IOP Publishing Mach. Learn.: Sci. Technol. 6 (2025) 015047 https://doi.org/10.1088/2632-2153/adb4ba

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
12 September 2024

REVISED
20 January 2025

ACCEPTED FOR PUBLICATION
11 February 2025

PUBLISHED
20 February 2025

PAPER

Validating large-scale quantum machine learning: efficient simulation of quantum support vector machines using tensor networks

Kuan-Cheng Chen[1,2,8,*], **Tai-Yue Li**[3,7,8], **Yun-Yuan Wang**[4,8], **Simon See**[5], **Chun-Chieh Wang**[3], **Robert Wille**[6], **Nan-Yow Chen**[7], **An-Cheng Yang**[7] and **Chun-Yu Lin**[7]

[1] Department of Electrical and Electronic Engineering, Imperial College London, London, United Kingdom
[2] QuEST, Imperial College London, London, United Kingdom
[3] National Synchrotron Radiation Research Center, Hsinchu, Taiwan
[4] NVIDIA AI Technology Center, NVIDIA Corp., Taipei, Taiwan
[5] NVIDIA AI Technology Center, NVIDIA Corp., Singapore, Singapore
[6] Chair of Design Automation, Technical University of Munich, Munich, Germany
[7] National Center for HPC, Narlabs, Hsinchu, Taiwan
[8] The first three authors contributed equally to this work.
[*] Author to whom any correspondence should be addressed.

E-mail: kuan-cheng.chen17@imperial.ac.uk

Keywords: quantum machine learning, quantum kernel estimation, quantum circuit simulation, tensor network, cuquantum SDK

Abstract

We present an efficient tensor-network-based approach for simulating large-scale quantum circuits exemplified by quantum support vector machines (QSVMs). Experimentally, leveraging the cuTensorNet library on multiple GPUs, our method effectively reduces the exponential runtime growth to near-quadratic scaling with respect to the number of qubits in practical scenarios. Traditional state-vector simulations become computationally infeasible beyond approximately 50 qubits; in contrast, our simulator successfully handles QSVMs with up to 784 qubits, executing simulations within seconds on a single high-performance GPU. Furthermore, utilizing the message passing interface for multi-GPU environments, our method demonstrates strong linear scalability, effectively decreasing computation time as dataset sizes increase. We validate our framework using the MNIST and Fashion MNIST datasets, achieving successful multiclass classification and highlighting the potential of QSVMs for high-dimensional data analysis. By integrating tensor-network techniques with advanced high-performance computing resources, this work demonstrates both the feasibility and scalability of simulating large-qubit quantum machine learning models, providing a valuable validation tool within the emerging Quantum-HPC ecosystem.

1. Introduction

In the rapidly evolving landscape of artificial intelligence (AI), machine learning algorithms stand out as pivotal components driving advancements across a multitude of domains [1]. These algorithms, distinguished into supervised and unsupervised learning paradigms, harness the power of data to uncover patterns or make predictions [2]. Supervised learning, in particular, leverages pre-labeled data to train models, with the support vector machine (SVM) being a cornerstone technique in this category [3]. SVMs excel in classifying data into distinct categories by finding an optimal hyperplane in either the original or a higher-dimensional feature space [4]. However, the computational demands of SVMs, especially in the context of large-scale 'big data' applications [5], pose significant challenges in terms of both computational resources and execution time.

Enter the realm of quantum computing, a burgeoning field offering profound computational speedups over classical approaches for certain problem types. Among these, quantum support vector machines

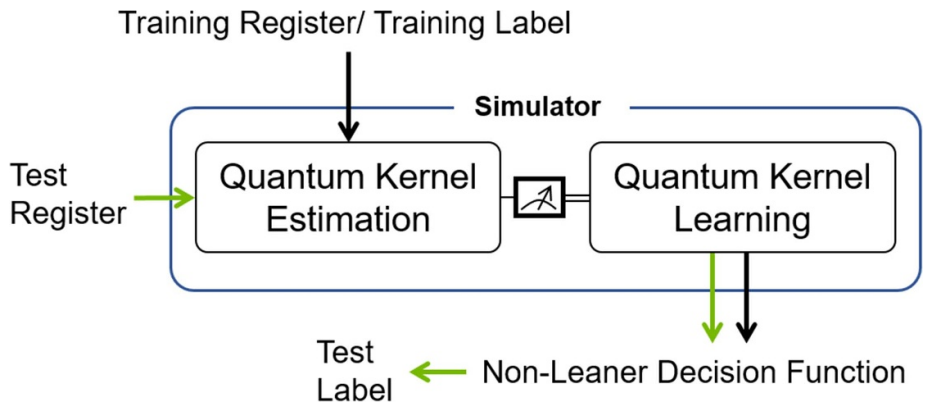

Figure 1. QSVM Simulator: Optimizes quantum kernel estimation and learning, enhancing phase operation and objective evaluation, leading to swift and precise classification outcomes.

(QSVMs) emerge as a promising quantum-enhanced technique for machine learning [6–8], capable of drastically reducing the computational resources required for SVMs. Leveraging quantum algorithms, QSVMs achieve exponential speedups in both training and classification tasks by performing calculations in parallel and employing quantum-specific optimizations [6, 9–11].

However, in the current noisy intermediate-scale quantum (NISQ) era [12], the practical utility of quantum computers is significantly constrained by their availability and imperfect technological state. Challenges such as the fidelity of qubits, the error rates of two-qubit gates, and the limited number of available qubits present substantial hurdles [13–15]. Despite the advent of several methodologies aimed at enhancing qubit fidelity-such as quantum error mitigation [16, 17] and Dynamical Decoupling [18]-these limitations persist, impeding the realization of quantum advantage on quantum computing platforms in the current NISQ era [19, 20]. Consequently, the design and validation of quantum-inspired algorithms, or hybrid classical-quantum algorithms, are predominantly conducted through high-performance classical simulations [11, 21]. Furthermore, quantum circuit simulators have shown considerable success in the near-term verification of quantum algorithms on small qubit systems [22, 23].

Within the scope of our research, we have engineered an advanced tensor-network simulation framework, purpose-built to expedite the development of QSVMs through the integration of the cuTensorNet library underlying cuQuantum SDK [24]. This library is meticulously optimized for NVIDIA GPUs and can facilitate QSVM algorithms, requiring noiseless simulations for quantum kernel estimation as depicted in figure 1. A pre-computation mechanism is embedded within this workflow, allowing for the reuse of an optimized tensor-network contraction path in the QSVM's complex learning stages, thereby bolstering the efficacy of both the training and classification phases.

Our tensor-network-based simulation is designed for parallel execution using the Message Passing Interface (MPI) and leverages the substantial computational power of GPU acceleration. This combination enables our QSVM simulator to efficiently manage large datasets while only modestly increasing memory requirements, thereby avoiding out-of-memory situations during large-scale quantum circuit simulations. Its flexibility ensures its utility across various quantum machine learning paradigms. Benchmark results show that our simulator achieves speedups often exceeding an order of magnitude compared with existing methods [25, 26], thereby underscoring its potential as a robust and scalable tool for quantum machine learning within the broader Quantum-High-Performance Computing (HPC) ecosystem [21, 24].

A key feature of our simulator is its capacity to handle up to 784 qubits, enabling an extensive scaling analysis of QSVM performance and shedding light on the potential of quantum kernel methods in realistic data classification scenarios. Furthermore, this approach is flexible enough to accommodate various quantum machine learning paradigms and can be extended to multi-GPU settings for large-scale simulations. By validating our methods on real-world datasets (such as MNIST and Fashion-MNIST), we demonstrate that QSVMs can tackle complex classification tasks in Quantum-HPC environments, marking a significant step toward practical quantum-enhanced machine learning. These strides in QSVM development signal a major progression towards practical deployment, charting a path for the application of quantum-enhanced methodologies to complex, real-world data classification challenges within the Quantum-HPC Ecosystem [11, 21, 27–29].

These results highlight not only the viability of QSVM algorithms but also the value of advanced simulation tools in guiding future quantum hardware development, such as offering ground truth for

benchmarking purpose. As such, this work contributes to bridging the gap between theoretical QSVM formulations and their eventual implementation on large-scale quantum devices, offering valuable insights for both algorithmic refinement and hardware optimization in the quantum information sciences.

2. Background

QSVMs represent a significant breakthrough in quantum machine learning, particularly for large-scale data classification. The pioneering work by Rebentrost *et al* [6] introduced a quantum algorithm that substantially enhances the computational efficiency of traditional SVMs. By harnessing quantum-mechanical principles such as superposition and interference, QSVMs can, under certain assumptions (e.g. quantum random-access memory, qRAM), achieve near-logarithmic complexity with respect to both the dimensionality of feature vectors N (qubit number) and the size of the training dataset M (data size). This approach suggests a potential exponential speedup over classical methods, although practical constraints like the realization of qRAM remain a major challenge.

More recent work on quantum machine learning has shifted toward quantum kernel estimation, emphasizing the capability of entangled quantum states to embed classical data in an exponentially large Hilbert space [30]. Rather than focusing solely on matrix-inversion routines, these methods evaluate inner products of quantum states (i.e. kernel functions) that would be prohibitively expensive to compute classically. By embedding data into high-dimensional quantum feature spaces, one can construct decision boundaries that may be unreachable with purely classical algorithms. Indeed, [31] demonstrates an end-to-end quantum speedup for a suitably constructed classification problem, providing concrete evidence that quantum kernels can yield practical gains in machine learning tasks.

Classical SVMs aim to find a hyperplane that maximizes the margin between two classes, typically formulated in its primal form as:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \tag{1}$$

subject to

$$y_j\left(\mathbf{w}\cdot\mathbf{x}_j + b\right) \geqslant 1,;\forall j, \tag{2}$$

where $\mathbf{w}$ is the normal vector to the hyperplane, b is the bias, $\mathbf{x}_j$ are feature vectors, and y_j are the class labels. In quantum extensions of this method, the data are mapped into a higher-dimensional Hilbert space via a quantum kernel, enabling efficient non-linear classification that would otherwise be computationally prohibitive on classical hardware.

Early QSVM formulations relied on quantum matrix-inversion routines, such as the HHL algorithm [32], to mitigate the computational bottleneck inherent to large-scale SVMs. Theoretical analyses indicated that QSVM could perform these matrix inversions with $\mathcal{O}(\log(NM))$ complexity [6], a significant improvement over classical approaches. Quantum parallelism further reduces computational overhead by allowing simultaneous calculation of many kernel matrix elements, crucial for SVM optimization.

Within the QSVM framework, data points x_i are non-linearly transformed into quantum states $\rho(x_i) = |\psi(x_i)\rangle\langle\psi(x_i)|$ within the Hilbert space. The inner product between these quantum states, crucial for constructing the kernel matrix $K(x_i, x_j)$, is given by:

$$K\left(x_i, x_j\right) = \mathrm{tr}\rho\left(x_i\right)\rho\left(x_j\right) = |\langle\psi\left(x_i\right)|\psi\left(x_j\right)\rangle|^2, \tag{3}$$

where $|\langle\psi(x_i)|\psi(x_j)\rangle|^2$ is computed using a unitary matrix U, defined as:

$$|\langle\psi\left(x_i\right)|\psi\left(x_j\right)\rangle|^2 = |\langle 0^{\otimes N}|U^{\dagger}\left(x_i\right)U\left(x_j\right)|0^{\otimes N}\rangle|^2, \tag{4}$$

with $|0^{\otimes N}\rangle$ representing the initial state with all qubits in the $|0\rangle$ state.

QSVM extends classical SVM by utilizing quantum superposition and entanglement to address large, complex datasets more efficiently. Quantum parallelism enables rapid evaluation of kernel functions across multiple data pairs, a task that is computationally expensive classically [33]. In light of these developments, modern QSVM research increasingly emphasizes quantum kernel methods, reflecting both the capabilities of near-term quantum devices and the desire to circumvent the strict requirements of fully functional qRAM. Consequently, the present work examines classical simulations of quantum kernel approaches, building on recent theoretical and experimental advances to investigate whether and how quantum-enhanced feature spaces can yield advantages in realistic classification scenarios.

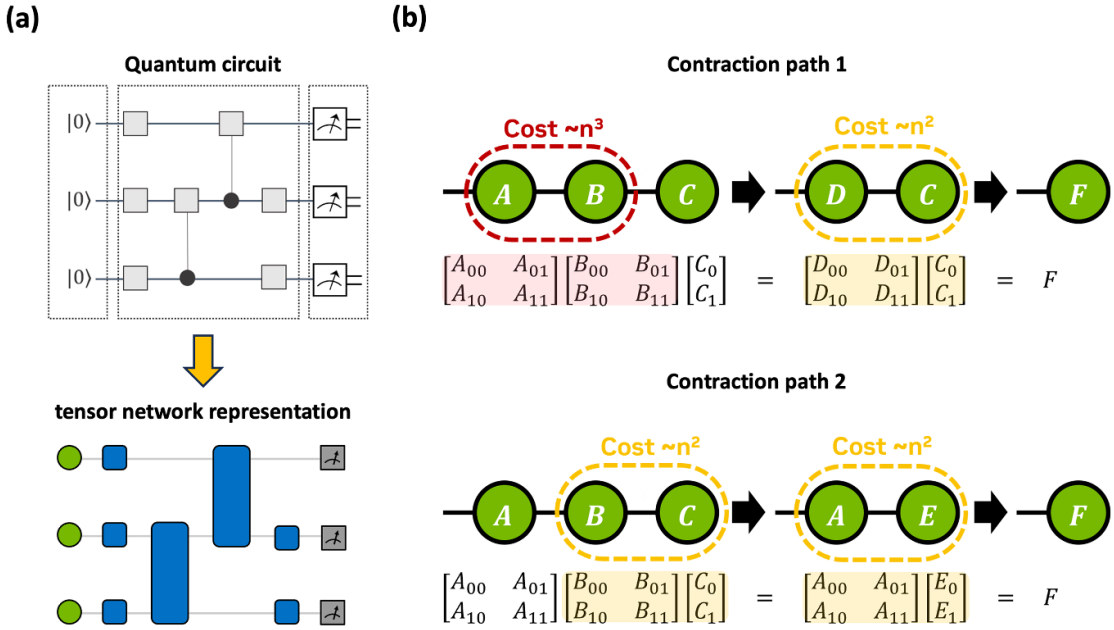

Figure 2. (a) Tensor network formulation of a quantum circuit. (b) Contraction paths determine the computational and memory costs of tensor network simulations: The upper path results in higher costs.

3. Simulating QSVM with tensor networks using the cuQuantum SDK and cuTensorNet library

3.1. Introduction of cuQuantum SDK and cuTensorNet library

As the fields of quantum computing and advanced numerical simulations rapidly expand, NVIDIA has introduced cuQuantum SDK [24], a comprehensive software development kit (SDK), to accelerate quantum circuit simulations with NVIDIA GPUs. It supports the programming model CUDA Quantum [24, 34] (CUDA-Q), and frameworks like Qiskit [35], Pennylane [36], and Cirq [37]. By offering a scalable and high-performance platform for quantum simulations, cuQuantum can democratize access to quantum computing research and even propel the field towards achieving real-world quantum machine learning applications.

The cuQuantum SDK consists of optimized libraries such as cuStateVec and cuTensorNet. cuStateVec is dedicated to state-vector simulation methods, providing significant acceleration and efficient memory usage, while cuTensorNet focuses on tensor-network simulations. For tensor-network methods, the quantum circuit is initially converted into a tensor-network representation (figure 2(a)). Subsequently, pairwise contraction paths are optimized to minimize computational complexity and memory footprint, followed by the execution of the computation. As shown in figure 2(b), the sequence of pairwise contractions plays a role in computational cost. cuTensorNet efficiently identifies high-quality contraction paths [24], accelerating quantum machine learning exploration, especially for high-dimensional data. The library offers advanced features like path optimization, approximate simulations, multi-GPU, and multi-node execution, enabling large-scale simulations and significantly advancing research into complex quantum algorithms across quantum physics, quantum chemistry, and quantum machine learning.

To boost the efficiency of tensor network computation, cuTensorNet delivers modular and finely adjustable APIs, as shown in figure 3, tailored for optimizing the pairwise contraction path on the CPU and improving contraction performance on the GPU. This optimization is essential for minimizing both computation cost and memory footprint. The pathfinder workflow is primarily structured in the following manner:

1) *Simplification:* This initial stage focuses on reducing the complexity of the whole tensor network and eliminating redundancies within the network. The implementation involves rank simplification to minimize the number of tensors by removing trivial tensor contractions from the network, resulting in a smaller network for subsequent processing.
2) *Division:* After simplification, the tensor network undergoes a recursive graph partitioning. This approach segments the network into multiple sub-networks and forms a contraction tree. The binary tree defines the contraction path and can be further optimized at the following stage.

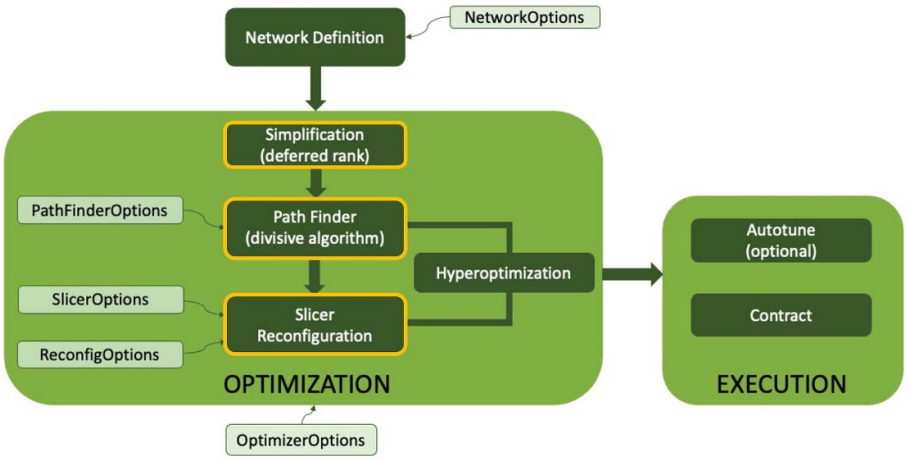

Figure 3. Building blocks of the cuTensorNet library.

3) *Slicing and Reconfiguration:* The slicing process selects a subset of edges from a tensor network for explicit summation. This technique results in lower memory requirements and allows parallel execution for each sliced contraction. Reconfiguration considers several small subtrees within the full contraction tree and attempts to reduce the contraction cost of the subtrees. cuTensorNet implements dynamic slicing, which interleaves slicing with reconfiguration.

3.2. Pipeline of QSVM simulation

In figure 4(a), the depicted pipeline of a QSVM commences with the initial quantum state preparation in a canonical basis state $|0\rangle$. The number of qubits depends on input data features, which can be adjusted using principal components analysis to evaluate QSVM with varying qubit counts. The QSVM circuit comprises a parameterized quantum circuit (QC) and its corresponding adjoint ($QC^{\dagger}$), which correspond to the unitary operators $U(\mathrm{x}_i)$ and $U^{\dagger}(\mathrm{x}_j)$ depicted in figure 4(b). The paired input data x_i and x_j are embedded into the left and right halves of the parameterized quantum circuit (QC and $QC^{\dagger}$), as shown in figure 4(b). At the measurement stage, the probability amplitude of the zero state $|0\rangle$ is used to represent the similarity between x_i and x_j in the quantum feature space. After computing the zero state amplitude for all paired data in the quantum feature space, the quantum kernel matrix is used to train the support vector classifier. Notably, only the probability of the all-zero state needs to be computed, allowing the tensor network simulation to reduce the overall computation by contracting the subspaces of the complete tensor structure. In this paper, we employ a parameterized QC based on Block-Product State (BPS) wavefunctions [38, 39]. This design mitigates accuracy degradation at moderate qubit counts, enabling the QSVM to maintain high classification performance. However, as the number of qubits becomes very large, exponential kernel concentration ultimately leads to a decrease in accuracy (see section 4.2). Notably, the circuit does not decompose into smaller blocks; instead, each qubit is entangled through CNOT gates arranged in a linear topology, ensuring compatibility with near-term quantum hardware (see appendix).

3.3. Complexity of QC simulation for QSVM

When executed on classical hardware such as CPUs and GPUs, the simulation of the QSVM algorithm poses significant computational challenges with state vector simulations. Figure 5 elucidates these challenges, indicating that the complexity scales exponentially with the number of qubits as $O(2^n)$ and quadratically with data size as $O(N^2)$. Additionally, the memory footprint of the full state vector grows exponentially with the number of qubits q, which map features into Hilbert space for QC simulations. This aspect underscores the inherent computational intensity of simulating quantum systems on classical infrastructure.

This scenario highlights the computational complexity advantages that QSVM offers in the realm of quantum machine learning. The simulation demands, in terms of computation time and memory size, grow exponentially with larger datasets and a greater number of qubits, a limitation not encountered when QSVM is run on quantum computers. As demonstrated by Rebentrost *et al* [6], the complexity advantage of QSVM can exhibit logarithmic scaling with respect to the product of the number of features and the size of the training set, denoted as $O(\log(NM))$. However, in the NISQ era, the verification of algorithms using traditional CPUs is inevitable. Therefore, this section focuses on leveraging GPU acceleration to address the

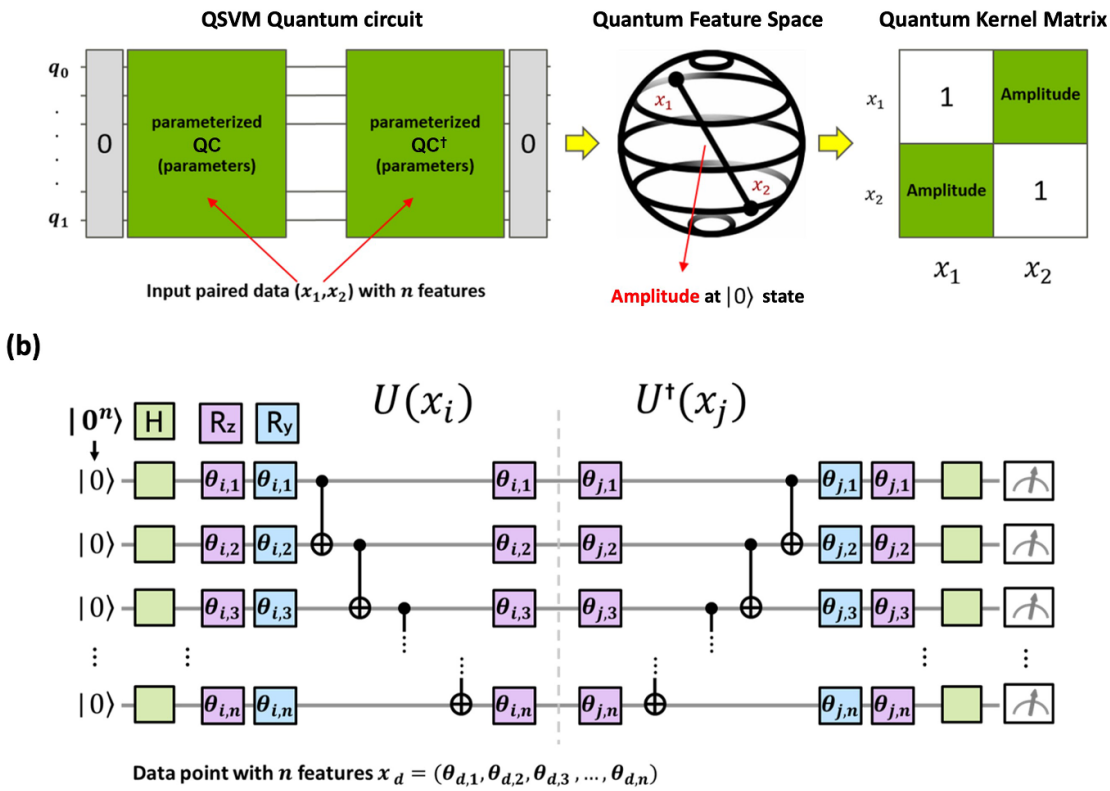

Figure 4. (a) The QSVM pipeline showcases the quantum circuit transformation of input data into feature space quantum states. (b) A schematic of the QSVM circuit.

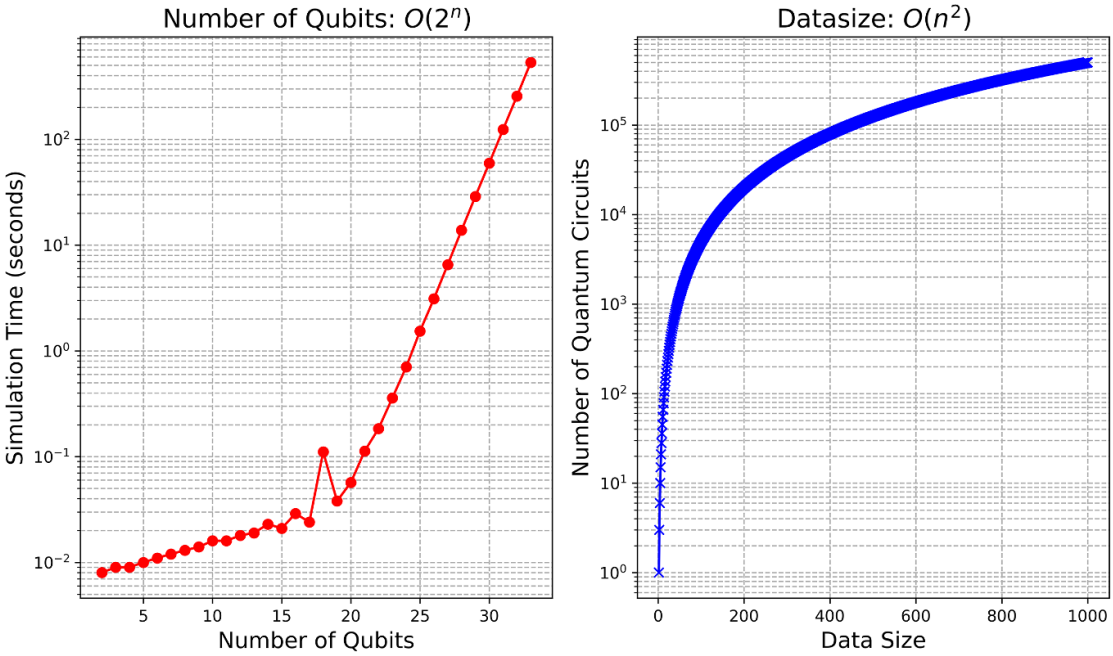

Figure 5. Computational complexity of QSVM simulation. The left graph demonstrates that simulation time scales exponentially with the number of qubits, as $O(2^n)$, while the right graph shows that the number of quantum circuits required scales quadratically with data size, as $O(n^2)$.

Algorithm 1. Get Kernel Matrix using cuStateVec.

Input :Number of data1 *datasize*1, Number of data2 *datasize*2, List of quantum circuits *circuits*, Index of data1 and data2 combinations *indices*, statevector simulator *simulator*

(i) Initialize $kernel\_matrix \in \mathbb{C}^{datasize1 \times datasize2}$ with all elements set to zero.

(ii) Set the current operand index i to -1.

(iii) **for** $i_1, i_2 \in \{1, \ldots, indices\}$ **do**

- (a) Update the circuits index $i \leftarrow i + 1$.
- (b) Save circuits [i] statevector.
- (c) Set transpile(circuits[i], simulator).
- (d) Run simulator and save result *result*.
- (e) Compute amplitude $amp \leftarrow$ result.get_statevector().
- (f) Calculate and store $kernel\_matrix[i_1 - 1][i_2 - 1] \leftarrow (\sqrt{\text{amp.real}^2 + \text{amp.imag}^2})$.

end

(iv) Symmetrize *kernel_matrix* by adding its transpose and an identity matrix: $kernel\_matrix \leftarrow kernel\_matrix + kernel\_matrix^T + \text{diag}(\mathbb{1}_{datasize1})$.

return *kernel_matrix*

computational bottlenecks encountered when simulating QSVM with large-scale qubit sizes and processing large datasets.

3.4. Simulating QSVM's quantum kernel matrix

In this section, we discuss three methods for simulating a QSVM algorithm: state-vector simulation on GPU using the cuStateVec library, tensor-network simulation on CPU using the opt-einsum library, and tensor-network simulation on GPU using the cuTensorNet library. In this research work, our interest lies in comparing a state-of-the-art CPU-centric approach (opt-einsum) to a state-of-the-art GPU-centric approach (cuTensorNet). This comparison highlights how GPU acceleration can significantly impact large-scale QC simulations.

3.4.1. Simulation of QSVM with state vector

State vector simulation is widely used for simulating QC-based quantum computing because QC operations can naturally be represented using state vectors. In this method, Qiskit is used to create the QSVM QC, and the state-vector simulation is implemented on a GPU via the cuStateVec backend, as described in algorithm 1. The advantage of using cuStateVec includes a speedup of the simulation time by leveraging GPU capabilities and enabling multi-GPU processing with MPI for distributed computing. The effectiveness of cuStateVec in enhancing quantum-circuit-simulation efficiency is evidenced in Lykov *et al*'s research work using cuStateVec and the cuQuantum SDK [22].

3.4.2. Simulation of QSVM with tensor network

Even with `cuStateVec` enabling GPU acceleration, challenges persist due to the complexity of encoding the number of qubits $O(2^n)$ CF and the size of the data $O(n^2)$. To surmount these challenges, we present an innovative approach using the `cuTensorNet` library for QSVM simulation. In the creation of the tensor network representation, we seamlessly integrate Qiskit and cuQuantum's built-in `CircuitToEinsum` object.

Initially, Qiskit is used to construct a `QuantumKernel` circuit, which is then transformed into 'expression' and 'operand' components by `CircuitToEinsum`. Due to the identical topological structure of the QC, the same 'expression' component can be reused for subsequent pairs of data. Meanwhile, the 'operand' is updated with parameters from the previously created operand. This approach rapidly transitions data pairs into tensor networks and preserves computational efficiency. The derivation of the kernel matrix-a pivotal component of the SVM-exploits a consistent 'path' to greatly minimize the repetition of contraction order calculations. The detailed algorithm is described in algorithm 2. This technique not only leverages the computational strength of GPUs but also ensures path reusability, resulting in a considerable acceleration of the simulation process and a dramatic reduction in computational complexity. We will demonstrate those improvements in the next section.

To ensure a fair comparison tensor network QSVM simulation between CPU and GPU performance, we utilize the opt-einsum package, which provides optimized tensor computation on CPUs similar to the cuQuantum SDK available for NVIDIA GPUs. The detailed algorithm for simulating the QSVM on CPUs, aimed at equalizing the computational environment to the extent possible, is described in algorithm 3.

Algorithm 2. Get Kernel Matrix using cuTensorNet.

Input :Number of data1 *datasize*1, Number of data2 *datasize*2, Circuit einstein summation expression exp, List of operands *operands*, Index of data1 and data2 combinations *indices*, network options *options*

(i) Initialize $kernel\_matrix \in \mathbb{C}^{datasize1 \times datasize2}$ with all elements set to zero.
(ii) Set the current operand index i to -1.
(iii) Initialize the network with given *options* to prepare for contraction operations.
(iv) **for** $i_1, i_2 \in \{1, \ldots, indices\}$ **do**
  (a) Update the operand index $i \leftarrow i + 1$.
  (b) Reset the network to its initial state before each contraction.
  (c) Prepare the operands for contraction based on i.
  (d) Compute amplitude $amp \leftarrow$ Contract within the network$(\exp, operands[i], options)$.
  (e) Calculate and store $kernel\_matrix[i_1 - 1][i_2 - 1] \leftarrow \sqrt{\text{amp.real}^2 + \text{amp.imag}^2}$.
end
(v) Symmetrize *kernel_matrix* by adding its transpose and an identity matrix:
$kernel\_matrix \leftarrow kernel\_matrix + kernel\_matrix^T + \text{diag}(\mathbb{1}_{datasize1})$.

return *kernel_matrix*

Algorithm 3. Get Kernel Matrix using opt-einsum.

Input :Number of data1 *datasize*1, Number of data2 *datasize*2, Circuit einstein summation expression exp, List of operands *operands*, Index of data1 and data2 combinations *indices*, Contraction path *path*

(i) Initialize $kernel\_matrix \in \mathbb{C}^{datasize1 \times datasize2}$ with all elements set to zero.
(ii) Set the current operand index i to -1.
(iii) **for** $i_1, i_2 \in \{1, \ldots, indices\}$
  (a) Update the operands index $i \leftarrow i + 1$.
  (b) Compute amplitude $amp \leftarrow$ opt_einsum.contract$(\exp, operands[i], path)$.
  (c) Calculate and store $kernel\_matrix[i_1 - 1][i_2 - 1] \leftarrow \sqrt{\text{amp.real}^2 + \text{amp.imag}^2}$.
end
(iv) Symmetrize *kernel_matrix* by adding its transpose and an identity matrix:
$kernel\_matrix \leftarrow kernel\_matrix + kernel\_matrix^T + \text{diag}(\mathbb{1}_{datasize1})$.

return *kernel_matrix*

4. Performance and benchmarking of QSVM with cuTensorNet

4.1. QSVM Simulation and cuTensorNet-Accelerated QSVM (cuTN-QSVM)

In the outlined simulation workflow, figures 1 and 4 illustrate the sequence from the initial input of data to the generation of a QC for the purpose of encoding. Subsequent steps involve the use of optimized compilation to compute and simulate the QCs, leading to the extraction of a quantum kernel matrix. This matrix is then applied to develop a support vector classifier (SVC).

However, in typical CPU-based workflows, bottlenecks arise in the progression from the construction of QCs to the calculation of the quantum kernel matrix, where the complexity of simulating the QSVM algorithm scales exponentially with the number of qubits, $O(2^n)$, and quadratically with data size, $O(N^2)$. To alleviate these bottlenecks, we incorporate the cuQuantum SDK into the QSVM workflow, employing a method of assigned parameters for the formulation of QSVM's QCs. We then maintain a consistent 'expression' for the simulation of these circuits. Ultimately, a 'path reuse' strategy is adopted for the tensor network contractions to compute the quantum kernel matrix, effectively reducing redundant computations when processing large datasets, reducing it from $O(N^2)$ to $O(1)$ for pathfinding. Importantly, as depicted in figure 6, the expressions and paths used in the cuTensorNet during the QSVM simulation process remain unchanged compared to those in CPU and cuStateVec, ensuring that no accuracy is compromised for the sake of expedience. In addition to the path reuse strategy, cuTensorNet offers concurrent execution for tensor network contractions. This technique allows the continued contractions on multiple GPUs asynchronously when tensors are already on the device, thus enhancing computational efficiency by continuing operations without delay. The pronounced speedup achieved through the implementation of path reuse within the

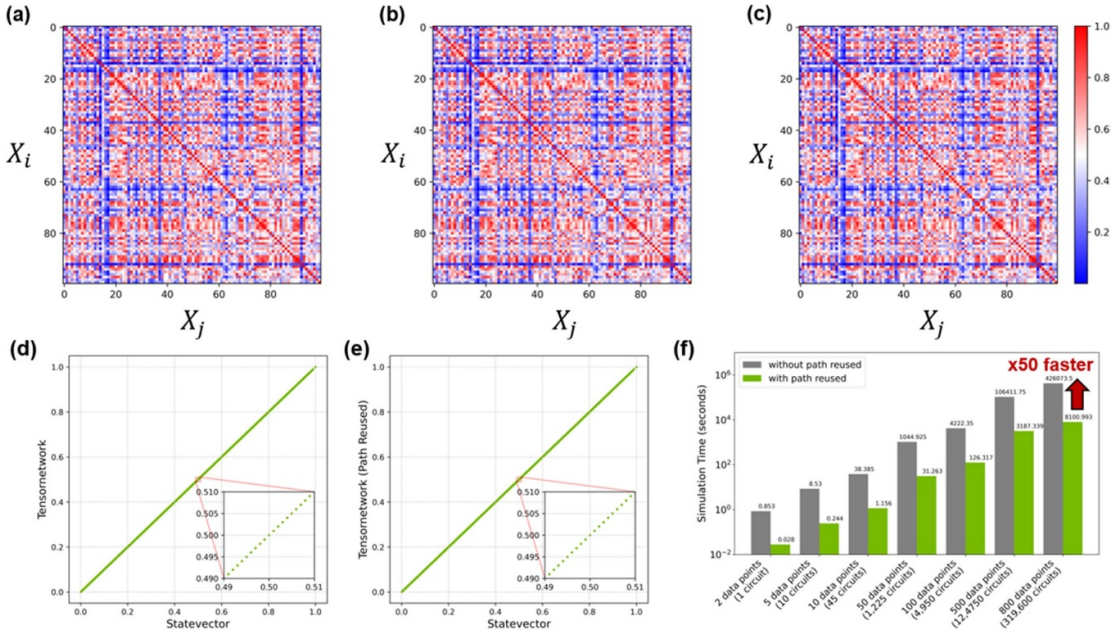

Figure 6. Comparative visualization of quantum kernel matrices and their computation speedups. (a)–(c) illustrate the quantum kernel matrices generated from state vector simulation, tensor network simulation, and tensor network simulation with path reuse strategies, respectively. (d) and (e) feature the parity plots for quantum kernel assessments comparing the outputs of state vector simulations with tensor network and tensor network with path reuse algorithms, demonstrating high concordance. (f) quantifies the performance enhancement attributable to path reuse in tensor network simulations, showcasing significant temporal reductions across an array of dataset sizes.

cuTensorNet library is detailed in figure 6(g), where we report a fiftyfold increase in speed compared to conditions without path reuse.

In the comprehensive workflow outlined in figure 7, the input data initiates QC construction, integrating frontends such as Qiskit or Cirq with the cuQuantum API, which generates Einstein summation expressions and tensor operands for the circuit. The process advances by converting QCs into tensor networks represented as CuPy arrays, enabling the utilization of in-place operations to update content for the same operands efficiently. Key to enhancing computational efficiency within this framework is the strategic deployment of direct conversion from data to operand, alongside expression reuse for optimizing computational pathways. This step is crucial in minimizing redundancy and ensuring the streamlined execution of the workflow. As the process proceeds, CuPy's capabilities are harnessed to accelerate the computation of the kernel matrix, culminating in the application of the SVC. Moreover, cuTensorNet, as part of the cuQuantum SDK, incorporates advanced strategies such as path reuse and non-blocking operations across multi-GPU configurations.

These approaches significantly reduce the computational overhead from a conventional complexity of $O(2^n)$ to a more scalable $O(n^2)$, thereby enhancing the practicality of executing extensive QSVM simulations with improved processing times and efficiency in resource utilization. Figure 8 illustrates that quantum simulation on the NVIDIA A100 GPU using cuStateVec becomes practically infeasible for more than 50 qubits. However, by employing cuTensorNet, single-contraction simulations can be completed within 0.2 seconds, even with up to 784 qubits. Additionally, figure 8 shows that the path reuse strategy can further enhance the speed, offering more than tenfold acceleration when increasing the number of qubits in the QSVM algorithm.

In the GPU-accelerated workflow utilizing cuTensorNet, as delineated in figure 7, we are able to expand the feature size (number of qubits) and scale up the data volume for our QSVM algorithm. The evaluation of accuracy resulting from these augmentations will be discussed in the following part, while an in-depth assessment of resource management will be presented in the subsequent section.

4.2. Accuracy benchmarking and validation of large-scale QSVM

4.2.1. Binary classification

We benchmarked the performance of a QSVM on high-dimensional image classification tasks using both 10-class MNIST and Fashion-MNIST datasets of 31 500 images each, split 80–20 for training and testing. As a classical baseline, we employed an SVM with a radial basis function kernel and systematically varied the scaling parameter γ between 0.001 and 1000 to identify optimal hyperparameters. Although SVM is not the

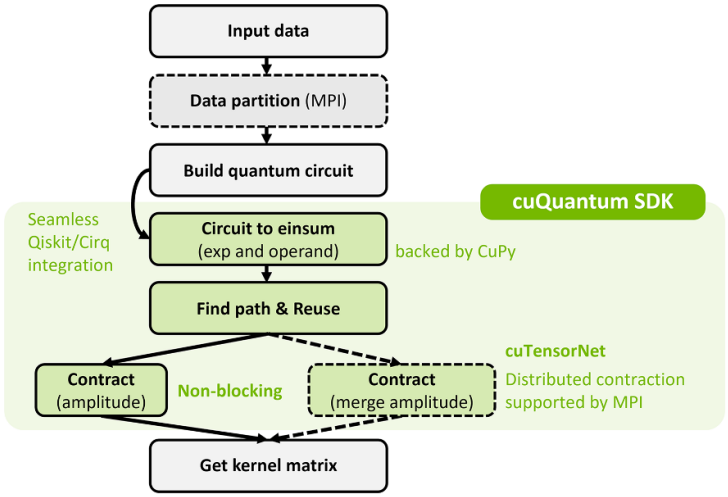

Figure 7. Workflow optimization for QSVM simulation through architectural enhancements, integrating Qiskit/Cirq with cuQuantum SDK. This transition from circuit building to tensor network conversion and kernel matrix computation reduces computational time complexity, leveraging GPU acceleration and multi-GPU strategies.

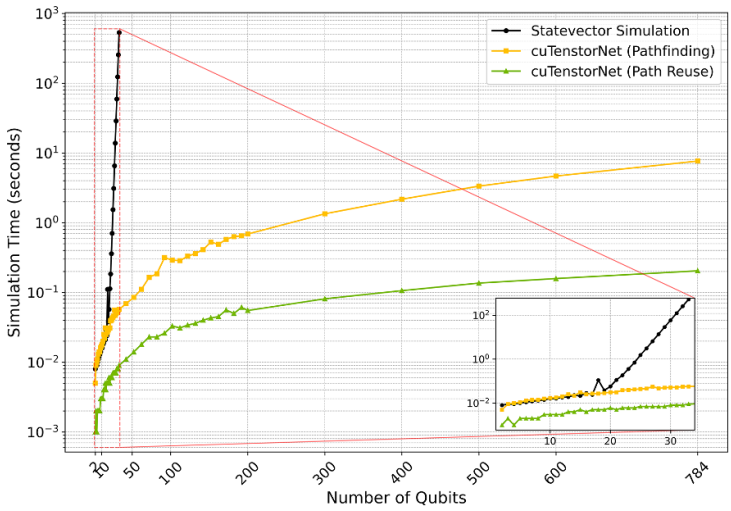

Figure 8. Simulation time comparison for quantum state vector simulation and two cuTensorNet approaches-path plus contraction (no-path reuse), and contraction only (path reuse)-across a range of qubit numbers, up to the equivalent of a 28x28 pixel grid (784 qubits).

most advanced machine learning model available, it provides a well-established framework that enables us to validate the feasibility and potential advantages of our quantum-enhanced approach.

In figure 9, our results indicate that QSVM offers competitive performance and can, under certain conditions, outperform the classical SVM for moderate circuit sizes. In particular, QSVM maintains high accuracy by leveraging quantum feature maps that embed data into larger Hilbert spaces. However, beyond a critical qubit threshold, we observe a decline in test accuracy, which we attribute to overfitting effects and the phenomenon of exponential kernel concentration-where off-diagonal kernel matrix elements diminish and the optimization landscape becomes exponentially flat [40]. This vanishing-gradient problem not only complicates the training of parameterized Qs but also underscores the practical limitations of naively scaling up circuit depth or qubit count.

In light of these observations, current quantum machine learning approaches still require careful feature engineering or hybrid methods to optimize model performance. Moreover, the amount of training data can significantly impact QSVM accuracy, as illustrated in figure 10, underscoring the importance of sufficiently large datasets.

Despite these challenges, our proof-of-concept study confirms that QSVM can serve as a promising foundation for large-scale quantum machine learning, particularly in scenarios where high-dimensional embeddings may confer a computational advantage. Further research on QC design, regularization

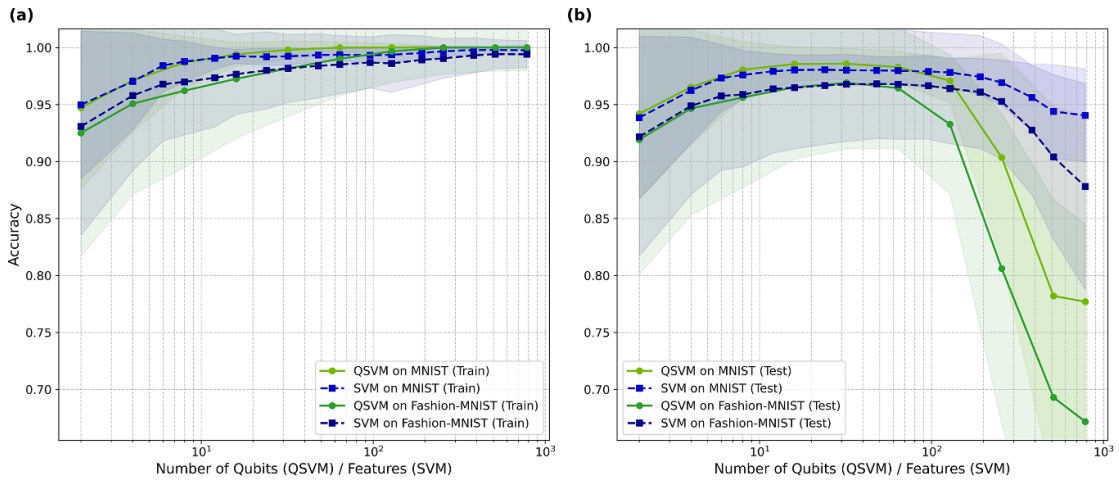

Figure 9. Benchmark of QSVM and SVM on MNIST and Fashion-MNIST (nine labels, 45 binary classification tasks). (a) Training accuracy vs number of qubits (QSVM) or features (SVM). (b) Test accuracy. QSVM outperforms SVM for moderate qubit counts, but overfitting leads to reduced test accuracy with more qubits. Shaded regions show one standard deviation around mean accuracy.

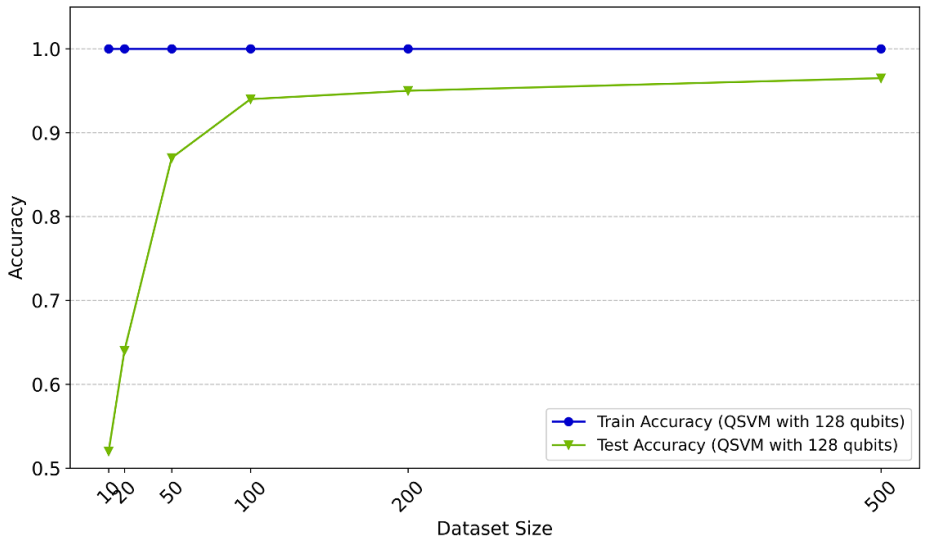

Figure 10. Classification Accuracy versus Dataset Size for Binary Classification of MNIST Digits 2 and 6 using a 128-Qubit QSVM model.

strategies, and optimization techniques will be crucial to fully harness the benefits of quantum-enhanced models and to mitigate the pitfalls associated with increasingly large quantum systems.

4.2.2. Multi-class classification

To assess the robustness of the QSVM beyond binary classification, we further evaluated its performance on the 10-class versions of the MNIST and Fashion-MNIST datasets. For each dataset, we select 1000 training samples and reserve an additional 500 samples for testing. For MNIST, both the classical baseline SVM and the QSVM employ 64 features/qubits, whereas for Fashion MNIST, they use 96 features/qubits. Figure 11 shows the confusion matrices for both datasets under SVM and QSVM, while table 1 summarizes several macro-level performance metrics (accuracy, sensitivity, specificity, and F_1 score). For MNIST, the QSVM yields slightly higher accuracy, along with marginal improvements in sensitivity and specificity. A similar trend emerges for the more challenging Fashion-MNIST dataset, where QSVM also achieves higher overall accuracy and macro-level metrics than the classical SVM.

These findings reinforce our earlier observations that, for moderate circuit sizes, QSVM can learn complex data distributions effectively. By embedding data points in a larger Hilbert space, the quantum kernel method can capture subtle features that improve class separability. However, as discussed previously, overfitting and the onset of Barren plateaus can degrade performance when the number of qubits becomes

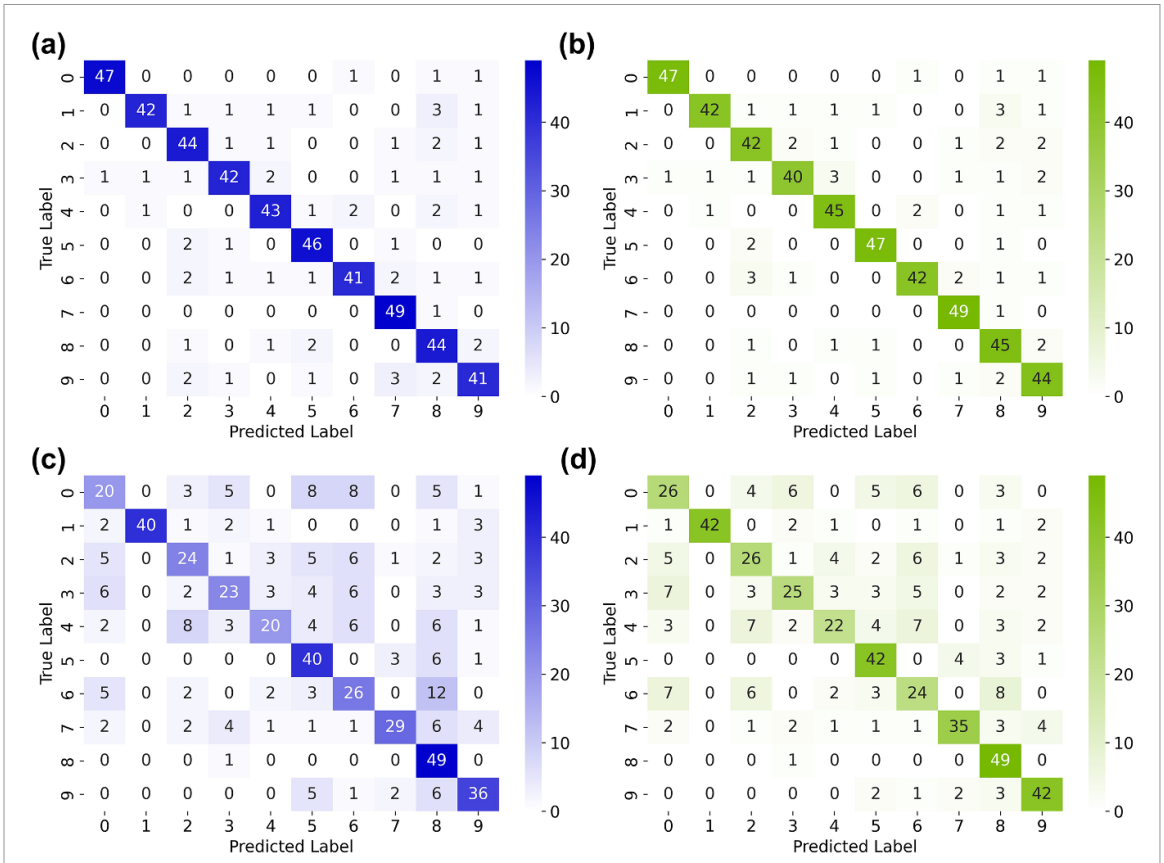

Figure 11. Confusion matrices for the classical SVM with 64 features (a) and 96 features (c), and for the QSVM with 64 qubits (b) and 96 qubits (d), evaluated on 10-class MNIST (a), (b) and 10-class Fashion-MNIST (c), (d). Each cell indicates the number of predictions for a given true label (vertical axis) and predicted label (horizontal axis). Brighter diagonal entries reflect a higher count of correctly classified samples. Overall, QSVM demonstrates consistently strong performance, often improving upon the classical SVM.

Table 1. Macro Metrics for SVM and QSVM on MNIST and Fashion MNIST Datasets. The bold numbers this table indicate better performance in macro metrics.

Dataset	Model	Accuracy	Sensitivity	Specificity	F1-score
MNIST	SVM	0.8780	0.8820	0.9865	0.8783
	QSVM	**0.8860**	**0.8900**	**0.9874**	**0.8864**
Fashion MNIST	SVM	0.6140	0.6388	0.9578	0.6088
	QSVM	**0.6660**	**0.6744**	**0.9633**	**0.6608**

excessively large. Consequently, the design of circuit architectures and the choice of hyperparameters-particularly in multiclass settings-remain critical in balancing expressivity and generalization.

4.3. Simulation with single CPU and GPU

In this section, we compare the performance of a CPU and a GPU, as illustrated in figure 12. To ensure a fair comparison, we employed Opt-Einsum for the contraction process on a single AMD EPYC 7J13 CPU, contrasting this with a single NVIDIA A100 GPU using cuTensorNet for the contraction process, with path reuse implemented. The detailed pseudocodes are discussed in section 3.4. Moreover, it was necessary to synchronize the contraction paths in Opt-Einsum with those of cuTensorNet to ensure consistency. As depicted in figure 12, the speedup provided by the GPU relative to the CPU becomes more pronounced as the number of simulated qubits increases. Consequently, for large-scale qubit simulations, GPUs demonstrate enhanced scalability and promise substantial benefits for future advanced qubit algorithms in simulation and emulation.

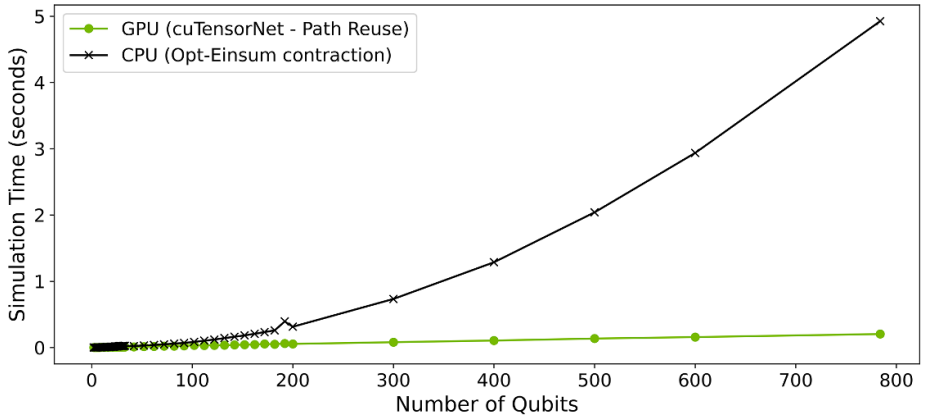

Figure 12. Benchmarking QSVM circuit simulation time using a single CPU and a single GPU. The GPU data shows minimal variation compared to the CPU's scale.

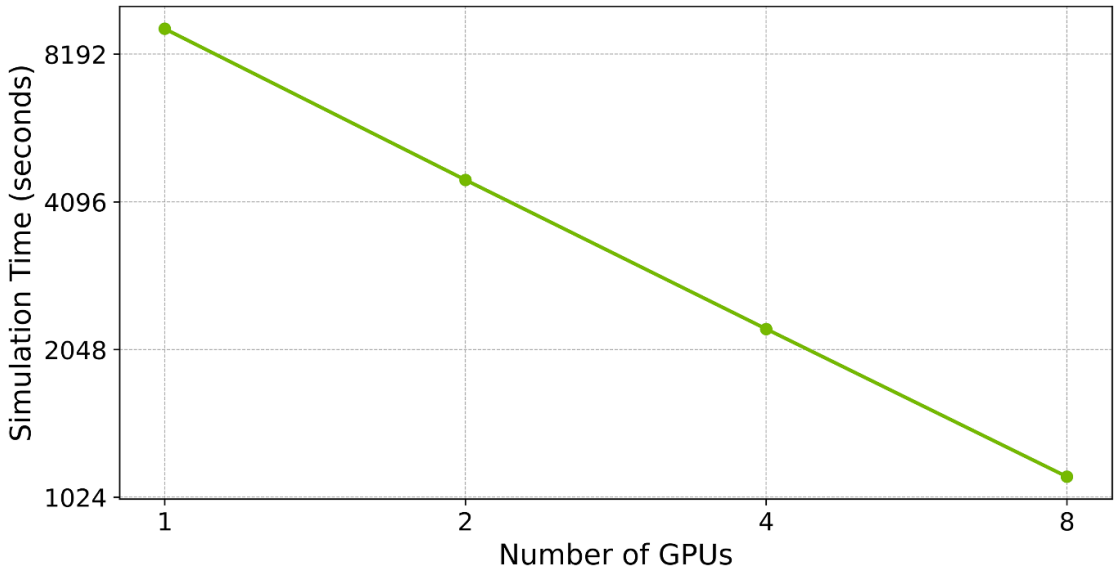

Figure 13. Strong scaling of the QSVM simulation is observed for 1000 data points across 1, 2, 4, and 8 GPUs, demonstrating linear speedup.

5. Distributed simulation and resource estimation in HPC

In the final section of our study, a multi-GPU instance was utilized to expand the QSVM model via cuTensorNet to accommodate a dataset comprising 1000 data points of 28x28 MNIST images. The implementation of multi-GPU resources to enhance QC simulation via cuStateVec is thoroughly detailed in the research conducted by Shaydulin *et al* [41]. Our emphasis lies on leveraging the data from these experiments to rigorously assess both the computational costs and the temporal demands inherent in the tensor-network simulation of the QSVM algorithm within a multi-GPU processing framework.

In our computational environment, each GPU within a node is interconnected using the high-bandwidth NVLink network and the NVIDIA Collective Communications Library (NCCL) to optimize intra-node communication. The input data is paired and evenly distributed across multiple GPUs via NCCL, where it is directly converted into a tensor network for computation. The results are then returned to a single GPU via NCCL to form the quantum kernel matrix for SVC classification. By harnessing these integrated technological benefits, we have successfully actualized the accelerated computational outcomes for managing large-scale qubit systems and complex datasets, as illustrated in figure 13. Comparative analysis indicates that our performance metrics are on par with distributed simulation results documented in the existing scientific corpus, as cited in Bayraktar *et al*'s and Lykov *et al*'s work [22, 24].

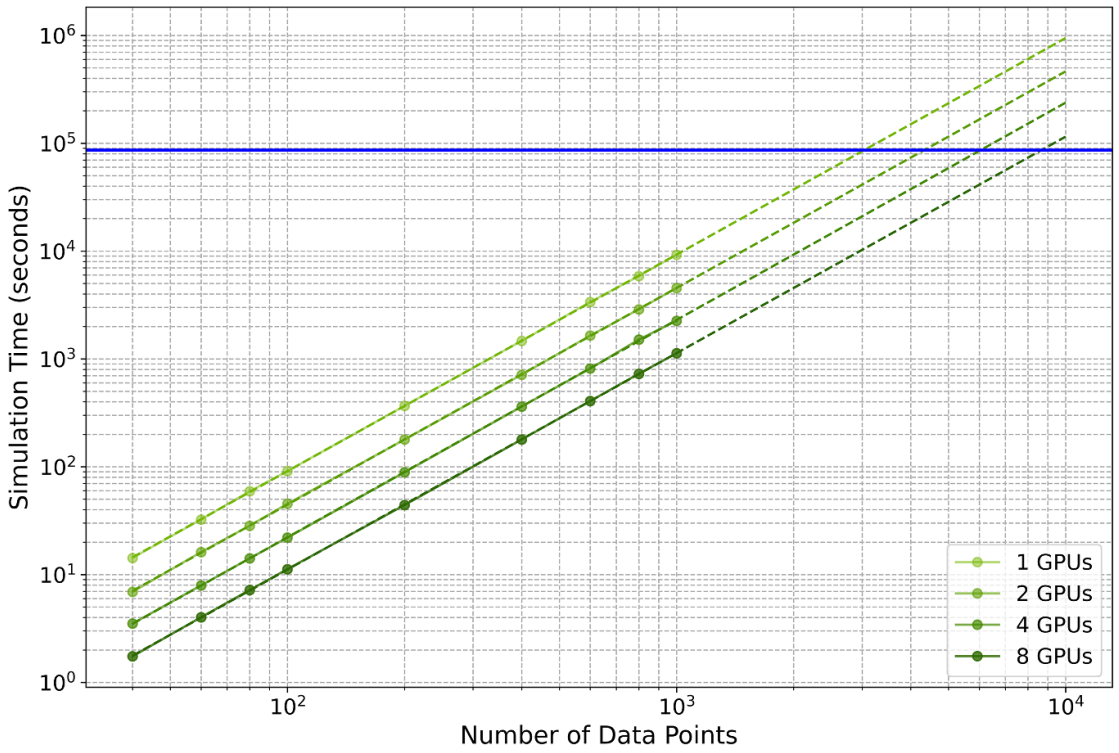

Figure 14. Execution time for quantum simulations against qubit count for a single A100 GPU and MPI-based 2, 4 and 8 multi-GPU setups. The performance enhancement with additional GPUs is evident, underscoring the benefits of parallelized computation.

5.1. Benchmarking cuTensorNet Multi-GPU with MPI

Figure 13 illustrates the execution time required for quantum simulations in relation to the number of qubits. The data compares the performance of a single A100 GPU to systems utilizing 2, 4, and 8 GPUs in conjunction with MPI and within a single NVIDIA DGX node. It is evident from the results that the incorporation of multi-GPUs significantly decreases computation time, highlighting the strong linear speedup of cuTenserNet with MPI. The trend indicates a substantial reduction in execution time as the number of GPUs is increased, affirming the efficacy of multi-GPU setups in handling large datasets.

5.2. Large dataset processing with multi-GPU

Figure 14 presents a comparative analysis of computational time across different configurations, ranging from a single GPU (A100, 80GB) to 2, 4, and 8 multi-GPU arrangements using MPI for processing datasets of various sizes. The results distinctly highlight the superior efficiency and scalability of multi-GPU systems, especially when managing large-scale datasets. A notable reduction in processing time is observed with the integration of an 8-GPU setup, underscoring the considerable advantages of parallel computing for large-scale data analysis. In figure 14, experimental data (solid line) from 40 to 1000 data points is extrapolated to estimate the processing time for 10 000 data points, corresponding to nearly 50 million circuits (dashed line). The projection indicates that an eight-GPU system could achieve linear acceleration, reducing a week-long processing task using the simulated QSVM to approximately one day (blue line).

6. Conclusion

This paper has presented a comprehensive investigation into the feasibility and performance of large-scale QSVM simulations using a tensor-network-based framework integrated with NVIDIA's cuQuantum SDK. By leveraging the cuTensorNet library on multi-GPU platforms, we significantly reduced the otherwise prohibitive computational overhead associated with simulating large qubit systems. Rigorous performance benchmarks demonstrated not only near-quadratic scaling for circuit simulations-thereby overcoming the exponential barriers of conventional state-vector approaches-but also robust speedups via MPI-based parallelization for QC simulation. Moreover, our experiments on benchmark datasets, including MNIST and Fashion-MNIST, revealed that QSVMs can achieve high classification accuracy, emphasizing the promise of quantum methods for complex, high-dimensional data. Crucially, the observed improvements in accuracy with increasing dataset size underscore the value of scalable simulation environments as a test bed for

algorithmic refinements and real-world applications. The successful integration of cuTensorNet and multi-GPU infrastructures thus serves as a critical validation of quantum-HPC synergy, pointing to a practical route toward bridging near-term quantum hardware limitations and large-scale quantum machine learning goals. These results lay a foundation for further advances in high-performance quantum simulations and reinforce the potential impact of quantum-enhanced algorithms within the rapidly evolving Quantum-HPC ecosystem.

Data availability statement

The code supporting the findings of this research is available on GitHub at the following repository: https://github.com/Tim-Li/cuTN-QSVM. This repository includes the scripts and data required to reproduce the results presented in this paper.

Acknowledgments

The authors express their gratitude to W C Qian (MediaTek) for invaluable assistance and insights, which were pivotal to the success of this research. This research was funded by the U K Engineering and Physical Sciences Research Council (EPSRC) under Grant No. EP/W032643/1. K C acknowledges financial support from the Turing Scheme for the Imperial Global Fellows Fund and the Taiwanese Government Scholarship to Study Abroad. This work was also supported by the National Science and Technology Council (NSTC), Taiwan, under Grants NSTC 112-2119-M-007-008- and NSTC 113-2119-M-007-013-. The authors thank the National Center for High-performance Computing of Taiwan for providing computational and storage resources, as well as the NVAITC and NVIDIA Quantum team for their technical support.

Conflict of interest

The authors declare that they have no conflict of interest related to this work.

Appendix. QSVM with block-product states

In this appendix, we provide additional technical details on our use of BPS and the contrast between our circuit constructions and the stricter block-encoding formalism in [38]. Our primary objective is to investigate how varying levels of entanglement (no entanglement, partial entanglement, and full entanglement) can affect the performance of QSVM. Although we embrace the idea of embedding features in 'blocks,' we do not strictly enforce the complete separability mandated in reference [38]. Instead, we consider a spectrum of entangling mechanisms within and between these blocks to assess scalability, expressive power, and classification accuracy [39].

One of the feature-map strategies we examine in this appendix is the fully-entangled ZZ Feature Map, which harnesses pairwise qubit interactions via controlled-phase gates to capture higher-order correlations in the input data. For an n-dimensional input vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, a standard realization of this encoding can be written as

$$U_{\mathrm{ZZ}}(\mathbf{x}) = \left[\prod_{1\leqslant i<j\leqslant n} \exp\left(-\frac{i}{2} x_i x_j Z_i Z_j\right)\right]\left[\prod_{i=1}^{n} \exp\left(-\frac{i}{2} x_i Z_i\right)\right], \tag{5}$$

where Z_i and Z_j are the Pauli-Z operators on the ith and jth qubits, respectively, with $1 \leqslant i < j \leqslant n$. These exponential operators can be decomposed into single-qubit rotations and two-qubit controlled-phase gates, and a layer of Hadamard gates is often introduced at the beginning and/or end of the circuit to spread amplitudes across the computational basis.

To explore more scalable quantum data embeddings, we employ a BPS approach, inspired by the framework in reference [38, 39]. In this method, the feature space is partitioned into smaller blocks that are mapped to distinct, though not necessarily fully separable, sections of the Hilbert space. Specifically, we let $\mathbf{x}$ be an (mn)-dimensional input vector grouped into m blocks,

$$\mathbf{x} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m]^\top, \quad \mathbf{s}_b = [s_{b,1}, s_{b,2}, \ldots, s_{b,n}]^\top,$$

and define the BPS wavefunction as

$$\left|\Psi^{\mathrm{BPS}}(\mathbf{x})\right\rangle = \left|\psi_1(\mathbf{s}_1)\right\rangle \otimes \left|\psi_2(\mathbf{s}_2)\right\rangle \otimes \cdots \otimes \left|\psi_m(\mathbf{s}_m)\right\rangle, \tag{6}$$

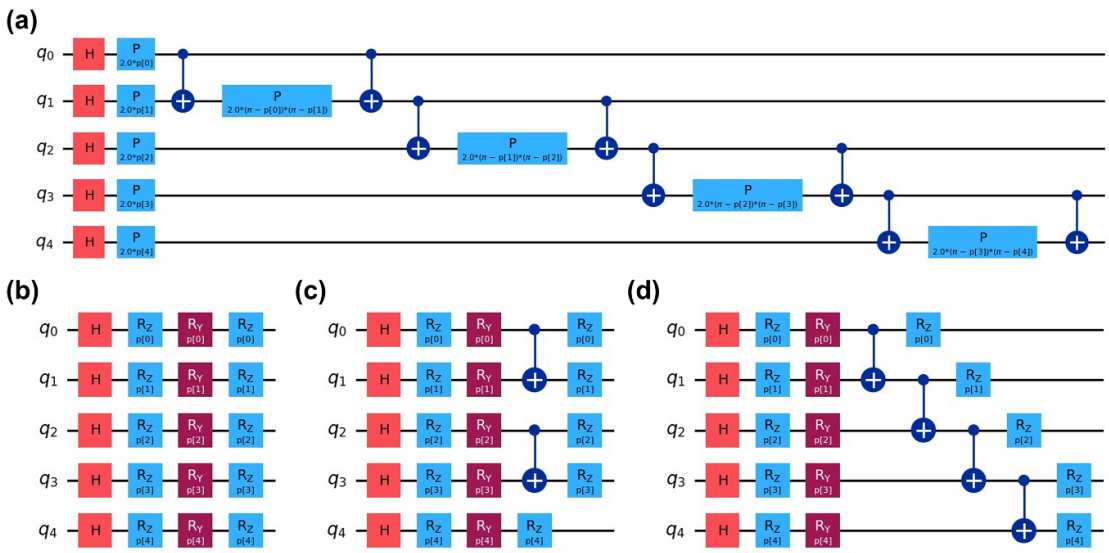

Figure 15. Four encoding schemes for 5-qubit QSVM: (a) Fully-entangled ZZ Feature Map, (b) Non-entangled BPS, (c) Half-entangled BPS, and (d) Fully-entangled BPS. By varying the entanglement among qubits and across blocks, these schemes strike different balances between expressiveness and resource requirements.

where each block $\left|\psi_b(\mathbf{s}_b)\right\rangle$ is constructed via single-qubit rotations and an internal entangling layer:

$$\left|\psi_b\left(\mathbf{s}_b\right)\right\rangle = \left(\bigotimes_{q=1}^{n} R_z\left(s_{b,q}\right)\right) U_{2n}^{\text{ent}} \left(\bigotimes_{q=1}^{n} R_y\left(s_{b,q}\right) R_z\left(s_{b,q}\right) H\right) \left|0^{\otimes n}\right\rangle. \tag{7}$$

The operator

$$U_{2n}^{\text{ent}} := \prod_{q=1}^{n-1} \text{CNOT}_{q,q+1} \tag{8}$$

adds a layer of entanglement among the n qubits in each block. While [38] enforces block separability across the entire wavefunction to allow classical multiplications of block overlaps, our approach relaxes this condition in subsequent layers to support partial or fully connected entanglement across blocks for comparison in this appendix.

Figure 15 illustrates four representative encoding circuits for a 5-qubit case: (a) a fully-entangled ZZ Feature Map employing multi-qubit controlled-phase operations, (b) a non-entangled BPS circuit that uses only single-qubit rotations (R_z, R_y) without CNOTs, (c) a half-entangled BPS incorporating partial entanglement within each block, and (d) a fully-entangled BPS that enables both intra- and inter-block entangling gates for maximum expressive power. These circuits capture the trade-off between entanglement depth, expressive capacity, and circuit overhead.

From a practical standpoint, BPS embeddings offer notable advantages: partitioning the encoding into separate blocks simplifies circuit design for large-scale qubit systems and enables tensor-network or classical simulation methodologies that assume limited entanglement. In instances where strong multi-qubit correlations are not essential, low-entanglement BPS embeddings can deliver satisfactory performance while maintaining relatively shallow circuits. It is worth mentioning that this could benefit distributed quantum computing's circuit partitioning strategy.

Nonetheless, strictly separable blocks may undermine the expressive power of an encoding, resulting in suboptimal classification accuracy for datasets that rely on higher-order correlations. As shown in figure 16, although partially or fully entangled circuits incur higher overhead, they typically achieve superior accuracy when the number of qubits increases. In our benchmarking, which targets MNIST digit {2,6} classification (with 500 training samples and 200 testing samples), BPS-based feature maps outperform the ZZ Feature Map. Among the three BPS variants (non-entangled, half-entangled, and fully-entangled), the fully entangled design consistently yields the best performance. These observations suggest that for more complex problems, the richer correlations intrinsic to fully entangled feature maps often warrant the additional resource cost. Nevertheless, partially entangled BPS configurations may still serve as a practical compromise

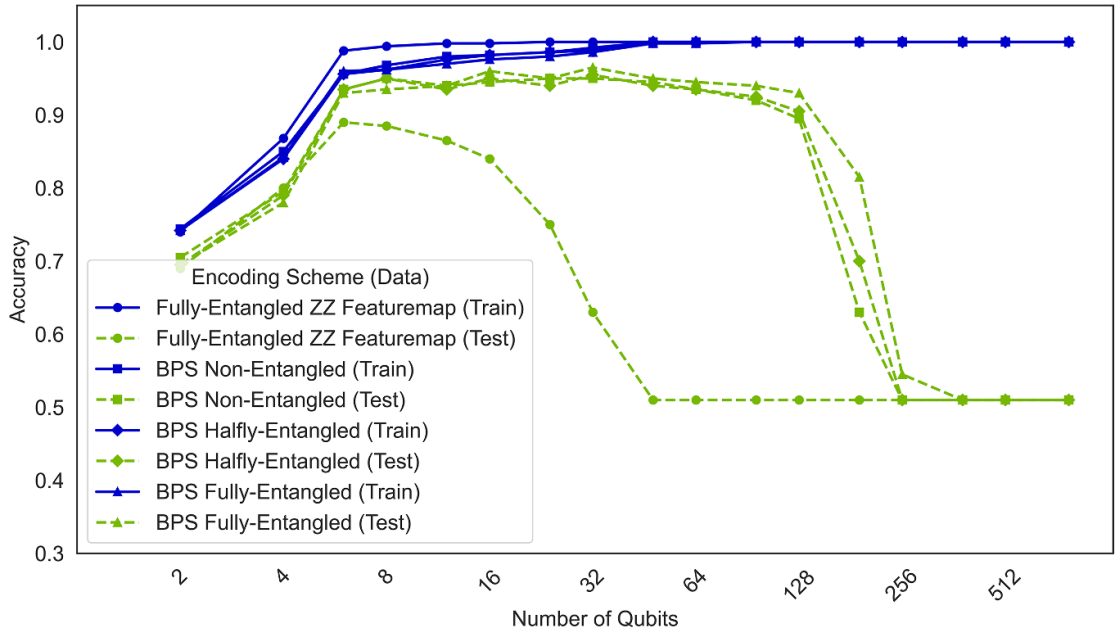

Figure 16. Classification accuracy vs. number of qubits for different encoding schemes. Solid lines denote training accuracy, while dashed lines denote test accuracy. The Fully-Entangled ZZ Feature Map (blue curves) consistently outperforms other approaches, emphasizing the importance of entanglement for large-scale QSVM. BPS schemes (green curves) help reduce circuit depth but may yield lower performance on complex datasets.

between performance and implementability, whereas the fully entangled BPS scheme is ultimately adopted in this research work on account of its superior accuracy to others.

Future studies might more thoroughly quantify the interplay between block separability and entanglement depth, revealing when strict block encoding (as prescribed in reference [38]) is most beneficial versus when the enhanced correlations of fully entangled circuits drive substantial improvements in performance.

ORCID iDs

Tai-Yue Li https://orcid.org/0000-0002-1993-1863
Chun-Chieh Wang https://orcid.org/0000-0001-5099-876X
Chun-Yu Lin https://orcid.org/0000-0002-7489-7418

References

[1] Jordan M I and Mitchell T M 2015 Machine learning: trends, perspectives and prospects *Science* **349** 255–60
[2] MacKay D J C 2003 *Information Theory, Inference and Learning Algorithms* (Cambridge University Press)
[3] Hearst M A, Dumais S T, Osuna E, Platt J and Scholkopf B 1998 Support vector machines *IEEE Intell. Syst. Appl.* **13** 18–28
[4] Ghaddar B and Naoum-Sawaya J 2018 High dimensional data classification and feature selection using support vector machines *Eur. J. Oper. Res.* **265** 993–1004
[5] Gaye B, Zhang D, Wulamu A and Tsai S-B 2021 Improvement of support vector machine algorithm in big data background *Math. Prob. Eng.* **2021** 1–9
[6] Rebentrost P, Mohseni M and Lloyd S 2014 Quantum support vector machine for big data classification *Phys. Rev. Lett.* **113** 130503
[7] Wang X, Du Y, Luo Y and Tao D 2021 Towards understanding the power of quantum Kernels in the NISQ era *Quantum* **5** 531
[8] Li T, reddy Mekala V, Ng K and Su C 2022 Classification of tumor metastasis data by using quantum kernel-based algorithms *2022 IEEE 22nd Int. Conf. on Bioinformatics and Bioengineering (BIBE) (IEEE)* pp 351–4
[9] Li Z, Liu X, Xu N and Du J 2015 Experimental realization of a quantum support vector machine *Phys. Rev. Lett.* **114** 140504
[10] Ding C, Bao T-Y and Huang H-L 2021 Quantum-inspired support vector machine *IEEE Trans. Neural Netw. Learn. Syst.* **33** 7210–22
[11] Chen K-C, Xu X, Makhanov H, Chung H-H and Liu C-Y 2024 Quantum-enhanced support vector machine for large-scale multi-class stellar classification *Int. Conf. on Intelligent Computing* (Springer) pp 155–68
[12] Preskill J 2018 Quantum computing in the NISQ era and beyond *Quantum* **2** 79
[13] Kjaergaard M, Schwartz M E, Braumüller J, Krantz P, Wang J I-J, Gustavsson S and Oliver W D 2020 Superconducting qubits: current state of play *Annu. Rev. Condens. Matter Phys.* **11** 369–95
[14] Bruzewicz C D, Chiaverini J, McConnell R and Sage J M 2019 Trapped-ion quantum computing: progress and challenges *Appl. Phys. Rev.* **6** 021314
[15] Evered S J *et al* 2023 High-fidelity parallel entangling gates on a neutral-atom quantum computer *Nature* **622** 268–72

[16] Cai Z, Babbush R, Benjamin S C, Endo S, Huggins W J, Li Y, McClean J R and O'Brien T E 2023 Quantum error mitigation *Rev. Mod. Phys.* **95** 045005
[17] Chen K-C 2023 Short-depth circuits and error mitigation for large-scale ghz-state preparation and benchmarking on ibm's 127-qubit system *2023 IEEE Int. Conf. on Quantum Computing and Engineering (QCE) vol 2 (IEEE)* vol 2 pp 207–10
[18] Souza A M, Alvarez G A and Suter D 2011 Robust dynamical decoupling for quantum computing and quantum memory *Phys. Rev. Lett.* **106** 240501
[19] Wei Zhong Lau J, Hwee Lim K, Shrotriya H and Chuan Kwek L 2022 Nisq computing: where are we and where do we go? *AAPPS Bull.* **32** 27
[20] Chen S, Cotler J, Huang H-Y and Li J 2023 The complexity of nisq *Nat. Commun.* **14** 6001
[21] Chen K-C, Li X, Xu X, Wang Y-Y and Liu C-Y 2024 Quantum-classical-quantum workflow in quantum-hpc middleware with gpu acceleration *2024 Int. Conf. on Quantum Communications, Networking and Computing (QCNC) (IEEE)* pp 304–11
[22] Lykov D, Shaydulin R, Sun Y, Alexeev Y and Pistoia M 2023 Fast simulation of high-depth qaoa circuits *Proc. SC'23 Workshops of The Int. Conf. on High Performance Computing, Network, Storage and Analysis* pp 1443–51
[23] Chen Z-Y, Zhou Q, Xue C, Yang X, Guo G-C and Guo G-P 2018 64-qubit quantum circuit simulation *Sci. Bull.* **63** 964–71
[24] Bayraktar H, Charara A, Clark D, Cohen S, Costa T, Fang Y-L L, Gao Y, Guan J, Gunnels J, Haidar A *et al* 2023 cuquantum sdk: a high-performance library for accelerating quantum science *2023 IEEE Int. Conf. on Quantum Computing and Engineering (QCE) vol 1 (IEEE)* vol 1 pp 1050–61
[25] Jones T, Brown A, Bush I and Benjamin S C 2019 Quest and high performance simulation of quantum computers *Sci. Rep.* **9** 10736
[26] Jamadagni Gangapuram A J, Läuchli A and Hempel C 2024 Benchmarking quantum computer simulation software packages: State vector simulators *SciPost Phys. Core* **7** 075
[27] Vasques X, Paik H and Cif L 2023 Application of quantum machine learning using quantum kernel algorithms on multiclass neuron m-type classification *Sci. Rep.* **13** 11541
[28] Huck Yang C-H, Li B, Zhang Y, Chen N, Sainath T N, Marco Siniscalchi S and Lee C-H 2023 A quantum kernel learning approach to acoustic modeling for spoken command recognition *ICASSP 2023-2023 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP) (IEEE)* pp 1–5
[29] Wu S L *et al* 2021 Application of quantum machine learning using the quantum kernel algorithm on high energy physics analysis at the LHC *Phys. Rev. Res.* **3** 033221
[30] Havlíček V, Córcoles A D, Temme K, Harrow A W, Kandala A, Chow J M and Gambetta J M 2019 Supervised learning with quantum-enhanced feature spaces *Nature* **567** 209–12
[31] Liu Y, Arunachalam S and Temme K 2021 A rigorous and robust quantum speed-up in supervised machine learning *Nat. Phys.* **17** 1013–7
[32] Duan B, Yuan J, Yu C-H, Huang J and Hsieh C-Y 2020 A survey on hhl algorithm: From theory to application in quantum machine learning *Phys. Lett.* A **384** 126595
[33] Gentinetta G, Thomsen A, Sutter D and Woerner S 2024 The complexity of quantum support vector machines *Quantum* **8** 1225
[34] Kim J-S, McCaskey A, Heim B, Modani M, Stanwyck S and Costa T 2023 Cuda quantum: The platform for integrated quantum-classical computing *2023 60th ACM/IEEE Design Automation Conf. (DAC)* (IEEE) pp 1–4
[35] Wille R, Van Meter R and Naveh Y 2019 Ibm's qiskit tool chain: working with and developing for real quantum computers *2019 Design, Automation & Test in Europe Conf. & Exhibition (DATE) (IEEE)* pp 1234–40
[36] Bergholm V *et al* 2018 Pennylane: automatic differentiation of hybrid quantum-classical computations (arXiv:1811.04968)
[37] Isakov S V *et al* 2021 Simulations of quantum circuits with approximate noise using qsim and cirq (arXiv:2111.02396)
[38] Martyn J, Vidal G, Roberts C and Leichenauer S 2020 Entanglement and tensor networks for supervised image classification (arXiv:2007.06082)
[39] Suzuki T, Miyazaki T, Inaritai T and Otsuka T 2023 Quantum ai simulator using a hybrid CPU–FPGA approach *Sci. Rep.* **13** 7735
[40] Thanasilp S S W, Cerezo M and Holmes 2024 Exponential concentration in quantum kernel methods *Nat. Commun.* **15** 5200
[41] Shaydulin R *et al* 2023 Evidence of scaling advantage for the quantum approximate optimization algorithm on a classically intractable problem (arXiv:2308.02342)