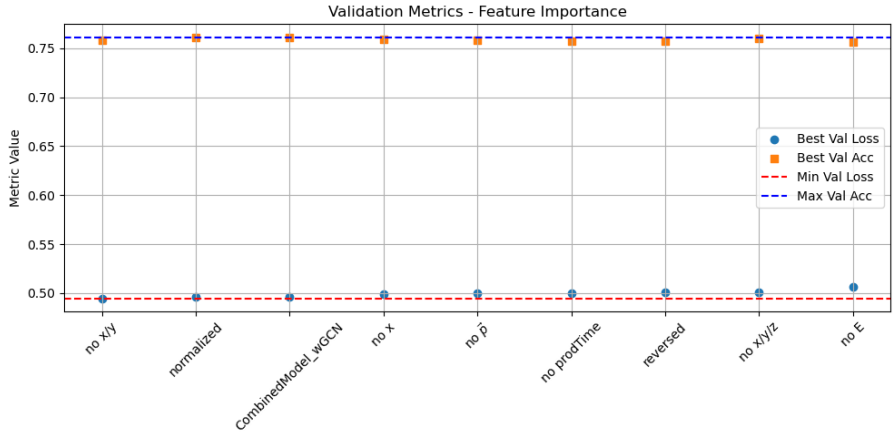

Figure 7: Validation accuracy and loss for different experiments. The horizontal dashed lines indicate the best performance achieved with the baseline model. Note: the **reversed** model refers to the configuration where the GCN and DeepSet layers were swapped.

From figure 7 it is evident that no significant improvement in model performance was observed when removing features. In all cases, the validation accuracy and loss remained close to the baseline values. The only slight performance gain was observed when applying normalization to the input features, confirming its importance as a preprocessing step.

4.4 Speedup

To evaluate the efficiency of using a NN as a filter before detector simulation, we define the speedup as the ratio of time needed to process events *without* and *with* the NN, for the same number of positively selected (skimmed) events.

Let:

- t_g: time for event generation
- t_s: time for detector simulation and reconstruction
- $r = 0.05$: fraction of events passing the original skim (without NN)
- f_1: true positive rate (TPR)
- f_0: false positive rate (FPR)
- X: number of events generated

Further, we define:

$$\begin{aligned} X_i &= r \cdot X \quad \text{(number of events passing skim)} \\ X_i &= f_1 \cdot X' \quad \text{(true positives from NN)} \\ \Rightarrow X' &= \frac{r}{f_1} \cdot X \quad \text{(number of events to be generated with NN)} \\ P &= f_1 \cdot X' + f_0 \cdot (1 - r) \cdot X \quad \text{(events passing NN)} \end{aligned}$$

The speedup is the ratio of total time without and with NN:

$$\text{Speedup} = \frac{X \cdot (t_g + t_s)}{X' \cdot t_g + P \cdot t_s}$$

Substituting X' and P into the equation:

$$\text{Speedup} = \frac{f_1 \cdot (t_g + t_s)}{t_g + t_s \cdot \left(f_1 + f_0 \cdot \frac{1-r}{r}\right)}$$

Finally, assuming $t_s = 100 \cdot t_g$, we simplify to:

$$\text{Speedup} = \frac{101 \cdot f_1}{1 + 100 \cdot \left(f_1 + f_0 \cdot \left(\frac{1-r}{r}\right)\right)}$$

4.5 Models architecture

From our comulative research we came to the conclusion that the following are the best models, we mention also more simplistic models as it is interesting to see the impact that additional layers or methods of regularization have on the performance.

DeepSet Model

This model follows the Deep Sets framework (as mentioned in 3.1), where the same transformation is applied independently to each element of the input set before aggregation.

Architecture. The model takes as input a batch of events, where each event consists of a list of particles. Each particle is represented by a feature vector. The architecture consists of the following components:

- Linear layer with Relu activation function
- A masking layer with averaging over the masked events or averaging if no masks exists
- Linear layer with Relu activation function
- Output layer (a single neuron)

Combined Model

This model extends the deepset architecture by introducing PDG code to each particles afterwards mapped to an embedding vector. The embedding vectors are added to the original particle feature before being processed by the rest of the network

Architecture. The model processes a batch of events, where each particle is represented by both a feature vector and a PDG code. The architecture consists of the following components:

- Embedding layer that maps each PDG code to a vector.
- Concatenation of the PDG embedding with the original particle features.
- Linear layer with ReLU activation function.
- A masking layer with averaging over the masked events or simple averaging if no mask is used.
- Linear layer with ReLU activation function.
- Output layer (a single neuron).

DeepSet with GCN

This model combines a Graph Convolutional Network (GCN) layer with the Deep Set architecture. The GCN captures local structure between particles as defined by an adjacency matrix (typically based on mother-daughter relations),

Architecture. The input consists of a set of particles (each with a feature vector) and a corresponding adjacency matrix encoding connections between particles. The architecture includes the following components:

- A GCN layer that applies a linear transformation to each particle this is done by multiplying the normalized adjacency matrix with a linear layer
- A DeepSet layer that aggregates the GCN output using a masked mean over particles, followed by a linear transformation with ReLU activation.
- An output layer (a single neuron) that produces the final prediction.

Combined Deepset with GCN

This model extends the DeepSet with GCN architecture by introducing a PDG code to each particles afterwards mapped to an embedding vector

Architecture. The input consists of particle features, PDG codes, and an adjacency matrix. The architecture includes the following components:

- An embedding layer that maps each PDG code to a vector.
- A droupout layer with a droupout rate of 0.3.

- Concatenation of the PDG embedding with the original particle features and the normalized adjacency matrix.
- A GCN layer applied to the concatenated inputs.
- A batch normalization layer
- A dropout layer with a droupout rate of 0.3
- A Deep Set layer.
- An output layer (a single neuron) that produces the final prediction.

Combined Deepset with GCN Normalized

This model extends the Combined Deepset with GCN architecture (model 4.5) by introducing a normalization of the inputs according to section 4.2.

Architecture. The architecture is the same as in the case of 4.5 with an additional normalization layer.

DeepSet with GCN variable

The architecture was designed to investigate the optimal number and arrangement of GCN and linear layers. To support this, an additional script (deepset_gcn_variator.py) was introduced, enabling dynamic naming based on layer configuration. The naming convention follows the pattern: `DS_{TotalLayers}_GCN_{GCNIndices}`. For example, the model `"DS_6_GCN_135"` consists of six layers in total, which are by default linear, with the 1st, 3rd, and 5th layer replaced by GCNs.

The model takes four arguments:

- **hidden_layers (int)**: Gives the number of hidden layers. The total number of layers is then determined by the number of hidden layers + 2 (in- and output).
- **gcn_layers (list)**: A list of layer indices to be implemented as GCN layers. For example `gcn_layers = [2,4]` replaces the 2nd and 4th layers with GCNs.
- **num_features (int)**: Defines the number of input features and determines the shape of the input layer.
- **units**: Sets the dimensionality of hidden layers.

If `gcn_layers` includes the index 1, the input layer is implemented as a GCN, otherwise a linear input layer is used by default. In both cases, the input layer has shape `num_features` $\times$ `units`.

Hidden layers not specified in `gcn_layers` are implemented as linear layers. All hidden layers, whether linear or GCN, have shape `units` $\times$ `units`. The final output layer is always a linear pooling layer of shape `units` $\times$ 1.

Transformer

5 Results

For the final results we evaluated our models on a new test dataset (i.e. a dataset not used throught the training process) and used the following models:

- deepset 4.5
- deepset combined 4.5
- deepset gcn 4.5
- deepset combined with gcn 4.5
- transformer 4.5
- deepset combined wwith gcn normalized 4.5

5.1 Model Evaluation

To evaluate the performance of each model, we use the Receiver Operating Characteristic (ROC) curve, a widely adopted metric for binary classification tasks. The ROC curve plots the **True Positive Rate (TPR)** against the **False Positive Rate (FPR)**.

Definitions.

- **True Positive Rate (TPR):** It measures the proportion of actual positives that are correctly identified:
$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
- **False Positive Rate (FPR):** It measures the proportion of actual negatives that are incorrectly identified as positives:
$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

The ideal model achieves a high TPR while maintaining a low FPR. The performance of a model can be quantified using the **Area Under the Curve (AUC)**. A higher AUC indicates better separability between the two classes.

Interpretation.

- AUC $= 1.0$: Perfect classifier.
- AUC $= 0.5$: Random guessing.
- AUC < 0.5: Model performs worse than random (potentially inverted labels).

5.2 Optimal Model

6 Conclusion

References

[1] Belle II Collaboration. Belle ii detector overview. https://software.belle2.org/development/sphinx/_images/detector_labeled.jpg, 2023. Accessed: 2025-07-12.

[2] Belle II Collaboration. Belle ii detector overview. https://software.belle2.org/development/sphinx/_images/Uresonances.svg, 2023. Accessed: 2025-07-12.

[3] Nikolai Krug. Preparation materials for particle physics lab group a. https://gitlab.physik.uni-muenchen.de/ai-lab/ss25/pp-lab-group-a/-/blob/main/preparation.md, 2025. Accessed: July 8, 2025.

[4] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.