Application of Deep Learning to Jet Charge Discrimination

Meisam Ghasemi Bostanabad* and Mojtaba Mohammadi Najafabadi†
School of Particles and Accelerators, Institute for Research in Fundamental Sciences (IPM) P.O. Box 19395-5531, Tehran, Iran

The Large Hadron Collider (LHC) generates an enormous volume of data every year. A significant challenge in this vast dataset is the analysis of hadronic jets, which are clusters of particles resulting from quantum chromodynamics processes. Determining the charge of the parton that initiates a light-quark jet can be highly valuable for testing components of the Standard Model and identifying potential signals of physics beyond the Standard Model. An overview of a study of using classical and quantum machine learning models to classify jet charges for up and anti-up quark jets is described here. In addition to better efficiency and rejection rate than the traditional methods, Jet charge offers a wide range of potential applications in both measurements and exploratory searches.

INTRODUCTION

The Large Hadron Collider (LHC) is the world's most powerful and complex particle accelerator, designed to probe the fundamental laws of nature [1]. Its primary goals include testing the predictions of the Standard Model of particle physics, such as the existence of the Higgs boson, while also searching for signs of new physics beyond the Standard Model, such as supersymmetry and extra dimensions. However, one of the greatest challenges in analyzing the enormous volumes of data generated by the LHC is the study of hadronic jets — collimated sprays of particles resulting from quantum chromodynamic (QCD) processes. These jets are produced in high-energy collisions and their detailed characterization is essential. Accurate identification of the origin of these jets, whether from quarks or gluons [12], is critical for testing the Standard Model's predictions and for uncovering any potential signals of new, unknown phenomena. Improving our ability to use electric charge to discriminate between jets originating from different particles will enhance our understanding of both well-established theories and potential discoveries at the LHC.

The concept of the jet charge observable, first introduced by Field and Feynman [3], has been explored in experimental studies such as measurement of top quark charge at the LHC [4], and identifying the charge of heavy bosons W'/Z' [5]. Previous studies of jet charge [6][7] involve measuring variants of a momentum-weighted jet charge, typically defined as a summation of the particle tracks within the jet, weighted by their transverse momentum (p_T).

$$\mathcal{Q}_j = \frac{1}{(p_T^{\text{jet}})^\kappa} \sum_{i \in \text{Tr}} Q_i (p_T^i)^\kappa \qquad (1)$$

where the summation runs over all particles tracks recorded for the jet, Q_i and p_T^i represent the charge of the object and the magnitude of its transverse momentum relative to the beam axis, and p_T^{jet} is the total transverse momentum of the jet. The parameter κ is a tuning parameter defined the weighting with values ranging from 0 to 1. When κ approaches 0, the jet charge becomes highly influenced by soft tracks, which are often not detectable. Conversely, as κ increases towards one, the jet charge becomes primarily determined by the charge of the leading track [8, 10]. Fig. 1 presents the distributions of $\mathcal{Q}_j$ for $u, \bar{u}$ jets at two values of κ. Although there is a large overlap between the positive and negative jet charge distributions, $\mathcal{Q}_j$ remains useful for distinguishing the charge of the originating parton.

ANALYSIS VARIABLES

By utilizing various deep learning models such as convolutional or graph neural network, we can capture the complex features of jet charge distributions, which are

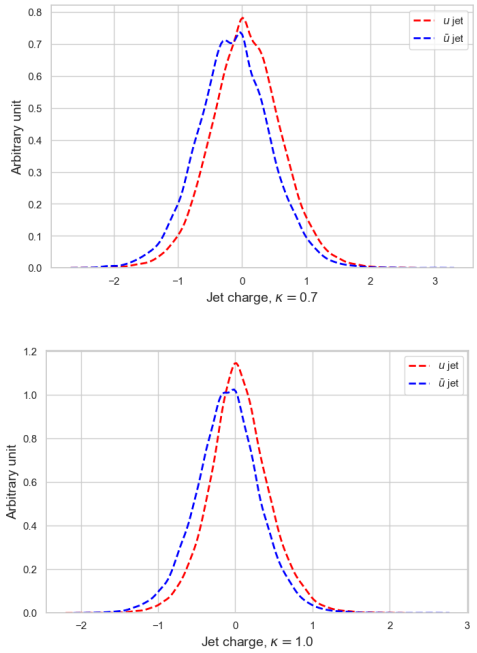

FIG. 1: Distributions of $\mathcal{Q}_j$ for $u, \bar{u}$ jets obtained from $pp \to ug$ or $pp \to \bar{u}g$ events with $\kappa = 0.7, 1$.

often challenging to interpret using traditional methods. This discrimination helps in reducing background noise in measurements, improving the accuracy of particle identification, and providing valuable insights into the behavior of quarks within jets.

The training dataset comprises 35000 simulated up quark and anti-up quark jet events within $pp \to ug$ and $pp \to \bar{u}g$ processes, simulated by MADGRAPH [11]. For this study, jets are clustered using anti-k_t algorithm with radius parameter $R = 0.4$ [12]. The PYTHIA [13] generator is used to collect information for every jet in an event such as p_T, η, ϕ, momentum weighted charge for κ of 0, 0.3, 0.5, 0.7, and 1. In addition to jet variables which are necessary for deep neural network, we provide a list of information for jet's components to train convolutional neural network, including the particle's p_T, η, ϕ, and charge. In addition to the variables for jets and subjets, we also used some features from traditional jet charge calculation searches []:

$$\begin{aligned} Q_{1,\kappa} &= \sum_{i \in Tr} q_i z_i^{\kappa} \\ Q_{2,\kappa} &= \frac{\sum_{i \in Tr} q_i \left|\Delta\eta_i\right|^{\kappa}}{\sum_{i \in Tr} \left|\Delta\eta_i\right|^{\kappa}} \end{aligned} \tag{2}$$

where $z_i = p_{T_i}/p_{T_{jet}}$ is the fraction of the jet momentum, and summation is over every particle in a jet, which provides much more information than just the particles with leading p_T. The last variables used for model training are charge ratio as the ratio of the sum of positive charges to the negative ones, and total jet charge as the sum of track's charge.

RESULT

In this section, we present the performance of different classical and quantum machine learning models in discriminating up-quark from anti-up-quark initiated jets. These models were trained using both high-level jet charge observables and low-level jet image representations constructed from charged particle constituents.

Deep Neural Network

A deep neural network (DNN) [14] is a machine learning model inspired by the human brain, consisting of multiple layers of interconnected artificial neurons. Each neuron in the network performs a computation based on its inputs, weights, and biases. The relationship between the inputs and the output of a neuron can be expressed as $z = \sum_{i=1}^{n} w_i x_i + b$, where w_i are the weights associated with each input x_i, and b is the bias. The output a of a neuron is obtained by applying a non-linear activation function σ, such that $a = \sigma(z)$. Weights and biases are crucial as they determine how the input data is transformed as it propagates through the network. Learning these parameters effectively allows the model to recognize patterns and make accurate predictions.

The training of a DNN involves forward and backward propagation. During forward propagation, the input data is passed through the network layer by layer, and each layer's output becomes the input for the next layer, ultimately generating a prediction. This process can be expressed as $\mathbf{a}^{(l+1)} = \sigma(\mathbf{W}^{(l)}\mathbf{a}^{(l)} + \mathbf{b}^{(l)})$, where $\mathbf{W}^{(l)}$ represents the weights, $\mathbf{b}^{(l)}$ represents the biases of layer l, and $\mathbf{a}^{(l)}$ is the activation. In the backward propagation, the goal is to minimize the error between the predicted output and the actual output using an error function, such as mean squared error or cross-entropy. Gradients of the error function with respect to each parameter are calculated using the chain rule of differentiation. These gradients are then used in an optimization algorithm, such as gradient descent, to iteratively update the weights and biases, following the equation $\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial L}{\partial \mathbf{W}^{(l)}}$, where η is the learning rate, and L is the loss function. This process allows the network to improve its performance through multiple iterations and effectively learn the complex relationships within the data.

DNN used in this study is a fully connected feedforward architecture designed for binary classification of up and anti-up quark jets. The model consists of three hidden layers, each containing 32 neurons with the hyperbolic tangent (tanh) activation function and weights initialized using a normal distribution. To prevent overfitting, dropout layers with a rate of 0.2 follow each dense layer. The final output layer is a single neuron with a sigmoid activation function, suitable for probabilistic binary output. The model is compiled using the stochastic gradient descent (SGD) optimizer and trained with the binary cross-entropy loss function. Prior to training, the input features were standardized using a StandardScaler method, ensuring that each feature had zero mean and unit variance to improve the convergence and stability of the DNN.

Hyperparameter optimization is performed using the RandomizedSearchCV method from the scikit-learn [15] library, wrapped around the Keras [16] model via SciKeras [17]. The search space includes variations in the number of layers, number of neurons per layer, dropout rate, activation functions, kernel initializers, batch size, and optimizer type. The search uses 5-fold cross-validation to evaluate model configurations and identify the one with the highest average accuracy on the training set. This strategy enables efficient tuning over a wide space of architectures and training settings while minimizing overfitting and improving generalization.

Fig. 2 top shows the accuracy of the deep learning model over training epochs, where both training accuracy and validation accuracy improve rapidly in the ini-

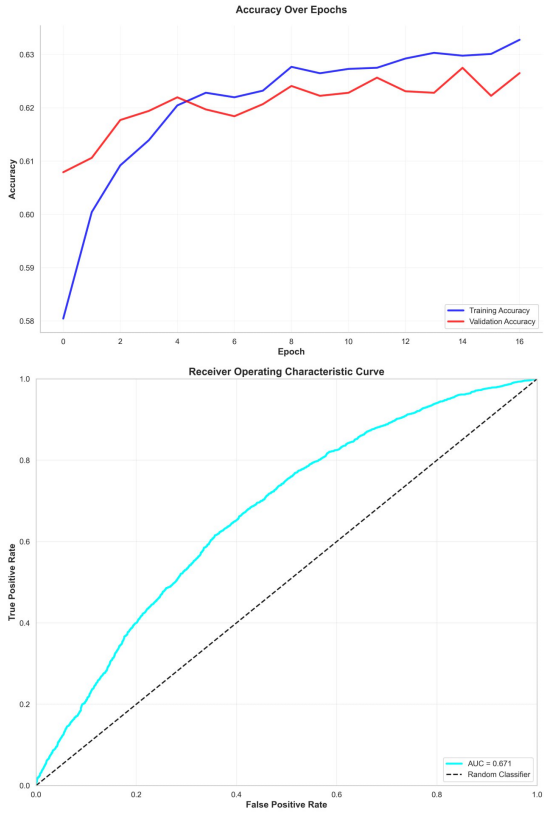

FIG. 2: Top: the accuracy progression over 16 training epochs, showing both training (blue) and validation (red) accuracy. Bottom: the Receiver Operating Characteristic (ROC) curve, with an Area Under the Curve (AUC) score of 0.671, which measures the model's ability to distinguish between the two classes.

tial epochs before converging around 63%. The relatively close alignment between training and validation accuracy suggests that the model generalizes reasonably well without significant overfitting. Fig. 2 bottom presents the receiver operating characteristic (ROC) curve for the model, with an area under the curve (AUC) of 0.671, indicating that the model has moderate discriminatory ability in distinguishing between classes. The model outperforms random classification, represented by the dashed line, highlighting its ability to identify meaningful patterns in the data.

Other Machine Learning Classifiers

The Fig. 3 illustrates the accuracy of various classical machine learning classifiers in discriminating between up and anti-up quark jets using charge-related features. Among the tested algorithms, the Gradient Boosting [18] achieved the highest accuracy, slightly outperforming the Support Vector Classifier (SVC) [19] and Random Forest classifiers [20], which are margin-based and ensemble methods known for their robustness and ability to

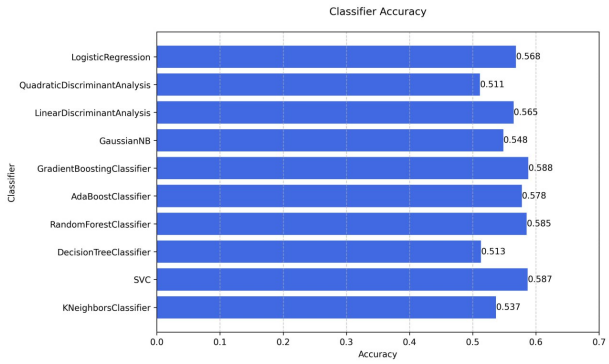

FIG. 3: Accuracy of various ML classifiers for discriminating up and anti-up quark jets.

capture nonlinear patterns. Logistic Regression, Linear Discriminant Analysis (LDA), and AdaBoost [21] also demonstrated competitive performance, suggesting that even relatively simple linear or shallow models can extract meaningful discriminative information from jet charge observables.

In contrast, the K-Nearest Neighbors (KNN) and Decision Tree classifiers performed less effectively, likely due to their sensitivity to high-dimensional feature spaces and potential overfitting to local fluctuations in the training data. The superior performance of margin-based methods (like SVC) and boosted ensembles (Gradient Boosting, AdaBoost) underscores their suitability for handling subtle differences in feature distributions, such as those present between up and anti-up quark jets.

Convolutional Neural Network

A convolutional neural network (CNN) [22] is an advanced type of deep learning model particularly well-suited for tasks involving spatial data, such as image recognition. It consists of several key layers, including convolutional, pooling, and fully connected layers. The convolutional layers apply filters to the input data to learn spatial features, allowing the network to recognize complex patterns by combining simple ones from previous layers. The pooling layers help to reduce the size of the data representation, making the model more computationally efficient and less prone to overfitting. Finally, the fully connected layers use the extracted features for classification. By stacking multiple convolutional layers, a CNN learns to represent increasingly abstract and complex features from the raw input, ultimately making accurate predictions.

In this case, the CNN is used to discriminate between up and anti-up quark jets by analyzing pixel charge information plotted in η (pseudorapidity) and ϕ (azimuthal angle) space. The input to the CNN consists of images like the Fig.4, where each pixel represents a region in

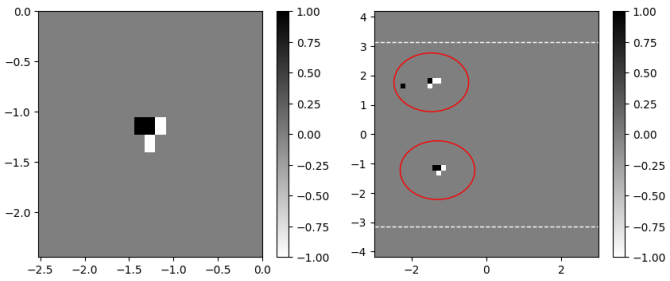

FIG. 4: Left: pixelated representation of a leading jet in the η-ϕ space, showing regions normalized by electric charge, used as input for the CNN model. Right: same representation for all jet's constituents in an event.

the η-ϕ space and is normalized by the electric charge of particles observed in that region. The positive and negative charges are represented by different colors, which the CNN learns to interpret. The convolutional layers extract features from the spatial distribution of the pixel charge, identifying distinctive charge patterns that differentiate up and anti-up quarks. By recognizing these spatial features in the η-ϕ distribution, the CNN is capable of learning the inherent differences between the two quark types and effectively classifying them based on their charge characteristics.

The baseline CNN architecture designed for this study processes jet images formatted as 16×22 charge-weighted (η, ϕ) pixel maps. The model begins with a convolutional layer of 32 filters with a 3×3 kernel and ReLU activation, followed by batch normalization, max pooling, and a 30% dropout layer for regularization. A second convolutional block with 64 filters, batch normalization, and similar pooling and dropout layers is added to deepen the representation. The output is then flattened and passed through a dense layer with 64 neurons, followed by batch normalization and a higher dropout rate of 40%. The final output layer is a single neuron with sigmoid activation, suitable for binary classification. The model is trained using the Adam optimizer with a learning rate of 10^{-4} and binary cross-entropy loss.

To improve performance, an optimized CNN was constructed using the Keras Tuner framework [23] with randomized hyperparameter search. This version dynamically adjusts the number of filters in each convolutional layer (ranging from 32 to 128), dropout rates (between 20% and 60%), dense layer size (64 to 128 units), and the learning rate (sampled logarithmically between (10^{-5} and 10^{-3}). A learning rate scheduler (ReduceLROnPlateau) was also used to reduce the learning rate when validation performance plateaued. The optimized model selected by the tuner achieved improved generalization by exploring a broader architectural and training configuration space, demonstrating the advantage of systematic hyperparameter optimization over fixed-structure models.

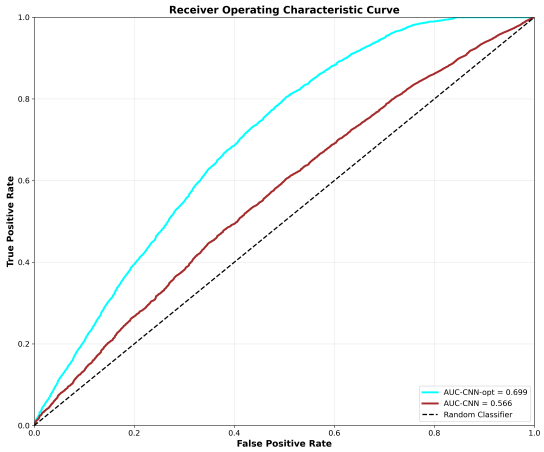

FIG. 5: ROC curves for baseline and optimized CNN models. The optimized model (AUC = 0.699) outperforms the baseline (AUC = 0.566), highlighting the benefit of hyperparameter tuning.

In Fig.5 the ROC curves comparing the baseline and optimized CNN models reveal a clear improvement in performance following the optimization process. The baseline model yields an AUC of 0.566, reflecting limited discrimination power—only slightly above random chance. In contrast, the optimized CNN, obtained through a systematic hyperparameter search using Keras Tuner, achieves a substantially higher AUC of 0.699. This improvement underscores the importance of careful model design and tuning when working with jet image data. By adjusting key architectural parameters such as convolutional filter counts, dense layer sizes, dropout rates, and learning rates, the optimization process enhances the model's ability to generalize and capture subtle patterns that differentiate up from anti-up quark jets.

Graph Neural Network

Graph Neural Networks (GNNs) [24] provide a powerful framework for analyzing data with an underlying graph structure. Unlike traditional neural networks that treat input features as independent, GNNs are designed to capture the relationships and interactions between connected elements. This makes them particularly well-suited for jet classification tasks, where particles within a jet are inherently linked through their spatial and kinematic correlations. By leveraging these connections, GNNs can model the internal structure of jets more effectively than standard architectures.

In our study, the jets are represented as fully connected graphs, where each particle within a jet corresponds to a node, and edges represent the relationships between particles. The node features include physical attributes such as transverse momentum, pseudorapidity, azimuthal angle, and momentum-weighted electric charge. The graph

connectivity is defined using an edge index tensor, which lists all directed connections between node pairs. In our initial setup, we use a fully connected graph, meaning each node is linked to every other node, except itself (Fig.6). While this approach simplifies the implementation, it can be refined by incorporating more physically motivated connections—for example, linking particles based on their spatial proximity or other shared properties relevant to jet structure.

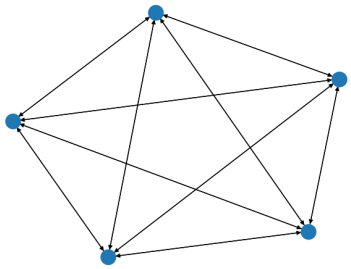

FIG. 6: Visualization of a fully connected directed graph used to represent jet constituents in the GNN model. Each node corresponds to a particle, and edges encode pairwise interactions without self-loops.

In this study, two graph neural network architectures—Graph Convolutional Network (GCN) and GraphSAGE are employed for the task of graph classification, aiming to distinguish between up and anti-up quark jets. Graph Convolutional Networks (GCNs) [25] are a natural extension of traditional convolutional networks to data with an underlying graph structure. Unlike standard neural networks that assume a fixed grid, GCNs are designed to work with data where elements are connected in more complex ways such as the particles within a jet. In a GCN, each node learns by gathering information from its neighboring nodes through a process known as message passing. This allows the model to capture not only the individual properties of particles but also how they relate to one another. With each layer, the network refines these representations by aggregating and transforming the input from surrounding nodes. As a result, GCNs are able to learn meaningful patterns from the graph's structure and are well-suited for tasks like classifying entire jets based on the relational information between their constituents.

The network consists of two graph convolutional layers, each followed by ReLU activation and dropout layers with a dropout rate of 0.5 to mitigate overfitting. A global mean pooling operation is used to aggregate node features into a fixed-size vector representing the entire graph. This vector is then passed through a fully connected layer to produce the final classification output. The model is trained using the Adam optimizer with a learning rate of 0.001 and weight decay of $10^{-}3$, minimizing the cross-entropy loss. Training is performed over multiple epochs with early stopping based on test accuracy, and the model achieving the highest validation performance is saved for evaluation. This setup enables the GCN to learn from the relational structure of jet constituents and effectively distinguish between quark flavors.

GraphSAGE [26] (Graph Sample and Aggregate) is a graph neural network architecture designed to efficiently learn node and graph representations by aggregating information from local neighborhoods using a learnable function—in this case, a simple mean aggregator. Unlike traditional GCNs, which rely on the full adjacency structure, GraphSAGE samples a fixed-size set of neighbors and can generalize to unseen graphs, making it well-suited for jet-level classification tasks. In this analysis, the model architecture includes two SAGEConv layers with ReLU activations, each followed by dropout to mitigate overfitting. Node e mbeddings are aggregated using global mean pooling and passed through a fully connected layer to perform binary classification of up versus anti-up quark jets. The model is trained using the Adam optimizer with a learning rate of 0.001 and weight decay, , using cross-entropy loss and early stopping to select the best-performing configuration. This setup allows the model to capture the structural and relational information within each jet graph while maintaining strong generalization performance.

The ROC curves in the Fig.7 show that both Graph Neural Network models (GCN and GraphSAGE) achieve strong classification performance in distinguishing up from anti-up quark jets. The GCN slightly outperforms GraphSAGE, with an AUC of 0.883 compared to 0.860, indicating that while both models effectively capture relational information within jet graphs, the GCN provides a marginally better overall separation in this task.

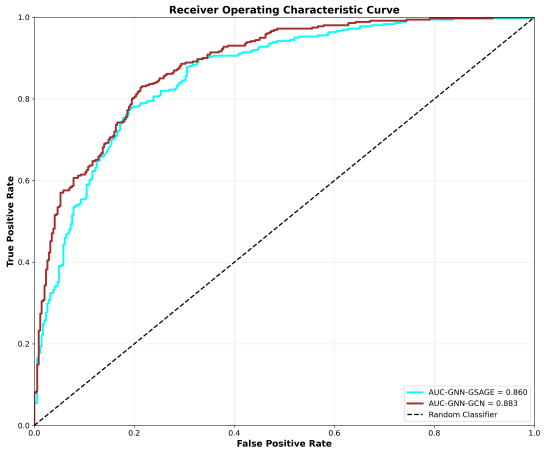

FIG. 7: ROC curves comparing GCN and GraphSAGE performance on jet classification.

Quantum Machne Learning Models

Quantum machine learning (QML) offers a new paradigm by utilizing the principles of quantum mechanics to enhance traditional machine learning methods. Two key concepts in QML are quantum feature maps and quantum kernels, which are essential for quantum support vector machines (QSVM) and variational quantum classifiers (VQC). These two concepts are explained below:

Quantum Feature Map: A quantum feature map encodes classical data into a quantum state. It transforms data into a high-dimensional feature space using the power of quantum computation. This is achieved by applying a series of quantum gates that map input features into a quantum state, typically parameterized by the input data, while leveraging the principles of superposition and entanglement. Superposition allows the quantum system to represent multiple states simultaneously, and entanglement enables the creation of correlations between qubits. In Fig.10 top, the quantum feature map utilizes the Hadamard gates (red) to create superposition states, while entanglement between two qubits is introduced through the use of control-not gates (dark blue). By representing data as quantum states (quantum encoding), the quantum feature map can capture complex relationships that may be difficult for classical algorithms to identify.

Quantum Kernel: The quantum kernel represents the inner product between two quantum states generated by the quantum feature map. It measures the similarity between data points in the high-dimensional feature space created by the quantum feature map. In classical machine learning, kernels are used in methods like SVM to assess the similarity between data points in a transformed feature space. Similarly, a quantum kernel considers quantum states to potentially explore richer feature spaces, facilitating a more effective separation between classes. The formula for the quantum kernel is the squared magnitude of inner product between two quantum states:

$$k(x_i, x_j) = | < \phi(x_i) | \phi(x_j) > |^2 \qquad (3)$$

where $\phi(x)$ is the quantum states produced by encoding the classical input data. In Fig.10 bottom, the quantum feature map employs the ZZFeatureMap, followed by its reverse, creating an encoding and decoding process that captures correlations between qubits, which contributes to building the quantum kernel. The quantum kernel computation is used to train models such as QSVM, providing a new approach to kernel-based learning with quantum advantage.

We applied Principal Component Analysis (PCA) [,] to condense the kinematic variables into 6 components, aligning the number of encoded variables with the 6 available qubits in this study. After performing PCA, the reduced set of variables is passed to quantum machine learning algorithms. For the quantum approach, we use 6 qubits on the ibmq QasmSimulator [] to classify up and anti-up processes. The quantum circuits are optimized to align with hardware constraints, including qubit connectivity, available gate sets, and hardware noise.

Quantum Support Vector Machine (QSVM): The quantum support vector machine is a quantum-enhanced version of the traditional SVM. By employing a quantum feature map, QSVM aims to leverage quantum states for mapping data into higher-dimensional feature spaces, which might improve the separation between classes when compared to classical SVMs. The kernel trick, a central concept in SVMs, is implemented through a quantum kernel that calculates inner products between quantum states. This allows QSVM to efficiently learn complex decision boundaries, potentially providing advantages in problems where classical SVMs may struggle, such as high-dimensional or non-linearly separable data.

Variational Quantum Classifier (VQC): The variational quantum classifier is a hybrid quantum-classical model that uses parameterized quantum circuits (also known as quantum neural networks) for classification tasks. A VQC involves a quantum circuit with tunable parameters that is trained using classical optimization methods. The training process adjusts these parameters to minimize a cost function, such as the binary cross-entropy loss, based on the classification performance. VQC typically consists of a quantum feature map to encode data, followed by a variational circuit that adjusts the parameters to fit the training data. The output is measured through quantum measurements, which are used to make predictions. VQC is particularly suit-

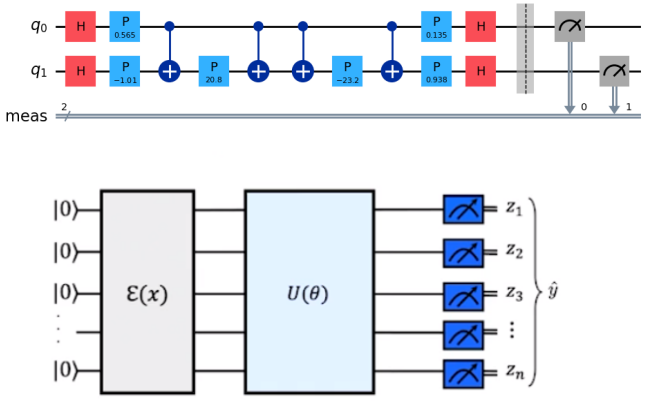

FIG. 8: Top: Quantum feature map using ZZFeatureMap, with superposition from Hadamard gates (red) and entanglement from CNOT gates (dark blue), contributing to quantum kernel construction. Bottom: Variational quantum circuit encoding input data, followed by trainable transformation and measurement for class prediction.

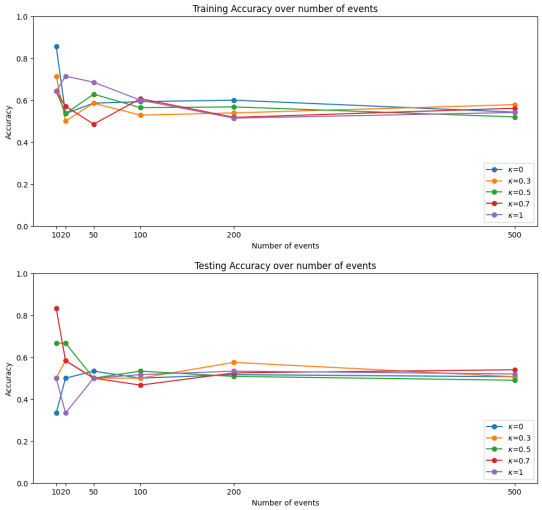

FIG. 9: Accuracy of the Variational Quantum Classifier (VQC) on the training set (top) and testing set (bottom) for different values of the hyperparameter κ. Results are plotted as a function of the number of training events, ranging from 10 to 500. Each curve represents a different κ value, showing how model performance varies with training set size and the impact of κ on accuracy.

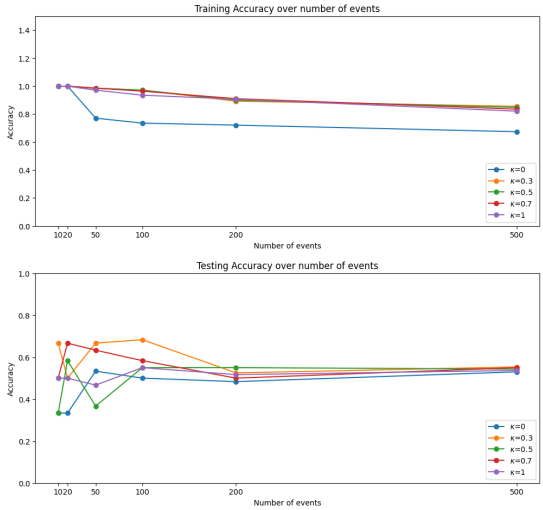

FIG. 10: Accuracy of the Quantum support vector machine (QSVM) on the training set (top) and testing set (bottom) for different values of the hyperparameter κ. Results are plotted as a function of the number of training events, ranging from 10 to 500. Each curve represents a different κ value, showing how model performance varies with training set size and the impact of κ on accuracy.

able for near-term quantum devices (NISQ era) as it allows for flexibility in circuit design and can potentially exploit quantum entanglement and superposition to capture complex patterns in data.

(should be updated) By using both QSVM and VQC, you can harness the unique advantages of quantum computing for classification tasks. The QSVM benefits from quantum-enhanced feature spaces through quantum kernels, while the VQC offers flexibility and adaptability through parameterized circuits, making them promising candidates for problems like jet charge discrimination. The QSVM benefits from quantum-enhanced feature spaces through quantum kernels, while the VQC offers flexibility and adaptability through parameterized circuits, making The QSVM benefits from quantum-enhanced feature spaces through quantum kernels, while the VQC offers flexibility and adaptability through parameterized circuits, makingThe QSVM benefits from quantum-enhanced feature spaces through quantum kernels, while the VQC offers flexibility and adaptability through parameterized circuits, making

To be added. The code is running on IBM real quantum machine.

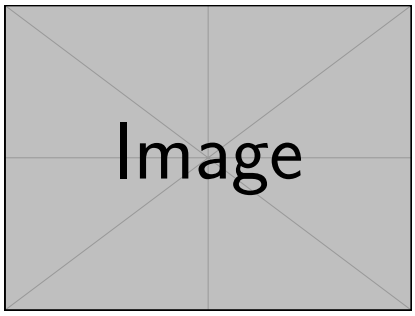

FIG. 11: To be added. Code is running on IBM real quantum machine.

Conclusion

Acknowledge

Data availability

* mghasemi@ipm.ir
† mojtaba@ipm.ir

[1] L. Evans, P. Bryant (2008), IOPscience.10.1088/1748-0221/3/08/S08001.
[12] M. Cacciari, G. P. Salam, G. Soyez (2008), arXiv:0802.1189v2 [hep-ph]NoStop.
[3] R. D. Field, R. P. Feynman (1978), A parametrization of the properties of quark jets, Nucl. Phys. B **136**, 1–76, Nuclear Physics B 10.1016/0550-3213(78)90015-9NoStop.
[4] ATLAS Collaboration (2013), Measurement of the top quark charge in pp collisions at $\sqrt{s} = 7$ TeV with the ATLAS detector, JHEP **11**, 031, arXiv:1307.4568 [hep-ex]NoStop.
[5] D. Krohn, M. D. Schwartz, T. Lin, W. J. Waalewijn (2013), Jet charge at the LHC, Phys. Rev. Lett. **110**, 212001, arXiv:1209.2421 [hep-ph]NoStop.
[6] D. Krohn, M. D. Schwartz, T. Lin, W. J. Waalewijn (2013), Jet charge at the LHC, Phys. Rev. Lett. **110**, 212001, arXiv:1209.2421 [hep-ph]NoStop.
[7] CMS Collaboration (2013), Measurement of jet charge in dijet events at $\sqrt{s} = 8$ TeV, CMS Physics Analysis Summary CMS-PAS-JME-13-002, CDS:cds.cern.ch/record/1599732.
[8] J. Berge *et al.*, Nucl. Phys. **B184**, 13 (1981).
[9] J. Albanese *et al.* (EMC), Phys. Lett. **B144**, 302 (1984).
[10] D. Decamp *et al.* (ALEPH), Phys. Lett. **B259**, 377 (1991)NoStop.
[11] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.-S. Shao, T. Stelzer, P. Torrielli, M. Zaro (2014), The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations, JHEP **07**, 079, arXiv:1405.0301 [hep-ph]NoStop.
[12] M. Cacciari, G. P. Salam, G. Soyez (2008), arXiv:0802.1189v2 [hep-ph]NoStop.
[13] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, P. Z. Skands (2015), An Introduction to PYTHIA 8.2, Comput. Phys. Commun. **191**, 159–177, arXiv:1410.3012 [hep-ph]NoStop.
[14] P. Baldi, P. Sadowski, D. Whiteson (2014), Searching for exotic particles in high-energy physics with deep learning, Nature Communications **5**, 4308, arXiv:1402.4735 [hep-ph]NoStop.
[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay (2011), Scikit-learn: Machine Learning in Python, J. Mach. Learn. Res. **12**, 2825–2830, NoStop.
[16] F. Chollet (2015), Keras: Deep Learning for humans, `https://keras.io`,NoStop.
[17] A. Nicolaou, P. André (2021), SciKeras: A Scikit-Learn API wrapper for Keras, `https://github.com/adriangb/scikeras`,NoStop.
[18] J. H. Friedman (2001), Greedy function approximation: A gradient boosting machine, Annals of Statistics **29**, 1189–1232, NoStop.
[19] C. Cortes, V. Vapnik (1995), Support-vector networks, Machine Learning **20**, 273–297, NoStop.
[20] L. Breiman (2001), Random forests, Machine Learning **45**, 5–32, NoStop.
[21] Y. Freund, R. E. Schapire (1997), A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. **55**, 119–139, NoStop.
[22] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner (1998), Gradient-based learning applied to document recognition, Proc. IEEE **86**, 2278–2324, NoStop.
[23] T. O'Malley, E. Bursztein, J. Long, H. Chollet, C. Jha, L. Invernizzi (2019), Keras Tuner, `https://github.com/keras-team/keras-tuner`,NoStop.
[24] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun (2020), Graph Neural Networks: A Review of Methods and Applications, AI Open **1**, 57–81, NoStop.
[25] T. N. Kipf, M. Welling (2017), Semi-Supervised Classification with Graph Convolutional Networks, ICLR, arXiv:1609.02907 [cs.LG]NoStop.
[26] W. L. Hamilton, R. Ying, J. Leskovec (2017), Inductive Representation Learning on Large Graphs, NeurIPS, arXiv:1706.02216 [cs.SI]

.