Enhancing Fraud Detection with Quantum Machine Learning: A Comparative Study of Quantum and Classical Approaches

William Jasmine

2024-12-15

Abstract

Recent advancements in quantum computing have given rise to the development of Quantum machine learning (QML), a field which studies how transforming classical data into a quantum feature space could potentially benefit predictions made by machine learning (ML) algorithms. The work presented here provides a study of how two of these QML algorithms - Quantum Support Vector Machines (QSVM) and Quantum Random Forests (QRF) - perform when attempting to solve a fraud detection machine learning problem: identifying fraudulent credit card transactions. While tuned versions of traditional Support Vector Machine (SVM) and Random Forest (RF) models achieved respectable F1 scores for the fraudulent class (0.8 and 0.91, respectively), they were outperformed by their quantum counterparts. A tuned SVM model implementing a quantum kernel produced via a tuned quantum circuit was able to construct a QSVM algorithm that resulted in a F1-score of 0.89 for the fraudulent class. A QRF algorithm constructed via a similar process expanded on this success, achieving a F1-score of 0.94 on the fraudulent class. Additionally, it was observed that increased quantum circuit complexity actually degraded the performance of QSVMs, while having little to no impact on QRF performance. The number of quantum gates (operations) present within the quantum circuits used to build these quantum kernels was also shown to exhibit a linear relationship with the computational processing time required to engineer the quantum kernel matrices used to fit the QML models.

1 Introduction

Recent years have seen groundbreaking research invested into both machine learning and quantum computing, positioning these fields at the forefront of some of the most anticipated scientific breakthroughs. Quantum computers promise to solve problems once deemed unsolvable by classical computing, while advancements in generative AI are poised to spark the next major technological revolution, with tech companies racing to develop and deploy their own AI-driven solutions. While much of the excitement surrounding each field has largely remained within their respective domains, researchers have been able to combine components of each into what is now called Quantum Machine Learning (QML). QML combines the newfound ability of being able to represent data via quantum states with existing machine learning models in an effort to theoretically optimize their performance.

The work presented here showcases how QML can be used to predict credit card fraud as an example of its potential fraud detection capabilities. Fraud detection has become an increasingly important area of research in recent years, given that the methodologies utilized by fraudsters are continuously evolving in complexity, posing novel challenges that push the limits of traditional fraud detection systems. As such, QML can be a possible solution to fight back against these increasingly sophisticated fraudulent attacks.

Using IBM's `qiskit` Python library, various aspects of simulated quantum circuits were examined in order to model how different system parameters affected the results of the QML process that implemented them. More specifically, these circuits were used to encode classical data into a quantum feature space, and this data was then ingested by machine learning algorithms that could predict the presence of fraud in credit card transactions. These results were also compared to those generated using classical data. This project focuses on two ML algorithms in particular - Support Vector Machines (SVM) and Random Forests (RF) - along with their QML counterparts - Quantum Support Vector Machines (QSVM) and Quantum Random Forests (QRF).

2 Literature Review

While being a relatively new field, QML has still seen a number of groundbreaking developments over the years that demonstrate how quantum mechanics can be adapted to tackle complex data processing tasks. Using quantum computers to solve classification problems was explored as early as the late 1990's, in which Farhi et al. (1998) theorized how quantum computers could be used to build decision tress [1]. Later, Khadiev et al. (2019) showed how Grover's algorithm (also known as the quantum search algorithm) could be exploited to improve the efficacy of quantum decision trees, laying the groundwork for their team to be able to eventually demonstrate how to build QRFs in practice (Khadiev et al. (2021) [2, 3]. Most recently, QRFs that utilize "quantum kernels" have been shown to further enhance their ability to process and analyze data, first evidenced by Srikumar et al. (2022) [4].

Research on how to apply quantum enhancements to machine learning has been extended to numerous additional ML algorithms, including SVMs. In the early 2000s Anguita et al. (2003) first laid out the theoretical foundation for how quantum computing could be used to train SVM models [5]. Rebentrost et al. (2014) and Chatterjee et al. (2017) expanded on this early work to show how QSVMs could be produced in practice by creating quantum versions of the kernel needed by SVM models to transform data into a higher dimensional space [6, 7].

While a myriad of other theory-based papers were identified over the course of this literature review, the following comprehensive studies provide brilliant summaries of the numerous techniques and developments seen over the course of QML's relatively brief history:

- Biamonte et al. (2017) provides a robust review of the theoretical framework behind QSVM and the potential benefits that can be achieved when compared to using traditional SVMs [8].
- Zeguendry et al. (2023) provides a concise overview of the quantum mechanics involved in building QML algorithms and provides numerous case studies of how these algorithms can be applied in practice [9].

Each of the above papers have numerous citations and were used as key points of reference while completing the work presented here.

There are also numerous instances of instances in which QML algorithms were applied to real world scenarios to test the potential upsides speculated in the aforementioned theoretical research. The list below includes some noteworthy examples:

- Shan et al. (2022) used QSVMs as a methodology for breast cancer detection and show that it can match accuracy results comparable to that of classical SVMs [10].
- Luo et al. (2024) have shown that QML algorithms have "promising potential" when being used to predict predict the presence of Alzheimer's disease, though have not yet surpassed their classical counterparts in terms of learning capability [11].
- Cherrat et al. (2023) were able to develop a reinforcement learning model via Quantum Neural Networks (QNNs) and used financial sector data to tackle the problem of hedging [12].

Numerous additional case studies of QML are summarized in a study done by Nguyen et al. (2024), which draws on findings from 32 previously written papers and gives a robust summary of the possible benefits QML algorithms can provide [13]. This paper was also used as a consistent point of reference when completing the work presented here. A review of these numerous case studies reveals that QML algorithms either very nearly approach of exceed the performance seen from classical ML algorithms, fostering excitement when considering the prospect of applying these algorithms to the realm of fraud detection.

That being said, there is already some existing work that shows how QML can be applied to fraud detection problems. Namely, Innan et al. 2023 demonstrates that QSVMs seem to outperform Quantum Neural Network (QNN) algorithms when solving fraud detection problems, while Kyriienko et al. (2022) shows that QSVMs have similar performance compared to classical SVMs when trying to identify fraudulent banking transactions [14, 15]. That being said, the purpose of this work is to extend beyond the existing research to specifically quantify how changing the features of a quantum circuit affects the results of both QSVMs and QRFs in regards to predicting credit card fraud, and compare the results of *both* algorithms against those of their classical counterparts.

It is worth noting that as of July 2024, it has been revealed that Deloitte is exploring using QML (specifically, QNNs) to detect fraudulent online payments [16]. While they have provided no study of their results, the work presented here could provide quantitative evidence that utilizing QML should be considered as a possible next step in the ongoing fight against fraud.

3 Hypothesis

Given the findings of the literature review, it appears clear that QSVMs should approach or exceed the performance of a classical SVM model in identifying credit card fraud, especially after the best quantum circuit for the scenario has been identified. Given the arguably superior ability of RFs over SVMs to handle high dimensional feature spaces, it is expected that QRFs should be able to build on the gains realized by QSVMs. Once again, this is only after the best tuned quantum circuit has been identified. Identifying ideal quantum circuits is an exercise in balancing complexity: adding to the circuit's ability to handle more complex data structures (by increasing the number of qubits, using more expansive entanglement patterns etc.) should increase model performance up a point, but will eventually witness decreased efficiency and suffer from added noise.

4 Data

4.1 Data Source

The dataset used in this project is sourced from Kaggle and contains data of over 550,000 credit card transactions in the EU during 2023 [17]. The original source of the data is the result of a research collaboration between Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelles)]18]. The data is anonymized to protect the identity of the credit card users and contains 29 predictor variables representing various aspects of each credit card transaction. Due to the anonymization, it is impossible to know exactly what each of these first 28 features represent (they are all labelled simply as V1 through V28), but it is shared that they represent the commonly captured features of a credit card transaction (i.e. time, location, etc.). These first 28 features have all been transformed to be continuous numerical predictors. The only titled predictor variable in the dataset is "Amount", which represents the total amount of the credit card transaction in Euros. The goal

when using this dataset will be to predict "Class", a binary field indicating whether or not the transaction was fraudulent.

4.2 Data Cleaning

Datasets for this project were considered only if they had a reputation for being clean and usable, ensuring the focus remains on model performance rather than being heavily influenced by data cleaning, imputation, or collection methods. As such, this dataset contains 568,630 records in which none of the fields have missing, misprinted, or nan values. The target variable, "Class", contains either 1 or 0, indicating whether the observation is fraudulent (1) or valid (0).

5 Methods

5.1 Data Sampling

As previously mentioned, the chosen dataset contains over half a million observations. Given the computational complexity of fitting certain models to a dataset of this size, a vast reduction in the number of samples was warranted. Additionally, Figure 1 highlights an additional complication with the original dataset: balanced classes.

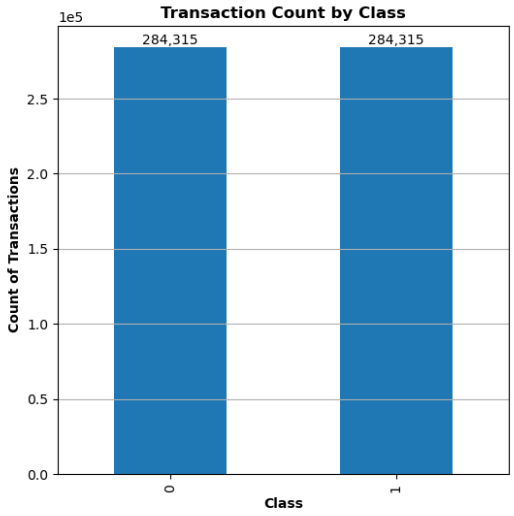

Figure 1: There are an equal number of fraudulent (1) and valid (0) transactions in the initial dataset.

Though the balanced dataset might make it easier to predict instances of fraud, it is not representative of what would occur in the real world. The European Banking Authority (EBA) estimates that "fraud accounted for 0.031% of total card payments in value and 0.015% of total card payments in volume terms in H1 2023" [18]. To combat both of these issues the following decisions were made in regarding the data:

1. Sample the dataset so that only 1% of observations represent fraudulent transactions.

2. Only include 10,000 observations in total.

To do this, 100 class 1 samples and 9,900 class 0 samples were randomly selected from the original dataset, without replacement. These subsets were then combined, forming a new sampled dataset to use for analysis and modeling. Although the percentage of fraudulent transactions is still inflated compared to the estimate provided by the EBA, this update shifts the machine learning objective to actual fraud detection while still ensuring that a sufficient number of fraudulent samples are retained. Figure 2 shows the the updated version the class distribution after these changes have been made.

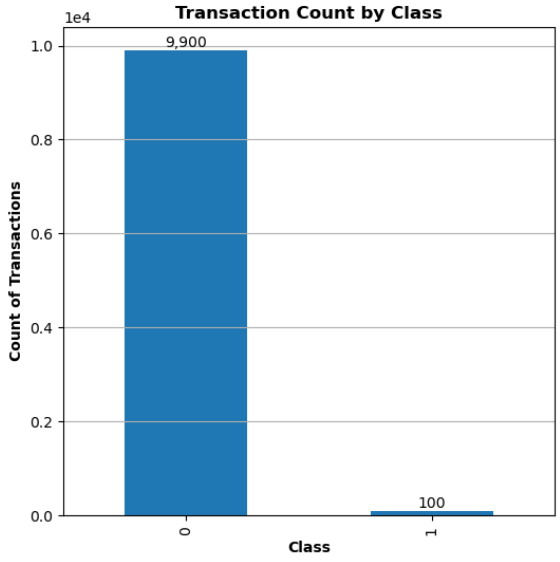

Figure 2: The updated breakdown by class represents a paints a more realistic picture of credit card fraud, and still maintains a large number of total transactions.

5.2 Data Preprocessing

5.2.1 Outliers

An outlier analysis was performed in which outliers from each predictor were identified via the well-established Tukey method [19]:

Given a vector X, a data point $x_i \in X$ is considered an outlier if it satisfies:

$$x_i < Q_1 - 1.5 \cdot \text{IQR} \quad \text{or} \quad x_i > Q_3 + 1.5 \cdot \text{IQR} \tag{1}$$

such that $Q1$ and $Q3$ represent the first and third quartiles of X, and $\text{IQR} = Q_3 - Q_1$ (the inter-quartile-range of X). Figure 3 shows the result of this analysis, and makes clear that this methodology identified a large number of outliers.

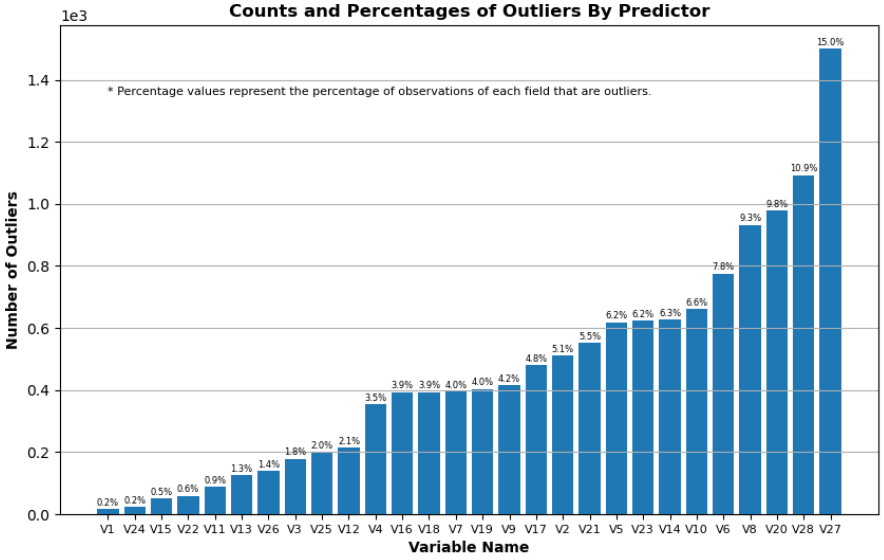

Figure 3: Outliers are present in each of the predictor variables when using the definition from Equation 1.

To reduce the quantity of outliers in the data and mitigate any potential negative impacts they might have when attempting to fit models, a Windsorization transformation was applied [20]. Winsorization is a statistical transformation technique used to limit extreme values in a dataset by replacing them with specified percentiles, and is defined as follows:

Given a vector X, a data point $x_i \in X$, and chosen percentiles $p_{\max}$ and $p_{\min}$, let $x_{\max}$ and $x_{\min}$ represent the values at each of the chosen percentiles. Windsorization transforms $x_i \to x_i'$ via:

$$x_i' = \begin{cases} x_i, & \text{if } x_{\min} \leq x_i \leq x_{\max} \\ x_{\min}, & \text{if } x_i < x_{\min} \\ x_{\max}, & \text{if } x_i > x_{\max} \end{cases} \tag{2}$$

This transformation ensures that values below the $p_{\min}$-th percentile are replaced by $x_{\min}$, and values above the $p_{\max}$-th percentile are replaced by $x_{\max}$. In this instance, the 5th and 95th percentiles were chosen as the limits of the Windsorization transformation.

5.2.2 Normality

The `normaltest` function from Python's `scipy.stats` library was used to determine if the data present in each of the predictor fields followed a gaussian distribution [21]. The implementation is based on statistical tests defined by D'Agostino and Pearson, and combines skewness and kurtosis to produce an omnibus test of normality [22, 23]. The null and alternative hypotheses for the test are as follows:

$$H_0 : \text{Data follows a normal distribution.} \tag{3}$$

$$H_a : \text{Data does not follow a normal distribution.} \tag{4}$$

Applying this test to each of the predictor fields resulted in p-values < 0.05, meaning that we can reject the null hypothesis and consider the data as non-normal. In an effort to address this (and also further limit the influence of the previously identified outliers), a Yeo-Johnson power transformation was applied (Box-Cox was not considered due to the presence of negative values in the dataset) [24]. A definition of this transformation is given below:

Given a vector X, a data point $x_i \in X$ and a transformation parameter λ, the

Yeo-Johnson transformation maps $x_i \to x_i'$ via:

$$x_i' = \begin{cases} \frac{(x_i+1)^\lambda - 1}{\lambda}, & \text{if } x_i \geq 0 \text{ and } \lambda \neq 0, \\ \log(x_i + 1), & \text{if } x_i \geq 0 \text{ and } \lambda = 0, \\ -\frac{(-x_i+1)^{2-\lambda} - 1}{2-\lambda}, & \text{if } x_i < 0 \text{ and } \lambda \neq 2, \\ -\log(-x_i + 1), & \text{if } x_i < 0 \text{ and } \lambda = 2. \end{cases} \tag{5}$$

λ in this case is determined when running the `normaltest` function, and is the value that maximizes the normality of the transformed data.

5.2.3 Multicollinearity

Figure 4 shows a correlation matrix of the dataset, which highlights the correlations between all pairs of predictor variables.

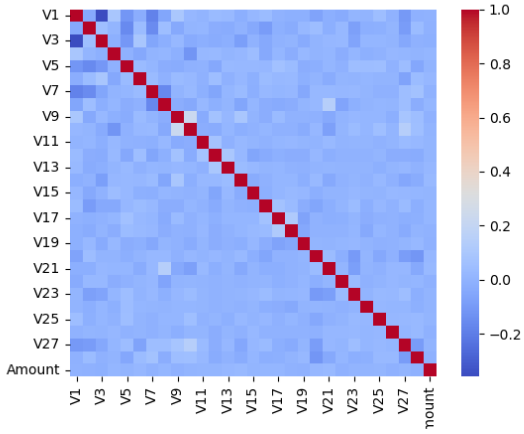

Figure 4: A correlation matrix of the transactions dataset. All correlations r were within $-0.36 < r < 0.23$.

Visual inspection of the correlation matrix reveals that no pair of unique predictor variables maintain a troublesome degree of correlation, providing evidence for the fact that dimensionality reduction transformations (i.e. PCA) are not necessary.

5.2.4 Scaling

Given that the data was not originally considered normal, standard scaling (in which $\mu = 0$ and $\sigma = 1$) is not appropriate. Additionally, min-max scaling (which scales all values to lie between 0 and 1) is highly sensitive to outliers, making it unsuitable

for this case. Robust scaling, on the other hand, is a preferred approach when handling data with outliers [19]. This scaling methodology centers each field around its median and scales it based on its interquartile range (IQR), effectively minimizing the influence of extreme values. As such, this was the scaling transformation applied in this case. A definition of robust scaling is provided below:

Given a vector X and a data point $x_i \in X$, robust scaling transforms $x_i \to x_i'$ via:

$$x_i' = \frac{x_i - Q_2}{\text{IQR}} \tag{6}$$

where Q_2 and IQR are the median of and inter-quartile-range of X, respectively.

5.2.5 Full Transformation Pipeline

In summary, the following data pre-processing transformations were applied to the dataset.

1. Split the data into testing and training sets. 25% of the sample was kept for testing, with the class distribution present in the original dataset preserved across both sets.
2. Fit and apply Windsorization transformation to the training set.
3. Use the fitted percentile values from step 2 to apply a Winsorization transformation to the testing set.
4. Fit and apply Yeo-Johnson transformation to the training set.
5. Use the fitted λ value from step 4 to to apply a Yeo-Johnson transformation to the testing set.
6. Fit and apply a robust scaling transformation to the training set.
7. Use the fitted median and IQR values from step 6 to apply a robust scaling transformation to the testing set.

The parameters used for each transformation are only determined via the training data in order to prevent data leakage. Figures 5 and 6 use box-plots to highlight the positive effects these transformations had on the distributions of the predictor variables.

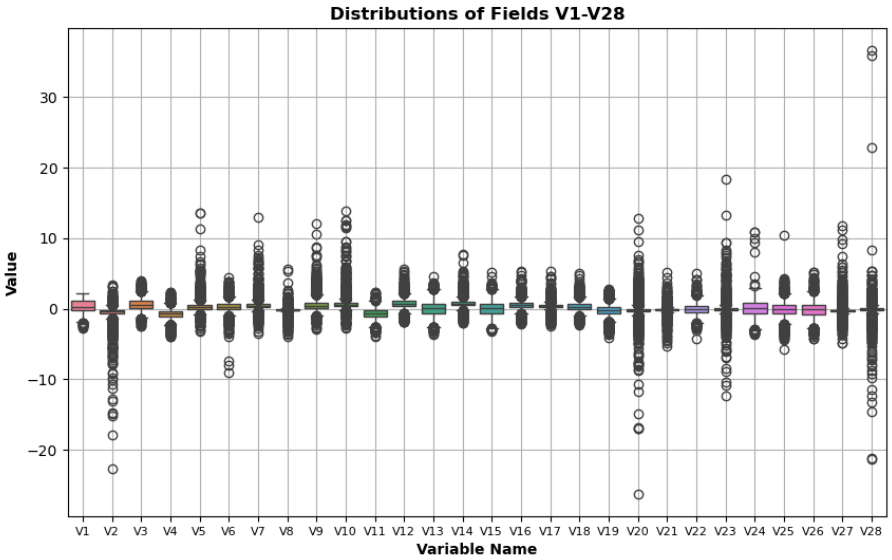

Figure 5: Represents the data before any pre-processing transformations. The predictors suffer from varying distributions, ranges, and exhibit a large number of outliers.

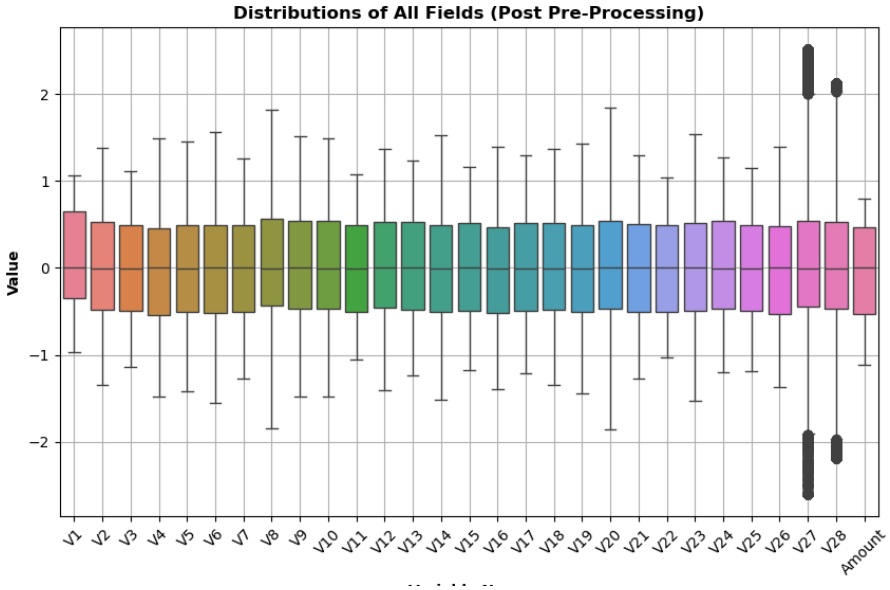

Figure 6: Represent the data after all data pre-processing transformations. All but two fields now have outliers, and the range of their distributions no longer vary from one another

5.3 Exploratory Data Analysis

Before applying any machine learning models to the data, an initial analysis was completed in an effort to determine which of the explanatory variables might be useful in making predictions. To do this, t-tests were performed that compared the difference of means for each field when separated by class [25]. Figure 7 shows the results of these t-tests, making it clear that many of the explanatory fields exhibited statistically significant differences. An example of one of these fields is shown in Figure 8.

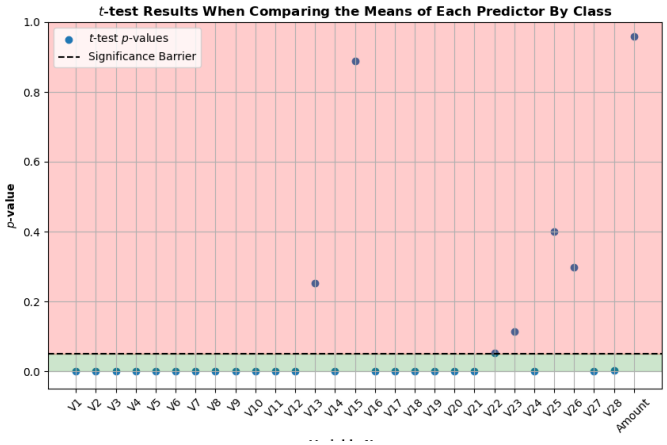

Figure 7: Many of the predictor variables exhibit statistically significant differences when comparing the means of each class.

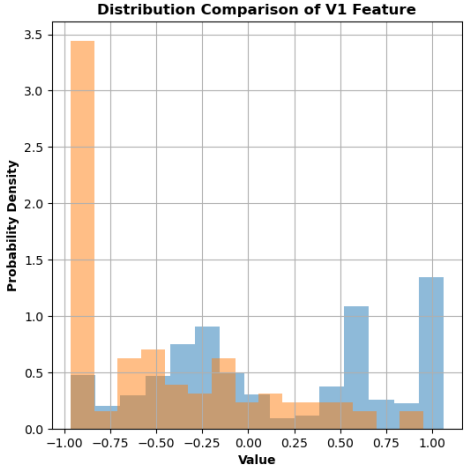

Figure 8: V1 was one of the statistically significant field identified in Figure 7.

The results of this analysis bode well for the quality of the models built, as it appears to contain features that will be useful in making predictions on the target variable. It provides confidence that identifying the fraudulent transactions via ML algorithms is possible.

5.4 Support Vector Machines (SVM)

Support Vector Machines (SVM) are a type of supervised learning algorithm used for classification tasks, especially when the data is not linearly separable [26. The key idea behind SVM is to find the hyperplane that best separates the data into different classes. More specifically, SVM attempts to find the hyperplane that maximizes what's known as the *margin* - the distance between the hyperplane and the nearest data points from each class (these distances are the "support vectors"). In a simple two-dimensional space, this hyperplane is essentially a line that divides the data points, but in higher-dimensional spaces, it becomes a complex multidimensional boundary. Once the best hyperplane is identified, the SVM classifier can use this boundary to classify new data points.

SVMs are a classic option when solving classification problems, and was one of the first to be adapted to be used on quantum computers according to the literature review. Consequently, they emerged as a clear choice when determining which ML algorithms to evaluate in this project.

SVM models consist of a number of tunable hyperparameters, such as:

- **Kernel Type:** The function used to transform the data into a higher-dimensional space where it is more likely for the classes to be separable. Different types of kernels work well for different applications. For example, linear kernels work best when data is linearly separable in higher-dimensional space, while radial basis function (RBF) kernels work well for data with complex, non-linear patterns.
- C**:** A regularization parameter that affects the shape of the decision boundary. Higher C values focus more on correctly classifying every observation in the training data, resulting in more complex decision boundaries that can lead to overfitting. Lower C values allow for more misclassifications on the training data, resulting in a more generalized decision boundary that can potentially underfit the data.

- γ**:** This parameter is only required when using a radial basis function (RBF) kernel. γ controls how far the influence of a single training point extends. Low γ values capture broader patterns with smooth decision boundaries (risking underfitting), while high γ values focus on local details (risking overfitting).

Each of the above parameters were tuned when building the SVM model implemented in this project. The tuning was performed using a three-fold cross-validation with the candidates for each hyperparameter provided in Table 1.

Hyperparameter	**Candidates**
Kernel Type	Linear, RBF
C	0.1, 1, 10
γ	0.01, 0.1, 1

Table 1: The hyperparameter candidates for the classical SVM model.

Given the choice of hyperparameter candidates, $2 \cdot 3 \cdot 3 = 18$ models were tested resulting in 54 total fits after taking into consideration the three-fold cross validation. The models were all built using `scikit-learn`'s `SVC` class [20]. To address the high level of class imbalance, `SVC`'s `class_weight` argument was set to `'balanced'`. Doing so creates weights (w_1, w_0) for each class such that C is scaled as $C \cdot w_1$ for class 1 and $C \cdot w_0$ for class 0. The `'balanced'` setting adjusts the weights so that they are inversely proportional to class frequencies of the input data, allowing the algorithm to account for class imbalance. More specifically:

The weight for class c, w_c is calculated as:

$$w_c = \frac{N}{n_{\text{classes}} \cdot n_c} \tag{7}$$

where N = total number of observations, n_c is the total number of observations of class c, and n_{classes} is the total number of classes.

Lastly, given that the models are attempting to solve a fraud detection problem, the metric that was used to asses their performance was the F1-score of the predictions made for the fraudulent class. The F1 score provides an average of both the precision and recall score, and thus provides a more holistic picture of the type 1 and type 2 error rates.

5.5 Random Forests (RF)

Random Forests belong to family of ML algorithms known as ensemble learning methods, meaning that they combine the output of several simpler models - decision trees, in case of the random forest - to improve accuracy [27]. Each decision tree in a Random Forest is trained on a different, random subset of the data, introducing randomness that helps to limit overfitting (a common problem with individual decision trees). By taking a majority vote across all trees, Random Forests create a more stable and generalizable model compared to using just one decision tree and thus have a reputation for being models that can reduce variance while maintaining low bias.

Random forests tend to have the advantage over SVMs when it comes to dealing with many features and complex pattern recognition. As such, they were chosen as a candidate to evaluate if their quantum counterparts (QRFs) could provide added benefits in regards to fraud detection, especially since there is already some evidence of QSVMs being successful in this area [10, 13, 15].

Random Forest models consist of a number of tunable hyperparameters, such as:

- `n_estimators`: The number of trees in the forest. Increasing this generally improves performance but also increases computational cost.
- `max_depth`: The maximum depth of each tree. Limits the growth of the tree to prevent overfitting and manage model complexity.
- `min_samples_split`: The minimum number of samples required to split a node. Higher values reduce tree growth and promote generalization.
- `min_samples_leaf`: The minimum number of samples required to be at a leaf node. Larger values prevent overly specific splits, improving model robustness.

Each of the above parameters were tuned when building the RF model implemented in this project. The tuning was performed using a three-fold cross-validation with the candidates for each hyperparameter provided in Table 2.

Given the choice of hyperparameter candidates, $2 \cdot 2 \cdot 2 \cdot 2 = 16$ models were tested resulting in 48 total fits after taking into consideration the three-fold cross validation. The models were all built using `scikit-learn`'s `RandomForestClassifier` class

Hyperparameter	Candidates
n_estimators	100, 200
max_depth	10, 20
min_samples_split	5, 10
min_samples_leaf	1, 2

Table 2: The hyperparameter candidates for the classical RF model.

[20]. To address the high level of class imbalance, the `RandomForestClassifier` maintains a `class_weight` argument that was set to `'balanced'`. Doing so creates weights (w_1, w_0) utilizing the same methodology implemented by the `SVC` class, and uses them to modify the Gini impurity G of the random forest. The value of G is used to determine the distribution of classes at each node in the random forest, meaning that proper weighting will address any class imbalance.

Similarly to the SVM models, all RF models had their performance assessed via the F1-score of the fraudulent class during training.

5.6 Quantum Computer Simulation

5.6.1 Additional Sampling

Quantum computing is still in its infancy in regards to mainstream accessibility and usage, with IBM appearing to be one of the only companies offering the ability to run jobs on utility scale quantum computers. That being said, their most limited plan comes at a cost of $96 per minute of runtime, which is outside the realm of feasibility for this project [28]. Thankfully, the open source Python package IBM maintains to handle all requests for their quantum computers (known as `qiskit`) provides the use of quantum simulators that can be implemented locally on classical computing devices [29]. Unfortunately, initial testing revealed that the dataset used for training and evaluation of the classical machine learning algorithms was much too large to be feasibly processed by the quantum computing simulators.

Because further direct sampling of the remaining 10,000 observations runs the risk of including too few (or possibly, no) fraudulent samples, the data was initially oversampled via the synthetic majority oversampling technique (SMOTE) [30] SMOTE generates synthetic samples for the minority class in an imbalanced dataset by interpolating between existing samples in the feature space. A mathe-

matical definition of this process is provided below:

Given the set of minority samples X_{minority} such that $X_{\text{minority}} \subset X$, synthetic samples x_s are created via the following steps:

1. For a given minority class sample $x_i \in X_{\text{minority}}$, find its k-nearest neighbors using a distance metric such as Euclidean distance:

$$d(x_i, x_j) = \sqrt{\sum_{l=1}^{d} (x_i^l - x_j^l)^2}, \quad x_j \in X \tag{8}$$

   where x_i^l and x_j^l are the l-th features of x_i and x_j, respectively.

2. Randomly choose one of the k-nearest neighbors, denoted as x_j.

3. Create a synthetic sample x_{s} by interpolating between x_i and x_j. The interpolation is controlled by a random number λ drawn uniformly from $[0, 1]$:

$$x_{\text{synthetic}} = x_i + \lambda \cdot (x_j - x_i) \tag{9}$$

The above process was applied to the training data and was repeated until the two classes were balanced (7,425 instances of each). 100 samples (50 from each class) were then randomly selected to produce a new training set that could be used by the quantum simulators implementing the QML algorithms. No synthetic data was added to the testing set, but its size was also reduced to only include 25 random samples of each class.

Though there was technically more than a sufficient number of fraudulent samples to create a balanced training dataset of this size, SMOTE was applied first to preserve the fact that these classes would be imbalanced in the real world. As such, it is improper to simply create a smaller, balanced training set via sampling the actual observations. The fact that the training data includes these synthetic samples is more representative of what would occur in a production model.

5.6.2 `ZZFeatureMap`

Quantum circuits consist of individual qubits (as opposed to bits) and the operations/connections that between them. The `ZZFeatureMap` is the `qiskit` class that

actually defines this quantum circuit and transforms classical data into quantum states [29]. Calling this method requires defining a number of key attributes of the circuit, such as the number of qubits, the entanglement pattern between the qubits, and how many times the sequence of operations in the circuit is repeated to encode the data (referred to as the number of repetitions). The result is a quantum circuit that encodes each feature $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ of a classical dataset into a quantum state $|\psi(\mathbf{x})\rangle$. All n qubits in the circuit start in the ground state ($|0\rangle\,|0\rangle\,...\,|0\rangle = |0\rangle^{\otimes n}$) , and are transformed into a final quantum state given by:

$$|\psi(x)\rangle = U_{\text{map}}(x)\,|0\rangle^{\otimes n} \tag{10}$$

where U_{map} is the transformation applied by the `ZZFeatureMap`. Equation 10 can be further broken down into the transformations applied via the qubit rotation U_{rotation} and entanglement pattern $U_{\text{entanglement}}$, over a series of R repetitions:

$$U_{\text{feature}}(x) = \prod_{r=1}^{R} \left(U_{\text{entanglement}}(x) \cdot U_{\text{rotation}}(x)\right) \tag{11}$$

The generalized formula for U_{rotation} shows how to apply the single-qubit Z-rotations based on the input data:

$$U_{\text{rotation}}(x) = \prod_{i=1}^{n} R_Z(x_i) \tag{12}$$

where:

$$R_Z(x_i) = \exp\left(-i\frac{x_i}{2}Z\right) \tag{13}$$

is the Z-axis rotation gate applied to the i-th qubit and Z is the Pauli-Z matrix given by:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{14}$$

Entanglements between qubits are produced in the quantum circuit via ZZ-interaction gates, with the ZZ interaction gate for a pair of qubits j and k being defined as:

$$ZZ(x_j, x_k) = \exp\left(-ix_j x_k Z_j \otimes Z_k\right) \tag{15}$$

where Z_j and Z_k are the Pauli-Z operators applied to qubits j and k, respectively. The full set of operations defined by the entanglement transformation $U_{\text{entanglement}}$ can then be defined as:

$$U_{\text{entanglement}} = \prod_{(j,k)\in E} ZZ(x_j, x_k) \tag{16}$$

where E is the set of qubit pairs to entangle, determined by the entanglement pattern.

Combining Equations 11, 12, and 16 gives:

$$|\psi(\mathbf{x})\rangle = \prod_{r=1}^{R} \left(\prod_{j=1}^{n-1} ZZ(x_j, x_k) \prod_{i=1}^{n} R_Z(x_i) \right) |0\rangle^{\otimes n} \tag{17}$$

Equation 17 represents the full transformation employed by a quantum circuit built via `ZZFeatureMap` to produce a quantum representation of classical input data. The aforementioned inputs of the `ZZFeatureMap` are defined in more detail below:

- **Number of qubits**: The number of qubits present in the circuit. This should be equal to the number of features used for training (a single feature is represented by a single qubit).
- **Entanglement**: The entanglement pattern that exists between the circuit. Linear entanglement means that each qubit is entangled only with its nearest neighbor in a sequential chain. Circular entangelment is the same as linear entanglement, with the addition of the first and last qubits also being entangled. Full entanglement means every qubit is entangled with every other qubit in the system.
- **reps**: The number of times the rotations and entanglement operations are repeated in the circuit.

5.6.3 `StateVectorSampler`

`qiskit`'s `StateVectorSampler` is a class used to actually simulate the result of applying the quantum circuit using classical hardware. More specifically, it uses state vectors to provide a mathematical representation of a quantum system using classical input data [29]. For example, Equation 18 shows the state vector representation

of a quantum state $|\psi(x)\rangle$ consisting of n qubits:

$$|\psi(x)\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle , \tag{18}$$

where:

- $|i\rangle$ are the computational basis states using traditional bits, such as $|000\rangle$, $|001\rangle$, ..., $|111\rangle$.
- $\alpha_i \in \mathbb{C}$ are complex amplitudes associated with each basis state.

Using this simulating method actually has some benefits, as these state vector representations are able to encode all the information within the quantum state, and avoid some of the possible noise that might be apparent when using an actual quantum computing system. Despite this, producing and running calculations with these state vectors is highly computationally complex.

5.6.4 `FidelityQuantumKernel`

While QML does contain the word "quantum," the implementation of the actual "machine-learning" algorithms they utilize has nothing to do with quantum mechanics. QML is actually achieved by utilizing classical ML methods using quantum representations of classical data. One way this is possible is by creating what's known as a "quantum kernel" that can be taken as input into these classical algorithms. These quantum kernels work the same way as a traditional kernel might when implementing a traditional SVM algorithm, the only difference being that the kernel matrix values are computed based on quantum state similarities (fidelities) in a quantum feature space, as opposed to similarities determined using classical feature transformations. Creating these quantum kernels is done using the `FidelityQuantumKernel` class, which takes as input a `ZZFeatureMap` to encode the classical data and a "fidelity" function that can be used to calculate the similarities between the quantum states of the data points (similar to a traditional kernel function). The fidelity function F used is this case was built via the `ComputeUncompute` class, which can calculate the fidelity between data point x_i - represented by quantum state $|\psi(x_i)\rangle$ - and data point x_j - represented by quantum state $|\psi(x_j)\rangle$ - via the following:

$$F(|\psi(x_i)\rangle , |\psi(x_i)\rangle) = |\langle\psi(x_i)|\psi(x_j)\rangle|^2 \tag{19}$$

Equation 19 essentially represents a quantum version of the kernel trick, enabling the data to be implicitly transformed into a higher-dimensional quantum feature space. The `ComputeUncompute` class function also invokes the previously mentioned `StateVectorSampler` as input, meaning that the quantum states determined via the quantum circuit are represented by classical state vectors, and that these state vectors are what was actually used when calculating the quantum kernel matrix. As such, the result of invoking an instance of the `FidelityQuantumKernel` class is newly formed quantum kernel that can be used produce a matrix that contains the fidelity between all pairs of data points represented in quantum feature space (the quantum kernel matrix). Note that a requirement of building these quantum kernels matrices is that the input data is scaled between $[0, \pi]$ (representing the angles of rotation of the qubits). In practice, a `MinMaxScaler` was used to apply this transformation.

5.7 Quantum Support Vector Machines (QSVM)

Utilizing SVMs was an easy choice to test QML's ability to solve fraud detection problems, as they can easily ingest the aforementioned quantum kernel matrices. In this case, once these matrices were built, they were fed directly into scikit-learn's `SVC` class to perform the fitting. This allows for the ability to tune two aspects of QSVMs:

- The parameters of the quantum circuits used to encode the data.
- Traditional SVM hyperparameter tuning.

The table below shows the shows the values of the different parameters that were used to test the quantum circuits:

Circuit Parameter	**Candidates**
`reps`	1, 2, 3
`entanglement`	linear, circular, full
`num_qubits`	2–10

Table 3: Quantum circuit candidates for the QSVM model.

The choices shown in Table 3 result in $3 \cdot 3 \cdot 10 = 90$ different circuits. Note that a maximum of only 10 qubits was used, which translates to a maximum of only 10 predictor variables used to evaluate the quantum circuits. While technically possible to use additional features by adding more qubits, the computational cost of doing

so proved too expensive for these more complex circuits. Thus, the permutation importance analysis presented in Figure 9 using a classical SVC model was used to pick the features included in the QRF models. As an example, a circuit with five qubits would utilize the five most important features.

To test the circuits, an "out-of-the-box" `SVC` model was used. Once the best quantum circuit was determined, traditional hyperparameter tuning was implemented using the same candidates listed in Table 1.

5.8 Quantum Random Forests (QRF)

Utilizing Random Forests as a baseline for testing QML's capability in solving fraud detection problems is a natural extension, particularly because they can also leverage quantum kernels. Random forests typically use the original feature data to make their predictions, but a kernelized random forest instead utilizes the similarity values from a kernel matrix as its input data. In this case, the same quantum kernel matrices resulting from the circuits described in Table 3 were used to test the predictions of a "out-of-the-box" `RandomForestClassifier`. For these models, features were chosen using the results of the classical RF feature importance analysis seen in Figure 10.

After determining the best performing quantum circuit, traditional hyperparameter tuning was conducted, which involved a grid-search over the same hyperparameter candidates seen in Table 2.

6 Results

6.1 Support Vector Machines

Table 4 shows an updated version of Table 1 that includes the values of the hyperparameters that resulted in the most successful SVM model:

Hyperparameter	**Candidates**	**Selection**
Kernel Type	Linear, RBF	RBF
C	0.1, 1, 10	10
γ	0.01, 0.1, 1	0.01

Table 4: Hyperparameters from the most successful SVM model after fitting.

This model was then used to make predictions on the test set, producing the following results:

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2475
1	0.73	0.88	0.80	25
accuracy			1.00	2500
macro avg	0.87	0.94	0.90	2500
weighted avg	1.00	1.00	1.00	2500

Figure 9 shows the estimated feature importances of the predictors used by the SVM model via their permutation importance scores.

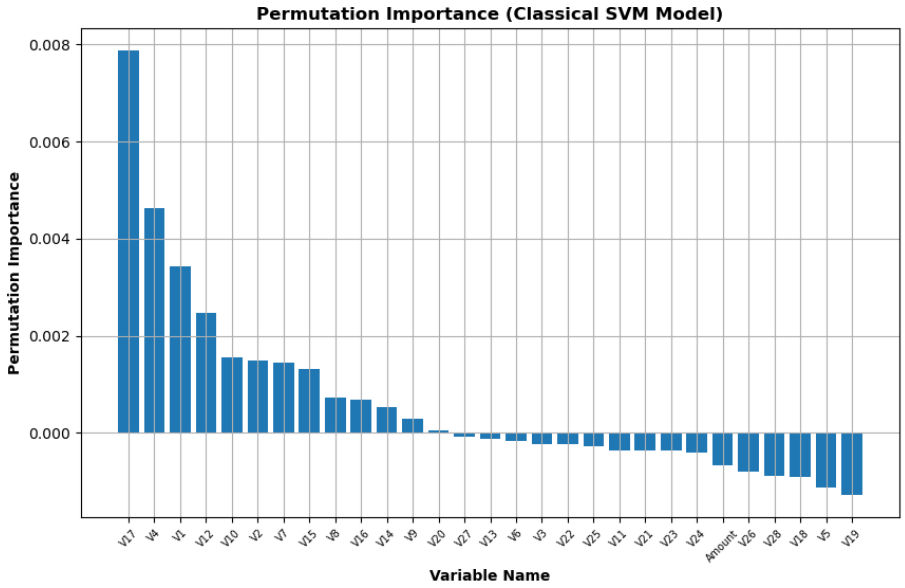

Figure 9: Permutation importance of all predictors using the best performing SVM model.

6.2 Random Forest

Table 5 shows an updated version of Table 2 that includes the values of the hyper-parameters that resulted in the most successful RF model.

Hyperparameter	Candidates	Selection
n_estimators	100, 200	100
max_depth	10, 20	10
min_samples_split	5, 10	5
min_samples_leaf	1, 2	1

Table 5: An updated version of Table 2 that shows the value of the hyperparameters from the most successful SVM model

This model was then used to make predictions on the test set, producing the following results.

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2475
           1       1.00      0.84      0.91        25

    accuracy                           1.00      2500
   macro avg       1.00      0.92      0.96      2500
weighted avg       1.00      1.00      1.00      2500
```

Figure 10 shows the estimated feature importances of the predictors used by the RF model:

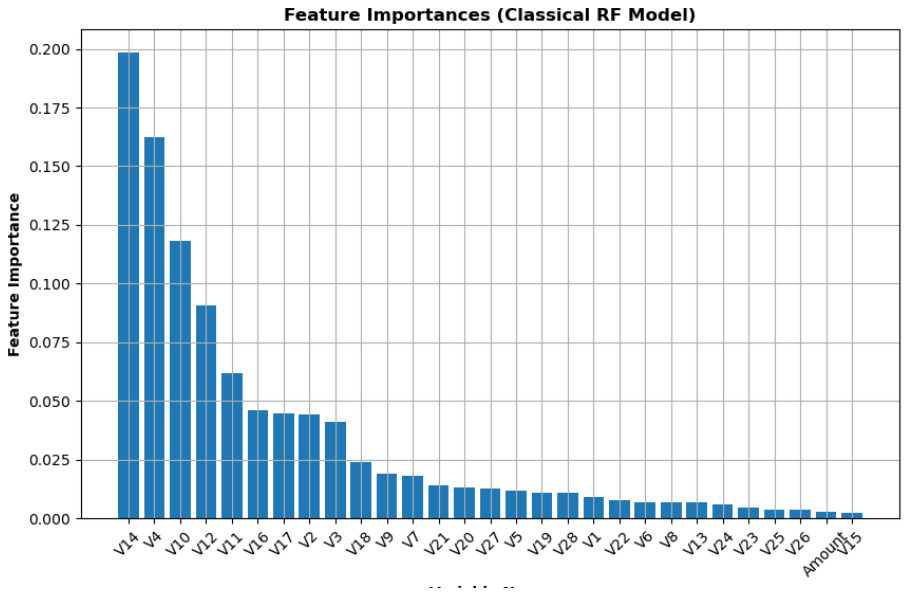

Figure 10: Feature importance of all predictors using the best performing RF model.

6.3 Quantum Kernel Matrix Build Time

Figure 11 shows a summary of how long it took to build the quantum kernel matrix using different quantum circuits.

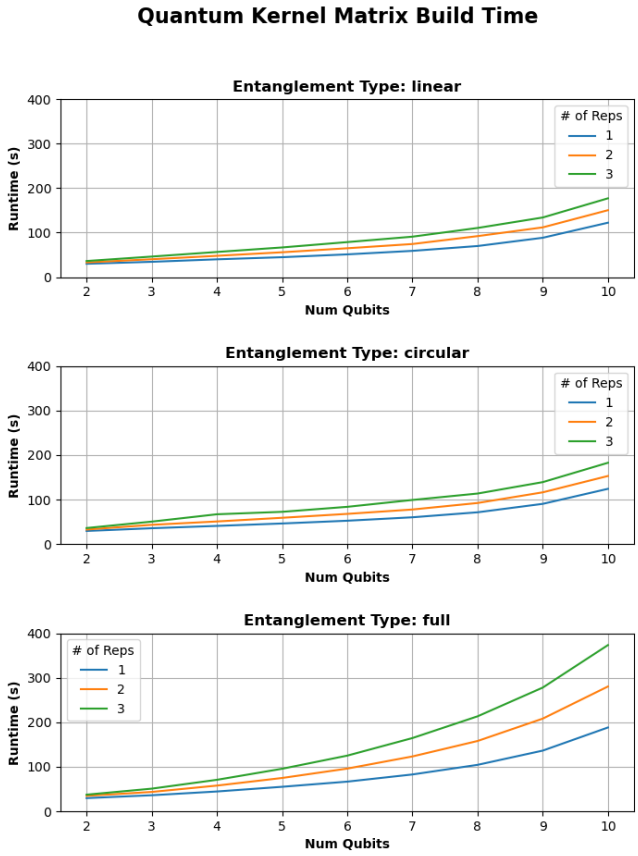

Figure 11: The time to build the quantum kernel matrix increases with increase quantum circuit complexity.

Further evaluation of these runtimes K revealed that they maintain a linear relationship with the number of repetitions R, and a non-linear second polynomial relationship with the number of qubits n:

$$K \propto R \cdot n^2 \tag{20}$$

Furthermore, while Figure 11 shows that proportionality defined in Equation 20 was observed across all entanglement types, models using full entanglement patterns exhibited longer runtimes compared to those with linear or circular structures

(especially as the number of repetition and qubits increases). To assist in modeling the effect of entanglement pattern on these runtimes, the total number of quantum gates or quantum operations G in each circuit was calculated. Determining G requires determining the number of operations stemming from both the quantum entanglement and rotation transformations:

- **For linear patterns**: $G = R(n-1+n) = R(2n-1)$.
  Explanation: $n-1$ gates from entanglement (gates exist between the first and second qubits, the second and third qubits, etc.), n rotation gates, multiplied by the number of repetitions.

- **For circular patterns:** $G = R(n+n) = 2Rn$.
  Explanation: Same as linear with an added entanglement gate between the first and last qubit.

- **For full patterns:** $G = R(\frac{n(n-1)}{2} + n) = R \cdot \frac{n^2+n}{2}$.
  Explanation: Same as circular but there are $\frac{n(n-1)}{2}$ entanglement gates (each qubit has a gate between all other qubits).

Based off the results shown in Figure 12, the relationship between the time it takes to build the quantum kernel matrix and the number of quantum gates follows is almost perfectly linear.

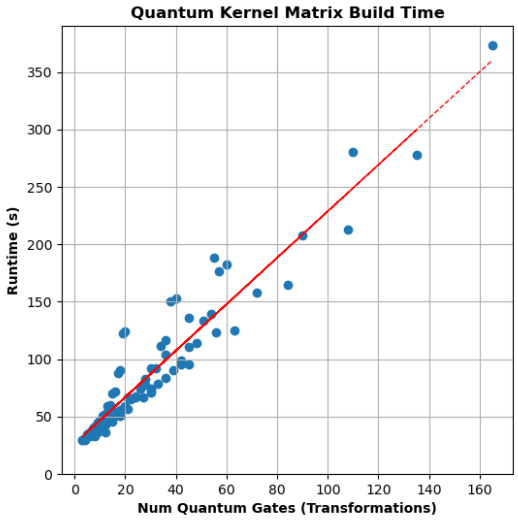

Figure 12: Based on the slope of the linear fit, each quantum circuit operation takes an average of 2.02s.

6.4 Quantum Support Vector Machines

Figure 13 shows how the F1 scores for the fraudulent class differ when using different quantum circuits to build the QSVM models.

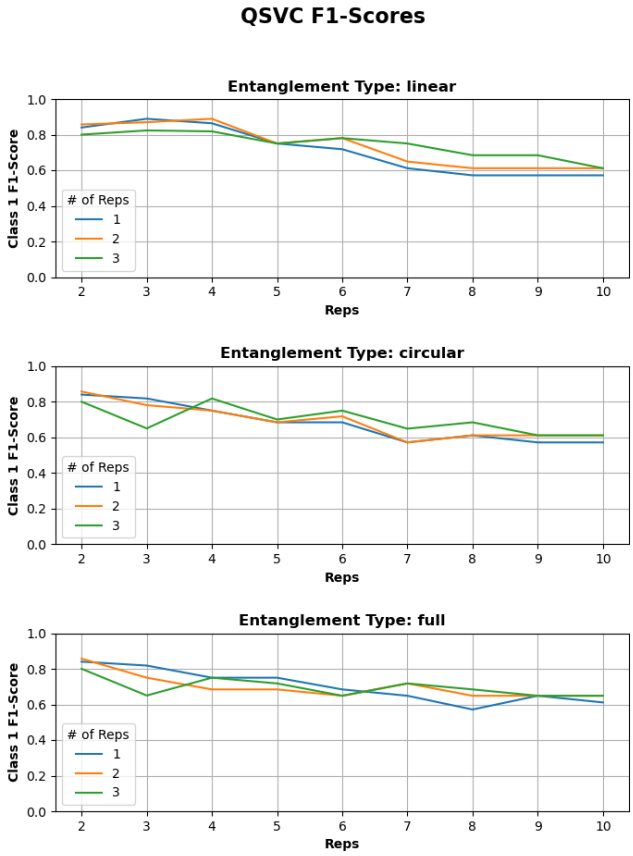

Figure 13: Testing data F1-scores achieved on the fraudulent class using QSVC as a function of the number of qubits, reps, and entanglement patterns.

Table 6 shows the quantum circuit parameters that resulted in the most effective "out-of-the-box" SVM model.

Circuit Parameter	Candidates	Selection
`reps`	1, 2, 3	1
`entanglement`	linear, circular, full	linear
`num_qubits`	2–10	3

Table 6: Quantum circuit candidates for the QSVM model.

The parameter choices in Table 6 were then used to a construct a quantum kernel

matrix in order to perform traditional hyperparameter tuning. These best values of each hyperparameter are shown in Table 7.

Hyperparameter	**Candidates**	**Selection**
Kernel Type	Linear, RBF	RBF
C	0.1, 1, 10	1
γ	0.01, 0.1, 1	0.01

Table 7: Testing data F1-scores for the fraudulent class achieved using QSVMs, as a function of the number of qubits, reps, and entanglement patterns.

These choices of quantum circuit parameters and traditional SVM hyperparameters were then used construct the "best" QSVM algorithm and make predictions on the test set. A classification report of these predictions is shown below:

	precision	recall	f1-score	support
0.0	0.83	1.00	0.91	25
1.0	1.00	0.80	0.89	25
accuracy			0.90	50
macro avg	0.92	0.90	0.90	50
weighted avg	0.92	0.90	0.90	50

6.5 Quantum Random Forests

Figure 13 shows how the F1 scores for the fraudulent class differ when using different quantum circuits to build the QRF models.

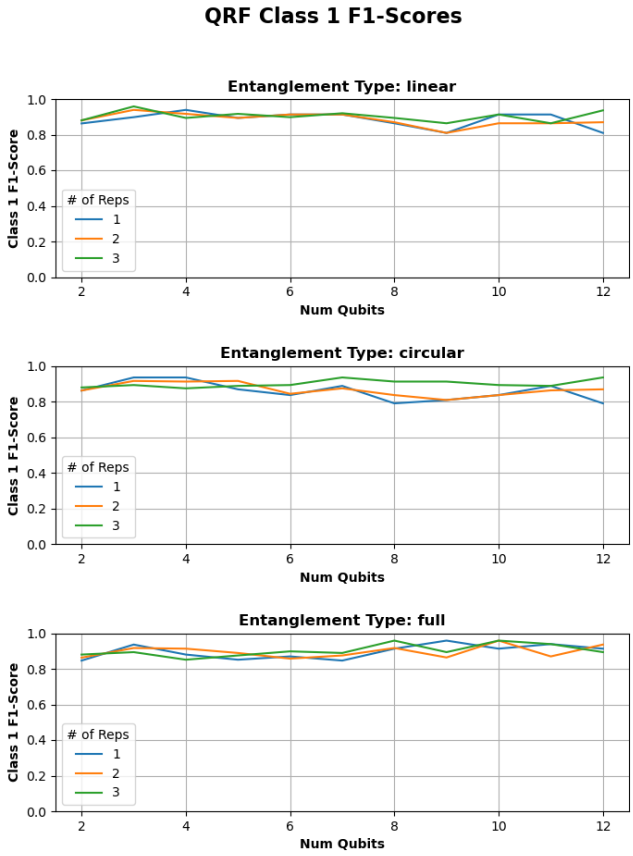

Figure 14: Testing data F1-scores for the fraudulent class achieved using QRFs, as a function of the number of qubits, reps, and entanglement patterns.

Table 8 shows the quantum circuit parameters that resulted in the most effective "out-of-the-box" RF model.

Circuit Parameter	Candidates	Selection
`reps`	1, 2, 3	3
`entanglement`	linear, circular, full	linear
`num_qubits`	2–10	3

Table 8: Quantum circuit candidates for the QRF model.

The parameter choices in Table 8 were then used to a construct a quantum kernel

matrix in order to perform traditional hyperparameter tuning. These best values of each hyperparameter are shown in Table 9.

Hyperparameter	Candidates	Selection
`n_estimators`	100, 200	100
`max_depth`	10, 20	10
`min_samples_split`	5, 10	10
`min_samples_leaf`	1, 2	1

Table 9: An updated version of Table 2 that shows the value of the hyperparameters from the most successful SVM model

These choices of quantum circuit parameters and traditional RF hyperparameters were then used construct the "best" QRF algorithm and make predictions on the test set. A classification report of these predictions is shown below:

	precision	recall	f1-score	support
0.0	0.92	0.96	0.94	25
1.0	0.96	0.92	0.94	25
accuracy			0.94	50
macro avg	0.94	0.94	0.94	50
weighted avg	0.94	0.94	0.94	50

7 Discussion

The results presented in the previous section reveal several key insights. While the best-tuned versions of the classical RF and QRF models performed well on the test set, their quantum counterparts (QSVM and QRF) achieved superior F1-scores for the fraudulent class, demonstrating their potential effectiveness for fraud detection. In practice however, implementing QML maintains a number of operational challenges: usage of actual quantum computers is expensive, and simulating their functionality on classical devices is highly computationally expensive. Figure 12 makes the case that the time to build quantum kernel matrices is a function of the number of quantum operations that take place within a circuit, meaning it is possible to estimate these runtimes these values before actually deciding to build the a quantum kernel matrix.

While the current complications might mean that we haven't yet reached a point where QML is feasible for quick results and/or large datasets, it is also worth noting

that the best versions of the quantum models in this case utilized far fewer features than the traditional SVM and RF models. The most successful QML models utilized circuits with only three qubits (features) and significantly fewer observations, showcasing the ability of quantum representations of classical data to capture complex patterns using fewer input features. In fact, the QSVM models seemed actually appeared to suffer as the quantum circuits they utilized increased in complexity. This same behavior was not observed when evaluating the QRF models, which seemed to maintain consistent F1-scores across the varying quantum circuits. One possible explanation for this is that exceedingly complex quantum circuits can result in quantum kernel degeneracy. An example of this is shown in Figure 15, which plots image representations of the quantum kernel matrices resulting from quantum circuits with different numbers of qubits.

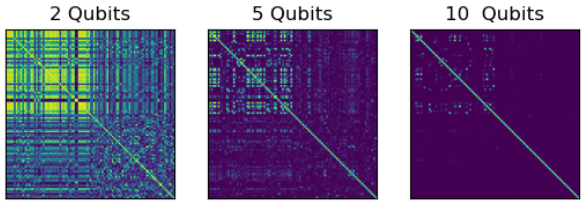

Figure 15: As the number of qubits increases, many of the values within the quantum kernel matrices contain similar or identical values, a hallmark aspect of kernel degeneracy

Because RFs rely on localized decision-making, they are more robust to degenerate kernels, especially when compared to SVM models that rather than requiring the kernel to define a global decision boundary. This allows RFs to still extract useful information from relative similarities in the data, even if the kernel lacks diversity or precision.

Overall, the QRF algorithm emerged as the most successful model for detecting credit card fraud using the data provided. This is especially exciting given that

they even outperformed the QSVMs models, which have already shown promise in the relam of fraud detection. These findings validate many aspects of the initial hypothesis.

8 Conclusion/Next Steps

This research demonstrates the potential of QML, specifically QSVM and QRF models, for enhancing fraud detection capabilities. Despite the promising results, several challenges remain before true QML can be implemented at scale, and the reliance on quantum simulators imposes significant computational costs and time constraints. However, advancements in quantum computing are progressing rapidly, meaning that it might soon be feasible to attempt to recreate these results using actual quantum computers. Additionally, it would be interesting to see if other QML models (such as Quantum Neural Networks) can build on the already impressive performance witnessed by QSVMs and QRFs.

9 References

[1] E. Farhi, and S. Gutmann, "Quantum computation and decision trees," *Physical Review A*, **58**(2):915–928, 1998.

[2] K. Khadiev, I. Mannapov, and L. Safina, "The Quantum Version Of Classification Decision Tree Constructing Algorithm C5.0," //*arXiv* preprint arXiv:1907.06840, 2019.

[3] K. Khadiev, and L. Safina, "The Quantum Version of Prediction for Binary Classification Problem by Ensemble Methods," //*arXiv* preprint arXiv:2112.13346, 2021.

[4] M. Srikumar, C.D. Hill, and Lloyd, "A kernel-based quantum random forest for improved classification," //*arXiv* preprint arXiv:2210.02355, 2022.

[5] D. Anguita, S. Ridella, F. Rivieccio, and R. Zunino, "Quantum optimization for training support vector machines," *Neural Networks*, **16**(5-6):763–770, 2003.

[6] Rebentrost, Patrick, et al. "Quantum Support Vector Machine for Big Data Classification." *Physical Review Letters*, **113**(13):25, 2014

[7] R. Chatterjee, and T. Yu, "Generalized Coherent States, Reproducing Kernels, and Quantum Support Vector Machines," *Quantum Information & Computation*, **17**(15-16), 2017.

[8] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, **549**(7671):195–202, 2017.

[9] A. Zeguendry, Z. Jarir, and M. Quafafou, "Quantum Machine Learning: A Review and Case Studies," *Entropy*, **25**(2):287, 2023.

[10] S. Zheng, J. Guo, X. Ding, X. Zhou, J. Wang, H. Lian, Y. Gao, B. Zhao, and J. Xu, "Demonstration of Breast Cancer Detection Using QSVM on IBM Quantum Processors," //*Research Square* preprint researchsquare:10.21203, 2022.

[11] Z.J. Luo, T. Stewart, M. Narasareddygari, R. Duan, and S. Zhao, "Quantum Machine Learning: Performance and Security Implications in Real-World Applications," //*arXiv* preprint arXiv:2408.04543, 2024.

[12] E.A. Cherrat, S. Raj, I. Kerenidis, A. Shekhar, B. Wood, J. Dee, S. Chakrabarti, R. Chen, D. Herman, S. Hu, P. Minssen, R. Shaydulin, Y. Sun, R. Yalovetzky, and M. Pistoia, "Quantum Deep Hedging," *Quantum* 7:1191, 2023.

[13] T. Nguyen, T. Sipola, and J. Hautamäki, "Machine Learning Applications of Quantum Computing: A Review," *European Conference on Cyber Warfare and Security*, **23**(1):322–330, 2024.

[14] N. Innan, M.A.-Z. Khan, and M. Bennai, "Financial Fraud Detection: A Comparative Study of Quantum Machine Learning Models," *International Journal of Quantum Information* **22**(2):2350044, 2024.

[15] O. Kyriienko, and E.B. Magnusson, "Unsupervised quantum machine learning for fraud detection," *//arXiv* preprint arXiv:2208.01203, 2022.

[16] M. Swayne, "Deloitte Italy Explores Quantum Machine Learning for Digital Payments Fraud Detection," *The Quantum Insider*, 2024.

[17] Nidula Elgiriyewithana, "Credit Card Fraud Detection Dataset 2023," www.kaggle.com, 2023.

[18] European Banking Authority, "2024 Report on Payment Fraud", 2024.

[19] J.W. Tukey, "Exploratory Data Analysis," (Addison-Wesley Pub. Co, Reading, Mass., 1977).

[20] C. Hastings, F. Mosteller, J.W. Tukey, and C.P. Winsor, "Low Moments for Small Samples: A Comparative Study of Order Statistics," *The Annals of Mathematical Statistics*, **18**(3):413–426, 1947.

[21] P. Virtanen, et. al., "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature Methods*, **17**(3):261–272, 2020.

[22] D'Agostino, R. B., "An omnibus test of normality for moderate and large sample size", *Biometrika*, 58:341-348, 1971.

[23] D'Agostino, R. and Pearson, E. S., "Tests for departure from normality", *Biometrika*, 60:613-622, 1973.

[24] I.-K. . Yeo, "A new family of power transformations to improve normality or symmetry," *Biometrika* **87**(4):954–959, 2000.

[25] Student, "The Probable Error of a Mean," *Biometrika*, **6**(1):1, (1908).

[26] Vladimir Naoumovitch Vapnik, "The Nature of Statistical Learning Theory," (Springer, Cop, New York, 2000).

[27] L. Breiman, "Random Forests," Machine Learning 45(1), 5–32 (2001).

[28] "IBM Quantum Computing — Pricing," www.ibm.com.

[29] A. Javadi-Abhari, et. al., "Quantum computing with Qiskit," *//arXiv* preprint arXiv:2405.08810, 2024.

[30] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, **16**(16):321–357, 2002.