QNet: A Scalable and Noise-Resilient Quantum Neural Network Architecture for Noisy Intermediate-Scale Quantum Computers

Mahabubul Alam * and Swaroop Ghosh

Department of Electrical Engineering, School of Electrical Engineering and Computer Science, Pennsylvania State University, University Park, PA, United States

Quantum machine learning (QML) is promising for potential speedups and improvements in conventional machine learning (ML) tasks. Existing QML models that use deep parametric quantum circuits (PQC) suffer from a large accumulation of gate errors and decoherence. To circumvent this issue, we propose a new QML architecture called QNet. QNet consists of several small quantum neural networks (QNN). Each of these smaller QNN’s can be executed on small quantum computers that dominate the NISQ-era machines. By carefully choosing the size of these QNN’s, QNet can exploit arbitrary size quantum computers to solve supervised ML tasks of any scale. It also enables heterogeneous technology integration in a single QML application. Through empirical studies, we show the trainability and generalization of QNet and the impact of various configurable variables on its performance. We compare QNet performance against existing models and discuss potential issues and design considerations. In our study, we show 43% better accuracy on average over the existing models on noisy quantum hardware emulators. More importantly, QNet provides a blueprint to build noise-resilient QML models with a collection of small quantum neural networks with near-term noisy quantum devices.

Keywords: quantum machine learning, quantum neural network, parameterized quantum circuit, noisy intermediate-scale quantum, quantum-classical hybrid algorithms

1 INTRODUCTION

Quantum computing (QC) is one of the major transformative technologies. The community is seeking computational advantages with quantum computers (i.e., quantum supremacy) for practical applications. Recently, Google has claimed quantum supremacy with a 53-qubit quantum processor to complete a computational task (of no practical relevance though) in 200 s that may take 10K years on the state-of-the-art supercomputers [1]. This experiment has been a significant milestone for quantum computing.

Quantum machine learning (QML) is a promising application domain to archive quantum advantage with noisy quantum computers in the near term. Numerous QML models built upon parametric quantum circuits (PQC) are already present in the literature [2–6]. A PQC is a quantum circuit with parameterized gates as shown in Figure 1 (w1, w2, . . . are the tunable parameters). The parameters can be tuned to create the desired output state. Quantum Neural Network (QNN) is one of the most promising QML models that has gained significant attention in the past few years.
A conventional QNN consists of a data encoding circuit and a PQC followed by measurement operations (Figure 1). The data encoder encodes the classical data as a quantum state. The quantum state is transformed by changing the parameters of the PQC. The output state is retrieved through the measurements. QNN models are claimed to be more expressive than the classical neural networks [8–10]. In other words, QNN models have a higher capability to approximate the desired functionality (e.g., classifying data samples) compared to the classical models of a similar scale (e.g., with the same number of tunable parameters/weights). Numerous choices exist for the encoding circuits, PQC, and measurements to build a QNN [9, 11–13]. Recent works have demonstrated application of QNN in image classification [14, 15], drug discovery [16, 17], finance [18, 19], and many other industrial problems [20].

The near-term quantum devices have a limited number of qubits. Moreover, they suffer from various errors such as, decoherence, gate errors, measurement errors and crosstalk. John Preskill famously coined the term Noisy Intermediate-Scale Quantum or NISQ computers to refer to these machines [21]. The size of the quantum circuit (in terms of qubit count, gate count, and depth) that can be executed reliably on a quantum computer is limited by the hardware noise characteristics [22]. Therefore, the QNN models need to be small to exploit near-term devices for QML applications. However, the size of the conventional QNN model grows with the scale of the ML task. For example, we may require a 4-qubit QNN model to solve a 4-features ML task with 1 variable per qubit angle encoding (Figure 2B). A 1000-features problem will require a 1000-qubit circuit that may not fit in any near-term quantum devices. Even if the larger model fits in target hardware, it may not execute reliably due to a higher accumulation of gate errors and decoherence. Therefore, development of QNN models that use smaller quantum circuits is required.

Two variants of the conventional QNN model are widely used to address the above scalability issue: 1) reduction of data dimension using a classical algorithm (e.g., PCA) before training a QML [17,23,24], and 2) repeatedly uploading the...
high-dimensional data in a small number of qubits using sequential rotation operations [25–27]. Technically, both these approaches can exploit arbitrarily small quantum hardware to solve ML tasks of any scale (a single qubit can be used to build a classification model involving 1,000 features). However, they have significant limitations that restrict their use in practical problems. For example, the performance of approach 1) will largely depend on the number of principal components (PC) taken from the PCA. A lower number of PC may not explain the variance in the data efficiently. This can affect the overall performance of this hybrid model. If we choose a large number of PCs, the succeeding QNN model may not be that small. It may not even fit in the target hardware. Uploading many features on a small number of qubits using approach 2) will easily cross the coherence limit of the quantum devices if the number of features is large. Therefore, the model will essentially generate random outputs on hardware and may not be trainable.

To address the above challenges, we propose QNet - a scalable quantum neural network architecture for small noisy quantum computers (Figure 1). We draw inspiration from classical multilayer perceptron network (MLP) [28]. QNet consists of several smaller QNN’s that are distributed in multiple QNN Layers (similar to the hidden layers in MLP). The number of qubits in a QNN and the circuit depth can be chosen based on the target hardware characteristics. Each QNN takes a fraction of the input vector as inputs and generates a transformed output. All the QNN’s in a QNN Layer together generates a new representation in the target hardware. Uploading many features on a small number of qubits using approach 2) will easily cross the coherence limit of the quantum devices if the number of features is large. Therefore, the model will essentially generate random outputs on hardware and may not be trainable.

To address the above challenges, we propose QNet - a scalable quantum neural network architecture for small noisy quantum computers (Figure 1). We draw inspiration from classical multilayer perceptron network (MLP) [28]. QNet consists of several smaller QNN’s that are distributed in multiple QNN Layers (similar to the hidden layers in MLP). The number of qubits in a QNN and the circuit depth can be chosen based on the target hardware characteristics. Each QNN takes a fraction of the input vector as inputs and generates a transformed output. All the QNN’s in a QNN Layer together generates a new representation in the target hardware. Uploading many features on a small number of qubits using approach 2) will easily cross the coherence limit of the quantum devices if the number of features is large. Therefore, the model will essentially generate random outputs on hardware and may not be trainable.

To address the above challenges, we propose QNet - a scalable quantum neural network architecture for small noisy quantum computers (Figure 1). We draw inspiration from classical multilayer perceptron network (MLP) [28]. QNet consists of several smaller QNN’s that are distributed in multiple QNN Layers (similar to the hidden layers in MLP). The number of qubits in a QNN and the circuit depth can be chosen based on the target hardware characteristics. Each QNN takes a fraction of the input vector as inputs and generates a transformed output. All the QNN’s in a QNN Layer together generates a new representation in the target hardware. Uploading many features on a small number of qubits using approach 2) will easily cross the coherence limit of the quantum devices if the number of features is large. Therefore, the model will essentially generate random outputs on hardware and may not be trainable.

To address the above challenges, we propose QNet - a scalable quantum neural network architecture for small noisy quantum computers (Figure 1). We draw inspiration from classical multilayer perceptron network (MLP) [28]. QNet consists of several smaller QNN’s that are distributed in multiple QNN Layers (similar to the hidden layers in MLP). The number of qubits in a QNN and the circuit depth can be chosen based on the target hardware characteristics. Each QNN takes a fraction of the input vector as inputs and generates a transformed output. All the QNN’s in a QNN Layer together generates a new representation in the target hardware. Uploading many features on a small number of qubits using approach 2) will easily cross the coherence limit of the quantum devices if the number of features is large. Therefore, the model will essentially generate random outputs on hardware and may not be trainable.

To address the above challenges, we propose QNet - a scalable quantum neural network architecture for small noisy quantum computers (Figure 1). We draw inspiration from classical multilayer perceptron network (MLP) [28]. QNet consists of several smaller QNN’s that are distributed in multiple QNN Layers (similar to the hidden layers in MLP). The number of qubits in a QNN and the circuit depth can be chosen based on the target hardware characteristics. Each QNN takes a fraction of the input vector as inputs and generates a transformed output. All the QNN’s in a QNN Layer together generates a new representation in the target hardware. Uploading many features on a small number of qubits using approach 2) will easily cross the coherence limit of the quantum devices if the number of features is large. Therefore, the model will essentially generate random outputs on hardware and may not be trainable.

To address the above challenges, we propose QNet - a scalable quantum neural network architecture for small noisy quantum computers (Figure 1). We draw inspiration from classical multilayer perceptron network (MLP) [28]. QNet consists of several smaller QNN’s that are distributed in multiple QNN Layers (similar to the hidden layers in MLP). The number of qubits in a QNN and the circuit depth can be chosen based on the target hardware characteristics. Each QNN takes a fraction of the input vector as inputs and generates a transformed output. All the QNN’s in a QNN Layer together generates a new representation in the target hardware. Uploading many features on a small number of qubits using approach 2) will easily cross the coherence limit of the quantum devices if the number of features is large. Therefore, the model will essentially generate random outputs on hardware and may not be trainable.

To address the above challenges, we propose QNet - a scalable quantum neural network architecture for small noisy quantum computers (Figure 1). We draw inspiration from classical multilayer perceptron network (MLP) [28]. QNet consists of several smaller QNN’s that are distributed in multiple QNN Layers (similar to the hidden layers in MLP). The number of qubits in a QNN and the circuit depth can be chosen based on the target hardware characteristics. Each QNN takes a fraction of the input vector as inputs and generates a transformed output. All the QNN’s in a QNN Layer together generates a new representation in the target hardware. Uploading many features on a small number of qubits using approach 2) will easily cross the coherence limit of the quantum devices if the number of features is large. Therefore, the model will essentially generate random outputs on hardware and may not be trainable.

We make the following contributions in this paper: 1) present a scalable quantum neural network architecture (QNet) for noisy small quantum devices, 2) analyze the performance of QNet against a variety of design choices through numerical experiments, 3) compare the performance of QNet against conventional QNN, and two existing proposals to exploit small quantum computers for larger ML problems, and 4) discuss potential issues and future developments of QNet. We perform the numerical experiments using ideal simulators and hardware emulators (ibmq_melbourne, ibmq_bogota, ibmq_casablanca). An extensive set of classification datasets e.g., Iris, Wine, Breast Cancer, Digits, MNIST, and Fashion-MNIST is used to showcase the superior performance of QNet over the existing proposals.

We cover the basics on quantum computing and quantum neural networks in Section 2, discuss related works in Section 3, present our QNet architecture in Section 4, describe the results in Section 5, discuss potential issues, applications, and future development of QNet in Section 6, and draw our conclusions in Section 7.

2 PRELIMINARIES

Qubits, Quantum Gates, and Measurements: Qubit is analogous to classical bits. However, unlike a classical bit, a qubit can be in a superposition state i.e., a combination of $|0\rangle$ and $|1\rangle$ at the same time. A variety of technologies exist to realize qubits such as, superconducting qubits, trapped-ions, neutral atoms, silicon spin qubits, to name a few [29]. Quantum gates (e.g., single qubit Pauli-X gate or 2-qubit CNOT gate) modulate the state of qubits and thus perform computations. These gates can perform a fixed computation (e.g., an X gate flips a qubit state) or a computation based on a supplied parameter (e.g. the RY($\theta$) gate rotates the qubit along the Y-axis by $\theta$). A two-qubit gate changes the state of one qubit (target qubit) based on the current state of the other qubit (control qubit). For example, the CNOT gate flips the target qubit if the control qubit is in $|1\rangle$ state. A quantum circuit contains many gate operations.

Qubits are measured to retrieve the final state of a quantum program. A complete state extraction requires the simultaneous computation of the position of a qubit along the X, Y, and Z-axis in the Bloch sphere. Unfortunately, it is impossible to simultaneously compute these components, thus requiring many executions of the same circuit. In addition, measurements are generally restricted to a computational basis (Z-basis in IBM quantum computers). Therefore, each qubit needs to be rotated to the standard Z-basis before measurement to access the X and Y components. Note that the majority of the existing quantum algorithms only require measurements in the computational Z-basis.

Expectation Value of an Operator: Expectation value is the average of the eigenvalues, weighted by the probabilities that the state is measured to be in the corresponding eigenstate. Mathematically, expectation value of an operator ($\sigma$) is defined as $\langle \psi | \sigma | \psi \rangle$ where $| \psi \rangle$ is the qubit state vector. For example, the expectation value of the Pauli-Z operator ($\sigma_z$) is $\langle \psi | \sigma_z | \psi \rangle$. If a qubit yields more $|0\rangle$ ($|1\rangle$) than $|1\rangle$ ($|0\rangle$), its Pauli-Z expectation value will be positive (negative). This value will vary in the range $[-1, 1]$. In practice, the expectation value can be approximated from a finite number of repeated qubit measurements: Pauli-Z expectation value of a qubit $\approx 2^N \text{(Number of measured 0’s/Total number of measurements)} - 1$.

Noise in Quantum Computation: Errors in quantum computing can be broadly classified into three categories: 1) Coherence errors: a qubit can retain its state for a short period (coherence time). The computation needs to be completed well within this limit. The coherence time restricts the depth of quantum circuits that can be executed reliably on any target hardware. 2) Gate errors: quantum gates are realized using microwave/laser pulses. It is impossible to generate and apply precise pulses in actual hardware. Hence, gate operations in quantum computers are erroneous. An intended rotation of $\theta$ along X-axis may end up being $\theta + \delta$ or $\theta - \delta$ when executed on...
Quantum Neural Network: QNN involves parameter optimization of a PQC to obtain a desired input-output relationship. QNN generally consists of three segments: 1) a classical to quantum data encoding (also referred to as embedding in the literature) circuit, 2) a parameterized circuit, and 3) measurement operations. A variety of encoding methods are available in the literature [12]. For continuous variables, the most widely used encoding scheme is angle encoding [7, 8, 12, 34] where a continuous variable input classical feature is encoded as a rotation of a qubit along the desired axis (X/Y/Z). For “n” classical features, we require “n” qubits. For example, RZ(f1) on a qubit in superposition (the Hadamard–H gate is used to put the qubit in superposition) is used to encode a classical feature “f1” in Figure 2B [8]. We can also encode multiple continuous variables in a single qubit using sequential rotations. For example, “f1” and “f2” are encoded using subsequent RZ(f1) and RX(f2) rotations on a single qubit in Figure 2C. As the states produced by a qubit rotation along any axis will repeat in $2\pi$ intervals (Figure 2A), features are generally scaled within 0 to $2\pi$ in a data pre-processing step. One can restrict the values between $-\pi$ to $\pi$ to accommodate features with both negative and positive values. Discrete/categorical variables can be encoded using rotations in discrete steps [35].

The parametric circuit has two components: entangling operations and parameterized single-qubit rotations. The entanglement operations are a set of multi-qubit operations between the qubits to generate correlated states [34]. The following parametric single-qubit operations search through the solution space. This combination of entangling and rotation operations is referred to as a parametric layer in the literature. Note that finding the optimal PQC architecture is an unresolved research problem. Descriptors such as, expressive power, entanglement capability, and effective dimension are proposed to measure the potency of various PQC choices [8, 9, 11]. Recent work indicates that these descriptors may have a significant correlation with the trainability of the quantum circuits [36]. In practical applications, such descriptors may be useful to choose better PQC architecture for the intended QML application. A widely used parametric layer architecture is shown in Figure 2D [8, 11]. Here, CNOT gates between neighboring qubits create the entanglement, and rotations along Y-axis using RY(θ) operations define the search space. Normally, these layers are repeated multiple times to extend the search space [7, 8]. We refer to this QNN model with a single PQC as our “Baseline.”

QNN Cost Functions: Qubits in a QNN circuit are measured in the computational basis to retrieve the output state. A cost function is derived from the measurements to train the network [6–8]. For example, in a binary classification problem, the authors measured all the qubits in the QNN model in Pauli-Z basis and associated class 0 with the probability of obtaining even parity, and class 1 with odd parity [8]. Then, the model is trained using binary cross-entropy loss (BCELoss).

$$\text{BCELoss} = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$  \hspace{1cm} (1)$$

Here, “N” is the number of samples in a batch, $y_i$ is the original class label of the ith sample (0 or 1), $p_i$ is the predicted probability of class 1.
In [37], the authors used the Pauli-Z expectation value of a single qubit (-1 associated with class 1 and +1 associated with class 0) for a binary classifier and trained it using mean-squared error loss (MSELoss).

$$MSELoss = \frac{1}{N} \sum_{i=1}^{N} (y_i - E(Q_i))^2$$ \hspace{1cm} (2)

Here, “N” is the number of samples in a batch, $y_i$ is the original class label of the $i$th sample (~1 or 1), $E(Q_i)$ is the Pauli-Z expectation value of the qubit.

Note that multiple one-vs-all classifiers are often used for multi-class classification problems, i.e., classification of more than two classes [7]. Huang et al. [24] fed the expectation values from the QNN to a classical neural network and trained it using binary cross-entropy loss function for binary classifications (Projected Quantum Kernel or PQK).

3 RELATED WORKS

**Baseline QNN:** The “Baseline” QNN is widely used in the literature where a single parametric quantum circuit is used as the QML model [7, 8, 12, 34]. Although they are quite useful to demonstrate various properties of QNN’s with toy datasets, they are not quite as useful for practical applications. The qubit requirements and the circuit depth/gate-count grows with the size (number of features) of the input dataset. The exact nature of this growth depends on the chosen encoding method and PQC architecture. For example, in angle encoding, qubit requirement and gate-count grow linearly while the depth remains constant for the encoding circuit [7, 8, 34]. In amplitude encoding, the qubit requirement grows logarithmically while the gate count/depth remains constant. For instance, the number of gates in DRC increases linearly with the size of the dataset. For a 1000 feature dataset, every single data upload requires 1000 sequential single-qubit operations. The coherence time in the target hardware is the order of microseconds while gate operation time is in the order of nanoseconds [39]. The DRC classifier for the 1000 feature dataset can cross the coherence limit in a single data uploading step.

**Data Re-Uploading Classifier:** Data Re-Uploading Classifier (DRC) is proposed by Pérez-Salinas et al. [25] for small quantum computers. In DRC, classical data are sequentially uploaded in qubits with rotation angles. Each set of data uploads is followed by a PQC. This combination of data uploading and PQC is repeated many times. Note that one can implement a classifier with a single qubit for an arbitrary number of feature variables using DRC. For example, 4 features of a dataset ($f_1, f_2, f_3, f_4$) can be uploaded in a single qubit with alternating RZ and RY rotations ([RZ($f_1$)–RY($f_2$)–RZ($f_3$)–RY($f_4$)]) (Figure 4B). Adding a suitable PQC to this single qubit [e.g., trainable RX(θ)], and repeating this combination many times, one can use a single qubit to build a QNN classification model for the dataset with 4-features. However, the DRC model has obvious scalability issues. For instance, the number of gates in DRC increases linearly with the size of the dataset. For a 1000 feature dataset, a single data upload requires 1000 sequential single-qubit operations. In the current generation of IBM quantum computers, the coherence time is in the order of microseconds while gate operation time is in the order of nanoseconds [39]. The DRC classifier for the 1000 feature dataset can cross the coherence limit in a single data uploading step.

4 PROPOSED QNET ARCHITECTURE

In this section, we present the architectural details of QNet which consists of several QNN Layers (Figure 1). Each QNN Layer contains multiple conventional QNN’s. A QNN processes/transforms a fraction of the input vector. A QNN Layer is followed by classical non-linear activation functions and random shuffling of the output vector. The final QNN Layer output is fed to a fully connected classical dense layer. The details of the architecture are discussed below.
Conventional QNN: The conventional QNN model lies at the heart of QNet. It consists of several conventional QNN models distributed in multiple QNN Layers. Unlike "Baseline" which encodes the whole input vector as a quantum state, each QNN in QNet encodes a fraction of the input vector in a quantum state and performs data transformation in the Hilbert space with a PQC. The width of the QNN (number of qubits) can be chosen based on the size of the target hardware. Each QNN in QNet can use any conventional data encoding technique, PQC architecture, and measurement operations. However, in this work, we use angle encoding (1 continuous variable per qubit as in Figure 2B) as our preferred encoding method and the entanglement layer shown in Figure 2D as our preferred PQC architecture following the recent trends in QML research [8, 11, 24]. We use Pauli-Z expectation values of the qubits as the output features from the QNN [6, 7]. In this chosen configuration, each QNN takes "k" continuous variables as inputs and generates "k" output features as qubit expectation values in the range of [−1, 1]. A toy example is shown in Figure 1. The toy dataset has 4 features. The "Baseline" processes all 4 features using a single quantum circuit. On the other hand, each QNN in QNet processes a fraction of the features (2).

QNN Layers: QNN's are distributed in multiple QNN Layers (Figure 1). The number of parallel QNNs in a Layer is chosen based on the size of the input vector, and the width of individual QNN's. For example, in a 30-feature tabular dataset, each QNN Layer can have five 6-qubit QNN's or ten 3-qubit QNN’s. Increasing the number of QNN Layers raises the total number of QNN in the network. If each QNN has a similar number of trainable parameters, the total number of trainable parameters in the network grows linearly with increasing QNN Layers. It is quite similar to increasing hidden layers in a multi-layer perceptron network [28]. Note that the width of individual QNN’s in a QNN Layer can differ. Moreover, the number of QNN’s in different QNN Layers can be varied as well. For simplicity, we have kept these values constant across the layers in this work. In the remaining paper, we refer to the number of Qubits/QNN as QQ, the number of Parametric Layers/PQC as PL, and the number of QNN Layers in QNet as QL.

Non-linearity: Classical deep neural networks consists of many neurons that perform linear transformations on the input data. However, linear transformation alone is not sufficient to generate complex non-linear decision boundaries encountered in practical ML classification/regression problems. Therefore, non-linear activation functions (e.g., sigmoid, relu, tanh, etc.) are used to provide non-linearity in classical neural networks [28]. In the quantum domain, every gate operation is a linear transformation of the quantum state [29]. During measurements of the qubits in the computational Z-basis, we lose the phase information associated with the quantum state (X and Y components). It provides the required non-linearity in QNN models [40]. In our QNet architecture, we have added an optional classical non-linear activation layer at the end of every QNN Layer to increase non-linearity (Figure 1).

Random Shuffling: In classical neural networks, each fully connected neuron processes the entire input vector generated by the previous layer. However, in QNet, each QNN processes a fraction of the input vector. Therefore, the correlation between features at different fractions is not utilized. To circumvent this issue, we have added a shuffling layer at the end of every QNN Layer that randomly shuffles the input vector generated by the previous layer. Consequently, the QNN’s in subsequent QNN Layers processes a different fraction of the input (transformed) vector. Note that the random shuffling order is generated when the QNet is instantiated. In the later training and inference steps, the order remains constant.

Output Layer and Cost Function: To date, the best QNN performance (accuracy/loss) has been reported by Huang et al. [24] using a hybrid quantum-classical architecture called Projected Quantum Kernel or PKQ. In PKQ, a "Baseline" QNN transforms the input data to a new feature space using a PQC. These new features are extracted through measurements of the quantum state in all three bases (X, Y, and Z). Next, these features are fed to an MLP. The classical network can be trained using any conventional loss function (e.g., cross-entropy loss).

Drawing inspiration from this work, we use a classical dense layer at the end of QNet. The final QNN Layer output is taken as the input to this classical layer. For classification problems, the number of neurons in this dense layer is equal to the number of classes in the input dataset. Unlike PKQ, which performs the quantum kernel transformation and training of the classical network separately, we train the whole network simultaneously. Moreover, PKQ uses a single large quantum circuit while QNet uses a collection of several small quantum circuits. We apply LogSoftmax activation at the output of the dense layer and use negative log-likelihood loss (NLLLoss) as our preferred loss function [41]. The LogSoftmax activation function transforms the output of the dense layer to class probabilities. This combination of LogSoftmax activation and NLLLoss is also known as Cross-Entropy Loss (CELoss) in the literature.

\[
CEF = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{k} y_{ij} \log(p_{ij})
\]

Here, “N” is the number of samples in a batch, “k” is the number of classes in the problem, \(y_{ij}\) is the truth label of the ith sample and jth class, and \(p_{ij}\) is the corresponding predicted probability. Note that one can choose any suitable loss function to train QNet (e.g., MSELoss).

Trainable Parameters: The number of trainable circuit parameters in QNet depends on chosen PL and QL. Since we have used identical QNN across the layers with single feature/qubit angle encoding, the total number of trainable circuit parameters is “PL*QL” where “P” is the number of features in the input dataset. Additionally, the network has “PL*QL” classical weights between the final QNN Layer and the output Dense layer where “C” is the number of classes in the dataset. Note that one can also add bias parameters/weights to the outputs of the QNN Layers and the classical dense layer.

Training and Inference: The QNet network can be trained using any gradient-based optimization algorithm such as, Adam or Adagrad [42, 43]. To apply backpropagation on QNet, QNN outputs have to be differentiable with respect to the inputs and the circuit parameters. Multiple methods exist to compute gradients...
in QNN [44–46]. However, not all of them are suitable for hardware. For instance, the adjoint method proposed by Luo et al. [44] requires the circuit intermediate state information to compute gradients which is not accessible when we execute the circuit on hardware. The parameter-shift rule is widely used to compute quantum gradients [45, 46]. It is quite similar to the age-old finite difference method which uses two evaluations of the target function at proximity to compute the gradients. However, in the parameter-shift rule, the two data points can be far from each other. It shows greater resilience to shot noise and measurement errors compared to finite difference [46]. It is also suitable for hardware. The PennyLane framework supports all these quantum gradient computation methods [47]. In this work, we use the Pytorch and the PennyLane frameworks to build and train QNet [41, 47].

Note that all the QNN’s in a QNN layer can go through the forward pass during training/inference at the same time as they do not have any dependency. Therefore, multiple hardware/simulators can be used to execute/simulate these QNN’s for faster computation both during training and inference. We can also use gradient-free optimizers such as, Nelder-Mead to train QNet [48]. However, they may perform poorly when the network has lots of parameters.

5 EVALUATION

In this section, we present a numerical analysis of QNet performance with varying QQ, QL, and PL. We compare QNet with “Baseline” both in noiseless simulations and hardware emulations. We also compare QNet with PCA + Baseline and DRC. Note that this is not an exhaustive study. For instance, encoding methods, parametric layer architecture, measurements in the QNN’s are also major design choices. The field is continuously evolving, and there is a wide variety of choices for each of them. We stick to the same design choices throughout the study to prevent deviating from our key goals. Future research can focus on a comprehensive analysis that takes into account all of the variables.

5.1 Setup for Numerical Studies

Datasets: We use 6 classification datasets that are often used to evaluate the performance of QML models in contemporary works [3, 8, 23, 24]–Iris (150 samples, 4 features, 3 classes), Wine (178 samples, 13 features, 3 classes), Breast Cancer (569 samples, 30 features, 2 classes), Digits (1797 samples, 64 features, 10 classes), MNIST (70000 samples, 784 features, 10 classes) and Fashion-MNIST (70000 samples, 784 features, 10 classes). To limit simulation time, we pick 1200 samples from the MNIST and the Fashion-MNIST datasets. For MNIST, we pick 400 samples/digits of digits 3, 5, and 8 [49]. For Fashion-MNIST, we use samples of T-shirt/top, Trouser, and Pullover [50] classes. We also downsampled both MNIST and Fashion-MNIST datasets from 28 × 28 to 14 × 14 using 2d max-pooling [41]. The Iris, Wine, Breast Cancer, and Digits datasets are downloaded through the scikit-learn Python package [51]. We divide each dataset into two equal sets and use them for training and validation, respectively.

Metrics: We use the average loss and accuracy as our preferred metrics to measure the performance of the QML models [28]. Note that the average loss value is calculated by performing a forward pass through the networks with each of the data samples, measuring the corresponding loss value, and then averaging all the losses. Accuracy is the fraction of the samples that are classified correctly. The training loss and accuracy indicate the trainability of the models. The validation loss and accuracy indicate the generalization capability of the models on unseen data.

Simulation Framework: We have developed a Python framework using Pytorch, PennyLane, and Qiskit packages to build and train all the QML models used in this work [39, 41, 47]. We use the Adagrad optimizer with a learning rate of 0.5 [41, 43]. We use the default state vector simulator available in the PennyLane framework to simulate the quantum circuits in noiseless training [47]. Currently, we have limited access to actual quantum computers through cloud-based quantum computing services. It is not possible to present a statistically significant amount of data from the hardware since training and inference of QML models need frequent calls to quantum computers. Therefore, we resort to hardware emulators (simulated hardware with noise). IBM calibrates the hardware available in their cloud service twice a day. After each calibration, they provide the hardware characterization data through IBM Quantum [52]. Qiskit provides hardware emulation models (e.g., FakeMelbourne, FakeBogota, FakeCasablanca, etc.) that can replicate the output from the target hardware available in IBM Quantum [39]. These emulators use the noise models derived from the calibration data. We use these hardware emulators to compare the performance of various QML models under noise. The adjoint method is used for gradient computation in noiseless simulations because it is faster than parameter-shift. We use the parameter-shift rule in training QNet with hardware emulators as it is robust against shot noise and measurement errors. The QNN outputs are calculated (Pauli-Z expectation values of the qubits) from a finite number of samples of the circuit output. The number of samples is kept to 128 throughout our experiments. In the noiseless simulation, we calculate QNN outputs analytically. The simulations are done in a Windows machine with an Intel Core i7-10750H processor and 16 GB RAM. The datasets and Python framework are available through the following GitHub repository.1

Noise Simulation: We measure the impact of gate errors, decoherence, and measurement errors on QML models using Qiskit hardware emulators–FakeBogota, FakeMelbourne, and FakeCasablanca [39, 53, 54]. Several contemporary research works have also used these emulators [55–58]. Errors in single-qubit gate execution are simulated by applying a single qubit depolarizing error (gate error) followed by a single qubit thermal relaxation error (decoherence). Similarly, errors in two-qubit gate execution are simulated by applying a two-qubit depolarizing error (gate error) followed by single-qubit thermal relaxation errors on both qubits in the gate.

1https://github.com/mahabub-alam/QNet
(decoherence). Each measured bit is flipped with a bit-flip probability to simulate the impact of measurement errors. The error parameter of the thermal relaxation errors is derived from T1 (longitudinal coherence time) and T2 (phase coherence time) parameters, and the gate execution time parameters [59]. Several characterization techniques are used in the IBM quantum systems to measure coherence time parameters (T1 and T2) such as Inversion Recovery, Ramsey Experiments, and Hahn Echoes. Gate errors are characterized using the Randomized Benchmarking protocol. Qubits are repeatedly prepared and measured in |0⟩ and |1⟩ states to calculate the corresponding bit-flip probabilities during measurements. IBM provides access to the hardware calibration data through its Qiskit framework [53]. During simulation, the hardware emulators load these noise models, decompose/compile the circuit with the basis/native gates of the hardware, and generate outputs under the aforementioned noises. Note that noise modeling is not the contribution of our work. One can use a different/more accurate noise model for comparative analysis of the QML models [60, 61, 62]. However, we believe that the conclusion will remain similar. The Supplementary Materials provide a more detailed overview of the noise models and attributes.

Note that these hardware emulators do not support simulation of crosstalk noise that stems from parallel/sequential execution of gate operations on the same hardware. The smaller QNN’s in QNet run on different hardware or on the same hardware at different points in time. Therefore, there are fewer concurrent gate operations in one particular hardware compared to the Baseline. As a result, QNet may accumulate less crosstalk noise compared to the Baseline. On top of that, we use noise simulations for comparative studies between the models/architectures. Therefore, we believe that the exclusion of crosstalk noise simulation does not affect our conclusions.

Recent studies show a high degree of similarity between the distributions generated by hardware simulators and the original hardware [55–58]. The difference can be as little as ≈ 7% in terms of total variation distance [37]. The minor difference can be attributed to noises that are not simulated (e.g., crosstalk). Therefore, we believe the comparative analysis will yield similar conclusions if the experiments are done on actual hardware.

### 5.2 Results

#### 5.2.1 QNet Design-Space Exploration

We sweep three major configurable attributes of QNet (PL, QL, and QQ) and train the networks for various classification tasks to gauge the impact of these attributes on training and validation. The training curves (over 100 epochs, averaged over 10 separate training runs) of Iris and Wine classifiers are shown in Figures 5A,D, respectively (QQ = 2, PL = 2, QL = 2). The training loss consistently goes down over training epochs, and the training accuracy improves for both these datasets. However, the validation loss decreases initially, and after a few epochs of training, it either goes up (Figure 5A) or remains at similar level (Figure 5D). This behavior resembles the classic overfitting issue of neural networks [28]. When a network is trained over a large number of epochs, the model tends to overfit. Consequently, it exhibits poor performance on unseen data. To avoid overfitting QNet, we can use the existing techniques such as early stopping [63].

We vary PL from 1 to 4 (for QQ = 2, QL = 2) and train the corresponding QNet networks for Iris and Wine classification. The corresponding results are shown in Figures 5B,E, respectively. Note that increasing PL from 1 to 2 doubles the number of trainable circuit parameters in the network. In both cases, the training loss goes down and the training accuracy improves with increasing PL. However, the validation loss or accuracy improve initially, then they start to saturate (Figure 5B) or degrade (Figure 5E). We observe similar behavior when we vary QQ from 1 to 4 (Figures 5C,F). Increasing QQ from 1 to 2 doubles the number of trainable circuit parameters in the QNet network used in this work. This behavior also resembles the classic overfitting issue in neural networks due to over-parameterization [28]. In practical applications, the size of the network should be chosen judiciously to avoid this issue. It may be worthwhile to explore techniques that exist in the classical domain to avoid overfitting in over-parameterized neural networks and apply them to QNet [64].

QNet performance metrics for the Wine, Breast Cancer and the Digits datasets with varying QQ’s and network sizes (PL = 2, QL = 2 Vs PL = 3, QL = 3) are shown in Table 1. Surprisingly, at lower QQ values, QNet performs better across all these datasets. The performance on the training set improves with increased network size. However, the validation performance deteriorates as expected due to overfitting. The performance on the truncated MNIST and Fashion-MNIST datasets are shown in Table 2. The performance metrics on both these datasets indicate the trainability and generalization capability of QNet on real-world datasets. Note that prior works on applying QML models on datasets such as MNIST or Fashion-MNIST reduced the input dimensions to an extremely low level (e.g., 30 in [24]). However, such reduction is not necessary for QNet. Nevertheless, we downsample the data from 28 × 28 (784) to 14 × 14 (196) to avoid large simulation time.

#### 5.2.2 QNet Vs. Baseline

We train the Iris and Wine classifiers using QNet and Baseline with a varying number of trainable circuit parameters (noiseless simulation). The results are shown in Figures 6A,B. Note that both these approaches provide similar performance in noiseless simulations. Baseline performs slightly better than QNet for the smaller Iris dataset (Figure 6A). QNet performs slightly better than Baseline for the bigger Wine dataset. Note that the Baseline models for Iris and Wine datasets require 4 and 13 qubits, respectively. We used QQ = 2, PL = 2, and QL = 2 in the QNet models. We can not train the Baseline models for other datasets (e.g., Breast Cancer, Digits, MNIST, Fashion-MNIST) in our computing environment due to the memory bottleneck of quantum simulation [29].

The superior performance of QNet becomes evident when we train these models under noise with a hardware emulator (ibmq_16_melbourne) as shown in Figure 7. On the smaller Iris dataset (Figure 7A), the performances are similar. However, on the bigger Wine dataset, there is a significant gap (Figure 7B). The QNet network training loss and validation loss consistently goes down over training epochs. However, the Baseline models
remain more or less constant. The Baseline model uses a 13-qubit circuit for the Wine dataset. The target device’s quantum volume (ibmq_16_melbourne) is 8, implying that 3-qubit (or similarly sized) circuits can be reliably executed on the hardware. Consequently, the Iris Baseline model performs well with this hardware emulator that uses a 4-qubit circuit. However, the 13-qubit Baseline QNN for the Wine dataset essentially generates random outputs, and therefore, this model does not train well as evident from Figure 7B. In both cases, QNet employs a sequence of 2-qubit circuits and is hardly affected by hardware noise.

5.2.3 QNet Vs. PCA + Baseline
As mentioned earlier, the performance of PCA + Baseline will largely depend on the level of compression using PCA. To further
illustrate this issue, we show the fraction of explained variance against the chosen number of principal components (PC) on the Digits dataset in Figure 8A [65]. The fraction improves with an increasing number of PC. For instance, 4 PC explains roughly 40% variance in the digits dataset compared to ≈70% in 8 PC. Therefore, the subsequent Baseline model in PCA + Baseline will work on a poor representation of the original dataset. Consequently, this PCA + Baseline model will perform poorly.

The accuracy of the trained PCA + Baseline models and QNet models on the Digits dataset with varying PCs is shown in Figure 8B (noiseless simulation). Note that, when we choose n-PC’s, the subsequent Baseline QNN in the PCA + Baseline model uses n-qubit circuits. For QNet, we use QQ = 2 (2-qubit circuits), PL = 2, and QL = 2. The QNet models outperform the corresponding PCA + Baseline models in terms of accuracy. For instance, QNet accuracy is 68% greater than PCA + Baseline with 2 PC. The difference is 30% for PCA + Baseline with 16 PC. If we use a larger PC (e.g., 32), the PCA + Baseline model may perform at a similar level to QNet in the noiseless simulation. However, the quantum circuit will be much larger (32-qubit), and it may perform very poorly on the near-term hardware.

To compare the performance between QNet and PCA + Baseline on hardware, we use the 5-qubit ibmq_bogota and 7-qubit ibmq_casablanca hardware emulators and measure the trained QNet and PCA + Baseline models performance for Wine, Breast Cancer, Digits, MNIST, and Fashion-MNIST datasets. The results are
We use 5 and 7 PCs in the PCA + Baseline models for ibmq_bogota and ibmq_casablanca, respectively. In all these cases, QNet uses QQ = 5 and the PCA + Baseline models use 5 principal components. For QNet, we use PL = 2, QL = 2. For a fair comparison, we vary the number of trainable parameters in PCA + Baseline models and report the best-measured performance against QNet.

5.2.4 QNet Vs DRC
As mentioned earlier, the DRC model exceeds the coherence time of the device when the number of features is large. It may be pointless to compare DRC with QNet for larger datasets on hardware emulators. Therefore, we choose the smallest Iris dataset for comparison. The measured performance of the trained QNet (QQ = 2, PL = 2, QL = 2) and two-qubit DRC models (on simulator and ibmq_bogota emulator) are shown in Figure 10. Note that the performances are similar in noiseless simulations (Figure 10A). However, the measured accuracy of QNet on noisy ibmq_bogota emulator is significantly higher on both the training (29%) and the validation (21%) sets (Figure 10B). Similarly, the loss is significantly lower in QNet.

6 DISCUSSION AND FUTURE OUTLOOK
Why is QNet Noise-Resilient and Scalable?: Unlike Baseline, PCA + Baseline, and DRC models, the circuit sizes can be
modulated arbitrarily in QNet. This flexibility provides noise resilience and scalability to QNet. Each QNN can have a small number of qubits (e.g., QQ = 2) and a small depth (e.g., PL = 2). One can execute these small circuits reliably in existing NISQ-machines. Due to the high accumulation of gate noise and decoherence in the Baseline and PCA + Baseline models, their corresponding QNN circuit outputs can be highly erroneous. Therefore, these models may perform poorly on hardware. The circuits in DRC have a lower number of qubits. They are, however, too deep for larger datasets, so decoherence and gate errors can have a bigger impact on their outputs. To solve larger problems in the Baseline or PCA + Baseline models, we need to add more qubits to the QNN circuit, whereas in QNet, we only need to add more small noise-resilient QNN circuits. As a result, QNet models can be scaled to fit any size dataset. PCA + QNet could be another promising solution for reducing the use of quantum resources. The PCA + QNet variation, unlike PCA + Baseline, does not require significant data compression to fit the quantum circuit on the target hardware. As a result, its performance will be less affected than that of the PCA + Baseline model due to information loss during data compression.

**Constructing QNN’s in QNet**: Throughout this study, we use the same encoding method (Figure 2B), parametric layer architecture (Figure 2D), and measurement operations (Pauli-Z expectation values of qubits) in all the models. We also use the same number of QNN in different QNN Layers. All these choices can have a significant impact on the network size and performance. For example, if we choose a 2-variables/qubit angle encoding method instead of the 1-variable/qubit angle encoding method, the number of QNN’s in the network can reduce greatly. For example, the QNN circuits in a QNet network for an 8-feature dataset (QQ = 2, QL = 2) will reduce from 8 to 3 as shown in Figure 11. The Baseline and PCA + Baseline models will also have similar benefits. However, a detailed study on this front is beyond the scope of this paper.

**QNet on Small Quantum Systems**: QNet enables the integration of multiple small and heterogeneous quantum computing systems in a single machine learning application.

The QNNs in a QNN Layer can be executed sequentially on a small quantum computer (at the expense of increased run-time) or concurrently in different hardware during the training/inference step’s forward pass. Seven 5-qubit systems, four 7-qubit systems, and one single-qubit system are available in the current IBMQ suite. One can use the 64 qubits available in these chips to create QNet models to handle 64-feature ML problems (e.g., digits classification discussed in the previous section). To create QNet with this set of hardware, we can utilize seven 5-qubit QNNs, four 7-qubit QNNs, and one single-qubit QNN in each QNN Layer. Each QNN model (parametric layer, encoding, and so on) should be tailored to the hardware used in the backend. For example, if we want all the QNNs in a QNN Layer to have a comparable forward pass execution time, the QNNs with a larger number of qubits should have fewer Parametric Layers than the QNNs with lower qubit counts.

**QNet on Heterogeneous Systems**: It’s worth noting that hardware based on a different quantum computing technology (for example, IonQ’s 11-qubit trapped-ion system) can be added to the mix without sacrificing generality. Individual QNN models can be modified for unique hardware in these circumstances. QNN’s for trapped-ion computers can be built using the technology’s native gates (e.g., GPI, GPI2, GZ, and MS) [66]. For example, instead of the CNOT gate, which is the native 2-qubit gate in IBM’s superconducting quantum computers, we can entangle the qubits using the two-qubit MS gate in trapped-ion systems [22]. Again, if we want all the QNNs in a single QNN Layer in QNet to have a comparable forward pass time, the PQCs of individual QNNs will need to be customized accordingly.

**QNet on Larger Quantum Systems**: Furthermore, we can implement QNet with small QNN’s on large hardware (e.g., the 65-qubit ibmq_brooklyn). Even if a Baseline fits in this large hardware, QNet might be preferred. The QV measure explains this. The ibmq_brooklyn hardware has QV of 32. Even though it has 65 qubits, we can reliably execute circuits that have ≈ 5-qubits and a depth of ≈ 5. As a result, for an ML job including a 64-feature dataset, a QNet model with sixteen 4-qubit QNNs per QNN Layer will have a better chance of success than a Baseline QNN created with a 64-qubit circuit.
QNet in Simulation: Quantum simulation is limited by the memory bottleneck [29]. One can represent an n-qubit system state with a state vector that has $2^n$ complex values. The memory requirement can easily go beyond the limit of classical simulation if we increase the number of qubits. Currently, the most powerful simulator available in the IBMQ suite can handle 32-qubits. Even QML applications based on classical simulators can benefit from QNet. Let’s say we’re solving a 64-feature dataset with QNet, and we’re employing 8-qubit QNNs. There will be eight such QNN’s in each QNN Layer. A quantum simulator will need to store $2^8$ or 256 complex values to process these QNN’s sequentially. Eight instances of the simulator (e.g., employing multi-processing) can be used for concurrent simulation, requiring storage of $8 \times 2^8$ or 2048 complex values. In comparison, a 64-qubit Baseline QNN circuit will require storage of $2^{64}$ complex values. It is far beyond the capabilities of existing supercomputers.

In this work, we have demonstrated QNet models on 196-feature datasets using quantum simulators available in the Pennylane [47] framework that runs on a single CPU-core of a standard computer (Intel Core i7-10750H, 16 GB RAM) reasonably fast. The simulation time can be improved by allocating QNN’s in a QNN Layer to different instances of the simulator using advanced computing techniques (e.g., multi-processing/multi-threading or distributed computing). QNet is not affected by the memory bottleneck as much as the Baseline when QQ is small. Figure 12 shows the single-epoch training time of QNet (Breast Cancer dataset) with varying QQ on our computing environment. At lower QQ, the simulation takes minutes. Therefore, QNet can be an exciting choice for simulation-based practical QML applications.

Other Applications of QNet: In this study, we exclusively employ classification datasets. By adjusting the cost functions, one can use QNet for different QML tasks (such as regression, auto-encoder, and so on). These can be worthwhile research topics for the future.

Quantum Supremacy: Quantum computing and quantum machine learning are still in their infancy. Existing classical ML algorithms have matured over a few decades of research. Therefore, we feel that seeking superior performance with QNet compared to the classical algorithms can be unfair. Nonetheless, the performance of QNet in this study is comparable to that of traditional neural networks. It may improve further with proper choice of encoding methods, parametric layer architecture, and measurements. QNet is a generic framework for creating quantum machine learning models that are noise-resistant. We believe it will have a profound impact on quantum machine learning research in the near term.

QNet Limiting Factors: QNet employs a large number of small QNNs, which can be advantageous if a user has access to a large number of small quantum computers for parallel processing. However, having only one piece of quantum hardware can be a bottleneck. In such circumstances, one needs to execute the QNNs in order. Furthermore, we will need to reset the qubits between each call to the quantum computers, which may raise the overall execution time of the QNet network because qubit resets can take much longer than actual gate executions [67]. Moreover, the number of qubit measurements increases with QL in QNet. A QNet with QL = 2 may have 2X measurements compared to QL = 1. Large measurement errors can affect QNet performance. However, measurement errors have decreased significantly in the IBM quantum devices in the past few years. For example, the 5-qubit ibmqx-2, a quantum device from the past generation, had an average measurement error of 8.2% while the current 5-qubit ibmq_bogota device has an average measurement error rate of 2.27%. On top of that, one can reduce measurement errors by applying classical post-processing [68–70]. The advantages of a reduced depth and gate-count may outweigh the additional measurement noise, as evidenced by comparison to prior approaches.

7 CONCLUSION

In this paper, we present QNet—a scalable and noise-resilient quantum neural network architecture. Through empirical study, we show that QNet outperforms existing techniques in terms of performance and noise resilience. We use six datasets (Iris, Wine, Breast Cancer, Digits, MNIST, Fashion-MNIST) and three noisy hardware emulators (ibmq_16_melbourne, ibmq_bogota, ibmq_casablanca) for this study. On average, trained QNet models show 43% better accuracy over the existing models on the hardware emulators. QNet also provides a framework for creating noise-resistant QML models using a collection of small quantum neural networks. In the not-too-distant future, it has the potential to have a significant impact on quantum machine learning research.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article/Supplementary Material.
AUTHOR CONTRIBUTIONS

All authors listed have made a substantial, direct, and intellectual contribution to the work and approved it for publication.

FUNDING

The work is supported in parts by NSF (CNS-1722557, CCF-1718474, OIA-2040667, DGE-1723687, DGE-1821766, and DGE-2113839) and seed grants from Penn State ICDS and Huck Institute of the Life Sciences.

REFERENCES

1. Arute F, Arya K, Babbush R, Bacon D, Bardin JC, Barends R, et al. Quantum Supremacy Using a Programmable Superconducting Processor. Nature (2019) 574:505–10. doi:10.1038/s41586-019-1666-5
2. Biamonte J, Wittek P, Pancotti N, Rebentrost P, Wiebe N, Lloyd S. Quantum Machine Learning. Nature (2017) 549:195–202. doi:10.1038/nature23474
3. Farhi E, Neven H. Classification with Quantum Neural Networks on Near Term Processors (2018). arXiv preprint arXiv:1802.06002.
4. Killoran N, Bromley TR, Arrazola JM, Schuld M, Quesada N, Lloyd S. Continuous-variable Quantum Neural Networks. Phys Rev Res (2019) 1:033063. doi:10.1103/physrevresearch.1.033063
5. Cong I, Choi S, Lukin MD. Quantum Convolutional Neural Networks. Nat Phys (2019) 15:1273–8. doi:10.1038/s41567-019-0646-8
6. Schuld M, Killoran N. Quantum Machine Learning in Feature hilbert Spaces. Phys Rev Lett (2019) 122:040504. doi:10.1103/PhysRevLett.122.040504
7. Schuld M, Bocharov A, Svore KM, Wiebe N. Circuit-centric Quantum Classifiers. Phys Rev A (2020) 101:032308. doi:10.1103/physreva.101.032308
8. Abbas A, Sutter D, Zoufal C, Lucchi A, Poggi D, Woerner S. The Power of Quantum Neural Networks. Nat Comput Sci (2021) 1:403–9. doi:10.1038/s43588-021-00084-1
9. Du Y, Hsieh M-H, Liu T, Tao D. Expressive Power of Parameterized Quantum Circuits. Phys Rev Res (2020) 2:033125. doi:10.1103/physrevresearch.2.033125
10. Wright IG, McMahon PL. The Capacity of Quantum Neural Networks. In: CLEO: QELS_Fundamental Science. Washington, DC: Optical Society of America (2020). p. JMMG–5. doi:10.1364/cleo_at.2020.jm4g.5
11. Sim S, Johnson PD, Aspuru-Guzik A. Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms. Adv Quan Tech (2019) 2:1900070. doi:10.1002/qute.201900070
12. Schuld M, Sweke R, Meyer JJ. Effect of Data Encoding on the Expressive Power of Variational Quantum-Machine-Learning Models. Phys Rev A (2021) 103:023240. doi:10.1103/physreva.103.023240
13. Funcke L, Hartung T, Jansen K, Kühn S, Stornati P. Dimensional Expressivity Analysis of Parametric Quantum Circuits. Quantum (2021) 5:422. doi:10.22331/q-2021-03-29-422
14. Hari A, Bromley TR, Izaac J, Schuld M, Killoran N. Transfer Learning in Hybrid Classical-Quantum Neural Networks. Quantum (2020) 4:430. doi:10.22331/q-2020-10-09-340
15. Alam M, Kundu S, Topaloglu RO, Ghosh S. Iccad Special Session Paper: Quantum-Classical Hybrid Machine Learning for Image Classification (2021). arXiv preprint arXiv:2109.02862.
16. Li J, Alam M, Sha CM, Wang J, Dokholyan NV, Ghosh S. Drug Discovery Approaches Using Quantum Machine Learning (2021). arXiv preprint arXiv:2104.00746.
17. Batra K, Zorn KM, Foil DH, Minerali E, Gawriluk V, Lane TR, et al. Quantum Machine Learning Algorithms for Drug Discovery Applications. In: Chem Inf Model., 61 (2021). p. 2641–7. doi:10.1021/acs.jicim.1c00166
18. Sakuma T. Application of Deep Quantum Neural Networks to Finance (2020). arXiv preprint arXiv:2011.07319.

AUTHOR CONTRIBUTIONS

All authors listed have made a substantial, direct, and intellectual contribution to the work and approved it for publication.

FUNDING

The work is supported in parts by NSF (CNS-1722557, CCF-1718474, OIA-2040667, DGE-1723687, DGE-1821766, and DGE-2113839) and seed grants from Penn State ICDS and Huck Institute of the Life Sciences.

REFERENCES

1. Arute F, Arya K, Babbush R, Bacon D, Bardin JC, Barends R, et al. Quantum Supremacy Using a Programmable Superconducting Processor. Nature (2019) 574:505–10. doi:10.1038/s41586-019-1666-5
2. Biamonte J, Wittek P, Pancotti N, Rebbentrost P, Wiebe N, Lloyd S. Quantum Machine Learning. Nature (2017) 549:195–202. doi:10.1038/nature23474
3. Farhi E, Neven H. Classification with Quantum Neural Networks on Near Term Processors (2018). arXiv preprint arXiv:1802.06002.
4. Killoran N, Bromley TR, Arrazola JM, Schuld M, Quesada N, Lloyd S. Continuous-variable Quantum Neural Networks. Phys Rev Res (2019) 1:033063. doi:10.1103/physrevresearch.1.033063
5. Cong I, Choi S, Lukin MD. Quantum Convolutional Neural Networks. Nat Phys (2019) 15:1273–8. doi:10.1038/s41567-019-0646-8
6. Schuld M, Killoran N. Quantum Machine Learning in Feature hilbert Spaces. Phys Rev Lett (2019) 122:040504. doi:10.1103/PhysRevLett.122.040504
7. Schuld M, Bocharov A, Svore KM, Wiebe N. Circuit-centric Quantum Classifiers. Phys Rev A (2020) 101:032308. doi:10.1103/physreva.101.032308
8. Abbas A, Sutter D, Zoufal C, Lucchi A, Poggi D, Woerner S. The Power of Quantum Neural Networks. Nat Comput Sci (2021) 1:403–9. doi:10.1038/s43588-021-00084-1
9. Du Y, Hsieh M-H, Liu T, Tao D. Expressive Power of Parameterized Quantum Circuits. Phys Rev Res (2020) 2:033125. doi:10.1103/physrevresearch.2.033125
10. Wright IG, McMahon PL. The Capacity of Quantum Neural Networks. In: CLEO: QELS_Fundamental Science. Washington, DC: Optical Society of America (2020). p. JMMG–5. doi:10.1364/cleo_at.2020.jm4g.5
11. Sim S, Johnson PD, Aspuru-Guzik A. Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms. Adv Quan Tech (2019) 2:1900070. doi:10.1002/qute.201900070
12. Schuld M, Sweke R, Meyer JJ. Effect of Data Encoding on the Expressive Power of Variational Quantum-Machine-Learning Models. Phys Rev A (2021) 103:023240. doi:10.1103/physreva.103.023240
13. Funcke L, Hartung T, Jansen K, Kühn S, Stornati P. Dimensional Expressivity Analysis of Parametric Quantum Circuits. Quantum (2021) 5:422. doi:10.22331/q-2021-03-29-422
14. Hari A, Bromley TR, Izaac J, Schuld M, Killoran N. Transfer Learning in Hybrid Classical-Quantum Neural Networks. Quantum (2020) 4:430. doi:10.22331/q-2020-10-09-340
15. Alam M, Kundu S, Topaloglu RO, Ghosh S. Iccad Special Session Paper: Quantum-Classical Hybrid Machine Learning for Image Classification (2021). arXiv preprint arXiv:2109.02862.
16. Li J, Alam M, Sha CM, Wang J, Dokholyan NV, Ghosh S. Drug Discovery Approaches Using Quantum Machine Learning (2021). arXiv preprint arXiv:2104.00746.
17. Batra K, Zorn KM, Foil DH, Minerali E, Gawriluk V, Lane TR, et al. Quantum Machine Learning Algorithms for Drug Discovery Applications. In: Chem Inf Model., 61 (2021). p. 2641–7. doi:10.1021/acs.jicim.1c00166
18. Sakuma T. Application of Deep Quantum Neural Networks to Finance (2020). arXiv preprint arXiv:2011.07319.

ACKNOWLEDGMENTS

We also thank Prof. Mehrdad Mahdavi from Penn State and Dr. Rasit Topaloglu from IBM Corp. for helpful discussions.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fphy.2021.755139/supplementary-material
37. Alam M, Ash-Saki A, Ghosh S. Addressing Temporal Variations in Qubit Quality Metrics for Parameterized Quantum Circuits. In: 2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). IEEE (2019). p. 1–6. doi:10.1109/isulped.2019.8824007

38. Mottonen M, Vartiainen J, Bergholm V, Salomaa MM. Transformation of Quantum States Using Uniformly Controlled Rotations (2004). arXiv preprint quant-ph/0407010.

39. Cross A. The IBM Q Experience and Qiskit Open-Source Quantum Computing Software. In: APS March Meeting Abstracts, 2018 (2018). p. 158–003.

40. Nghiem NA, Chen SY-C, Wei TC. Unified Framework for Quantum Classification. Phys Rev Res (2021) 3:033056. doi:10.1103/physrevresearch.3.033056

41. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. Pytorch: An Imperative Style, High-Performance Deep Learning Library. Adv Neural Inf Process Syst (2021) 32:8026–8037.

42. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization (2014). arXiv preprint arXiv:1412.6980.

43. Duchi J, Hazan E, Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. J Machine Learn Res (2011) 12.

44. Luo X-Z, Liu J-G, Zhang P, Wang L, Yao J. Extensible, Efficient Framework for Quantum Algorithm Design. Quantum (2020) 4:341. doi:10.22331/q-2020-10-11-341

45. Banchi L, Crooks GE. Measuring Analytic Gradients of General Quantum Evolution with the Stochastic Parameter Shift Rule. Quantum (2021) 5:386. doi:10.22331/q-2021-01-25-386

46. Schulz B, Bergholm V, Gogolin C, Izaaq J, Killoran N. Evaluating Analytic Gradients on Quantum Hardware. Phys Rev A (2019) 99:032331. doi:10.1103/physreva.99.032331

47. Bergholm V, Izaaq J, Schulz M, Gogolin C, Alam MS, Ahmed S, et al. PennyLane: Automatic Differentiation of Hybrid Quantum- Classical Computations (2018). arXiv preprint arXiv:1811.04968.

48. Lavrijsen W, Tudor A, Müller J, Izaac J, Schuld M, Gogolin C, Ahmed S, et al. SkitLearn: Learning in python. J Machine Learn Res (2011) 12:2825–30.

49. [Dataset] Ibm quantum. Qiskit/qiskit-tensor (2021). Available at: https://github.com/qiskit/qiskit-tensor/tree/main/qiskit/test/mock/backends (Accessed July 28, 2021).

50. [Dataset] Ibm quantum. Qiskit (2021). Available at: https://github.com/qiskit/qiskit-main/test/mock/test/mock/backends (Accessed July 28, 2021).

51. [Dataset] Ibm quantum. High-Performance Simulator Tutorials (2021). Available at: https://qiskit.org/documentation/tutorials/simulators/ (Accessed July 28, 2021).

52. Liu J, Zhou H. Reliability Modeling of Nisq-Era Quantum Computers. In: 2020 IEEE International Symposium on Workload Characterization (IISWC). IEEE (2020). p. 94–105. doi:10.1109/iiswc5251.2020.00018

53. Wang Y, Krstic PS. Prospective of Using grover’s Search in the Noisy-Intermediate-Scale Quantum-Computer Era. Phys Rev A (2020) 102:042609. doi:10.1103/physreva.102.042609

54. Kazemzadeh D, Haeberle R, Naveh Y, Rausendorf R, Sela E, Dalla Torre EG. Identification of Symmetry-Protected Topological States on Noisy Quantum Computers. Phys Rev Lett (2020) 125:120502. doi:10.1103/physrevlett.125.120502

55. Kandala A, Temme T,Temme T, Mezzacapo A, Neece J, Wang Y, Vainsencher A, Takabi T, Endo J, Schuster DJ, Wang M, Maslov K, Titov V, Nemoto K, Nagamatsu Y. Hardware-efficient variational quantum eigensolver. Nature (2017) 554:83. doi:10.1038/nature25965

56. Wang Y, Krstic PS. Prospect of Using Grover’s Algorithm for Quantum Computations on Noisy-Intermediate-Scale Quantum Computers. In: 2020 IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE (2020). p. 215–28. doi:10.1109/micro5266.2020.00039

57. Ash-Saki A, Alam M, Ghosh S. Quret: Qubit Re-allocation in Noisy Intermediate-Scale Quantum Computers. In: Proceedings of the 56th Annual Design Automation Conference 2019 (2019). p. 1–6.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher’s Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2022 Alam and Ghosh. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.