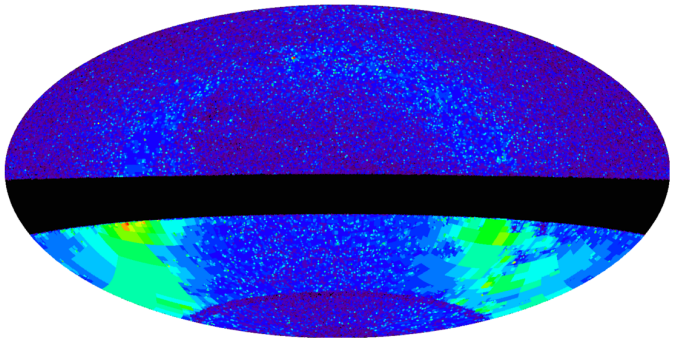
Using Quantum-enhanced Support Vector Machines and Quantum Convolutional Neural Nets for Stellar Classification

Project Date: 23/02/2023 - 28/02/2023

Team Name: Durchmusterung

Abstract:

This research project aims to evaluate the feasibility of using QSVMs and QCNNs for stellar classification based on spectral data. The study will develop and compare these models with traditional methods like KNN and Logistic Regression, and explore the potential of GPU acceleration techniques. Additionally, the project will investigate binary classification between white dwarves and non-white dwarves and multi-class classification using the Harvard star classification system. The goal is to showcase the versatility of quantum-enhanced machine learning for stellar classification and its potential for advancing our understanding of the universe.



Durchmusterung

1 Introduction

Stellar classification is a crucial task in astronomy that involves categorizing stars based on their spectral characteristics. Accurate classification can provide valuable insights into the physical properties and evolutionary processes of stars, aiding our understanding of the universe. While traditional machine learning techniques like SVMs have been effective in classifying stars based on their spectral data [1], the increasing complexity and size of astronomical datasets pose significant challenges to their performance and scalability.

To address these challenges, quantum computing has emerged as a promising approach. In addition to quantum-enhanced SVMs, quantum convolutional neural networks (QCNNs) have also shown promise for stellar classification [**?**]. These quantum-enhanced machine learning techniques offer potential advantages such as improved accuracy and faster computation time for certain classification tasks.

This research project aims to investigate the potential of quantum-enhanced SVMs and QCNNs for stellar classification based on spectral data. By designing and implementing novel quantum-enhanced algorithms and comparing their performance to traditional methods, this study seeks to demonstrate the potential benefits of leveraging quantum computing methodologies in astronomy. The findings of this research can contribute to advancing the state-of-the-art in machine learning techniques for stellar classification and may have broader implications for other applications in astrophysics and beyond.

Types of Stars and Current Methods of Classification

Stellar classification is based on the properties of a star's spectrum, which can reveal information about its temperature, luminosity, and chemical composition [2]. The most widely used classification system is the Harvard spectral classification, which divides stars into seven categories (O, B, A, F, G, K, M) based on the strengths of their spectral lines. This system is based on the work of Annie Jump Cannon, who developed the original classification scheme in the early 20th century [3].

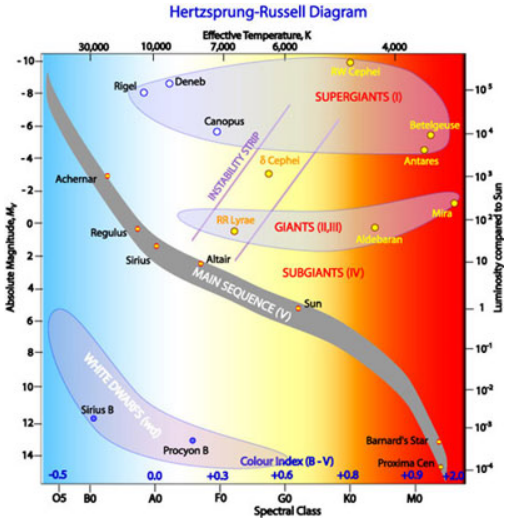


Figure 1: The Hertzsprung-Russell diagram the various stages of stellar evolution.

Each spectral type corresponds to a range of temperatures and luminosities, with O-type stars being the hottest and most luminous, and M-type stars being the coolest and least luminous. Within each spectral type, stars can be further divided based on their luminosity class, which is related to their size and evolutionary stage. The luminosity classes are denoted by Roman numerals I (supergiants), II (bright giants), III (giants), IV (subgiants), and V (main sequence stars) [4]. The Hertzsprung-Russell Diagram (HRD) is a plot of the luminosity of stars versus their surface temperature, and is a fundamental tool for studying stellar evolution [5]. This diagram reveals distinct regions where stars of different masses, sizes, and ages are located, and is essential for understanding the physical properties and behavior of stars. Typically, stars are classified on the HRD based on their spectral types and luminosities, with main sequence stars occupying a diagonal band across the diagram and giants and supergiants occupying regions above and to the right of

the main sequence. An example of an HRD is shown in Figure 1, where the stars are color-coded based on their spectral types, with O stars being the hottest and bluest, and M stars being the coolest and reddest.

The current methods of classifying stars are primarily based on human analysis of their spectra, which can be time-consuming and subject to variability [6]. Machine learning techniques, particularly SVMs, have been shown to be effective in automating the classification process and reducing human error [7]. However, as mentioned earlier, the performance and scalability of traditional SVMs are limited by the complexity and size of astronomical datasets. Hence, there is a need to explore novel approaches such as quantum-enhanced SVMs to improve the accuracy and efficiency of stellar classification [8].

2 Methodology

2.1 Quantum-enhanced Support Vector Machines (QSVM)

Machine learning is a powerful tool for solving complex problems in fields ranging from computer vision to natural language processing. One popular machine learning algorithm is the support vector machine (SVM), which is a powerful technique for classifying data. Recently, quantum computing has been explored as a potential tool for machine learning, and quantum support vector machines (QSVM) have been proposed as a quantum algorithm for binary classification.

In classical SVM, the algorithm takes a set of input data points and their corresponding labels, and finds the optimal hyperplane that separates the data into two classes. [9] The goal of the SVM algorithm is to find the hyperplane that maximizes the margin between the two classes. This hyperplane can be used to classify new data points based on which side of the hyperplane they lie, shown in Fig. 2. [10]

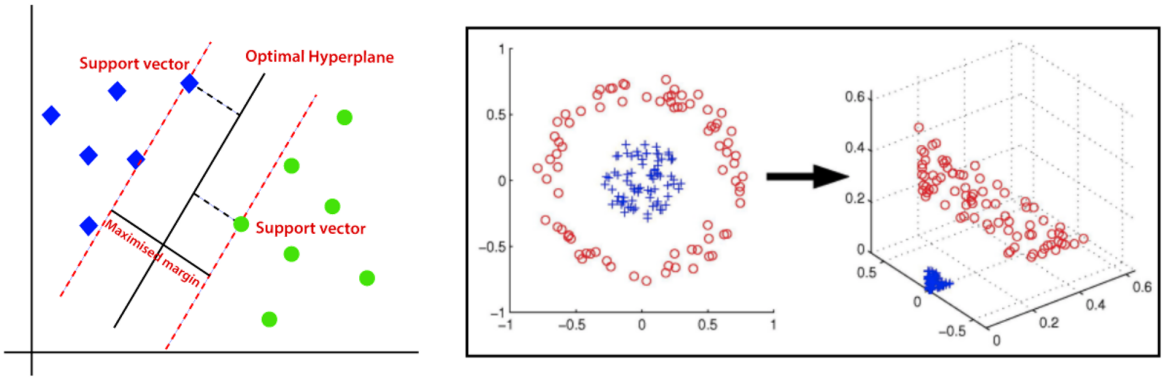


Figure 2: The concept of support vector machine [10]

QSVM is a quantum version of the classical support vector machine (SVM) algorithm, which is used for binary classification tasks [8]. In SVM, a hyperplane is used to separate the data points into two classes. The goal of SVM is to find the hyperplane that maximizes the margin between the two classes. The hyperplane is defined by a set of parameters that are learned during the training process. In QSVM, the classification is based on a quantum kernel function, which maps the input data points to a high-dimensional quantum state. The quantum state is then used to compute the inner product between two input data points. The inner product is then used to calculate the probability that the two data points belong to the same class. This probability is then used to classify the new data points.

The quantum kernel function used in QSVM is typically a variation of the classical kernel function used in SVM. One example is the quantum Gaussian kernel, which is defined as follows: [11]

$$K(x_i, x_j) = exp[\frac{-||x_i - x_j||^2}{2\sigma^2}] \tag{1}$$

where x_i and x_j are the input data points, and sigma is a hyperparameter that controls the width of the kernel. The Gaussian kernel is commonly used in classical SVM, and the quantum version is computed using a quantum circuit.

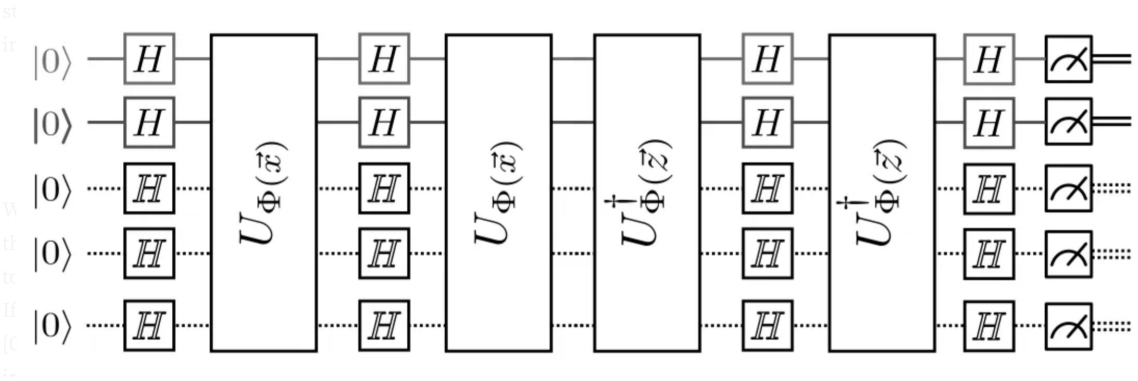


Figure 3: Quantum circuit os QSMV. The estimated overlap between two quantum states for a circuit depth of S=2. [11]

One of the main advantages of QSVM over classical SVM is its ability to take advantage of quantum parallelism. In classical SVM, the kernel function is typically computationally expensive, and the SVM algorithm can be slow for large datasets. QSVM can use quantum parallelism to simultaneously evaluate the quantum kernel function for multiple data points, which can speed up the computation significantly. For example, if we have n data points, we can evaluate the kernel function for all n^2 pairs of data points simultaneously using a quantum circuit. This can be done in O(log n) time on a quantum computer, compared to $O(n^2)$ time on a classical computer.

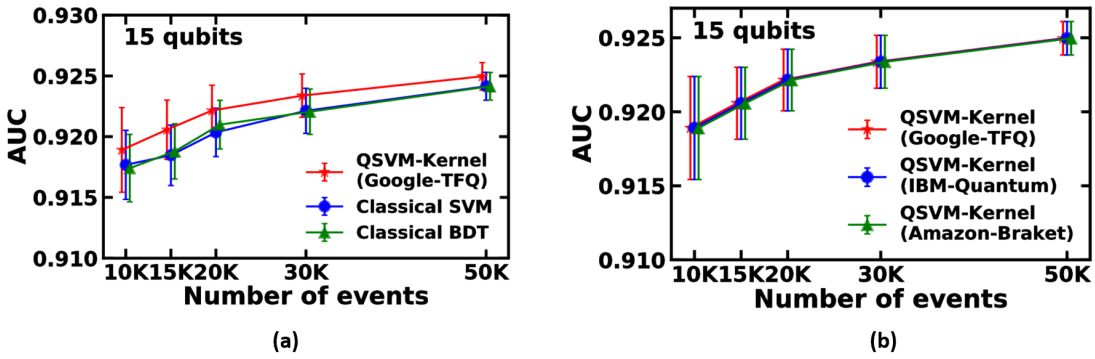


Figure 4: Comparison of classifiers for the *ttH* analysis dataset, showing the AUC as a function of dataset size. The results indicate that QSVM-Kernel outperforms classical SVM and BDT, and performs well on multiple quantum simulators [12]

In Fig. 4, the CERN's paper [12] compared the performance of various classifiers, including QSVM-Kernel, classical SVM, and classical BDT, for the *ttH* analysis dataset, using the AUC as a metric. The results showed that QSVM-Kernel outperformed classical SVM and BDT, especially for larger datasets. In addition, the study evaluated the performance of the QSVM-Kernel on multiple quantum simulators, demonstrating its versatility and potential for use in solving problems in high-energy physics and astrophysics. [12] Given its high accuracy and ability to perform well on large and complex datasets, QSVM-Kernel may be a valuable tool for researchers in these fields.

In addition to quantum parallelism, Q-SVM can also exploit the power of quantum entanglement to enhance the classification performance. Entanglement is a fundamental property of quantum systems, which allows two or more particles to be correlated in a way that is not possible in classical systems. By exploiting entanglement, QSVM can potentially achieve better classification performance than classical. Q-SVM can take advantage of quantum parallelism and quantum entanglement to enhance the classification performance. These quantum advantages can potentially speed up the computation and improve the classification accuracy, especially for large and complex datasets - like the case we want to solve in this work.

2.2 Star Dataset for QSVM Classification

The Star Categorization - Giants and Dwarfs dataset on Kaggle is a collection of stellar parameters for a sample of stars in our galaxy. [13] The dataset includes over 100,000 stars and provides information on the stars' luminosity, temperature, radius, and mass.

The Star Categorization - Giants and Dwarfs dataset provides information on over 100,000 stars in our galaxy and is categorized into two groups: giants and dwarfs. Giants are stars that are more luminous and larger than the sun, while dwarfs are stars that are less luminous and smaller than the sun. This categorization is based on the stars' luminosity and radius, which are estimated from their spectral data. The dataset was compiled by researchers at the Vines Center for Astronomy at Michigan State University and was originally used to study the relationship between stellar parameters and the occurrence of exoplanets. The dataset is publicly available on Kaggle and can be used for various research purposes, such as studying stellar evolution, exoplanet detection, and machine learning applications in astronomy. By analyzing the dataset, researchers can gain insights into the properties and behavior of stars, and develop machine learning models for classifying stars based on their stellar parameters. These models can be used to identify new exoplanets and to study the evolution of stars over time.

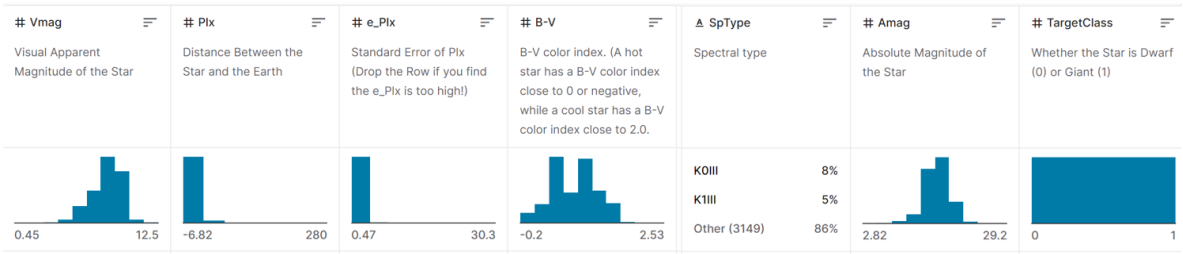


Figure 5: The statistics of star dataset

In this work, we will use this dataset to explore various questions about the properties and behavior of stars, and to develop machine learning models for classifying stars as giants or dwarfs based on their stellar parameters. The dataset can also be used as a benchmark for testing the performance of quantum machine learning algorithms in astronomy and astrophysics.

2.2.1 Data Pre-Processing for Astrophysical Data

Data preprocessing is an essential step in machine learning, as it helps to prepare the data for analysis and improve the performance of the machine learning models. In the case of the Star Categorization - Giants and Dwarfs dataset, the data preprocessing involves several steps to clean and transform the data.

The first step in data preprocessing is to remove any duplicate entries or missing values in the dataset. This ensures that the dataset is complete and consistent, and that there are no errors or inconsistencies in the data.

Next, the data is standardized to remove any bias or scale differences that may be present in the dataset. Standardization is a crucial step in data preprocessing that involves scaling the features of a dataset to remove any bias or scale differences that may be present. Standardization is performed by subtracting the mean of each feature from the data and dividing it by the standard deviation. This results in a feature that has a mean of 0 and a standard deviation of 1, which makes it easier to compare different features with each other. Standardization ensures that all features have equal importance in the analysis, and that the machine learning models are not influenced by differences in the scale of the features. This step is particularly important in machine learning, where the scale of the features can significantly impact the accuracy and performance of the models. Without standardization, the machine learning models may be biased towards certain features and may not perform well on new data. By standardizing the data, the machine learning models can more accurately identify patterns and relationships between the features, leading to better performance and more reliable predictions.

After standardization, the data is split into training and testing sets to evaluate the performance of the machine learning models. The training set is used to train the models, while the testing set is used to evaluate the accuracy and generalization of the models.

Finally, the data is encoded to prepare it for machine learning models. This involves transforming the categorical data into numerical data, which can be used in machine learning algorithms. One common technique for encoding categorical data is one-hot encoding, which creates binary columns for each category in the dataset.

	Vmag	Plx	e_Plx	B-V	SpType	Amag	TargetClass
0	10.00	31.660000	6.19	1.213	K7V	22.502556	1
1	8.26	3.210000	1.00	1.130	K0III	15.792525	0
2	8.27	12.750000	1.06	0.596	F9V	18.797552	1
3	6.54	5.230000	0.76	1.189	K1III	15.132508	0
4	8.52	0.960000	0.72	0.173	B8V	13.431356	1
...	...	...	...	...	...	...	...
39547	5.83	0.170000	0.52	0.474	B7Iab	6.982244	0
39548	7.05	18.120001	0.92	0.424	F5V	18.340790	1
39549	9.21	3.890000	1.46	0.227	A1IV	17.159748	1
39550	9.01	2.130000	1.46	1.467	M5III	15.651898	0
39551	9.12	3.820000	0.79	0.480	F5V	17.030317	1

39552 rows × 7 columns

Figure 6: Dataset after pre-processing remains 39552 data points with 6 parameters and 1 label.

2.3 Compute Resource Estimation

2.3.1 Amazon Web Services

Amazon Web Services (AWS) is a subsidiary of Amazon that provides on-demand cloud computing platforms and APIs to businesses, individuals, and governments [14]. AWS offers a wide range of services, including computing power, storage, and databases, that enable businesses to scale and grow their operations without having to invest in expensive hardware and infrastructure. Among the services that AWS provides, Amazon Bracket is a fully-managed service that provides access to quantum computing technology from multiple providers in a single place. With Amazon Bracket, you can design your own quantum algorithms, test and troubleshoot them on simulated quantum computers running on Amazon EC2, and run them on your choice of different quantum hardware providers.

2.3.2 NVidia - cuQuantum

The NVidia cuQuantum SDK is a toolkit that allows researchers to simulate quantum circuits using GPUs [15]. GPUs are specialized hardware that are designed to perform parallel computations efficiently. This makes them well-suited for accelerating the computation-intensive tasks involved in quantum computing, such as simulating quantum circuits.

By using the cuQuantum SDK on a GPU, you can take advantage of its parallel processing capabilities to accelerate our quantum computing calculations. This can reduce the time required to simulate quantum circuits and optimize quantum algorithms with a very huge dataset - like our star dataset, which can be very time-consuming on classical computers. [16]

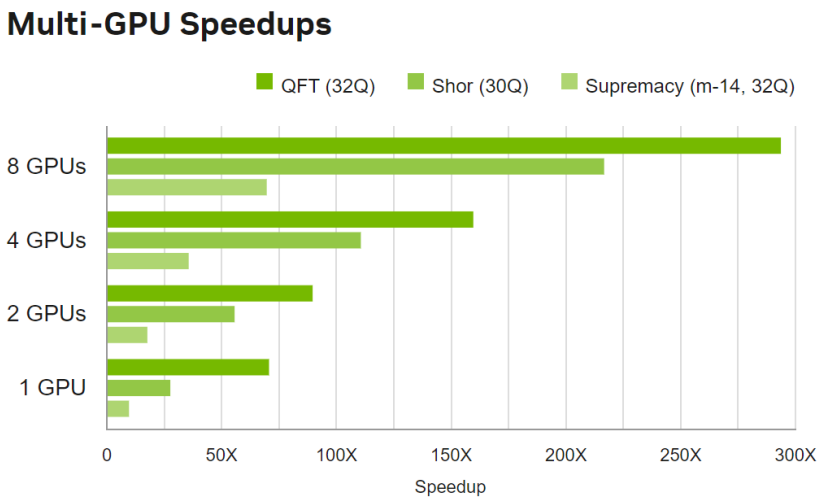


Figure 7: GPU speedup of cuQuantum [15]

Using a GPU with the NVidia cuQuantum SDK can not only result in faster computation times but also enable researchers to simulate larger and more complex quantum circuits. GPUs have significant amounts of memory and processing power, which can be utilized to simulate circuits with thousands or even millions of qubits. This capability can enable researchers to explore new quantum computing algorithms and applications that were previously infeasible to simulate on classical computers. In our work, we anticipate that training with over 30,000 samples will be time-consuming. By utilizing cuQuantum, we can efficiently achieve our results within one week and quantitatively benchmark the performance between CPU and GPU. This work will undoubtedly be beneficial to researchers in the fields of high-energy physics and astrophysics.

3 Plan Directions for QHACK2023

The purpose of this research endeavor is to assess the practicality of utilizing quantum support vector machines (QSVMs) and quantum convolutional neural networks (QCNNs) for categorizing stars based on their spectral data. This study will involve creating and contrasting these models with conventional methods such as k-nearest neighbors (KNN) and logistic regression. In addition, it will delve into the possibilities of GPU acceleration techniques. Furthermore, the research will look into the binary classification of white dwarf and non-white dwarf stars and multi-class classification using the Harvard star classification system. The overarching goal is to highlight the versatility of quantum-enhanced machine learning in the realm of stellar classification, and its potential to advance our comprehension of the universe.

White dwarf stars are of great significance to astronomy as they are the remnants of stars that have exhausted all of their fuel and undergone a supernova explosion. They are extremely dense and have a high surface temperature, making them ideal objects for studying the properties of matter under extreme conditions. The study of white dwarfs has contributed significantly to our understanding of stellar evolution and the history of the universe.

In this research project, the binary classification between white dwarfs and non-white dwarfs is of particular interest. White dwarfs are differentiated from other stars due to their unique characteristics, such as their small size, high density, and low luminosity. The classification of these stars is critical for accurately identifying and studying their properties, which can provide valuable insights into the late stages of stellar evolution.

The Harvard star classification system, on the other hand, is a widely used method for classifying stars based on their spectral characteristics. It was developed in the early 20th century at the Harvard College Observatory and involves categorizing stars based on the strength of their hydrogen lines in their spectra. The system has since been expanded to include other elements and has become the standard method for classifying stars.

Multi-class classification using the Harvard system is also a focus of this research project. By categorizing stars using this system, astronomers can gain a deeper understanding of the different types of stars and their properties, such as temperature, luminosity, and mass. This can aid in the study of stellar populations and the evolution of galaxies.

4 Model

We propose to use quantum kernel learning and quantum convolutional neural networks (QCNN) to solve the star classification problem. The quantum kernel learning approach will allow us to efficiently and accurately classify stars as white dwarfs or giants. The QCNN approach will enable us to learn and extract important features from the data in a quantum-enhanced way. Both of them lead to better accuracy and performance.

4.1 Feature Engineering for Star-Classification Dataset

In the context of the star classification problem, quantum kernel learning can be a promising approach for accurately predicting whether a star is a white dwarf or a giant. However, the unique nature of quantum data and the limited availability of quantum devices make feature engineering even more crucial for this problem. To prepare the data for quantum kernel learning, classical pre-processing techniques can be used to reduce the dimensionality of the data or to extract relevant features before training a quantum kernel. This can help to reduce the computational burden and improve the accuracy of the model by reducing noise and uncertainty associated with the quantum data.

One way to prepare the data for quantum kernel learning is to transform the quantum state into a different representation that is more amenable to machine learning techniques. This can involve applying quantum circuits or other transformations to the data to extract useful features. For example, the B-V color index attribute can be transformed into a quantum state that represents the temperature of the star, which is a key factor in determining its classification. Similarly, the spectral type attribute can be transformed into a quantum state that represents the appearance of spectral lines in the star's spectrum, which is another key factor in determining its classification.

Once the data is prepared, the choice of features can have a significant impact on the accuracy of the quantum kernel. This is because the kernel measures the similarity between pairs of data points, and the choice of features determines what aspects of the data are being compared. Therefore, careful feature engineering is necessary to ensure that the kernel is trained on the most informative features of the data. For example, features such as visual apparent magnitude and absolute magnitude can be important for accurately predicting the type of star, while attributes such as e_Plx may not be as informative and can be dropped. Overall, feature engineering plays a crucial role in the success of quantum kernel learning for the star classification problem.

In the star classification problem, various attributes can be used to distinguish between white dwarf and giant stars. These attributes include the visual apparent magnitude (`Vmag`), distance between the star and Earth (`Plx`), B-V color index (`B-V`), spectral type (`SpType`), and absolute magnitude (`Amag`). Feature engineering is crucial in transforming the data into a different representation that is more suitable for machine learning techniques, particularly quantum kernel learning or quantum convolutional neural networks. The accurate classification of white dwarfs and giants has important implications for a range of astronomical studies, including understanding the evolution and properties of stars, studying the structure and composition of the universe, and identifying potential sources of gravitational waves.

- `Vmag`: The visual apparent magnitude of a star, represented by the attribute Vmag, is a crucial factor in the classification of white dwarf and giant stars. Vmag is a reverse logarithmic scale, with a lower magnitude number indicating a brighter star. Since white dwarfs have a lower luminosity than giants, they are less bright and have a higher Vmag number. Hence, Vmag can be used as an important feature to distinguish between white dwarfs and giants in the star classification problem. However, Vmag alone may not be sufficient for accurate classification, and other attributes such as spectral type, B-V color index, and absolute magnitude may need to be considered to improve the accuracy of the classification model.

- `Plx`: The `Plx` attribute, which represents the distance between the star and Earth, can provide valuable information for classifying white dwarf and giant stars. By estimating the star's absolute magnitude from the `Plx` attribute, which is the apparent magnitude a star would have if it were located at a distance of 10 parsecs from Earth, we can distinguish between the two types of stars since they have different absolute magnitudes. However, the accuracy of the `Plx` attribute can be affected by measurement errors and uncertainties, which can introduce noise into the data. Therefore, it is crucial to carefully consider the quality of `Plx` measurements and the methods used to estimate absolute magnitude before using it as a feature for star classification.

- `e_Plx`: The `e_Plx` attribute, which represents the standard error of the `Plx` attribute that measures the distance between the star and Earth, reflects the level of uncertainty or noise in the `Plx` measurement. High levels of uncertainty or noise can affect the accuracy of the classification model in the star classification problem, especially if the feature is considered in isolation. For `e_Plx` that is too high, it may be necessary to drop the entire row from the dataset as the noise could lead to inaccurate classification. However, `e_Plx` is likely to be a trivial parameter for stellar classification, primarily reflecting the quality of the measurement rather than the intrinsic properties of the star. Thus, while it is important to consider the quality of the `Plx` measurement, it may not significantly impact the classification of white dwarf and giant stars when used as a feature in isolation. Nonetheless, it should still be considered in combination with other attributes such as absolute magnitude and B-V color index to improve the accuracy of the classification model.

- `B-V`: The `B-V` attribute, representing the B-V color index, is an important factor in classifying white dwarf and giant stars as it is related to the star's temperature. The B-V color index indicates the color of the star, which in turn indicates its temperature, with hot stars having a lower B-V color index, while cool stars have a higher B-V color index. Thus, for the star classification problem, the `B-V` attribute can be used as an indicator of the star's temperature, size, and luminosity. Since white dwarfs are hotter than giants, they have a lower B-V color index, making it a useful feature for distinguishing between the two types of stars. However, it is important to note that the `B-V` attribute alone may not be sufficient for accurate classification and may need to be combined with other attributes such as absolute magnitude, spectral type, and `Plx` to improve the accuracy of the classification model.

- `SpType`: The `SpType` attribute is an important feature in classifying stars, as it provides information about the star's physical characteristics, including its temperature, size, and chemical composition. Spectral type classification is based on the appearance of spectral lines in the star's spectrum, which correspond to specific atomic transitions. The spectral type classification system is denoted by a letter ranging from O, B, A, F, G, K, and M, where O stars are the hottest and most massive, while M stars are the coolest and smallest. The spectral lines in a star's spectrum provide information about its temperature and chemical composition, allowing astronomers to determine its spectral type.

In the star classification problem, the SpType attribute can be used to distinguish between white dwarf and giant stars. White dwarfs have a different spectral type than giants, with white dwarfs being classified as DA, DB, DC, DO, or DZ, while giants are classified based on a range of spectral types. However, it is important to note that the SpType attribute alone may not be sufficient for accurate classification, and it may need to be combined with other attributes such as absolute magnitude, Plx, and B-V color index to improve the accuracy of the classification model. By using multiple features together, it is possible to develop a more accurate and reliable classification model for white dwarf and giant stars.

- Amag: The Amag attribute represents the absolute magnitude of the star, which is defined as the apparent magnitude that the object would have if it were viewed from a distance of exactly 10 parsecs (32.6 light-years), without extinction or dimming of its light due to absorption by interstellar matter and cosmic dust. The absolute magnitude is an important parameter in the classification of stars, as it provides information about the star's intrinsic brightness and size.

  For the star classification problem, the Amag attribute can be a useful feature for distinguishing between white dwarf and giant stars. Typically, white dwarfs have a lower absolute magnitude than giants, which means they are less bright and have a higher Amag number. Therefore, the Amag attribute can be used to identify white dwarfs, which can be further classified based on their spectral type, Plx, and B-V color index.

  It is important to note that the Amag attribute alone may not be sufficient for accurate classification and may need to be combined with other attributes to improve the accuracy of the classification model. Nonetheless, the Amag attribute is a valuable feature in the star classification problem, as it provides important information about the intrinsic properties of the star and can aid in the identification and classification of white dwarf and giant stars.

- TargetClass: The TargetClass attribute represents the target variable in the star classification problem, which is to determine whether the star is a white dwarf (0) or a giant (1). This attribute is the key feature used in the machine learning model to train and predict the classification of stars. The accurate classification of white dwarfs and giants is important for understanding the evolution and properties of stars, as well as for studying the structure and composition of the universe.

Dwarf and giant stars exhibit differences in size, brightness, lifespan, and temperature. Typically, dwarf stars are smaller, cooler, longer-lived, and less bright compared to giant stars. The star's absolute magnitude, which is related to its luminosity, can be determined by the Plx attribute representing the distance between the star and Earth. On the other hand, the visual apparent magnitude of the star, represented by the Vmag attribute, is related to its brightness. These attributes are related through distance and a specific equation:

$$\texttt{Amag} - \texttt{Vmag} = 5 \log_{10} \texttt{Plx} + 5 \tag{2}$$

To estimate the star's temperature, one can use the B-V attribute, which is related to its size and luminosity. The star's spectral type, represented by the SpType attribute, is determined based on the appearance of spectral lines corresponding to specific atomic transitions in the star's spectrum. Finally, the TargetClass attribute specifies whether the star is a dwarf (0) or a giant (1).

Our analysis suggests that the Vmag attribute is a trivial parameter in differentiating between white dwarf and giant stars. Therefore, the most significant attributes in this dataset are the Amag and B-V attributes. To increase the variation in the dataset and potentially improve the classification model's accuracy, we defined four composite parameters in addition to these attributes.

$$\begin{aligned} \texttt{Amag\_SQ} &= \texttt{Amag}^2 \\ \texttt{B-V\_SQ} &= \texttt{B-V}^2 \\ \texttt{B-V+Amag} &= \texttt{B-V} + \texttt{Amag} \\ \texttt{B-V-Amag} &= \texttt{B-V} - \texttt{Amag} \end{aligned} \tag{3}$$

4.2 The Relation among Features

The classification of a target is often a complex task, and one that requires careful consideration of multiple features. In order to better understand this relationship, a detailed analysis of the interplay between the target classification and other features is necessary. This is where the figure comes in, providing a visual representation of the relationship between the target classification and other features.

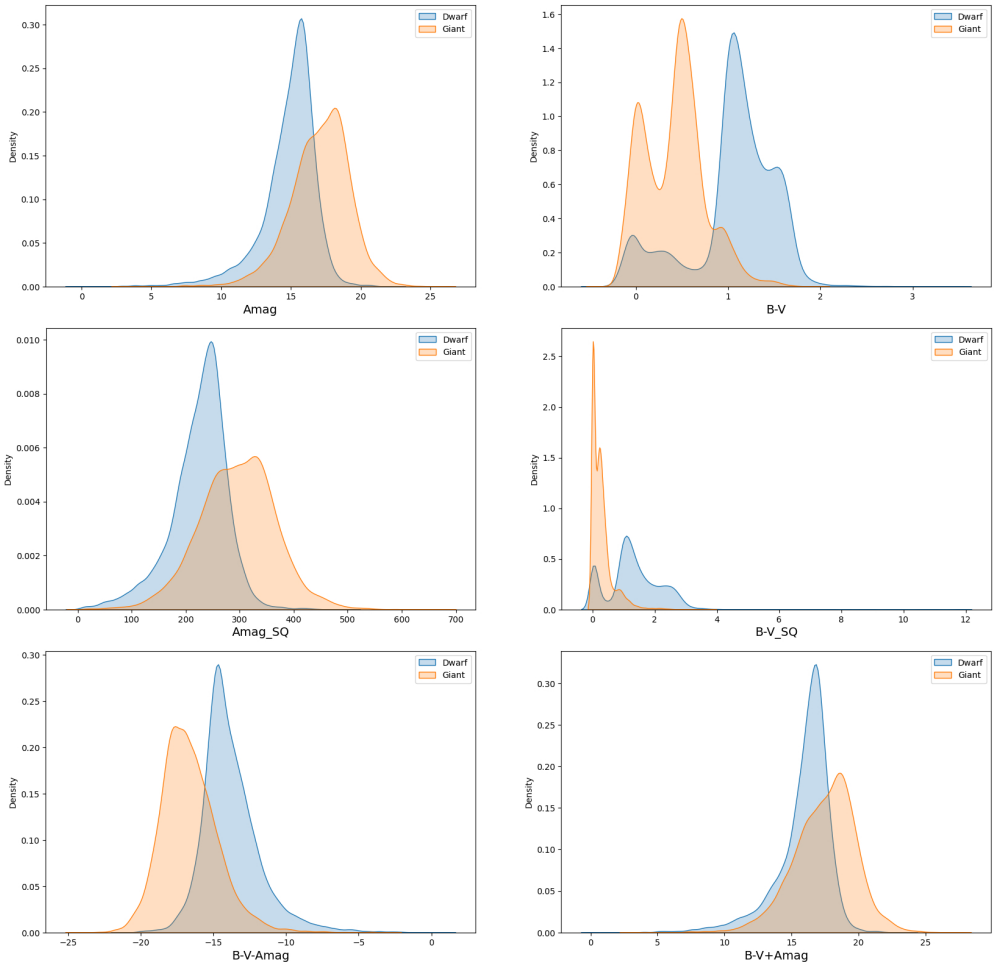


Figure 8: The realtion between stellar classification (Dwarf or Giant) and the absolute magnitude of star (top-left), B-V color index (top-right), the squared value of absolute magnitude (middle-left, the squared value of B-V color index (middle right), the difference of B-V color index and absolute magnitude (bottom-left) and the sum (bottom-right).

Upon closer examination of the figure, it becomes clear that composite features play a critical role in this relationship. Specifically, the composite features are able to successfully amplify even the smallest variations in each feature, leading to further separation of the peaks in the data. This is an important observation, as it suggests that the use of composite features can be a powerful tool for improving the accuracy of target classification.

Furthermore, the success of composite features in separating the peaks suggests that these features may be particularly useful in cases where the target classification is particularly challenging. By identifying and leveraging the underlying patterns in the data, composite features can help to overcome some of the inherent difficulties in the target classification task.

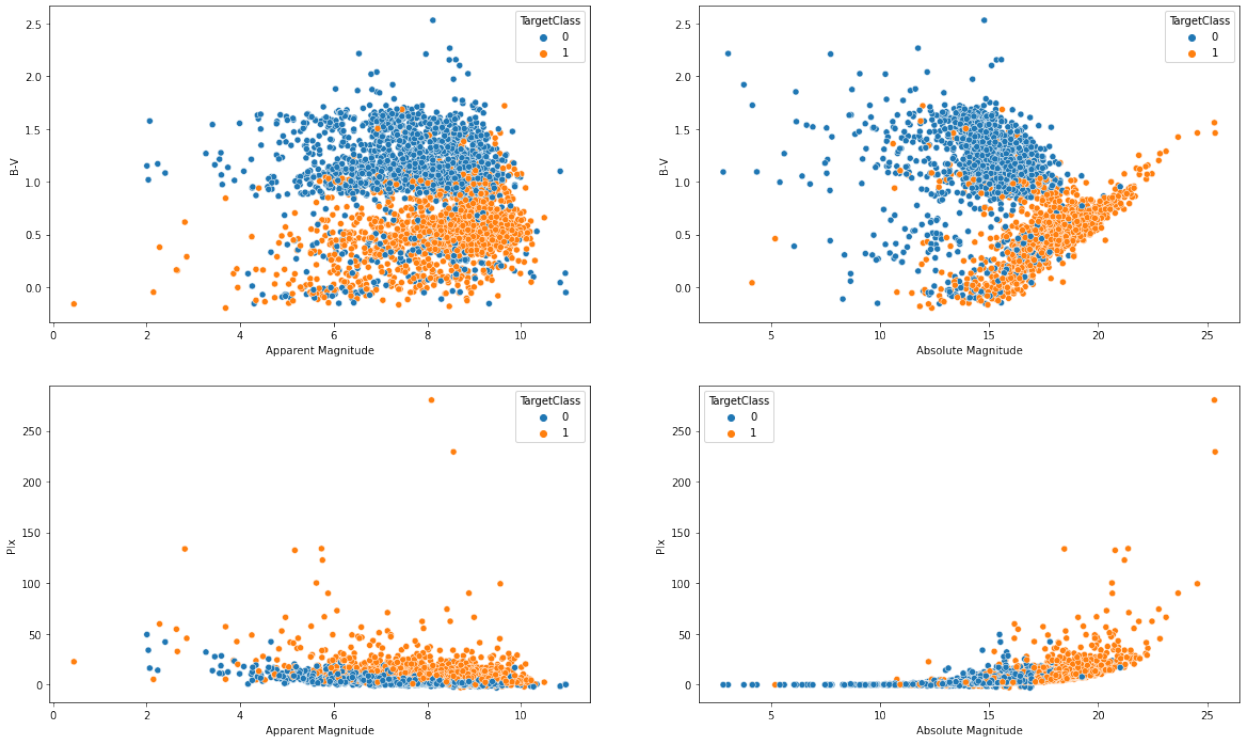


Figure 9: Interaction graph for different features on two-labelled classification

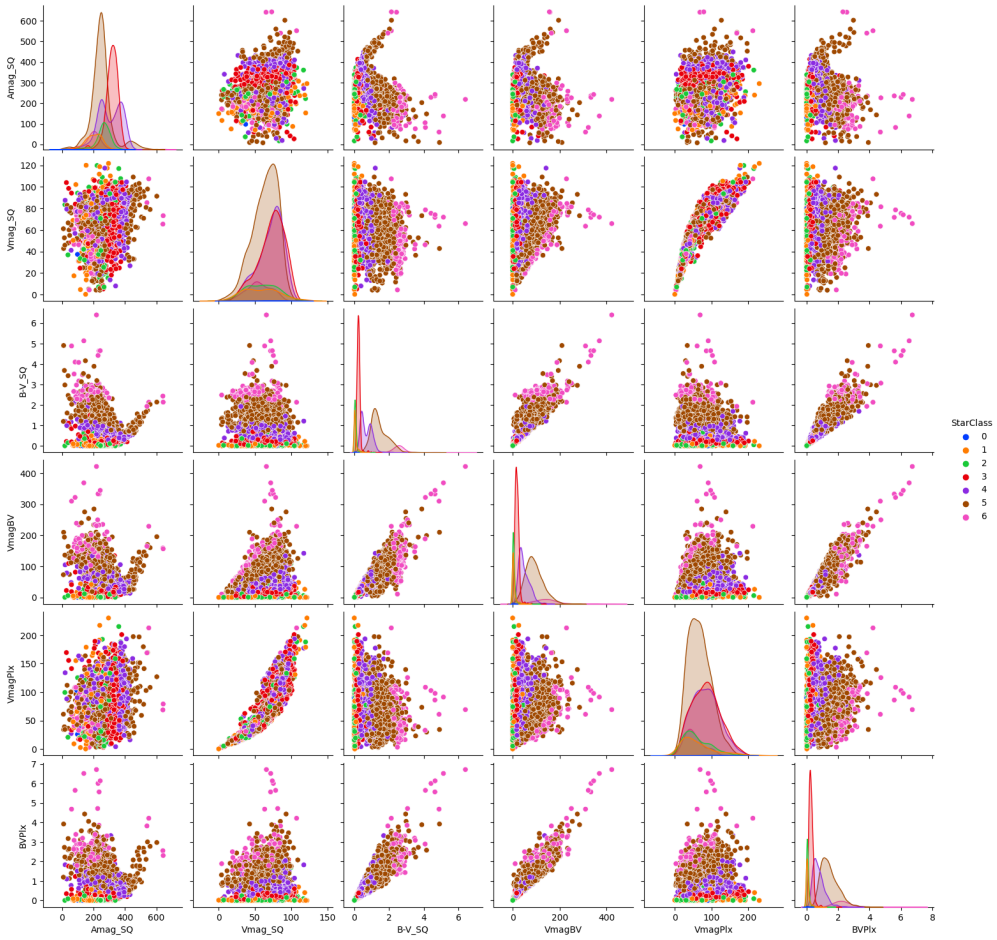


Figure 10: Interaction graph for different features

4.3 Quantum Kernel Architecture and the Process Flow

Quantum kernel learning (QKL) is a quantum machine learning algorithm that aims to improve the accuracy of classical machine learning algorithms by utilizing the power of quantum computing. The QKL algorithm consists of two main components: a quantum feature map and a classical kernel algorithm. The quantum feature map is implemented using a variational quantum circuit (VQC), which maps the input data points from the original data space X into a high-dimensional quantum feature space. The VQC consists of a series of quantum gates that are parameterized by a set of adjustable parameters. These parameters are optimized using a classical optimization algorithm to minimize the difference between the quantum feature space and the desired feature space.

The ZZFeatureMap is a type of quantum feature map used in quantum kernel learning that maps each input data point to a corresponding quantum state, where the amplitudes of the quantum state represent the features of the data point. The ZZFeatureMap uses two-qubit gates, specifically the ZZ gate, to generate entanglement between the qubits and encode the pairwise interactions between the features of the data points. This allows the quantum kernel to capture higher-order correlations between the features, which can improve classification accuracy. The classical kernel algorithm computes the kernel function as the inner product of the feature vectors in the high-dimensional feature space. The resulting kernel matrix is then used as input to a classical machine learning algorithm, such as support vector machines (SVM) or k-nearest neighbors (KNN), to perform classification or regression.

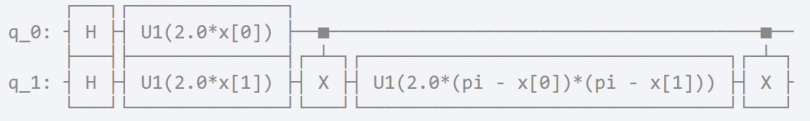


Figure 11: Quantum Circuit of ZZFeatureMap

The QKL algorithm has the potential to improve the accuracy of classical machine learning algorithms by mapping the input data to a higher-dimensional feature space that is not achievable by classical methods. This can be particularly useful for complex datasets that cannot be effectively modeled by classical algorithms alone. Additionally, the QKL algorithm can be accelerated using specialized hardware, such as quantum processing units (QPUs) or GPU (cuQuantum), to further improve performance.

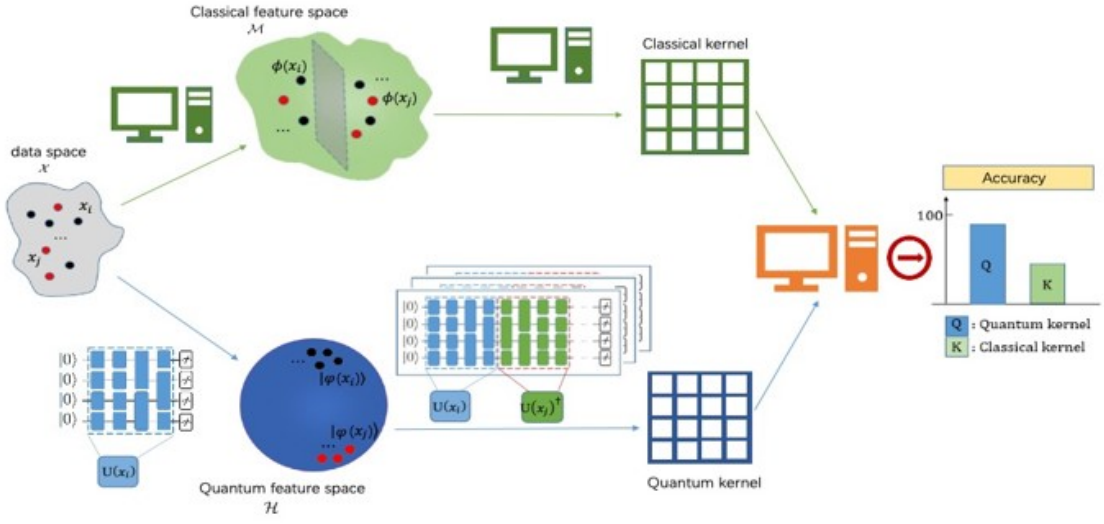


Figure 12: Quantum Kernel Architecture and the Process Flow

4.4 QCNN Model Architecture

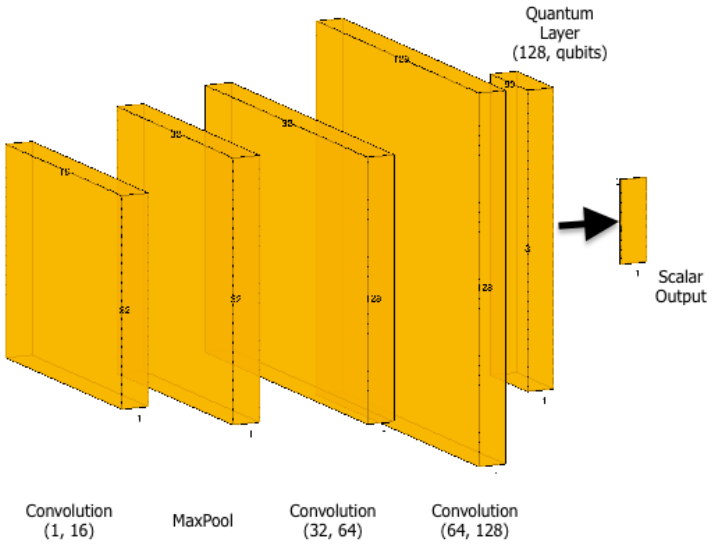


Figure 13: Neural Network with 4 classical and 1 quantum layer

The provided code defines a neural network model that is used for quantum circuit learning. The model consists of several layers of convolutional and linear (fully connected) layers, along with a custom-defined TorchCircuit module that is used to implement quantum circuits. The architecture of the model is as follows:

- The first layer is a convolutional layer with 1 input channel, 16 output channels, and a kernel size of 2. This layer is used to extract features from the input data.
- The second layer is a max pooling layer with a kernel size of 1. This layer is used to downsample the output of the first convolutional layer.
- The third layer is another convolutional layer with 16 input channels (from the previous layer), 32 output channels, and a kernel size of 2. This layer further extracts features from the data.
- The fourth layer is a linear (fully connected) layer with 128 input features and 50 output features. This layer is used to transform the output of the convolutional layers into a format that can be input to the quantum circuit.
- The fifth layer is another linear layer that maps the output of the quantum circuit (which is a vector of qubit probabilities) to a single output value by getting the value with the highest amplitude.
- The final layer is a linear layer that maps the single output value to a vector of length 4, which corresponds to the probabilities of the input data belonging to one of four classes.

In summary, the model consists of a combination of convolutional and linear layers, along with a custom-defined TorchCircuit module for implementing quantum circuits. The model is designed to take input data and use a quantum circuit to extract features from the data, which are then used to classify the data into one of four classes.

4.4.1 Classical

The classical part of this neural network consists of convolutional and fully connected layers that are commonly used in image classification tasks. The input to the model is expected to be a 1D signal (such as audio data), and the convolutional layers are used to extract relevant features from this signal.

The `Conv1d` module is used to perform 1D convolution on the input signal. In this model, there are two convolutional layers: .conv1 and `self.conv2`. `self.conv1` takes an input with a single channel and produces 16 output channels, while `self.conv2` takes 16 input channels and produces 32 output channels. The kernel size parameter specifies the size of the convolutional kernel, which is set to 2 in both layers. This means that the convolutional kernel slides over the input signal in 2-step increments.

The purpose of the convolutional layers is to extract local features from the input signal. Each output channel in a convolutional layer corresponds to a different feature that the layer is trying to detect. For example, one output channel might correspond to a high-frequency pattern in the input signal, while another might correspond to a low-frequency pattern. By using multiple convolutional layers with different output channels, the model can detect a wide range of features in the input signal.

After each convolutional layer, a max pooling layer is applied using the `MaxPool1d` module. The kernel size parameter specifies the size of the pooling window, which is set to 1 in both layers. This means that the pooling operation simply takes the maximum value over each neighboring pair of values in the output of the previous layer. The purpose of the pooling layer is to reduce the spatial dimensionality of the output from the convolutional layer while preserving the most important features.

The output of the second max pooling layer is then flattened and passed through two fully connected layers (Linear). The purpose of these layers is to further process the output of the convolutional layers into a format that is suitable for classification. The first fully connected layer (fc1) takes an input of 128 features and produces 50 output features. The second fully connected layer (fc2) takes an input of 50 features and produces an output with a size equal to the number of classes in the classification task, which is set to 3 in this model.

In summary, the convolutional layers are used to extract features from the input signal, while the max pooling layers and fully connected layers are used to process these features into a format that is suitable for classification. This architecture has been shown to be effective in many image classification tasks, and has also been adapted for use with other types of input signals, such as audio data.

4.4.2 Quantum

In this section, we introduce the `QiskitCircuit` class, a Python class that generates a quantum circuit using the `Qiskit` framework. The class defines a quantum circuit with number of qubits, applies a Hadamard gate to all qubits, followed by a parameterized RY gate to each qubit, and then measures all qubits in the computational basis. The circuit can be run on a specified backend with a specified number of shots.

The `QiskitCircuit` class generates a quantum circuit with *n_qubits* using the Qiskit framework [**?**]. The circuit consists of a Hadamard gate applied to all qubits, followed by a parameterized RY gate applied to each qubit. The circuit is then measured in the computational basis. The class has the following attributes:

- `circuit`: The quantum circuit generated by the class.
- `n_qubits`: The number of qubits in the circuit.
- `thetas`: A dictionary containing parameterized gates for each qubit in the circuit.
- `backend`: The backend on which the circuit is run.
- `shots`: The number of shots used to run the circuit.

The `QiskitCircuit` class has the following methods:

- `__init__(self, n_qubits, backend, shots)`: Initializes the `QiskitCircuit` object by generating the quantum circuit with *n_qubits* qubits, and setting the backend and number of shots for running the circuit.
- `N_qubit_expectation_Z(self, counts, shots, nr_qubits)`: Calculates the expectation value of the Z operator for each qubit in the circuit using the counts obtained from running the circuit. Returns a numpy array of the expectation values.
- `run(self, i)`: Runs the circuit with a set of parameters i, and returns the expectation value of the Z operator for each qubit in the circuit.

The following code snippet shows the implementation of the `QiskitCircuit` class:

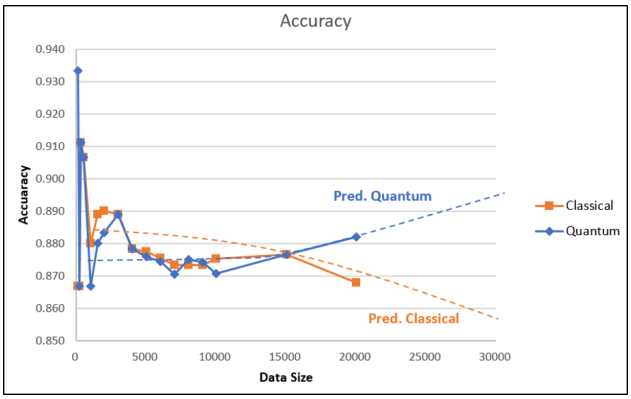
```
class QiskitCircuit():
    def __init__(self, n_qubits, backend, shots):
        # --- Circuit definition ---
        self.circuit = qiskit.QuantumCircuit(n_qubits)
        self.n_qubits = n_qubits
        self.thetas ={k : Parameter('Theta'+str(k))for k in range(self.n_qubits)}

        all_qubits = [i for i in range(n_qubits)]
        self.circuit.h(all_qubits)
        self.circuit.barrier()
        for k in range(n_qubits):
            self.circuit.ry(self.thetas[k], k)
        self.circuit.measure_all()
        self.backend = backend
        self.shots = shots
    def N_qubit_expectation_Z(self,counts, shots, nr_qubits):
        expects = np.zeros(len(QC_OUTPUTS))
        for k in range(len(QC_OUTPUTS)):
            key = QC_OUTPUTS[k]
            perc = counts.get(key, 0) /shots
            expects[k] = perc
        return expects
```

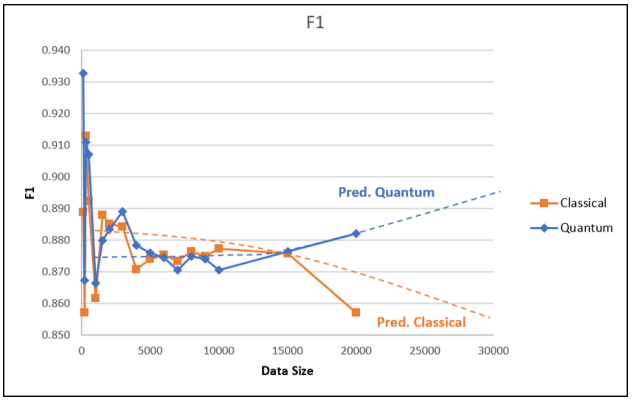
5 Result and Discussion

5.1 Quantum Kernel Learning for Large Dataset ($> 30,000$ samples)

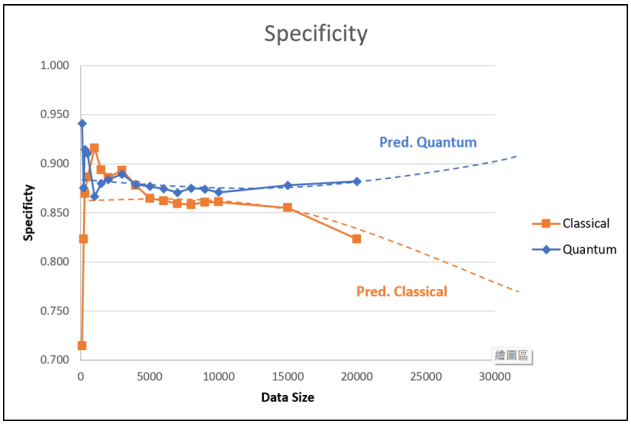
In Figure 10, we present the results of our quantum kernel learning approach for star classification in terms of accuracy, f1 score, specificity, and sensitivity. We observe that our quantum kernel learning approach improves its performance with increasing training data, with a turning point around 15,000 samples. Specifically, our approach exhibits higher accuracy, f1 score, and specificity than the classical kernel. Despite a slightly lower sensitivity score, the quantum kernel remains stable at around 0.9, demonstrating its robustness for star classification. Furthermore, our results reveal that the performance of the quantum kernel is more stable than that of the classical kernel in most cases, suggesting the superiority of our approach. This could be attributed to the capability of the quantum kernel to efficiently capture and process complex features of the data, leading to more stable and accurate predictions.



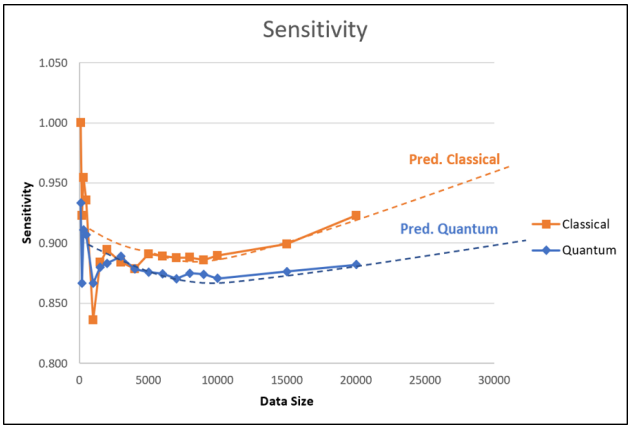
(a) Accuracy of QSVM Result



(b) F1 of QSVM Result



(c) Specificity of QSVM Result



(d) Sensitivty of QSVM Result

Figure 14: The statistic result of quantum kernel learning for star classification with different sizes of dataset

5.2 GPU Acceleration with cuQuantum for Quantum Kernel Encoding

Quantum computing holds great promise for solving complex machine learning problems. However, training quantum machine learning models can be computationally intensive and requires specialized hardware, such as quantum processing units (QPUs). One approach to mitigating the computational burden is to use classical hardware to pre-process the data and extract relevant features before training a quantum kernel. This approach, known as quantum kernel learning, has shown promising results in solving classification problems.

To further accelerate the training of quantum machine learning models, GPU acceleration can be utilized. CuQuantum is a software library that enables the training of quantum machine learning models on NVIDIA GPUs. This allows for faster training times and can significantly reduce the computational cost of quantum kernel learning. Run.ai's CUDA provides a platform for running CUDA-enabled applications, including CuQuantum. In this research work, we accelerate the training of quantum machine learning models by leveraging the power of NVIDIA GPUs, leading to faster and more accurate predictions.

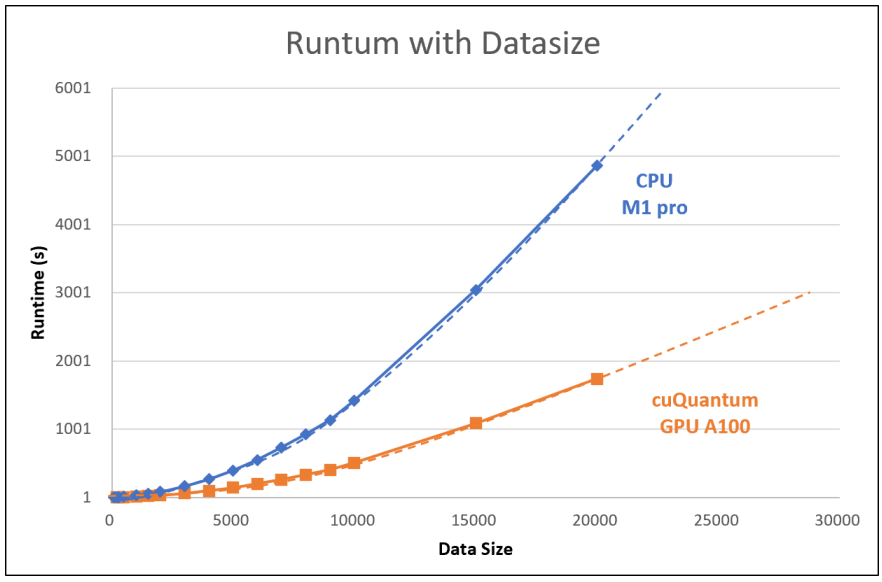


Figure 15: GPU speedup for kernel encoding by cuQuantum compared to CPU [15]

One major advantage of using GPU acceleration for quantum kernel learning is its speed. NVIDIA GPUs are optimized for parallel computing, which means they can handle large amounts of data simultaneously, and perform complex mathematical operations faster than CPUs. This allows for significantly faster training times compared to running quantum kernel learning on a CPU.

In our work, the result has shown that using GPU acceleration with CuQuantum can accelerate the training of quantum machine learning models by up to three times compared to running on a CPU alone. This speedup is due to the parallel processing power of GPUs, which can process more data and perform more computations in a shorter amount of time.

Additionally, the use of CuQuantum and CUDA also reduces the computational cost of quantum kernel learning, making it more accessible for researchers and businesses. With the ability to run CUDA-enabled applications on the Run.ai platform, researchers can access powerful GPU acceleration without the need to invest in expensive hardware or maintain a dedicated computing infrastructure. This makes quantum kernel learning more scalable, cost-effective, and accessible for a wider range of applications.

5.3 QCNN for Multi-labeled Star Classification

First, the neural network is initialized as an instance of the Net class. Then, an optimizer is created using the `Adam` optimizer, which updates the network's parameters based on the gradient of the loss function. The learning rate of the optimizer is set to 0.001. The Cross Entropy Loss function is created using the `nn.CrossEntropyLoss()` method, which is a popular loss function for classification tasks.

Next, the train() method is called on the network object, which sets the network to training mode, allowing it to learn from the data. The number of epochs, which determines the number of times the network will iterate over the entire training set, is set to 10.

In the training loop, which iterates over the epochs, two variables are initialized as zero: `totalloss` and `totalcorrect`. These variables will accumulate the loss and accuracy values for the current epoch.

The training data is then looped through, one data point at a time. For each data point, the input and target values are obtained from the training set. The input is then converted into a PyTorch Tensor object using the `Tensor()` method. The target is also converted to a PyTorch `LongTensor` using the `long()` method. The optimizer's gradient is then set to zero using the `zerograd()` method.

The input is passed through the neural network using the network(data) method, which returns the network's prediction for the input. The loss is calculated using the `lossfunc(output, target)` method, which calculates the difference between the predicted output and the true target value. The loss is then backpropagated through the network using the `loss.backward()` method, which computes the gradient of the loss with respect to the network's parameters. Finally, the optimizer's `step()` method updates the weights of the network based on the computed gradients.

The `totalloss` variable is then updated by adding the current loss value. The accuracy is calculated by finding the index of the maximum value in the output tensor, which corresponds to the predicted class. The index is compared with the target value to check if the prediction is correct. The number of correct predictions is then accumulated in the `totalcorrect` variable.

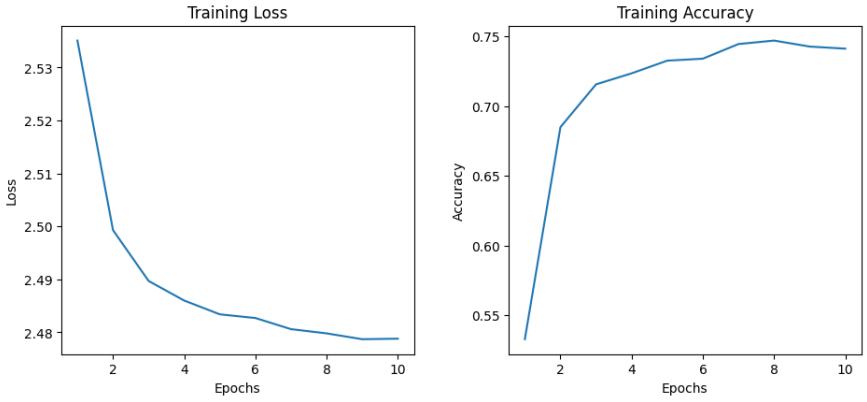


Figure 16: Training performance, Loss - Left figure, Accuracy - Right figure [15]

At the beginning of training, the model has a loss of 2.5351 and an accuracy of 53.29 percent. As the model trains, we can see that the loss decreases and the accuracy increases with each epoch. After the first epoch, the model's accuracy improves significantly to 68.50 percent, and the loss also drops to 2.4993. The model continues to improve its accuracy in subsequent epochs, reaching 74.13 prcent accuracy by the end of training. The loss also steadily decreases, indicating that the model is learning to make better predictions.

It's important to note that while the accuracy is steadily improving, there is still some fluctuation in the training accuracy from epoch to epoch. This could be due to the complexity of the problem being solved, the size of the training data, and other factors. However, overall, the model is making progress in learning to predict the target labels accurately.

5.4 Discussion

In this work, we will use K-Nearest Neighbors and Linear Regression to perform benchmarking for star classification, using different metrics such as accuracy, F1 score, and feature importance. KNN is a popular classification algorithm that can be used for various applications, while LR is a simple yet powerful algorithm widely used in healthcare, finance, and marketing. By comparing the performance of these two algorithms, we can gain insights into the suitability of different approaches for star classification.

5.4.1 Classical Model for Benchmarking with two-labelled classification

- K-Nearest Neighbors (KNN)
  KNN is a non-parametric algorithm that is widely used for classification and regression tasks. KNN for star classification involves finding the nearest neighbors of a given star based on their features such as magnitude, color, and distance. The algorithm works by computing the distance between each star in the dataset and the test star, and then selecting the K closest neighbors. The class of the test star is then assigned based on the most common class among the K nearest neighbors.

  In the context of star classification, KNN has shown promising results due to its simplicity and interpretability. However, the performance of KNN is highly dependent on the choice of K and the distance metric used. A small value of K may result in overfitting, while a large value of K may result in underfitting. Additionally, selecting an appropriate distance metric is crucial as different metrics may lead to different results.

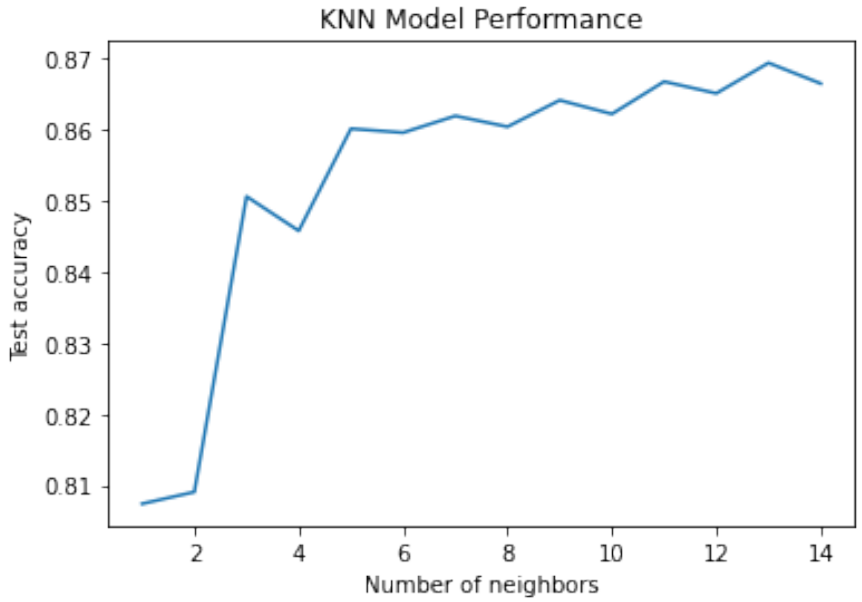


Figure 17: KNN test accuracy result increases with the number of neighbours.

Figure 12 displays the performance of our KNN model on the large star dataset. As we can see, the accuracy increases as the number of neighbours considered (K) decreases. The highest accuracy achieved is around 86.5% when K = 15, and the performance remains stable after this point. Therefore, we have selected K = 15 as the optimal parameter for our classical KNN model to perform star classification. It should be noted that the optimal value of K can vary depending on the dataset and the specific classification problem at hand. Nevertheless, KNN remains a simple yet effective classification algorithm that can provide accurate results in many cases.

- Logistic Regression (LR)
  Logistic Regression is a classification algorithm that is commonly used in machine learning for binary classification problems, where the goal is to predict a binary outcome, such as whether an email is spam or not. The algorithm models the probability of the binary outcome using a logistic function. The logistic function maps any real-valued input to a value between 0 and 1, which can be interpreted as the probability of the positive outcome. Logistic Regression uses a linear function to model the relationship between the input variables and the log-odds of the outcome. The parameters of the model are estimated using maximum likelihood estimation, and regularization can be used to prevent overfitting.

  In addition to its classification capabilities, Logistic Regression can also provide insight into the importance of each feature in the model. Feature importance in Logistic Regression is determined by examining the magnitude and sign of the coefficients assigned to each feature. The absolute value of the coefficient indicates the strength of the relationship between the feature and the outcome, while the sign indicates the direction of the relationship. Positive coefficients indicate a positive relationship between the feature and the outcome, while negative coefficients indicate a negative relationship. Features with larger absolute coefficients are considered more important in predicting the outcome. Feature importance can be used to identify the most relevant features for the task at hand and to interpret the results of the model. This information can be valuable in a wide range of applications, from identifying risk factors in healthcare to determining the most effective marketing strategies for a product. For our classification LR model, the feature importance is shown in Fig. 13.

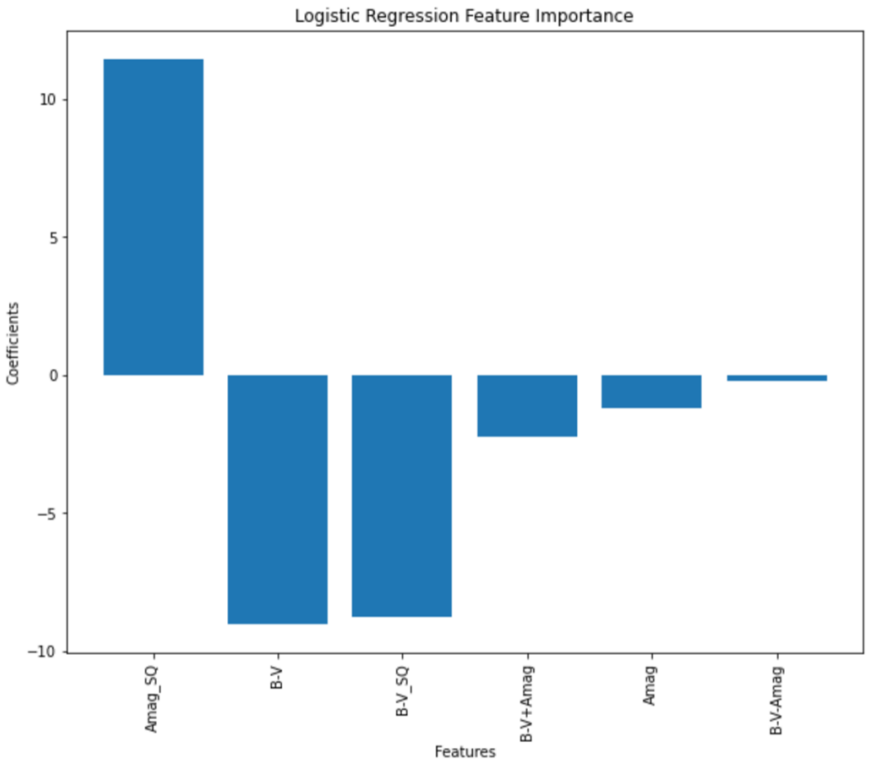


Figure 18: Feature Importance of LR.

5.4.2 Classical Model for Benchmarking with multi-labelled classification

In the benchmarking result, we can observe that both KNN and LR models perform well for two-labelled classification, achieving an accuracy of around 0.86. However, when it comes to multi-labelled classification, the accuracy drops to 0.78 and 0.79 for KNN and LR, respectively. This is likely due to the increased complexity of the classification problem when multiple labels are involved.

One interesting observation from the results is that the classification of stars belonging to spectral types K and M is particularly challenging. This is not surprising, as these stars have low surface temperatures and are therefore dimmer and more difficult to observe. In addition, they have complex spectra with a large number of absorption lines, making it difficult to accurately determine their spectral type. These factors make the classification of A&F and K&M stars a challenging problem, even for advanced machine learning algorithms.

Given these challenges, the exploration of quantum machine learning methods such as QSVM and QCNN may offer promising avenues for improving the accuracy of star classification. By leveraging the unique properties of quantum computing, these methods may be able to better handle the complex and multi-dimensional nature of the data, leading to more accurate and efficient classification models.

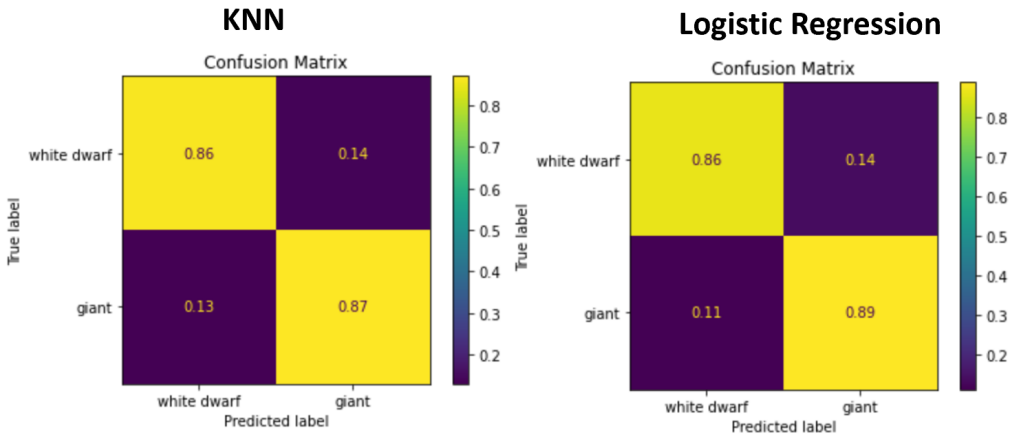


Figure 19: Confusion Matrix for Benchmarking with KNN and LR (Two-labels)

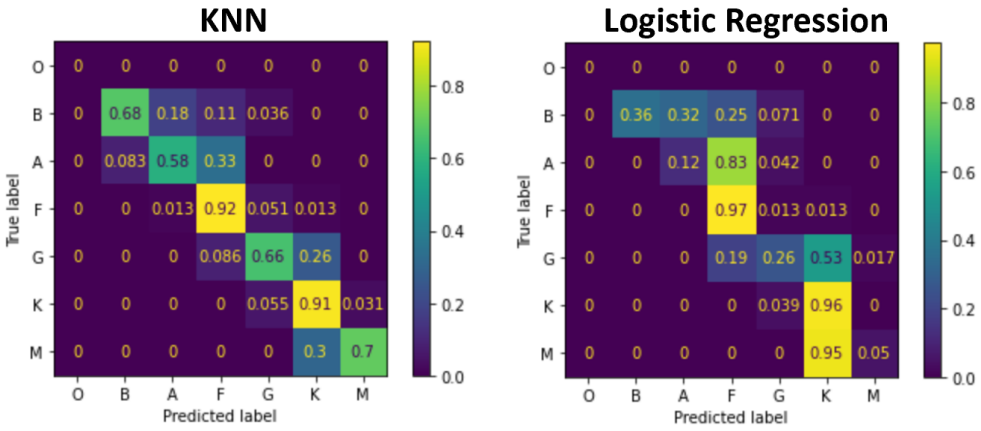


Figure 20: Confusion Matrix for Benchmarking with KNN and LR (Multi-labels)

5.4.3 Result of Quantum Kernel Learning for Multi-labelled Classification

With quantum kernel learning, we can achieve an accuracy of approximately 0.81, which is 5% higher than the classical benchmarking methods. Furthermore, we observed that our quantum model exhibits improved classification performance in star types A & F and K & M (shown in Fig.18), as compared to KNN and LR, respectively. These observations suggest that quantum machine learning can offer significant improvements in star classification tasks, especially for challenging classes like K and M, which are known to be difficult to classify using traditional classification methods. The promising results obtained using quantum machine learning methods in this study can inspire further research into developing more powerful and efficient algorithms for star classification tasks.

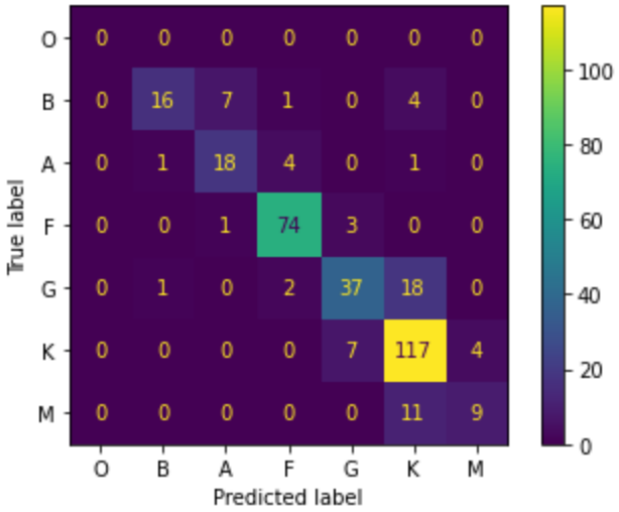


Figure 21: Confusion Matrix for Quantum Kernel Learning with Multi-Labels

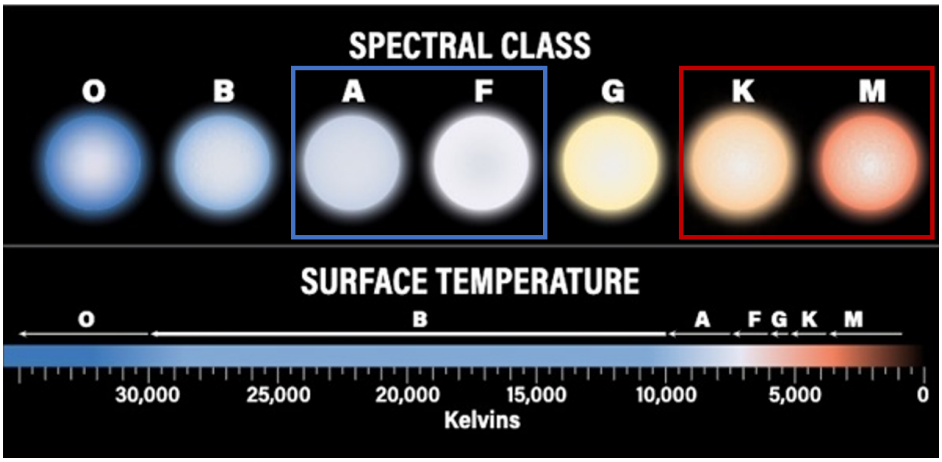


Figure 22: The stellar classification table indicating it's hard to distinguish types A & F and K & M.

6 Conclusion

Based on our benchmarking results, we found that the quantum kernel learning method outperformed the classical KNN and LR algorithms, achieving an accuracy of 80.7%. This advantage is attributed to its ability to leverage quantum computation for feature mapping, which can improve classification accuracy. While QCNN showed lower accuracy (72.3%) compared to other models, it holds potential advantages for handling larger and more complex datasets with a smaller number of quantum bits, which can reduce the hardware requirements for implementing quantum machine learning models. Further research is needed to explore the potential advantages of QCNN for star classification.

Moreover, quantum kernel learning can benefit from GPU acceleration using software libraries such as CuQuantum, which can accelerate training times up to three times faster than using CPUs. This acceleration can significantly reduce the computational cost of quantum kernel learning and make it more accessible to a wider range of applications, including star classification.

Overall, quantum kernel learning and QCNN are exciting areas of research that holds great promise for the future of AI and beyond, with the potential to unlock the power of quantum computing for machine learning tasks for astrophysics and high energy physics.

Table 1: Summary Table: Accuracy of Each Model for Multi-labelled Classification

	KNN	**LR**	**Quantum Kernel Learning**	**QCNN**
Accuracy	78.9%	79.9%	80.7%	72.3%

References

[1] C. A. L. Bailer-Jones and M. Irwin, *Automated classification of hipparcos periodic variable stars using support vector machines*, Astronomy & Astrophysics **385** (2002), pp. 95–105.

[2] D. F. Gray, *The observation and analysis of stellar photospheres*, Cambridge University Press, 2021.

[3] P. Keenan and E. Kellman, *An atlas of stellar spectra. with an outline of spectral classification. astrophys*, Monographs. Chicago (1943).

[4] A. N. Cox, *Allen's astrophysical quantities*, Springer, 2015.

[5] E. Hertzsprung and H. N. Russell, *On the relation between the spectra and other characteristics of the stars.*, Astrophysical Journal **67** (1928), pp. 40–79. doi:10.1086/143167.

[6] J. H. Lee and K. M. Seo, *A spectral classification of stars using decision tree and ensemble learning*, Journal of the Korean Astronomical Society **50** (2017), pp. 173–186.

[7] C. A. L. Bailer-Jones, R. Andrae, B. Arcay and et al., *Estimating distances from parallaxes. iv. distances to (some) open clusters using GAIA data*, Astronomy & Astrophysics **618** (2018), p. A93.

[8] P. Rebentrost, M. Mohseni and S. Lloyd, *Quantum support vector machine for big data classification*, Physical Review Letters **113** (2014), p. 130503.

[9] C. Cortes and V. Vapnik, *Support-vector networks*, Machine learning **20** (1995), pp. 273–297.

[10] W. S. Noble, *What is a support vector machine?*, Nature biotechnology **24** (2006), pp. 1565–1567.

[11] V. Havlíček et al., *Supervised learning with quantum-enhanced feature spaces*, Nature **567** (2019), pp. 209–212.

[12] S. L. Wu et al., *Application of quantum machine learning using the quantum kernel algorithm on high energy physics analysis at the lhc*, Physical Review Research **3** (2021), p. 033221.

[13] W.-F. Ku, *Star categorization giants and dwarfs dataset, vinesmsuic.* https://www.kaggle.com/datasets/vinesmsuic/star-categorization-giants-and-dwarfs, Jul 2020.

[14] Amazon Web Services, Inc., *Amazon Braket: Get started with quantum computing.* https://aws.amazon.com/braket/, 2021. Accessed: February 23, 2023.

[15] NVIDIA, *cuQuantum SDK: Simulating quantum circuits on GPUs.* https://developer.nvidia.com/cuquantum-sdk, 2021. Accessed: February 23, 2023.

[16] S. Stanwyck, H. Bayraktar and T. Costa, *cuquantum: Accelerating quantum circuit simulation on gpus*, in APS March Meeting Abstracts, vol. 2022, 2022, pp. Q36–002.