Google Summer of Code Proposal 2025

Arnav Nigam

April 2, 2025

Quantum Graph Neural Networks for High Energy Physics Analysis at the LHC

arnav3108nigam@gmail.com

+91 7834902251

Delhi Technological University
Bachelor of Technology, Computer Science Engineering
India / GMT+5:30
GitHub *LinkedIn*

To give feedback, please contact the email address

1. Overview

1.1 Project Abstract

The High Luminosity LHC (HL-LHC) program will generate unprecedented volumes of data in the coming decades, imposing enormous demands on computing resources. In order to improve the sensitivity for new physics—by efficiently identifying rare signals amid vast backgrounds—innovative computational methods are essential. This project aims to develop and implement Quantum Machine Learning (QML) methods for High-Energy Physics (HEP) analysis at the LHC. Specifically, we propose to design and benchmark a Quantum Graph Neural Network (QGNN) using the Pennylane framework. By leveraging quantum computing's potential for exponential scaling, our goal is to enhance the HEP community's ability to process and analyze LHC data with improved performance over classical methods.

1.2 Benefits to the HEP and Quantum Communities

HEP Analysis: The QGNN approach promises to better identify rare signal events in the immense background of collision data, potentially accelerating discoveries of new physics.

Resource Efficiency: Quantum algorithms may provide computational advantages in handling the complex, graph-structured data typical in jet tagging and event classification tasks.

Open-Source Contribution: The developed code and documentation (tutorial-like Jupyter Notebooks) will be released as open-source resources, serving as a foundation for future research in QML for HEP.

Community Impact: By using established frameworks such as Pennylane, the project will bridge quantum computing research with practical HEP analysis, enhancing the exposure and usability of QML methods within the HEP community.

2. Goals and Deliverables

2.1 Goals

Develop a Quantum Graph Neural Network (QGNN) using Pennylane:

The primary objective is to implement a Quantum Graph Neural Network (QGNN) using the Pennylane framework. This involves designing a quantum circuit architecture capable of encoding graph-structured data from the Large Hadron Collider (LHC), where particle jets can be represented as graphs. The model should effectively perform node and edge aggregation in Hilbert space, allowing quantum-enhanced feature extraction from high-energy physics data.

Benchmark the QGNN against Classical Machine Learning Methods:

To assess the efficiency and advantages of the QGNN, it will be applied to a benchmark high-energy physics (HEP) problem, such as jet tagging. The model's performance will be compared against classical machine learning approaches, including Graph Neural Networks (GNNs), Convolutional Neural Networks (CNNs), and Support Vector Machines (SVMs). The evaluation will consider classification accuracy, area under the ROC curve (AUC), training and inference time, and resource utilization. This benchmarking will help determine the quantum model's strengths, limitations, and potential benefits in HEP data analysis.

2.2 Deliverables

Comprehensive Code Repository:

A well-structured Python implementation of the QGNN using Pennylane, including modules for data encoding, quantum circuit design, and training. Classical baseline methods (GNNs, CNNs, SVMs) will be included for fair benchmarking.

Documentation and Tutorials:

Jupyter Notebooks covering the complete pipeline, with tutorial-style explanations and detailed documentation to facilitate reproducibility and ease of use.

Benchmark Results and Comparative Analysis:

A report summarizing key performance metrics (accuracy, AUC, loss curves, ROC curves) and a comparative study highlighting the advantages and limitations of QGNN over classical models.

Bug Reports and Contributions to Pennylane:

Any encountered issues with Pennylane will be documented and reported to the community to improve the framework for quantum machine learning applications.

2.3 Optional Contributions

Research Publication:

If the project achieves promising results, a research paper will be written in collaboration with mentors. The paper will document the methodology, findings, and implications of the study. The final manuscript will be submitted to high-impact journals, conferences, or workshops focused on quantum computing and high-energy physics. This will help disseminate the knowledge and encourage further research in this domain.

Open-Source Contributions:

The best-performing QGNN architectures developed during this project will be integrated into relevant open-source quantum machine learning frameworks for high-energy physics. Contributions will also be made to quantum computing libraries such as Pennylane, ensuring that the developed methods are accessible to the broader research community. Additionally, efforts will be made to provide well-documented code, model architectures, and usage guidelines to support future research and development in this area.

3. Dataset

Dataset Description:

We will use the "Pythia8 Quark and Gluon Jets for Energy Flow" dataset, which provides two collections of jets generated with Pythia 8. One collection includes all kinematically realizable quark jets, and the other excludes charm and bottom quark jets. This dataset is widely used in high-energy physics studies for jet analysis tasks.

Data Structure:

Each of the 20 files in the dataset (stored in compressed NumPy format) contains two arrays. The first array, X, has the shape (100000, M, 4), where each file includes exactly 50k quark jets and 50k gluon jets. Here, M represents the maximum particle multiplicity in a jet, with shorter jets padded with zero-particles. Each particle is characterized by four features: transverse momentum (pt), rapidity, azimuthal angle, and pdgid. The second array, y, is a label array of shape (100000,), where 1 indicates a quark jet and 0 a gluon jet.

Raw Data Visualizations:

Initial visualizations of the raw dataset include a 3D graph and a series of 2D histograms. The 3D graph plots key kinematic features—such as transverse momentum, rapidity, and azimuthal angle—demonstrating the spatial and kinematic structure of the jets. Additionally, three separate 2D histograms have been generated, each mapping one feature on the x-axis to another on the y-axis, to illustrate the pairwise correlations between different particle features. These visualizations help to understand the underlying structure and distribution of the data before further processing. The data is plotted from only one file as dataset is large for plotting the entire data.

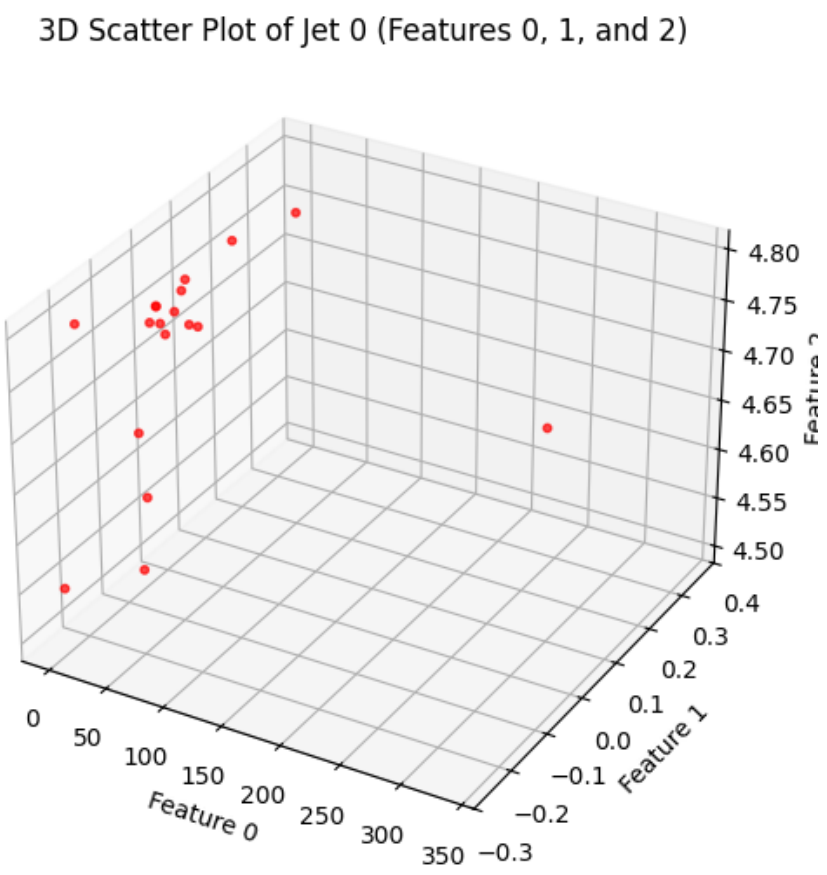
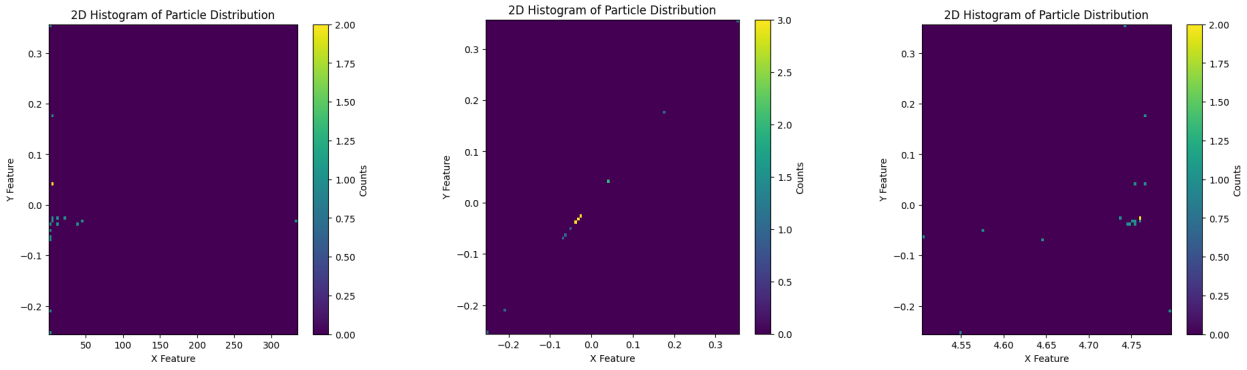


Figure 1: 3D Visualization of Particle Jets



(a) First feature of X with Y (b) Second feature of X with Y (c) Third Feature of X with Y

Figure 2: Plot of all 3 features with Y

4. Preprocessing

Handling Padded Zeros:

Load the NumPy files and identify the padded zero-particles. Remove or mask these values so that only valid particle data contribute to the subsequent processing steps.

Normalization:

Normalize the particle features (transverse momentum, rapidity, azimuthal angle, pdgid) to an appropriate range, for example using min-max scaling to [0, 1] or z-score standardization. This step ensures consistency and compatibility with the quantum encoding process.

Dimensionality Reduction:

We can try applying PCA to reduce the dimensions which will help in managing the limited number of qubits available for quantum encoding. While trying to plot the graph I was stuck as the size of graph was to big to be plotted so we have to try some method to reduce the data size and dimensions.

```
Dataset contains 5001 graphs.
Visualizing graph 0 ...
Visualizing graph 0 with 29856846 nodes and 298545148 edges.
```

Figure 3: Actual graph after preprocessing

Graph Construction:

In our approach, each jet is treated as an individual graph. Every particle within a jet becomes a node, and the node values (or features) are derived from the particle's kinematic properties—namely, transverse momentum (pt), rapidity and azimuthal angle. To capture the spatial and kinematic relationships between particles, we establish edges based on a distance metric in the (ϕ, y) space. Specifically, we calculate the distance R using the formula:

$$\Delta R = \sqrt{(\Delta\phi)^2 + (\Delta y)^2}$$

An edge is then created between two nodes if they are sufficiently close according to this metric, with the edge value typically being the computed ΔR. This setup allows the graph to encode both the individual particle properties and the relational structure between them, which is crucial for subsequent analysis using graph neural networks.

5. Data Encoding

Quantum Graph Neural Networks (QGNNs) require an efficient way to encode classical graph data (nodes, edges, and features) into quantum states. The choice of encoding impacts computational efficiency, expressivity, and overall model performance.

5.1 Basic Encoding

This is the simplest method to convert classical data into quantum states. It maps binary data directly into the computational basis of qubits. Given a dataset D consisting of M binary strings x^m of length n, the entire dataset can be represented as:

$$|D\rangle = \frac{1}{\sqrt{M}} \sum_{m=1}^{M} |x^m\rangle$$

Each classical binary string is mapped to a computational basis state $|x^m\rangle$.

5.2 Amplitude Encoding

Amplitude encoding represents classical data as amplitudes of quantum states, allowing for efficient storage of high-dimensional data using a logarithmic number of qubits. Given a feature vector $x = (x_1, x_2, \ldots, x_N)$it is encoded as:

$$|x\rangle = \frac{1}{\sqrt{\sum_{k=1}^{N} x_k^2}} \sum_{k=1}^{N} x_k |i_k\rangle$$

where $|i_k\rangle$ is the computational basis state. This method enables an exponential reduction in the required qubits but is sensitive to noise and requires specialized quantum circuits for loading data.

5.3 Angle Encoding

Angle encoding maps classical features to quantum gate rotation angles. A feature vector $x = (x_1, x_2, \ldots, x_N)$ is encoded by applying rotation gates such as R_x, R_y, or R_z:

$$|x\rangle = R_y(x_1) R_y(x_2) \ldots R_y(x_N) |0\rangle$$

where the R_y gate is defined as:

$$R_y(x_i) = \begin{bmatrix} \cos(x_i/2) & -\sin(x_i/2) \\ \sin(x_i/2) & \cos(x_i/2) \end{bmatrix}$$

This encoding is efficient and commonly used in variational quantum algorithms. However, it may require multiple layers to capture complex relationships in the data.

5.4 Hamiltonian Encoding

Hamiltonian encoding represents structured data, such as graphs, using a quantum Hamiltonian. The quantum state evolves according to:

$$|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle$$

where H is the Hamiltonian operator describing the system:

$$H = \sum_{i,j} w_{ij} \sigma_i \otimes \sigma_j$$

where w_{ij} are edge weights, and σ are Pauli matrices.

This method is particularly useful for quantum graph neural networks (QGNNs) and variational quantum algorithms.

6. Graph Neural Network(GNN)

Graph Neural Networks (GNNs) are a powerful class of models designed to process graph-structured data. In the context of high-energy physics (HEP), GNNs enable the extraction of meaningful patterns from jets by treating them as graphs, where nodes represent particles and edges capture spatial or kinematic relationships.

Architecture and Message Passing:

GNNs operate by iteratively updating node representations through a message-passing framework. At each layer, nodes aggregate information from their neighbors using functions (such as summation or mean pooling) and then update their own features using trainable weight matrices and activation functions. This process allows the network to capture local connectivity and global structure, making it highly effective for tasks such as jet tagging.

Implementation for HEP Analysis:

In our project, classical GNNs will serve as baselines to compare our Quantum Graph Neural Network (QGNN). The chosen GNN architecture will be tailored for jet classification, where preprocessed jet graphs are fed into the network. We will implement layers that aggregate node features and incorporate edge attributes derived from kinematic distances. Performance metrics like classification accuracy and AUC will be used to evaluate the classical GNN models and compare them with the quantum approach.

I have already explained 4 GNN architecture in Task 2. Please see the task2.ipynb file for detailed implementation.

7. Quantum Graph Neural Network (QGNN)

A Quantum Graph Neural Network (QGNN) integrates quantum computing principles with graph-based learning. It leverages quantum circuits to perform node and edge feature transformations, potentially providing computational advantages over classical models. QGNNs can exploit quantum entanglement and superposition to enhance relational learning tasks in high-energy physics (HEP).

Quantum Circuit Design for Graphs

- Node Encoding: Each node's features (jet properties) are encoded into quantum states using Parameterized Quantum Circuits (PQCs).
- Edge Information Processing: Quantum operations model interactions between connected nodes, similar to classical message passing.
- Quantum Aggregation: Quantum measurements replace classical pooling operations (e.g., mean or max aggregation).
- Graph Representation Learning: The final quantum state is measured and post-processed for downstream tasks such as classification (quark vs. gluon jets).

Advantages of QGNN in HEP

- Parallel Processing: Quantum superposition allows multiple graph transformations simultaneously.
- Expressivity: Quantum gates capture complex graph dependencies, potentially outperforming classical methods.
- Computational Scaling: Quantum GNNs may scale better for large particle-physics datasets compared to classical GNNs.

Implementation Strategy

- Framework: Implement QGNN using Pennylane and integrate with PyTorch for hybrid quantum-classical training.
- Quantum Layers: Use variational quantum circuits (VQCs) to replace standard neural network layers.

One example of QGNN is already discussed in Task5, please refer task5.ipynb to see. I will try new architecture and refine them.

8. Optimization and training of QGNN

8.1 Training process

Training a Quantum Graph Neural Network (QGNN) involves optimizing quantum and classical parameters simultaneously. We use a hybrid training approach, where a classical optimizer updates both quantum circuit parameters and graph-based transformations.

Forward Pass:

- The input graph is encoded into a quantum state using Variational Quantum Circuits (VQCs).
- Quantum computations transform node and edge features through unitary operations.
- A quantum measurement extracts graph embeddings for final prediction (e.g., jet classification).

Backward Pass:

- The model computes a loss function (e.g., cross-entropy for classification).
- Quantum gradients are computed using parameter shift rules (since quantum circuits are non-differentiable by standard backpropagation).
- Classical optimizers like Adam, RMSProp, or Stochastic Gradient Descent (SGD) update both classical and quantum parameters.

8.2 Optimization Strategies

Quantum Circuit Optimization:

- Reduce quantum depth to mitigate noise and hardware constraints.
- Use ansatz selection (choosing the right quantum architecture) to avoid barren plateaus.

Hybrid Classical-Quantum Optimization:

- Implement layer-wise training (training classical and quantum layers separately to improve convergence).
- Transfer learning is used by initializing QGNN layers with pre-trained classical GNN weights.

Regularization Techniques:

- Apply Dropout and Batch Normalization in classical layers.
- Use quantum noise-aware training to improve stability.

9. GSoC Timeline

9.1 Application Review Period (April 8 – May 8)

During this initial phase, the focus will be on learning the necessary quantum frameworks (Pennylane, Qiskit, TensorFlow Quantum, and Cirq). I will review the current state of quantum machine learning, study recent literature, and implement small-scale examples from previous work. This period will serve as the foundation for all subsequent work.

9.2 Community Bonding (May 8 - June 1)

9.2.1 Review (May 8 - May 15)

Engage with mentors to discuss the overall proposal. Obtain feedback to refine goals, deliverables, and timeline details. Finalize the choice of datasets, the specific QGNN model architecture, loss functions, gradient techniques, optimizers, and evaluation strategies. Discuss hardware options (local resources vs. cloud platforms).

9.2.2 Data Preprocessing(May 15 - May 25)

Set up the development environment and install all required packages. In a series of Jupyter Notebooks, perform the necessary data preprocessing steps—such as cropping, pooling, applying PCA, and normalizing the values—to prepare the LHC HEP data for model training.

9.2.3 Community Bonding (May 25- June 1)

I would like to learn more about the mentors, organization and network with other contributors.

9.3 Coding Time (June 2 - August 25)

9.3.1 Core Functions and Metrics(June 2 - June 8)

Develop all essential functions for training, testing, and visualizing models. Establish performance metrics (e.g., model accuracy and AUC) in consultation with mentors. This foundational code will be reused in subsequent phases.

9.3.2 Implement Classical Baselines(June 9 - June 15)

Develop and train classical machine learning models (such as GNNs, CNNs, and SVMs) on a benchmark HEP dataset. Validate these implementations using datasets like the Pythia8 Quark and Gluon Jets. These classical models will serve as baselines for later comparisons.

9.3.3 Develop the QGNN(June 16 - July 6)

Focus on building the QGNN by designing the quantum circuit architecture, including node encoding, quantum message passing, and quantum aggregation. Experiment with different variational ansatzes (such as angle or amplitude encoding) and fine-tune hyperparameters. Use small-scale datasets (e.g., MNIST or a subset of HEP data) for initial testing and iteratively improve the model.

9.3.4 Hybrid Testing and Visualization(July 7 - July 18)

Integrate the QGNN into the full pipeline. Visualize training results, generate outputs, and validate performance metrics. Compare intermediate results against classical benchmarks. Submit an initial progress report for mentor review and Midterm evaluation.

9.3.5 Full Quantum Model Development(July 19 - August 1)

Shift focus to developing a fully quantum model, exploring different architectures. Optimize the loss functions, gradient computation (using parameter shift rules), and network evaluation. Adjust the number of trainable parameters in both quantum and classical models to ensure fair comparisons. Continue regular mentor updates.

9.3.6 Final Model Tuning(August 2 - August 15)

Finalize the fully quantum QGNN model by completing hyperparameter tuning and running comprehensive benchmarks. Generate all required visualizations (loss curves, ROC curves, etc.) and prepare comparative analyses between quantum, hybrid, and classical approaches.

9.3.7 Documentation and Final Report Preparation(August 16 - August 25)

Write detailed documentation and tutorial-like Jupyter Notebooks. Finalize the project report/white paper summarizing methods, experiments, and results. Ensure the repository is complete with proper comments and a README file.

9.4 Students Submit Code and Final Evaluations(August 26 - September 1)

Use this final week as a buffer to implement any mentor feedback, polish the documentation, and ensure all deliverables meet the project requirements. Final evaluations and submission of the complete code and final report will be performed during this phase.

10. About Me

I am Arnav Nigam, a passionate developer and machine learning enthusiast currently pursuing a B.Tech in Computer Science and Engineering at Delhi Technological University.

My projects range from supply chain management (MediNexus - SIH 2024 Winner) to LLM-based chatbot development during my research internship at DRDO. I have published my researh paper in International Springer Conference on LLM and received Best Paper Award. Beyond development, I am a Specialist on Codeforces with 700+ solved problems on LeetCode, demonstrating my strong problem-solving skills.

I have started to study about Quantum Computing and Quantum ML recently but this field really facinates me and I would love to work with the organization and expand my knowledge base and will try my best to contribute to the open-source.

You can find all the tasks on my github under QML Repository