Quantum-Enhanced Support Vector Machines: Engineering Quantum Kernels and the Kernel Trick

2589802, 19 May 2025

I. Introduction

HIGH-dimensional datasets often cause classical Support Vector Machines (SVM) to overfit and incur high computational cost. Quantum kernels exploit superposition and entanglement to embed data into exponentially large Hilbert spaces, enabling richer similarity measures [1]. In this lab, a Quantum Support Vector Machine (QSVM) is implemented by first reducing a binary classification dataset to 2-4 principal components, then encoding each point with Qiskit's ZZFeatureMap and training via the QuantumKernel and QSVC primitives on both simulators and IBM quantum hardware. Test accuracy, support vector count, and runtimes are benchmarked against a classic Radial Basis Function (RBF)-kernel SVM, visualise the quantum kernel matrix as a heatmap, and critically assess the practical advantages and challenges of quantum-kernel methods on near-term devices.

II. Theoretical Background

QSVMs extend classical SVMs by encoding data points into quantum states and using quantum circuits to compute inner products in a high-dimensional Hilbert space, enabling richer similarity measures than classical kernels [1]. By leveraging the "quantum kernel trick," QSVMs can in principle compute certain kernel entries exponentially faster than any known classical algorithm for specific datasets [2]. Training a QSVM reduces to solving the usual convex quadratic optimisation in the dual formulation, substituting classical kernel evaluations with quantum kernel estimates. In practice, QSVMs on near-term devices face challenges such as noise, finite sampling overhead, and careful feature-map design which cause issues in achieving reliable results [1].

A. The Kernal Trick

The kernel trick allows algorithms to operate in high-dimensional feature spaces by computing inner products through kernel functions, such as $k(x, y) = \langle\varphi(x), \varphi(y)\rangle$, without ever explicitly mapping data into those spaces [2]. In support vector machines, this means replacing every standard dot product with a kernel evaluation $k(x, y)$, enabling a linear classifier to separate data that are not linearly separable in the original input space [3]. Common choices include the linear kernel $k(x, y) = x \cdot y$, the polynomial kernel $k(x, y) = (\gamma\, x \cdot y + r)^d$, and the radial basis function kernel $k(x, y) = \exp(-\gamma\|x - y\|^2)$, each corresponding to a different implicit feature mapping [2]. By sidestepping the explicit computation of high-dimensional vectors, the kernel trick delivers both computational efficiency and flexibility, allowing classifiers like SVMs to capture complex, nonlinear patterns in data with only pairwise similarity computations [4].

B. Classical Support Vector Machines

A Support Vector Machine (SVM) is a supervised method that finds a "maximum-margin" hyperplane separating two classes in feature space, maximising the distance (margin) to the nearest training points [3]. Those nearest points, called support vectors, fully determine the position of this hyperplane [3]; if any support vector were removed, the solution would change [3]. When data are not linearly separable, the kernel trick replaces dot-products with a kernel function $k(x, y)$, enabling separation in a higher-dimensional (possibly infinite) feature space without explicit mapping [3].

C. Quantum Support Vector Machine (QSVM)

Quantum support vector machines (QSVMs) are hybrid classifiers that replace the classical kernel function with a quantum-computed kernel, enabling potentially richer similarity measures in exponentially large Hilbert spaces [1]. Classical feature vectors $\mathbf{x}$ are encoded into parameterised quantum states $|\phi(\mathbf{x})\rangle$ via a quantum feature map, and the kernel matrix is estimated as

$$K_{ij} = \left|\langle\phi(\mathbf{x}_i) \mid \phi(\mathbf{x}_j)\rangle\right|^2 \tag{1}$$

by measuring state fidelities on a quantum device or simulator [1]. A standard SVM solver then finds the maximum-margin hyperplane using this quantum kernel, leaving the rest of the training and inference workflow unchanged [1]. High-level libraries like Qiskit's 'QSVC' primitive abstract away the underlying circuits, making QSVMs accessible even to users with minimal quantum background and allowing seamless execution on both simulators and real hardware [1].

III. Implementation and Experimental Setup

The entire workflow was implemented in Python3, using `pandas` and `scikit-learn` for data handling and pre-processing, and Qiskit together with Qiskit Machine Learning for the quantum-enhanced support vector machine (QSVM). All experiments were executed within a Jupyter notebook; figures and kernel heatmaps were saved under a dedicated `Heatmaps/` directory for inclusion in this report.

A. Choice of Dataset

The "Users vs Bots Classification" dataset was selected as it matches the requirements of using a binary classification problem from a real-world source [5]. It contains user-profile features (e.g. number of friends, posts, likes) gathered from VK.com (Russia's largest social network) which provides a mix of numerical and categorical attributes that require the preprocessing, imputation, normalisation, and one-hot encoding steps one must implement [5]. Moreover, at approximately 112 KB (one CSV file) and a few thousand records, its size is large enough to demonstrate how PCA reduction (to 2–4 components) and both quantum and classical SVMs behave as one varies sample size, yet small enough to keep runtimes under the lab's target of $\leq$ 100 samples for quick iterations.

B. Data Loading, Cleaning and Dimensionality Reduction

Initially `bots_vs_users.csv` is loaded into a `pandas` DataFrame. Columns with more than 75% missing values are dropped. Remaining numeric features have NaNs filled with 0, and for each a "`_was_na`" indicator is appended. Boolean strings are mapped to 1/0, other categoricals one-hot encoded (dropping one level), and exact duplicates removed. A `VarianceThreshold` (threshold=0.01) then eliminates very low-variance features. The cleaned feature set is reconstructed, target labels are reattached, any residual missing values are imputed via `SimpleImputer` (mean), and standardise all features to zero mean and unit variance using `StandardScaler`.

Each PCA dimension $n \in \{2, \dots, 8\}$, is applied to the project scaled data then into an n-dimensional subspace. A stratified subsample of size $m \in \{10, 100, 1000, 3000\}$ (to preserve class balance) is then drawn, and split each subsample into an 80% training fold and a 20% test fold for downstream QSVC and SVM experiments.

C. Quantum Machine Learning Model (QSVM)

For each PCA-reduced dataset, a two-repetition `ZZFeatureMap` of dimension n embeds each training vector into a quantum state. The `FidelityStatevectorKernel` evaluates the pairwise overlaps on a statevector simulator. Qiskit's `QSVC` is then invoked on this quantum kernel to solve the SVM dual problem. Model fitting and scoring are performed on an 80% train / 20% test split of a stratified subsample (sizes of 10, 100, 1000, or 3000), and test-set accuracy is recorded. The resulting kernel matrix for each run is visualised as a heatmap (Matplotlib's `imshow` with "viridis") and saved for qualitative analysis.

D. Classical Machine Learning Method

A classical support vector machine is trained in parallel as a baseline. Using scikit-learn's `SVC(kernel='rbf', gamma='scale')`, the same train/test splits and PCA-reduced inputs are employed. After fitting, test-set accuracy is obtained via the classifier's `score` method. This direct comparison allows assessment of any accuracy gains or computational trade-offs introduced by the quantum kernel approach.

E. Quantum Hardware Implementation

The quantum kernel is evaluated on real IBM hardware using Qiskit's `QiskitRuntimeService` and the `Sampler` primitive. A `ZZFeatureMap` circuit is constructed to encode each PCA-reduced sample into a parameterised quantum state, then build fidelity circuits that prepare $|\phi(x)\rangle$, un-prepare $|\phi(y)\rangle$, and measure the probability of the all-zero outcome. Pairwise fidelities are collected into a Gram matrix via `compute_kernel_matrix`, which is then passed to scikit-learn's `SVC(kernel='precomputed')` for training and evaluation. This workflow mirrors the classical baseline while leveraging hardware-calibrated quantum kernels to explore potential accuracy improvements and runtime trade-offs. Due to hardware 10 minute time restriction, the program could only run 128 shots with a PCA dimension 3 and 80 sample model with the `ZZFeatureMap` reps set to 1. All simulations were done on ibm_sherbrooke.

IV. Results

TABLE I: Classical and Quantum SVM test accuracies (in %) for varying PCA dimensions (n) and sample sizes (m)

PCA (n)	Classical Accuracy				Quantum Accuracy			
	10	100	1000	3000	10	100	1000	3000
2	100	85	90	90	100	80	78	78
3	100	90	91	90	100	80	85	84
4	100	90	91	92	100	85	84	86
5	100	90	91	91	100	85	86	88
6	100	85	91	92	100	85	87	90
7	100	90	93	93	100	85	88	90
8	100	95	94	94	100	85	88	91

Table I summarises classical-RBF and quantum-kernel SVM test accuracies for PCA dims $n \in \{2, \dots, 8\}$ and training sizes $m \in \{10, 100, 1000, 3000\}$. Several trends stand out:

- Overfitting at $m = 10$: both models score 100%, memorising the tiny dataset.
- Accuracy drop at $m = 100$: learning true decision boundaries causes classical accuracy to fall to 85-95% and quantum (noisy) to 80–85%.
- Recovery with more data: at $m = 1000$ and 3000, classical stabilises around 90-94% and quantum climbs from 78% to 91%, reflecting reduced variance.
- Dimensionality trade-off: increasing n hurts both methods when m is small (curse of dimensionality) but, for $m = 3000$, higher n (up to 8) captures more variance and boosts quantum accuracy most sharply (to $\tilde{9}1\%$), with classical gains more modest.

In summary, both classical and quantum SVMs require a balance between dimensionality and sample size. Too few points in a high-dimensional PCA subspace leads to under-sampling and poor generalisation, whereas too few components (over-aggressive PCA) can discard discriminative variance. Not shown in Table 1 was the hardware result which had a 81.25% accuracy showing solid results considering quantum noise. With sufficient data ($m \geq 1000$) and a moderate number of components ($n \leq 8$), the quantum kernel approaches the performance of the classical RBF SVM, demonstrating its viability for larger datasets and richer feature representations.

A. Model Training Times

TABLE II: Average train times (s) for QSVC vs classical SVM over all PCA dimensions

Sample size m	QSVC train time (s)	Classical SVM train time (s)
10	0.07	0.00
100	0.71	0.00
1000	10.05	0.01
3000	53.73	0.04

Table II shows the mean training time of our quantum-kernel SVM (QSVC) and the classical RBF SVM across all PCA dimensions for each training set size m. The QSVC's training time grows steeply with m, from only ≈ 0.07 s at $m = 10$ up to ≈ 53.7 s at $m = 3000$. This is because QSVC's 'fit' requires computing an $m \times m$ Gram matrix by running $O(m^2)$ fidelity circuits (each deeper when the number of PCA components is increased), plus the associated transpilation overhead. By contrast, the classical SVM remains extremely fast (sub-0.05 s even at $m = 3000$) thanks to highly optimised C-based solvers and efficient RBF kernel computations. Not shown in Table 2 is the time taken for hardware at 143 seconds taken a significant amount of time. These results underscore that, while quantum-kernel methods show promise in accuracy, they currently incur substantially higher computational cost, which will need to be addressed by future improvements in circuit compilation and sampling primitives.

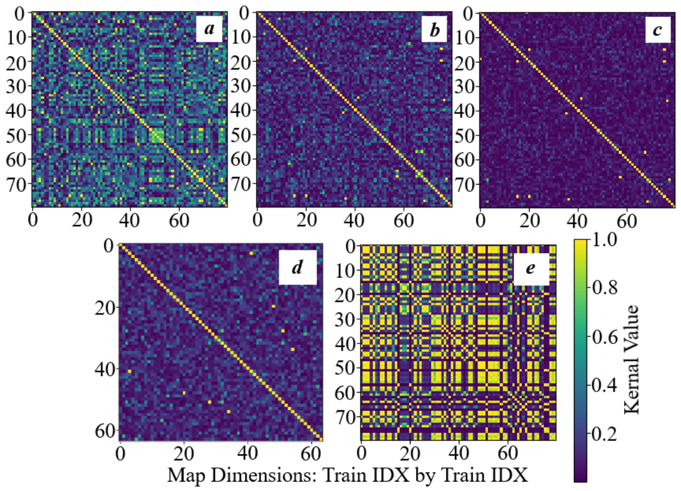

Fig. 1: Kernel-matrix heatmaps for QSVM (a–c: simulator, $m = 100$, $n = 2, 3, 4$; d: hardware, $m = 80$, $n = 3$) and classical RBF SVM (e: $m = 100$, $n = 3$)

B. Kernel-Matrix Heatmaps

Fig. 1 presents five kernel matrices K ($K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$) for various settings. In the quantum cases (a–c,e), k is the state-fidelity $P(0 \dots 0)$ after preparing $\phi(\mathbf{x}_i)$ and un-preparing $\phi(\mathbf{x}_j)$; in the classical case (d) it is the RBF kernel. Both axes ("Train IDX" $0 \dots m-1$) feature a diagonal indicating perfect self-similarity. For Fig. 1 (a)–(c) QVM, $m = 100$:

- Fig. 1a $n = 2$: High, noisy off-diagonals reflect poor separation in 2D.
- Fig. 1b $n = 3$: A sharper diagonal and lower off-diagonals show improved discrimination.
- Fig. 1c $n = 4$: Additional variance raises some off-diagonal values, separation remains better than $n = 2$.

Fig. 1d Classical SVM, $m = 100$, PCA $n = 3$ has a smooth, block-patterned heatmap characteristic of the continuous Gaussian kernel. Fig. 1e Quantum hardware, $m = 80$, PCA $n = 3$ retains the diagonal under noise, but exhibits a grainier background due to finite shots and device errors.

Across all maps, the contrast between diagonal and off-diagonals quantifies kernel effectiveness. The simulator sequence (a→c) confirms that moderate PCA dimensionality enhances quantum discrimination, while (d) and (e) illustrate how the classical Gaussian kernel and noisy hardware fidelities differ in their similarity landscapes.

V. Discussion and Future Work

The QSVM matched classical SVM accuracy for large training sets ($m \geq 1000$) and produced richer kernel structures, highlighting its potential to capture complex correlations. However, NISQ-induced noise, $O(m^2)$ circuit costs, and limited qubit counts constrain its practicality. To advance QSVMs, future work should focus on:

- *Error mitigation*: apply readout calibration and zero-noise extrapolation to reduce hardware noise.
- *Adaptive feature maps*: use trainable or data-reuploading circuits to lower depth while retaining expressivity.
- *Scaling benchmarks*: extend to multi-class tasks and larger datasets to rigorously assess quantum advantage.

VI. Conclusion

This lab benchmarked quantum-kernel and classical RBF SVMs on a real-world binary classification task, exploring PCA dimensions (2–8) and sample sizes (10–3000). The QSVM matched classical accuracy for $m \geq 1000$ and revealed richer feature discrimination in kernel heatmaps, but incurred substantially higher $O(m^2)$ training overhead and sensitivity to hardware noise. Future work should target error mitigation, kernel-approximation methods, and circuit-compilation optimisations to reduce runtime costs and fully leverage QSVM's expressive power on near-term quantum devices.

References

[1] Veloso, T. (2022). *Quantum Support Vector Machine (QSVM) - MIT 6.s089 — Intro to Quantum Computing - Medium*. [online] Medium. Available at: https://medium.com/mit-6-s089-intro-to-quantum-computing/quantum-support-vector-machine-qsvm-134eff6c9d3b [Accessed 18 May 2025].

[2] Amira, V.Y. (2025). *Understanding the Power of Kernel Trick in Machine Learning Algorithms*. [online] Medium. Available at: https://medium.com/@vaishnaviyada/understanding-the-power-of-kernel-trick-in-machine-learning-algorithms-b7ed9ad48e11 [Accessed 18 May 2025].

[3] IBM (2023). *Support Vector Machine*. [online] IBM. Available at: https://www.ibm.com/think/topics/support-vector-machine [Accessed 18 May 2025].

[4] Iyer, R. (2024). *Demystifying Support Vector Machines: Kernel Machines*. [online] mldemystified.com. Available at: https://mldemystified.com/posts/basics-of-ml/support-vector-machines/svm-kernel-machines/ [Accessed 18 May 2025].

[5] Zagorskii, A. (2025). *Users vs bots classification*. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets/juice0lover/users-vs-bots-classification [Accessed 19 May 2025].