# Generalized Quantum Hadamard Test for Machine Learning

Vivek Mehta[1], Arghya Choudhury[1], Utpal Roy[1*]

[1*]Department of Physics, Indian Institute of Technology Patna, Bihta, Patna, 801106, Bihar, India.

*Corresponding author(s). E-mail(s): uroy@iitp.ac.in;
Contributing authors: vivek_1921ph05@iitp.ac.in; arghya@iitp.ac.in;

**Abstract**

Quantum machine learning models are designed for performing learning tasks. Some quantum classifier models are proposed to assign classes of inputs based on fidelity measurements. Quantum Hadamard test is a well-known quantum algorithm for computing these fidelities. However, the basic requirement for deploying the quantum Hadamard test maps input space to $\boldsymbol{L2}$-normalize vector space. Consequently, computed fidelities correspond to cosine similarities in mapped input space. We propose a quantum Hadamard test with the additional capability to compute the inner product in bounded input space, which refers to the Generalized Quantum Hadamard test. It incorporates not only $\boldsymbol{L2}$-normalization of input space but also other standardization methods, such as Min-max normalization. This capability is raised due to different quantum feature mapping and unitary evolution of the mapped quantum state. We discuss the quantum circuital implementation of our algorithm and establish this circuit design through numerical simulation. Our circuital architecture is efficient in terms of computational complexities. We show the application of our algorithm by integrating it with two classical machine learning models: Logistic regression binary classifier and Centroid-based binary classifier and solve four classification problems over two public-benchmark datasets and two artificial datasets.

**Keywords:** Quantum Hadamard Test, Amplitude encoding, Machine Learning

## 1 Introduction

Quantum machine learning emerges as a field where we exploit the mysterious phenomena of quantum mechanics, specifically quantum superposition and entanglement, to provide quantum advantages for solving machine learning problems with less computational complexity than classical machine learning models [1, 2]. Hence, we see that quantum mechanics is applied other than its conventional purpose for describing the microscopic physical system. A variety of quantum classifiers are tested in quantum machine learning such as quantum algebraic-based classifiers [3–6], quantum centroid-based classifiers [7–9], quantum neural networks [10–15], and variational quantum classifiers[16–20]. Often, variational classifiers are designated as quantum neural network models. In particular, in quantum centroid-based classifiers [21, 22], we measure the similarities of unseen input to each input of both classes and assign the label to that class whose inputs have high similarity with it. These similarities are usually determined by cosine similarity since we use *amplitude encoding* to map the input space to the quantum Hilbert space. $L2$-normalization of input space is the classical pre-processing requirement for amplitude encoding. Once the classical information is embedded into the probability amplitude of the basis of quantum states, we compute their similarities via fidelities using the quantum algorithm, known as the quantum Hadamard test. This amplitude encoding is used not only with quantum centroid-based classifiers but also with other classes of quantum classifiers too [23].

In this paper, we propose the quantum Hadamard test with additional capability for machine learning that computes the inner product in the bounded real input space where cosine similarity is taken as a *special case* when our input space is $L2$ normalized [21, 24]. Bounded real input space means components of any vector that belongs to this space should be bounded between -1 to 1. This version of the quantum Hadamard test refers to the Generalized quantum Hadamard test (GQHT) since it generalizes the utility of the already well-known quantum Hadamard test (QHT) beyond $L2$ normalized inputs. For instance, we can state that GQHT incorporates not only $L2$ normalization of input space but also other standardization techniques, such as Min-max normalization for machine learning tasks, since in some cases $L2$ normalization of inputs which is a basic requirement of QHT, changes the distribution of inputs in the input space, consequently important information may also be destroyed due to this normalization, while it does not happen with the Min-max normalization method. We validate the effectiveness of our GQHT algorithm for classification tasks over the public-benchmark datasets (Iris and Seeds) as well as synthesized datasets (Blobs and Two Half Moons), by incorporating it as subordinate with two linear classifiers: a parametric classification algorithm, *Logistic regression binary classifier* [25] and a non-parametric classification algorithm, *Centroid-based binary classifier* [26]. Our sole purpose for using these two algorithms is to show the workings of our scheme for machine learning when we consider the bounded input vector other than the $L2$ normalized ones. Beyond machine learning, we can use this GQHT algorithm anywhere, when we require an inner product between bounded vectors. The additional capability of our algorithm arises due to the exploitation of another quantum feature mapping scheme instead of the amplitude encoding scheme. We also discuss the quantum circuit architecture for implementing our encoding scheme that uses the quantum diagonal computation method. This

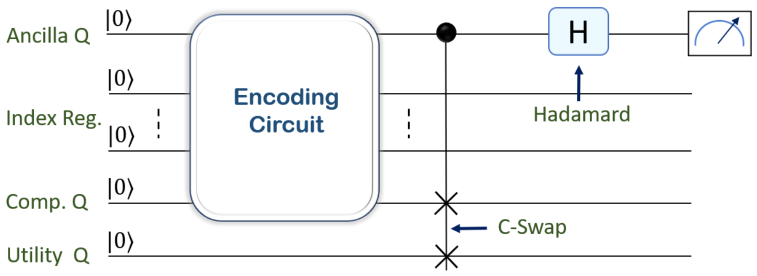

**Fig. 1** Schematic for the quantum circuit for GQHT algorithm. The abbreviation is used here: Ancilla qubit as Ancilla Q, Index Register as Index Reg., Component qubit as Comp. Q and Utility qubit as Utility Q.

architecture scheme provides certain circuit-complexity advantages over the amplitude encoding scheme like fewer interacting qubits and low classical overhead for designing the quantum circuit. Hence, GQHT can be used in place of QHT even if we are working with $L2$ normalized input space like for a Quantum centroid-based classifier for exploiting the efficiency of GQHT circuital implementation. Indeed, a quantum algorithm that exploits fewer resources is beneficial. We perform numerical simulations of quantum circuits that establish the workability of the classical information mapping scheme and our generalized algorithm. We use PennyLane, a Python-based quantum machine learning library for numerical simulations [27].

We organize this paper as follows: In Section (2), we discuss the GQHT algorithm and also compare it with the traditional QHT. This discussion will be carried out on the quantum circuital model for implementation of GQHT, highlighting the advantages that arise from its design. In the next Section (3), we discuss the training and testing of two binary classifiers: the Logistic regression binary classifier and the Centroid-based binary classifier. These classifiers use our GQHT as an important subordinate to get quantum advantages like less space complexity and faster computation. We also evaluate their performance on both public-benchmark datasets: the Iris dataset and the Seeds dataset, and artificial datasets: the Blobs dataset and the Two Half Moons datasets. Finally, we present our concluding remarks in the last section (4).

## 2 Generalize Quantum Hadamard Test

We begin by computing the inner product between two *bound* vectors, $\mathbf{x_p}, \mathbf{x_q}$, which belong to a set of $N$-dimensional real space, $X \subset \mathbb{R}^N$. We need to use a quantum circuit-based quantum feature mapping scheme $U_X$, that maps this input real space to quantum composite Hilbert space, $\mathcal{H}$; $U_X : X \to \mathcal{H}$, where all the input data points are represented as uniform superposition. This composite Hilbert space consists of two subspaces, one subspace corresponds to a single ancilla qubit, while the other corresponds to a quantum register that stores the individual quantum states as

$$|\mathbf{x}\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle \left( x_j |0\rangle - \sqrt{1 - x_j^2}\, |1\rangle \right), \tag{1}$$

where, $|j\rangle \in \{0,1\}^n$ are the computational basis of a quantum Hilbert space of dimension $2^n$, which consists of $n = \lceil log_2 N \rceil$ single qubits, which is used to enumerate the components of the given vector, and the value of their corresponding components, $x_j$ are stored in another single qubit, referred to as *component* qubit. Here, we need to pad the input vectors with zeros when the number of components of the real input space is less than the dimension of quantum Hilbert space, $N < 2^n$. The quantum space encoding unitary mapping circuit maps the reference state, $|0\rangle^{\otimes(n+2)}$ to

$$\begin{aligned} |\Psi_0\rangle &= U_X |0\rangle^{\otimes(n+2)} \\ &= \frac{1}{\sqrt{2}} \left[ |0\rangle |\mathbf{x}_p\rangle + |1\rangle |\mathbf{x}_q\rangle \right] \end{aligned} \tag{2}$$

Once we encode the input space, then we now prepare for the next stage where we transform the quantum state Eq. (2) as

$$|\psi_1\rangle = H \; C - swap \, |\Psi_0\rangle \, |0\rangle \, , \tag{3}$$

$$= \frac{1}{2} \left[ |0\rangle \left( |\tilde{\mathbf{x}}_p\rangle + |\tilde{\mathbf{x}}_q\rangle \right) + |1\rangle \left( |\tilde{\mathbf{x}}_p\rangle - |\tilde{\mathbf{x}}_q\rangle \right) \right] . \tag{4}$$

Here, $H$ and $C - swap$ are Hadamard gate and controlled-swap gate, respectively. We introduce another qubit that we call *utility* qubit in Eq. (3). We apply a three-qubit $C - swap$ gate, where ancilla is a control qubit and its target qubits are component qubit and utility qubit, and a single $H$ gate on ancilla qubit in sequence to get the quantum state in Eq. (4). In Eq. (4), two states are

$$\begin{aligned} |\tilde{\mathbf{x}}_p\rangle &= \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle \left( x_{pj} |0\rangle - \sqrt{1 - x_{pj}^2} |1\rangle \right) |0\rangle \, , \text{ and} \\ |\tilde{\mathbf{x}}_q\rangle &= \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle |0\rangle \left( x_{qj} |0\rangle - \sqrt{1 - x_{qj}^2} |1\rangle \right) . \end{aligned}$$

Next, we calculate the expectation value of the Pauli-Z operator acting over the ancilla qubit. The spectral decomposition Pauli-Z operator is a linear composition of computational basis projection operators, $\hat{Z} = |0\rangle\langle 0| - |1\rangle\langle 1|$. Hence, the expectation value of the Pauli-Z operator on the ancilla qubit is

$$\langle \hat{Z} \rangle = Tr(|\Psi_1\rangle \langle \Psi_1| \, \hat{Z}) \equiv \langle \Psi_1 | \hat{Z} | \Psi_1 \rangle$$

$$= |\langle 0|\Psi_1\rangle|^2 - |\langle 1|\Psi_1\rangle|^2 \tag{5}$$

$$= \frac{1}{2^n} \langle \mathbf{x}_p, \mathbf{x}_q \rangle, \tag{6}$$

where $Tr(.)$ is an trace operation. From Eq. (6), we get the inner product of two vectors, $\langle \mathbf{x}_p, \mathbf{x}_q \rangle$, form our Generalized quantum interference test. Since from Eq. (5), we find that the information of this inner product is concealed in the probability amplitudes of the computational basis of ancilla qubit, therefore we need to run this algorithm from a large but fixed number of times and use statistical inference to get the estimated value of probability amplitude. Fig. (1) depicts the GQHT algorithm.

**Comparison with QHT:** We now compare our protocol with the well-known and established quantum similarity-measurement protocol, the quantum Hadamard test. In this protocol, we need to compute the $L2$-norm of each input, before mapping it to the quantum state; and so, the classical information-embedded quantum state that represents the $L2$-normalized input is,

$$|\mathbf{x}'\rangle = \sum_{j=0}^{2^n-1} x'_j \, |j\rangle \,, \tag{7}$$

where $(.)'$ denotes that this vector is $L2$-normalized. This quantum feature mapping refers to amplitude encoding, where the components of the normalized vector are stored in the probability amplitude of the basis of $n$-qubit register, therefore it does not require an extra qubit, say component qubit as it is required in our protocol to store the information. It is important to note that amplitude encoding is a linear mapping of input space to quantum Hilbert space, while our quantum feature mapping is nonlinear, Eq. (1). GQHT does not use this extra component $\sqrt{1-x_j^2}$, however, this nonlinear quantum feature mapping allows it to compute inner products beyond $L2$ normalized inputs. Let's start the steps to get the similarity between two vectors, $|\mathbf{x}'\rangle_p$ and $|\mathbf{x}'\rangle_q$, using the quantum Hadamard test. Here, the first step is identical to our first step, as shown in Eq. (2). For the second step, since we do not require utility qubit and hence there is no need for a controlled-swap gate. Therefore, we only apply a Hadamard gate on the ancilla qubit and then in the final step, measure the expectation value of the Pauli-Z operation on the ancilla qubit, which returns

$$\langle \hat{Z} \rangle = \langle \mathbf{x}'_p, \mathbf{x}'_q \rangle. \tag{8}$$

On comparing the basis of qubit requirement, the quantum Hadamard test requires two fewer qubits as well as one less transformation. These benefits also limit its capacity to work with a more generalized representation of inputs as we cover with our protocol. In the realm of quantum machine learning, our protocol can replace the quantum Hadamard test with an additional need of scaling factor $\frac{1}{2^n}$. In a nutshell, we may find that where the quantum Hadamard test is used for quantum machine learning tasks, there our protocol may take its place. In the following section where we discuss the quantum circuit architecture of GQHT.

## 2.1 Quantum Circuit Design for GQHT

In this subsection, we discuss the design of a quantum circuit for implementing our protocol. It is crucial to construct an efficient circuit using the universal set of gates, since a non-efficient implementation may forbid getting fruitful quantum advantage. The efficient circuit design should minimize the requirement of two-qubit gates ( two-qubit gates since it requires quantum correlation) while keeping the depth of the circuit

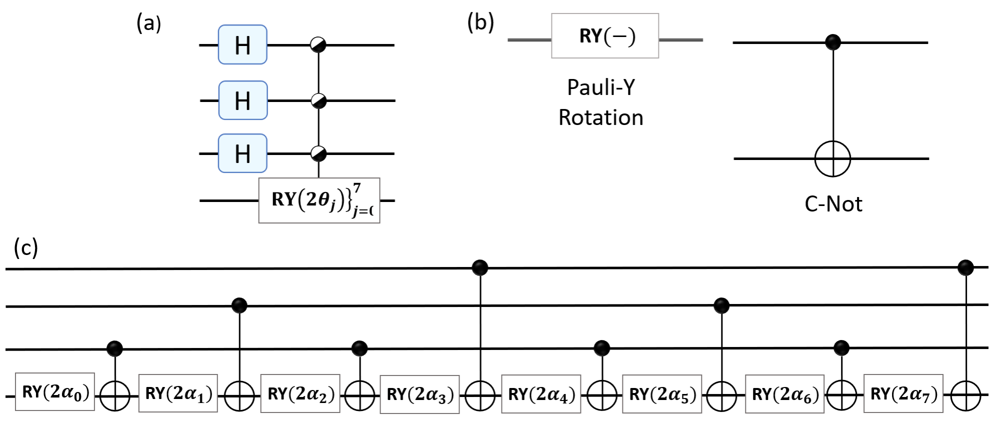

**Fig. 2** The compact form of the encoding map circuit is shown in figure **(a)**. In figure **(b)**, two gates are shown that are required for efficient decomposition. The quantum circuit architecture in figure (**c**), for efficient decomposition of the 3-qubits uniformly-controlled Pauli-Y rotation gates, is shown in figure (**a**).

low. The consequence of efficient circuit design is to save from the risk of quantum noises along with keeping the cost of valuable quantum resources in check.

In our protocol, only the input space encoding unitary map, $U_X$ requires special attention for quantum circuit designing. The quantum state representation of the input space, as shown in Eq. (2), provides the evidence of use of diagonal unitary evolution of the reference state. Hence, the encoding map, $U_X$, requires a diagonal computation encoding circuit scheme. To illustrate this scheme, we design a prototypical quantum circuit. Fig. (2, a) shows the compressed form of the unitary circuit, $U_X$. From Fig. (2, a), we see that it consists of two parts: in the first part, it requires the application of Hadamard gates on subsystems, and then needs to apply 3-qubits uniformly-controlled parameterized Pauli-Y rotations in sequence. Here, we need to get the efficient decomposition of this sequence of uniformly controlled gates in terms of single parameterized Pauli-Y rotation gates and two-qubit C-not gates, as shown in Fig (2, b). Möttönen, *et. al.* [28] proposed an efficient protocol to get such decomposition. Such a decomposition for 3-qubits uniformly-controlled rotation gates is shown in Fig. (2, c). It is noted that we need to apply the angle vector, $\boldsymbol{\theta}$, such that its entries are $\theta_j = arccos(x_j)$, as shown in (2, a) to represent the quantum state in Eq. (2), but we see from Fig. (2, c), we are applying angle vector, $\boldsymbol{\alpha}$. This is because entries of $\alpha_j, j = 0, \ldots, 7$, is a linear combination of components in angle vector $\boldsymbol{\theta}$. The relationship between three vectors is given by the linear transformation,

$$M\boldsymbol{\alpha} = \boldsymbol{\theta}, \tag{9}$$

where $M$ is $8\times8$ orthogonal and unitary matrix. Elements of this matrix, as well as the design of our quantum circuit, are driven from the representation of the computation basis of 3-qubit controlled quantum state systems in terms of Gray code, as shown in Table (1). Corresponding to the flip of a single bit in successive Gray code order $(g_j)$

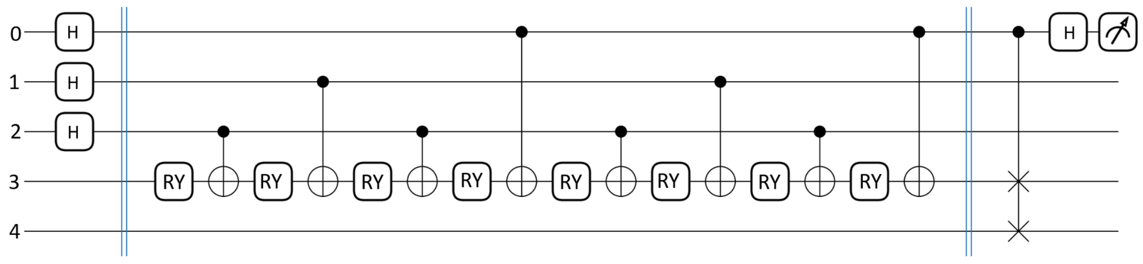

**Fig. 3** This is a quantum circuit to implement the GQHT algorithm for getting the inner product between two vectors having four components. This quantum circuit develops directly in PennyLane during the numerical simulation. Here, the indices of qubits 0, (1,2), 3, and 4 correspond to ancilla qubit, index register, component qubit, and utility qubit, respectively.

determines the control-qubit place in the quantum circuit architecture, as illustrated in Table (1) and Fig. (2, c). Following we get the components of matrix M, as

$$M_{sq} = \frac{(-1)^{b_s \cdot g_q}}{2^3}, \quad s, q = 0, \ldots, 7, \tag{10}$$

where $s$ and $q$ are arbitrary indices, and $b_s.g_q$ is element-wise binary dot product. Matrix M is shown in Table (2) is the required linear transform as shown in Eq. (9). It is a real matrix that belongs to an orthogonal class, and hence its inverse $M^{-1}$ is equal to its transpose, $M^T$. Fianlly, we compute the $\boldsymbol{\alpha}$ in linear combination of $\boldsymbol{\theta}$ using, $\boldsymbol{\alpha} = M^T\boldsymbol{\theta}$. In addition, we need not compute each element explicitly using Eq. (10). However, we use a trick that cancels the requirement to explicitly compute each component of this matrix. The trick is that we permute the columns of the corresponding dimension Hadamard gate according to the order correspondence between gray and binary codes. This order-correspondence can be seen in columns $IV$ and $V$ of Table (1) and correspondingly columns-wise permuted matrix is seen in Table (2). Both mathematical tricks, regarding the matrix inverse and preparation of the matrix, minimize the classical computing cost of getting $\boldsymbol{\alpha}$ in terms of $\boldsymbol{\theta}$. Additionally, to further reduce computation, we avoid repeatedly computing the matrix product $M^T\boldsymbol{\theta}$. Instead, we prepare a function that performs the matrix product for a given matrix $M^T$ and variable vector $\boldsymbol{\theta}$. This function takes $\boldsymbol{\theta}$ as an argument and returns the vector $\boldsymbol{\alpha}$.

Now, we discuss the advantages of our quantum circuit scheme concerning the amplitude-encode quantum states required for implementing the QHT algorithm [29]. These advantages are as follows:

1. It requires a less-connected quantum register. From Fig. (2), we conclude that a large number of qubits need to be connected with a single qubit, rather than to each other.
2. The depth of our quantum circuit architecture is low and consequently low risk of quantum decoherence. Our quantum encoding circuit's depth is $\mathcal{O}(N)$ to map any $N$-dimensional input space. The depth of our quantum circuit architecture is

| MSB | MB | LSB | Gray code | Binary code |
|---|---|---|---|---|
| 0 | 0 | 0 | $g_0$ | $b_0$ |
| 0 | 0 | 1 | $g_1$ | $b_1$ |
| 0 | 1 | 1 | $g_2$ | $b_3$ |
| 0 | 1 | 0 | $g_3$ | $b_2$ |
| 1 | 1 | 0 | $g_4$ | $b_6$ |
| 1 | 1 | 1 | $g_5$ | $b_7$ |
| 1 | 0 | 1 | $g_6$ | $b_5$ |
| 1 | 0 | 0 | $g_7$ | $b_4$ |

**Table 1** Gray code representation for three-bit strings. Abbreviations used are most significant bit as MSB, middle bit as MB, and least significant bit as LSB. The first three columns present gray code representation, while the last two columns show gray code order correspondence with binary code.

| | $g_0 \equiv b_0$ | $g_1 \equiv b_1$ | $g_2 \equiv b_3$ | $g_3 \equiv b_2$ | $g_4 \equiv b_6$ | $g_5 \equiv b_7$ | $g_6 \equiv b_5$ | $g_7 \equiv b_4$ |
|---|---|---|---|---|---|---|---|---|
| $b_0$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $b_1$ | 1 | -1 | -1 | 1 | 1 | -1 | -1 | 1 |
| $b_2$ | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 |
| $b_3$ | 1 | -1 | 1 | -1 | -1 | 1 | -1 | 1 |
| $b_4$ | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 |
| $b_5$ | 1 | -1 | -1 | 1 | -1 | 1 | 1 | 1 |
| $b_6$ | 1 | 1 | -1 | -1 | 1 | 1 | -1 | -1 |
| $b_7$ | 1 | -1 | 1 | -1 | 1 | -1 | 1 | -1 |

**Table 2** Orthogonal matrix gives the linear relation between $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$. $\frac{1}{2^3}$ should be multiplied with this matrix. This table is prepared in the following two steps; in step 1, we take $8 \times 8$ Hadamard transformation matrix, and in the final step, we permute the columns according to the gray code order correspondence with binary code as shown in the last two columns in Table (1). This correspondence relation between the two codes is also shown in the column header.

identical to the quantum circuit for amplitude-encode state preparation. However, it requires more classical overhead since we first need to compute required angles, as discussed in algorithm 1 of this paper [29] and then we need to use Möttönen, *et. al.* [28] scheme to deploy uniformly controlled rotation gates.

3. Whenever required, we may individually encode the information of vectors. See appendix (5.1), where we encode the training dataset and test point individually. This approach minimizes the requirement of classical overhead as well as quantum resources. However, we are not able to use this method to prepare amplitude-encoded quantum states.

It is worth mentioning that GQHT has a different quantum feature mapping protocol for mapping the input space. This quantum feature mapping scheme can be used as quantum feature mapping for variational classifiers [23].

Finally, we perform a numerical simulation of our circuit using *PennyLane*, a Python-based library for the numerical simulation of quantum circuits. Here, we compute scaled-inner product two vectors, $(0.1, 0.25, -1, 0.9)$ and $(-1, 0.75, 0.65, 0.89)$ using our protocol. The required quantum circuit is shown in Fig. (3). The result that we get after implementing this circuit is the required scaled-inner product between the given two vectors, 0.06. In the appendix (5.1), we briefly discuss the simultaneous

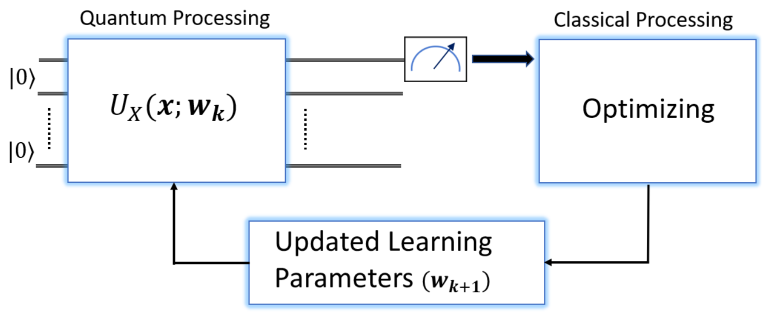
Quantum Processing
Classical Processing
$|0\rangle$
$|0\rangle$
$U_X(\boldsymbol{x}; \boldsymbol{w_k})$
Optimizing
Updated Learning Parameters $(\boldsymbol{w_{k+1}})$

**Fig. 4** It is a schematic of training of Logistic regression binary classifier. We use the quantum processing unit to run the GQHT algorithm that returns the inner product between inputs and learning parameters, then its result feeds into the classical processing unit for learning parameters optimization. It is an iterative process, therefore after each iteration, we get the updated learning parameters that again feed into the quantum processing unit.

computation of the inner product between the number of pairs of vectors. Next, when our vectors are binary-valued, they can be used directly to implement a quantum circuit model of an artificial neuron that is discussed by Tacchino *et. al.* [30].

## 3 Machine Learning Classifiers

It is impressive to show the working of our proposed Generalized test. It is used as the important subordinate of two classical machine learning classifiers: Logistic Regression binary classifier and Centroid-based binary classifier. Noted that integration of our quantum algorithm-based subordinate with required classical subordinates for the given classifiers put these on the same breed of machine learning models that follow the hybrid quantum-classical approach. The interesting aspect of our hybrid classifiers is that they have inherent basic characteristics like trainability and generalization from their corresponding classical counterparts. Our approach is underlying quantum-enhanced machine learning approaches where we are provided with quantum subordinates in view of getting computational speedups for established machine learning algorithms [3, 4, 31].

We now discuss the binary classification problem setup. Consider we have labelled a data set $\mathcal{D} = \{(\mathbf{x}^m, y^m)\}_{m=0}^{M-1} \in (X, \{0,1\})$, consists of $M$ inputs that live in $N$-dimensional real inner product space, $X \subset \mathbb{R}^N$. This labeled data set is split into two sets, as the training set $(\mathcal{T})$ and test set $(\mathcal{S})$. One part will be used for training during the training stage and the other part will be used to evaluate the performance of the trained model during the testing stage. A learning model is preferable when it performs well at the testing stage, i.e. on generalization.

### 3.1 Logistic Regression Binary Classifier

The Logistic regression binary classifier belongs to the class of parameterized machine learning algorithms that follow the approach of machine learning design as an optimization problem. We now describe its main components: hypothesis space, objective function, decision function, and performance analysis. Let's describe its first

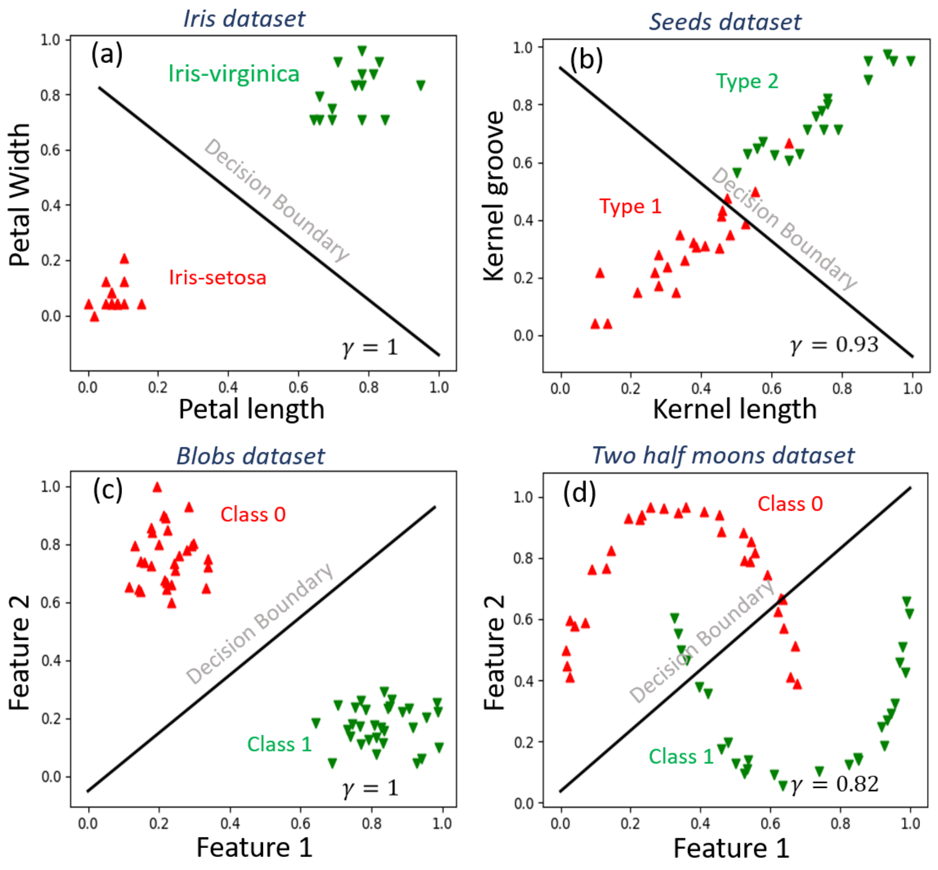

**Fig. 5** Decision boundary plots over four datasets are shown. Names of datasets and their generalization accuracies are mentioned above over subfigures and the lower-right side position of plots, respectively. In all plots, only test inputs are shown.

component, hypothesis space:

$$h(\mathbf{x}^m; \mathbf{w}, w_0) = \sigma(\langle \mathbf{w}, \mathbf{x}^m \rangle + w_0), \tag{11}$$

where $\mathbf{x}^m \in X$, $(\mathbf{w}, w_0)$ constitutes the learning parameters and bias, respectively that can be exploited to search the hypothesis space during optimization. The sigmoidal function, $\sigma$ is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

where $z$ is an arbitrary variable. For our work, we add one more dimension in the input space by providing an additional component $x_{m0} = 1$ to each input, such that the equation in parenthesis of Eq. (11) is rewritten as $\langle \mathbf{w}, \mathbf{x}^i \rangle + w_0 x_{m0}$. This inner product can be efficiently computed from our Generalized quantum interference test, while other calculations are carried out by classical processors. Once we come up with

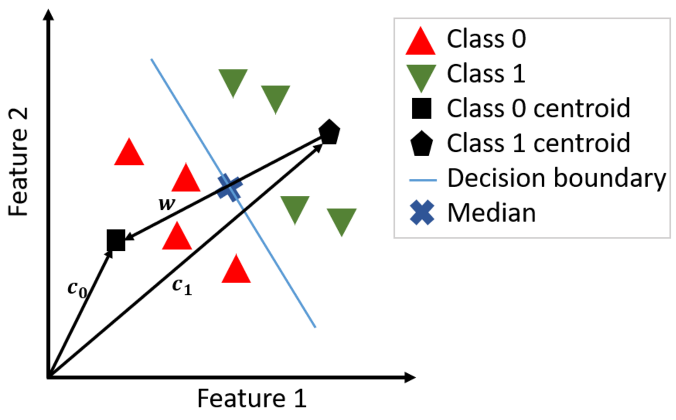

**Fig. 6** Schematic of Centroid-based binary classifier. We can see how two terms, median value $\mathbf{c}$ and $\mathbf{w}$ that is the difference between centroid-vectors $\mathbf{c_0}$ and $\mathbf{c_1}$ that decides the orientation of the decision boundary.

the hypothesis space, then we need to define the objective function,

$$J(\mathbf{w}, w_0, \mathcal{T}) = \frac{1}{B}\sum_{b=0}^{B-1}\mathcal{L}(h(\mathbf{x}^b; \mathbf{w}, w_0), y^b), \tag{12}$$

where $B$ is a randomly selected batch size of training inputs. We compute total loss over this batch inputs via the loss function, $\mathcal{L}()$. Here, a binary cross-entropy loss function is selected. Next, we select the optimizer which iteratively minimizes the objective function for the given number of times. At the end of this step, we come up with the optimized parameter value $\mathbf{w}^*, w_0^*$. We choose the Stochastic batch gradient descent as our optimizer. Finally, we need to design the decision function as

$$y'^m = \begin{cases} 1, & \sigma(h(\mathbf{x}^m; \mathbf{w}^*, w_0^*)) > 0.5 \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

To check the performance of our learning model on generalization, we conduct performance analysis over the test set, $\mathcal{S}$. We run this algorithm in four datasets: the first two belong to public-benchmark datasets such as the Iris and Seeds datasets that are shown in the first row of Fig.(5), while the last two datasets are artificial datasets like Blobs and two half moons dataset, shown in the second row of Fig. (5). Here, we use basic Python code to run this classifier. We use 70% of data for training and the remaining instances for testing.

It is important to highlight certain advantages of using GQHT as a subordinate in this classifier. Firstly, in comparison to classical machines, we minimize the space complexity exponentially, i.e., we need only $\mathcal{O}(log_2 N)$, the number of qubits for representing any arbitrary $N$-dimensional vector; whereas in the classical machine, we need given number $2^N$ in numbers for storing the indices of the vector plus additional binary strings to store their corresponding components with a given precision. Secondly, we discuss regarding the statistical learning perspective of our GQHT scheme. From Fig. (5), we find that we get the decision function in the Min-max pre-processed input space. There is no change in the distribution of input space due to Min-max normalization. Our Logistic regression binary classifier is inspired by the variational class of quantum classifiers, where the learning parameters of parameterized quantum circuits are iteratively optimized by classical processors. Implicitly, we can say that we propose a variational classifier that works on other than $L2$ normalized input space. On the other hand, we also find that the working of this classifier is identical to an artificial neuron, and hence we can exploit it to formalize a hybrid quantum-classical quantum neural network, as discussed by Tacchino *et. al.* [32].

## 3.2 Cetroid-based Binary Classifier

We adopt an interesting approach to designing a centroid-based binary classifier model that is different from the conventional centroid-based binary classifier model. This approach is driven from *Learning with Kernels* book [26]. The decision equation for assigning the label $y$ of test data point $\mathbf{x}$ by this model is,

$$y = sgn\langle(\mathbf{x} - \mathbf{c}, \mathbf{w}\rangle, \tag{14}$$

where two terms are elaborated as: $\mathbf{c} := (\mathbf{c}_0 + \mathbf{c}_1)/2$ is the mean of the centroids and $\mathbf{w} := (\mathbf{c}_0 - \mathbf{c}_1)$ is difference between centroids ( mean is synonyms of centroid). Here, $sgn(.)$ is a step function that returns the class label according to the sign of *arc cosine* of value under parenthesis of Eq. (14). Centroids for class 0 and class 1 are given, respectively, as

$$\mathbf{c}_0 = \frac{1}{M} \sum_{\{m|y^m=0\}} \mathbf{x}^m \tag{15}$$

$$\mathbf{c}_1 = \frac{1}{M} \sum_{\{m|y^m=1\}} \mathbf{x}^m \tag{16}$$

where $M$ is the number of samples in class 0 (class 1), i.e., our training data points are balanced. When we put the Eqs. (15) and (16) in Eq. (14), we get the equation

$$y = sgn\left(\frac{1}{M} \sum_{\{m|y^m=0\}} \langle \mathbf{x}^m, \mathbf{x}\rangle - \frac{1}{M} \sum_{\{m|y^m=1\}} \langle \mathbf{x}^m, \mathbf{x}\rangle + b\right), \tag{17}$$

where the offset, $b$ is defined as:

$$b = \frac{1}{2}\left(\frac{1}{M^2} \sum_{\{(m,\underset{\sim}{m})|y^m,y^{\underset{\sim}{m}}=0\}} \langle \mathbf{x}^m, \mathbf{x}^{\underset{\sim}{m}}\rangle - \frac{1}{M^2} \sum_{\{(m,\underset{\sim}{m})|y^m,y^{\underset{\sim}{m}}=1\}} \langle \mathbf{x}^m, \mathbf{x}^{\underset{\sim}{m}}\rangle\right). \tag{18}$$

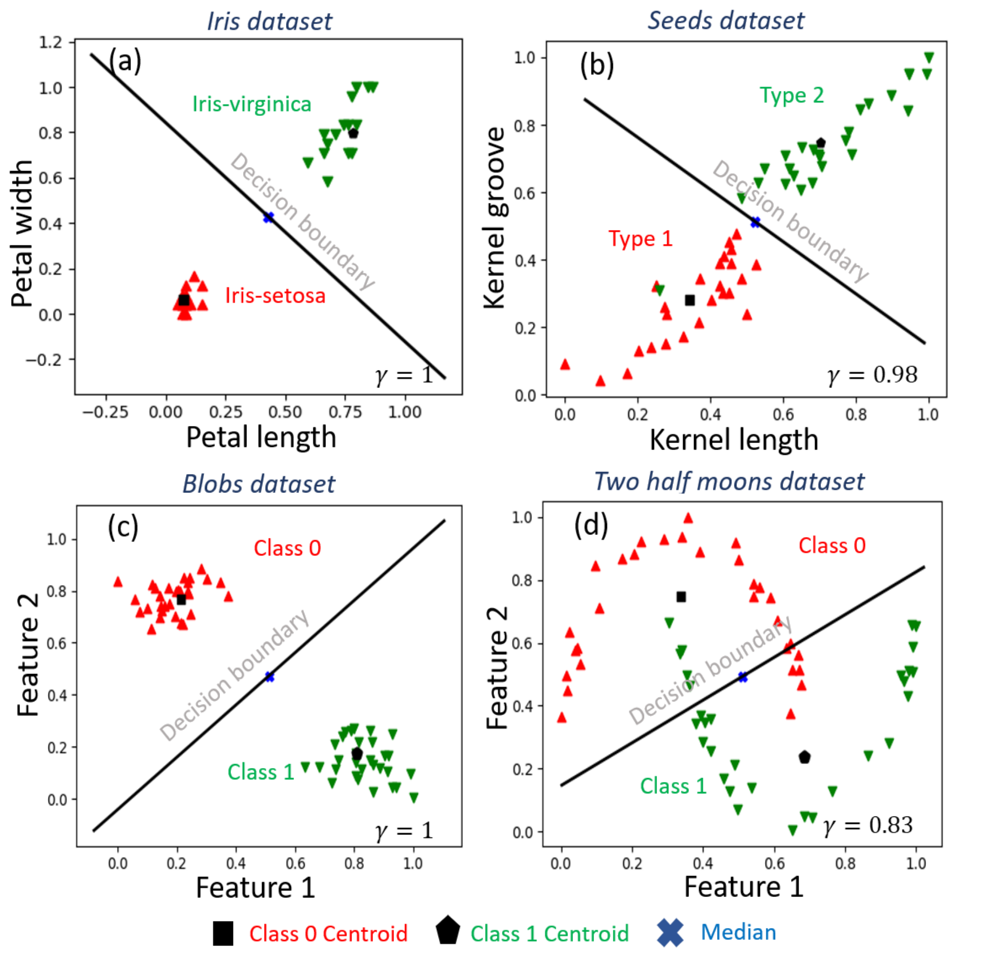

**Fig. 7** Decision boundary plots over four datasets are shown. The names of the dataset and their corresponding generalization accuracy are shown as titles over subfigures and as legends in the lower-right side of plots, respectively. Other notations like class centroids and medians are presented in the lower bar of the figure. In all plots, only test inputs are shown.

From Eqs. (17) and (18), we find that all the terms are redefined in terms of inner products between test data points to all training data points as in Eq. (17) or one training data point to all other training data points that belongs to the same class as in Eq. (18). These inner products can be implemented in one go due to quantum parallelism (see appendix (5.1)).

As the earlier classifier discussed in subsection (3.1), we again work with the same datasets, as shown in Fig. (7). Here, again we perform the Python-based code to do numerical simulation which returns inner products and these are used to get the class of test inputs. For this classifier, a balanced dataset is required. Except for the seeds dataset, other datasets are balanced. To address this, we balance the seeds dataset by selecting an equal number of inputs from both classes during training.

Finally, we talk about the merit of GQHT concerning this classifier. Firstly, it is important to note that this is not a computationally efficient approach, however, we are taking this in particular to exhibit the strength of our protocol to do a classification task when we work with only bounded input space. Here, we find that the model requirement of this classifier demands the inner product between bounded vectors. This is the merit of our algorithm which may work for this requirement along with other tasks where the QHT algorithm is already used like quantum centroid-based classifiers. Another important point is that when we integrate our GQHT algorithm for the quantum centroid-based quantum classifier, we can minimize the quantum resources and classical calculation overhead during quantum circuit implementation.

## 4 Conclusions

We design the Generalized version of the amplitude-mapped quantum Hadamard test. In addition, we discuss the efficient quantum circuit scheme for implementing this algorithm. We also validate the design of this scheme by numerical simulation of a prototype quantum circuit by using PennyLane, a Python-based quantum computation library. We further demonstrated the application of this scheme by incorporating it into two binary classifiers: Logistic regression binary classifier and Centroid-based binary classifier; and solving classification learning problems over four datasets. In addition, our GQHT may be applied in the field of quantum neural networks. Tacchino *et. al.* [30] presented the quantum circuital implementation of a binary-valued artificial neuron. They present the hypergraph-based implementation of this neuron which requires a lengthy classical backend. We can directly replace their quantum computational model with our GQHT. There is another artificial neuron quantum computational implementation method that is discussed by Yan *et. al.* [15], and our GQHT can replace the part of the algorithm where the inner product is computed. In summary, our GQHT algorithm shows great promise for a broad range of machine learning applications and its full potential can be further explored in future works.

## 5 Appendix

### 5.1 GQHT getting Inner Product between the Training set and a Test point

We compute the inner product of a training set, say $\mathcal{D} = \{\mathbf{x}_m\}$ consists of $M$ samples and a test point $\tilde{\mathbf{x}}$ simultaneously, using our GQHT algorithm. First, we need to represent the training dataset in a quantum state as

$$|\mathcal{D}\rangle = \frac{1}{\sqrt{2^{(p+n)}}} \sum_{m=0}^{2^p-1} |m\rangle \sum_{j=0}^{2^n-1} |j\rangle \left( x_{mj} |0\rangle - \sqrt{1 - x_{mj}^2} |1\rangle \right), \tag{19}$$

where the index of samples is stored in the first quantum register consisting of $p = \lceil log_2 M \rceil$ qubits, and the test point is stored as in Eq. (1). Then, we need to apply the GQHT algorithm as discussed in section (2). When we do get the Pauli-Z operator

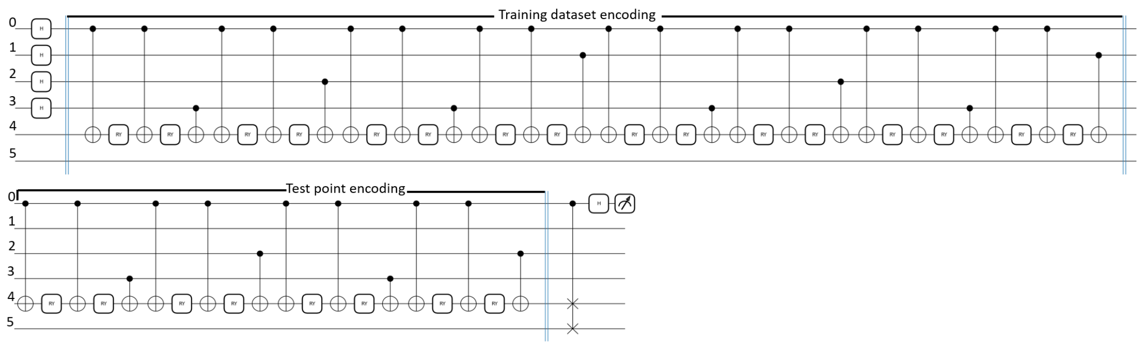

**Fig. 8** Quantum circuit schematic for getting the inner product between the training dataset and the test point. It is generated from the PennyLane. Index 0, 1, (2, 3), 4, and 5 correspond to ancilla qubit, sample index qubit, component index register, component qubit, and utility qubit, respectively. In the training dataset encoding section, we need to apply a 0 basis controlled Pauli-Y rotation gate, so it can be implemented by applying this sequence of gates $C - NotR_y(\theta)C - NotR_y(\theta)$, where $\theta$ is a parameter. While in the test point encoding, we need to apply 1 basis controlled Pauli-Y rotation gate, and it can be implemented with the same sequence of the gates as discussed earlier with a single change is that the first single qubit Pauli-Y rotation is replaced by its Hermitian conjugate version.

expectation value over the ancilla qubit, we get

$$\langle \hat{Z} \rangle = \frac{1}{2^{(p+m)}} \sum_{m=0}^{2^p-1} \langle \mathbf{x}_m, \tilde{\mathbf{x}} \rangle. \tag{20}$$

In Fig. (8), we illustrate a prototype where the training dataset consists of two vectors, vector 1 $(1.0, 0.25, -0.36, -0.98)$ and vector 2 $(-0.1, 0.37, 0.65, 0.45)$ and the test point is $(0.75, 0.1, 0.25, 0.25)$. The final result comes after the implementation of the quantum circuit is 0.069. What is important with quantum circuit implementation for this task is that we can encode the training dataset and test dataset *separately.* It is interesting since it can not be possible when we try to encode information in the probability amplitude of the quantum system, which is required for the QHT. It supports us to minimize the load of classical pre-processing.

## References

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

[2] Schuld, M., Petruccione, F.: Machine learning with quantum computers (2021)

[3] Rebentrost, P., Mohseni, M., Lloyd, S.: Quantum support vector machine for big data classification. Physical Review Letters **113**(13), 130503 (2014)

[4] Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum principal component analysis. Nature Physics **10**(9), 631–633 (2014)

[5] Li, Z., Liu, X., Xu, N., Du, J.: Experimental realization of a quantum support vector machine. Physical Review Letters **114**(14), 140504 (2015)

[6] Schuld, M., Sinayskiy, I., Petruccione, F.: Prediction by linear regression on a quantum computer. Physical Review A **94**(2), 022342 (2016)

[7] Schuld, M., Fingerhuth, M., Petruccione, F.: Implementing a distance-based classifier with a quantum interference circuit. Europhysics Letters **119**(6), 60002 (2017)

[8] Blank, C., Da Silva, A.J., Albuquerque, L.P., Petruccione, F., Park, D.K.: Compact quantum kernel-based binary classifier. Quantum Science and Technology **7**(4), 045007 (2022)

[9] Das, S., Zhang, J., Martina, S., Suter, D., Caruso, F.: Quantum pattern recognition on real quantum processing units. Quantum Machine Intelligence **5**(1), 16 (2023)

[10] Schuld, M., Sinayskiy, I., Petruccione, F.: Simulating a perceptron on a quantum computer. Physics Letters A **379**(7), 660–663 (2015)

[11] Mangini, S., Tacchino, F., Gerace, D., Macchiavello, C., Bajoni, D.: Quantum computing model of an artificial neuron with continuously valued input data. Machine Learning: Science and Technology **1**(4), 045008 (2020)

[12] Rebentrost, P., Bromley, T.R., Weedbrook, C., Lloyd, S.: Quantum hopfield neural network. Physical Review A **98**(4), 042308 (2018)

[13] Zhao, J., Zhang, Y.-H., Shao, C.-P., Wu, Y.-C., Guo, G.-C., Guo, G.-P.: Building quantum neural networks based on a swap test. Physical Review A **100**(1), 012334 (2019)

[14] Shao, C.: Data classification by quantum radial-basis-function networks. Physical Review A **102**(4), 042418 (2020)

[15] Yan, S., Qi, H., Cui, W.: Nonlinear quantum neuron: A fundamental building block for quantum neural networks. Physical Review A **102**(5), 052421 (2020)

[16] Benedetti, M., Lloyd, E., Sack, S., Fiorentini, M.: Parameterized quantum circuits as machine learning models. Quantum Science and Technology **4**(4), 043001 (2019)

[17] Mitarai, K., Negoro, M., Kitagawa, M., Fujii, K.: Quantum circuit learning. Physical Review A **98**(3), 032309 (2018)

[18] Havlíček, V., Córcoles, A.D., Temme, K., Harrow, A.W., Kandala, A., Chow, J.M., Gambetta, J.M.: Supervised learning with quantum-enhanced feature spaces. Nature **567**(7747), 209–212 (2019)

[19] Caro, M.C., Huang, H.-Y., Cerezo, M., Sharma, K., Sornborger, A., Cincio, L., Coles, P.J.: Generalization in quantum machine learning from few training data. Nature communications **13**(1), 4919 (2022)

[20] Huang, R., Tan, X., Xu, Q.: Variational quantum tensor networks classifiers. Neurocomputing **452**, 89–98 (2021)

[21] Park, D.K., Blank, C., Petruccione, F.: The theory of the quantum kernel-based binary classifier. Physics Letters A **384**(21), 126422 (2020)

[22] Blank, C., Park, D.K., Rhee, J.-K.K., Petruccione, F.: Quantum classifier with tailored quantum kernel. npj Quantum Information **6**(1), 41 (2020)

[23] Schuld, M., Bocharov, A., Svore, K.M., Wiebe, N.: Circuit-centric quantum classifiers. Physical Review A **101**(3), 032308 (2020)

[24] Rethinasamy, S., Agarwal, R., Sharma, K., Wilde, M.M.: Estimating distinguishability measures on quantum computers. Physical Review A **108**(1), 012409 (2023)

[25] Bishop, C.M., Bishop, H.: Deep learning: Foundations and concepts (2023)

[26] Scholkopf, B., Smola, A.J.: Learning with kernels: support vector machines, regularization, optimization, and beyond (2018)

[27] Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Ahmed, S., Ajith, V., Alam, M.S., Alonso-Linaje, G., AkashNarayanan, B., Asadi, A., et al.: Pennylane: Automatic differentiation of hybrid quantum-classical computations. arXiv preprint arXiv:1811.04968 (2018)

[28] Möttönen, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M.: Quantum circuits for general multiqubit gates. Physical Review Letters **93**(13), 130502 (2004)

[29] Araujo, I.F., Park, D.K., Petruccione, F., Silva, A.J.: A divide-and-conquer algorithm for quantum state preparation. Scientific reports **11**(1), 6329 (2021)

[30] Tacchino, F., Macchiavello, C., Gerace, D., Bajoni, D.: An artificial neuron implemented on an actual quantum processor. npj Quantum Information **5**(1), 26 (2019)

[31] Dunjko, V., Taylor, J.M., Briegel, H.J.: Quantum-enhanced machine learning. Physical Review Letters **117**(13), 130501 (2016)

[32] Tacchino, F., Barkoutsos, P., Macchiavello, C., Tavernelli, I., Gerace, D., Bajoni, D.: Quantum implementation of an artificial feed-forward neural network. Quantum Science and Technology **5**(4), 044010 (2020)