INDUSTRIAL AND SYSTEMS ENGINEERING

ISE

Quantum DeepONet: Neural Operators Accelerated by Quantum Computing

PENGPENG XIAO[1,2], MUQING ZHENG[3], ANRAN JIAO[1], XIU YANG[3], AND LU LU[1,4]

[1]Department of Statistics and Data Science, Yale University, New Haven, CT 06511, USA

[2]Department of Physics, Fudan University, Shanghai 200437, China

[3]Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA, 18015 USA

[4]Wu Tsai Institute, Yale University, New Haven, CT 06510, USA

ISE Technical Report 24T-011

LEHIGH UNIVERSITY

Quantum DeepONet: Neural operators accelerated by quantum computing

Pengpeng Xiao[1,2], Muqing Zheng[3], Anran Jiao[1], Xiu Yang[3,*], and Lu Lu[1,4,*]

[1]Department of Statistics and Data Science, Yale University, New Haven, CT 06511, USA
[2]Department of Physics, Fudan University, Shanghai 200437, China
[3]Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015, USA
[4]Wu Tsai Institute, Yale University, New Haven, CT 06510, USA
[*]Corresponding author. Email: xiy518@lehigh.edu, lu.lu@yale.edu

Abstract

In the realm of mathematics, engineering, and science, constructing models that reflect real-world phenomena requires solving partial differential equations (PDEs) with different parameters. Recent advancements in DeepONet, which learn mappings between infinite-dimensional function spaces, promise efficient evaluations of PDE solutions for new parameter sets in a single forward pass. However, classical DeepONet entails quadratic time complexity concerning input dimensions during evaluation. Given the progress in quantum algorithms and hardware, we propose utilizing quantum computing to accelerate DeepONet evaluations, resulting in time complexity that is linear in input dimensions. Our approach integrates unary encoding and orthogonal quantum layers to facilitate this process. We benchmark our Quantum DeepONet using a variety of equations, including the first-order linear ordinary differential equation, advection equation, and Burgers' equation, demonstrating the method's efficacy in both ideal and noisy conditions. Furthermore, we show that our quantum DeepONet can also be informed by physics, minimizing its reliance on extensive data collection. We expect Quantum DeepONet to be particularly advantageous in applications in outer loop problems which require to explore parameter space and solving the corresponding PDEs, such as forward uncertainty propagation and optimal experimental design.

1 Introduction

Partial differential equations (PDEs) play a crucial role in modeling complex phenomena that are fundamental to both natural and engineered systems. Traditional numerical methods, such as finite difference, finite element, and finite volume methods, typically involve discretizing the solution space and solving finite-dimensional problems. These approaches, however, are computationally intensive and require a complete re-solving of equations with even minor adjustments to the system. Recently, neural networks have been employed to learn the solutions of PDEs [1, 2, 3, 4, 5, 6]. In particular, physics-informed neural networks (PINNs) embed the PDE residual into the loss term [7, 8, 1], demonstrating potential in solving both forward and inverse problems [9, 10, 11, 12]. Despite their promise, many of these methods remain mesh-dependent or require re-training when new functional parameters are introduced.

To address these limitations, deep neural operators have gained popularity for learning the mapping between infinite-dimensional spaces of functions through data [13, 14, 15, 16, 17, 18]. Once

trained, neural operators are able to efficiently evaluate the PDE solutions for a new PDE instance in a single forward pass. Additionally, the output of neural operators can be discretized at different levels of resolutions or evaluated at any points. The training of neural operators can also incorporate physics priors [19, 20], aligning the concept of PINNs, which has been shown to enhance accuracy significantly. The main categories of neural operators include integral kernel operators [15, 14, 21], transformer-based neural operators [17, 16], and DeepONet [13]. Integral kernel operators, such as Fourier neural operator (FNO) [14], leverage iterative learnable kernel integration, but are usually restricted to grids. Transformer-based neural operator has larger model capacity, but relies on sufficient data to achieve optimal performance. DeepONet, grounded in universal approximation theorem [22], on the other hand, can evaluate the solution of PDEs at any points in a mesh-free manner. There have been a wide range of developments of DeepONet [23, 24, 25, 26, 27], highlighting its adaptability in various complex systems.

While classical developments greatly expand the potential of neural networks, quantum neural networks (QNNs) have also drawn much attention due to the potential of better complexity and higher capacities compared to their classical counterparts [28, 29, 30]. Such advantages often directly come from the ability to efficiently encode and explore the exponentially large space on quantum computers [31]. Specifically, there are quantum algorithms that demonstrate the quadratic speedup in online perceptron [32] and reinforcement learning [33], as well as the exponential speedup in linear-system solving [34, 35], least-square fitting [36], Boltzmann machine [37], principal component analysis [38], and support vector machine [39].

Neural operators present an ideal application scenario for quantum neural networks designed for accelerating the evaluation process, especially in situations where they are evaluated repeatedly in "outer-loop problems", such as forward uncertainty propagation and optimal experimental design. There is a recent development of quantum Fourier neural operator (QFNO) [40]. Utilizing a new form of the quantum Fourier transform, QFNO is expected to be substantially faster than classical FNO in evaluation: requiring a logarithmic number of evaluations of the initial condition function, an improvement from the linear dependency in the classical FNO. The success of QFNO motivates us to explore the possibility of accelerating other neural operators, such as DeepONet.

However, as suggested by Refs. [41, 42, 43, 44, 45], the data embedding of classical datasets on quantum computers and hardware noise can induce barren plateaus and local minima that damage the trainability of quantum neural networks. It is even more problematic for the optimizers relying on the Fisher information matrix because they require exponentially many measurement shots to achieve accurate computation in barren plateaus [43].

In this study, we design an architecture for quantum DeepONet and quantum physics-informed DeepONet (QPI-DeepONet). To circumvent the trainability issue in QNN, we incorporate classical training and quantum evaluation by employing the orthogonal neural network structure outlined in Ref. [46]. Our work preserves the quadratic speed-up with respect to the input dimension in the feed-forward pass from the quantum orthogonal neural network, with a minimal cost for classical data preprocessing before training. The results of our numerical experiments suggest the effectiveness of neural networks in solving different PDEs in both ideal and noisy environments. We also analyze the impact of quantum noise on our quantum DeepONet.

The paper is organized as follows. We first present the algorithm and architecture of quantum DeepONet in Section 2. In Section 3, we illustrate the ideal quantum simulation results of different applications of our quantum DeepONet. Then we investigate quantum noise and show the performance of the quantum DeepONet under two different noise models in Section 4. Finally, we conclude our work and discuss the limitations in Section 5. The background concepts related to quantum computing is provided in Appendices A and B.

2 Methods

In this section, we first introduce a specific quantum circuit for network layers in Section 2.1, referred to as "quantum layers", which are designed for constructing quantum orthogonal neural networks in Section 2.2. Building on these foundations, we propose a novel quantum DeepONet structure by synthesizing multiple quantum layers in Section 2.3. The training method and loss function are detailed in Section 2.4. Furthermore, in addition to data driven training, we also propose to use physics-informed loss function, developing quantum physics-informed DeepONet (QPI-DeepONet) in Section 2.5.

2.1 Quantum methods for network layers

A classical neural network layer, with the input $\mathbf{x} \in \mathbb{R}^n$ and output $\mathbf{x}' \in \mathbb{R}^m$, takes the form $\mathbf{x}' = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$. Here, $\mathbf{W} \in \mathbb{R}^{m\times n}$ represents the weight matrix, $\mathbf{b} \in \mathbb{R}^m$ is the bias, and σ is the activation function. As demonstrated by Ref. [46], the matrix multiplication $\mathbf{W}\mathbf{x}$ can be accelerated by substituting the classical matrix multiplication with quantum matrix multiplication. The neural network layer accelerated by this quantum algorithm is referred as a quantum layer. We provide a detailed explanation of each step of a quantum layer, beginning with an introduction to the basic gate, the reconfigurable beam splitter (RBS) gate, used in our method (Section 2.1.1). The whole process involves three key steps to handle classical data on a quantum computer: (1) loading the classical data onto the quantum circuit (Section 2.1.2), (2) performing matrix multiplication on quantum computer (Section 2.1.3), and (3) converting the resulting quantum data back into classical data (Section 2.1.4). We summarize and provide the complexity of each step in Section 2.1.5.

2.1.1 Reconfigurable beam splitter gate

We first introduce reconfigurable beam splitter (RBS) gate [46] as a basic tool used in our quantum layer:

$$U_{RBS}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

where its basis-gate decomposition is illustrated in Appendix A. It is basically performing rotation operation on state $|01\rangle \mapsto \cos\theta\,|01\rangle - \sin\theta\,|10\rangle$ and $|10\rangle \mapsto -\sin\theta\,|01\rangle + \cos\theta\,|10\rangle$, while leaving $|00\rangle$ and $|11\rangle$ unchanged. By carefully designing the circuit using RBS gates and setting θ to required value, we efficiently load data (Section 2.1.2) and perform specialized matrix multiplication operations (Section 2.1.3).

2.1.2 Loading classical data input

For a classical vector $\mathbf{x} \in \mathbb{R}^n$, to perform operations on quantum computers, this classical vector must be converted into a quantum state. It is essential to ensure that the norm $\|\mathbf{x}\|_2 = 1$, as required by the probabilistic nature of quantum mechanics.

If the condition is not met, normalization should be performed. For the first quantum layer in the neural network, to avoid losing information of input data during normalization, we append an additional dimension to $\mathbf{x}$, which keeps the norm of $\mathbf{x}$ at 1 and in the meantime store the information of the original norm of $\mathbf{x}$. In detail, each element of $\mathbf{x}$ is first rescaled to the range $[-1, 1]$. Then the value $\sqrt{1 - \sum_i x_i^2/d}$ is assigned to the new dimension, where d represent the

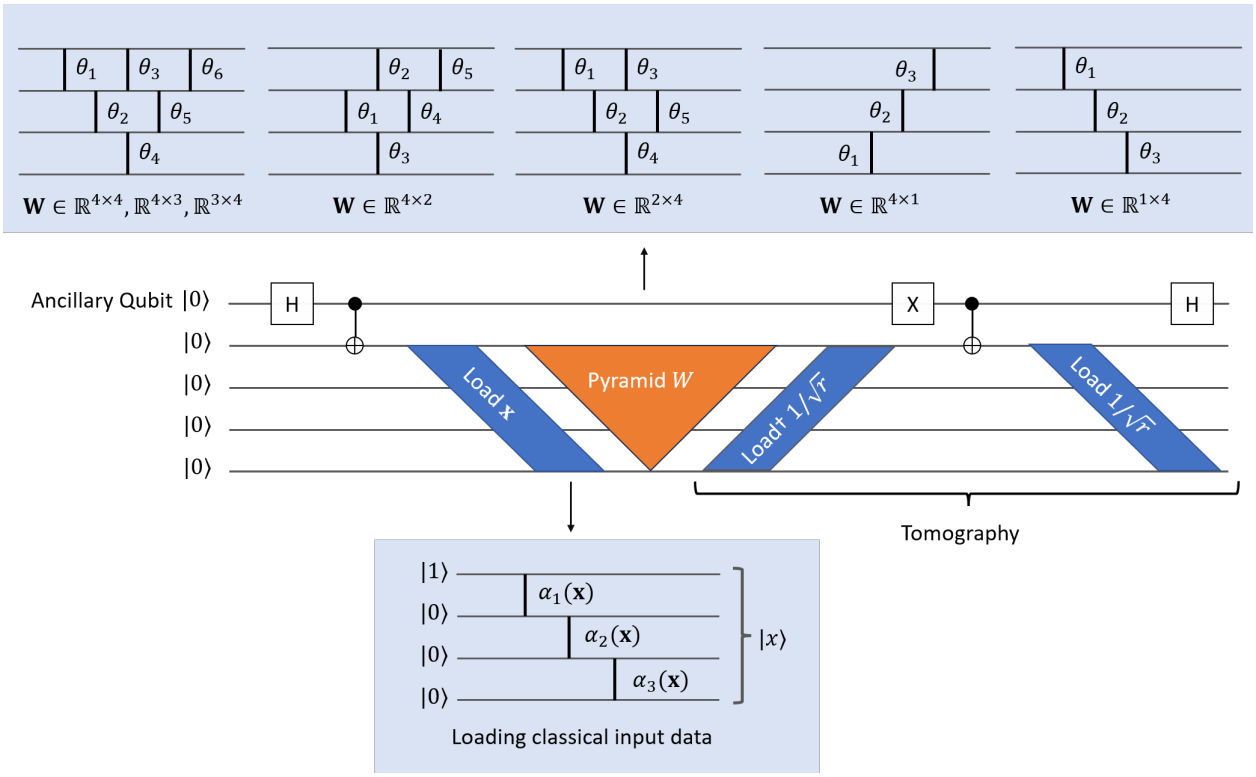

Figure 1: **The circuits of a quantum layer.** A quantum layer is composed of data loading, pyramidal circuit, and tomography. An ancillary qubit is included for the purpose of tomography. The vertical lines represent the two-qubit RBS gates, while the θ_1 and α_i correspond to the parameter of the gate. We provide the example of data loader for loading the classical vector $\mathbf{x} \in \mathbb{R}^4$ with $\|\mathbf{x}\|_2 = 1$. We demonstrate quantum pyramidal circuit using all of the seven examples. $\mathbf{W} \in \mathbb{R}^{4\times4}$, $\mathbf{W} \in \mathbb{R}^{4\times3}$, and $\mathbf{W} \in \mathbb{R}^{3\times4}$ share the same pyramidal circuit. The following circuit are other examples of $m \neq n$ cases: $\mathbf{W} \in \mathbb{R}^{4\times2}$, $\mathbf{W} \in \mathbb{R}^{4\times1}$, $\mathbf{W} \in \mathbb{R}^{2\times4}$ and $\mathbf{W} \in \mathbb{R}^{1\times4}$.

original dimensionality of $\mathbf{x}$. This procedure can be viewed as data preprocessing before training, transforming the original $\mathbf{x}$ into

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_m \\ \sqrt{1-\sum_i x_i^2/d} \end{pmatrix},$$

where x_i is the ith element of $\mathbf{x}$. For subsequent quantum layers in the neural network, we simply dividing $\mathbf{x}$ by $\|\mathbf{x}\|_2$ before loading the data.

The circuit for loading data is shown in Fig. 1 bottom. If $m = n$, the quantum circuit we adopt will have n qubits, initialized such that the first qubit is at state $|1\rangle$, while remaining qubits are $|0\rangle$. Then we apply a series of RBS gates parameterized by $(\alpha_1, \alpha_2, \dots, \alpha_{n-1})$, where

$$\begin{aligned} \alpha_1(\mathbf{x}) &= \arccos(x_1), \\ \alpha_2(\mathbf{x}) &= \arccos\left(x_2 \sin^{-1}(\alpha_1)\right), \\ \alpha_3(\mathbf{x}) &= \arccos\left(x_3 \sin^{-1}(\alpha_2) \sin^{-1}(\alpha_1)\right), \end{aligned}$$

and so on. This sequence of operations converts the initial quantum state to

$$\begin{aligned} |x\rangle &= \cos\alpha_1 |10\dots0\rangle + \sin\alpha_1 \cos\alpha_2 |01\dots0\rangle + \dots + \sin\alpha_1 \sin\alpha_2 \dots \sin\alpha_{n-1} |00\dots1\rangle \\ &= x_1 \underbrace{|10\dots0\rangle}_{|\mathbf{e}_1\rangle} + x_2 \underbrace{|01\dots0\rangle}_{|\mathbf{e}_2\rangle} + \dots + x_n \underbrace{|00\dots1\rangle}_{|\mathbf{e}_n\rangle} \end{aligned}$$

On the other hand, when input and output dimensions are different ($n \neq m$), the number of qubits required in the circuit will be $\max(m, n)$. If $n > m$, the data is loaded onto all of the n qubits. Conversely, if $n < m$, the classical vector $\mathbf{x}$ is loaded on to the bottom n qubits in the circuit, leaving upper $m - n$ qubits at $|0\rangle$.

2.1.3 Quantum pyramidal circuit

When classical data $\mathbf{x}$ is loaded onto the quantum circuit, matrix multiplication $\mathbf{y} = \mathbf{W}\mathbf{x}$ can be performed in quantum space, where $\mathbf{y} \in \mathbb{R}^m$. Here we adopt the quantum pyramidal circuit proposed in Ref. [46]. Such pyramidal circuit features orthogonal matrix multiplication, i.e., the corresponding $\mathbf{W}$ is orthogonal.

We first introduce the quantum pyramidal circuit for $m = n$ cases. The basic idea of this method is to decomposes the orthogonal matrix $\mathbf{W}$ into a series of rotation matrices, which can be represented by RBS gates. These decomposed rotation matrices can be parameterized with angles $\theta_1, \theta_2 \dots, \theta_d$, where $d = n(n-1)/2$. All of the parameterized RBS gates are arranged in a pyramid configuration. We take the $\mathbf{W} \in \mathbb{R}^{4\times4}$ matrix in Fig. 1 as an example. This circuit conducts the following operation on the loaded vector x:

$$\mathbf{y} = \underbrace{\begin{pmatrix} C_{\theta_1} & S_{\theta_1} & & \\ -S_{\theta_1} & C_{\theta_1} & & \\ & & 1 & \\ & & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & C_{\theta_2} & S_{\theta_2} & \\ & -S_{\theta_2} & C_{\theta_2} & \\ & & & 1 \end{pmatrix} \begin{pmatrix} C_{\theta_3} & S_{\theta_3} & & \\ -S_{\theta_3} & C_{\theta_3} & & \\ & & C_{\theta_4} & S_{\theta_4} \\ & & -S_{\theta_4} & C_{\theta_4} \end{pmatrix} \begin{pmatrix} 1 & & & \\ & C_{\theta_5} & S_{\theta_5} & \\ & -S_{\theta_5} & C_{\theta_5} & \\ & & & 1 \end{pmatrix} \begin{pmatrix} C_{\theta_6} & S_{\theta_6} & & \\ -S_{\theta_6} & C_{\theta_6} & & \\ & & 1 & \\ & & & 1 \end{pmatrix}}_{\mathbf{W}} \mathbf{x},$$

where C_{θ_j} and S_{θ_j} are $\cos\theta_j$ and $\sin\theta_j$ for any j, respectively. Therefore, the resulting quantum state is

$$|y\rangle = |Wx\rangle = \sum_{ij} W_{ij} x_i |\mathbf{e}_j\rangle.$$

If $m \neq n$, the construction of pyramidal circuit is the same as Ref. [46]. Examples of this include $\mathbf{W} \in \mathbb{R}^{4\times 1}$, $\mathbf{W} \in \mathbb{R}^{1\times 4}$ and so on, as shown in Fig. 1. Note that for $|m-n|=1$ cases, the pyramidal circuit is the same as $m=n$ cases. However, due to the difference in data loading and tomography process, the quantum layer is actually distinct.

2.1.4 Tomography for extracting classical output

After performing matrix multiplication in quantum space, it is necessary to convert the quantum information back to classical form for further processing, such as adding bias and applying non-linear transformation. This process is known as tomography. Tomography could commonly be expensive when extract complete information from quantum states [47, 48, 49]. However, in our method, the usage of unary state sparsely encodes information in Hilbert space and provides a feasible, cheap, and efficient tomography method. This tomography method, proposed by Ref. [46], is illustrated in Fig. 1 middle.

Here, exactly one qubit is in state $|1\rangle$ and all others are in state $|0\rangle$, where state is referred to as "unary state". For simplicity, the jth unary state is denoted as $|\mathbf{e}_j\rangle$. Therefore, the information of $\mathbf{x}$ is encapsulated in $|x\rangle$ represented as the superposition of these unary states. Once data is loaded into the superposition of unary states, all of our subsequent operations, which utilize the RBS gate and only include transformations between $|\mathbf{e}_j\rangle$ states, are effectively confined to these unary states. This implies that the unary subspace throughout entire process, allowing us to employ the tomography.

We introduce an ancillary qubit and implement a Hadamard (H) and a CNOT gate between the ancillary qubit and the first data loader qubit before loading data (see Appendix A for the definition of gates). After the pyramid gate, the circuit performs an adjoint operation of the data loader of a uniform norm-1 vector $\left(\frac{1}{\sqrt{r}}, \frac{1}{\sqrt{r}}, \ldots, \frac{1}{\sqrt{r}}\right)$, where $r = \max(m,n)$ represents the number of qubits excluding the ancillary qubit. This is followed by an X gate and CNOT gate.

Finally, we load $\left(\frac{1}{\sqrt{r}}, \frac{1}{\sqrt{r}}, \ldots, \frac{1}{\sqrt{r}}\right)$ and a Hadamard gate. In this way, the output can be represented by

$$y_j = \sum_i W_{ij} x_i = \sqrt{r}(\Pr[0, \mathbf{e}_j] - \Pr[1, \mathbf{e}_j]), \tag{1}$$

where $Pr[\xi, \mathbf{e}_j]$ means the the ancillary qubit is measured as a classical bit ξ, for $\xi \in \{0,1\}$, and the rest qubits are measured as $\mathbf{e}_j$. As a result, the value of $\sum_i W_{ij}x_i$ can be simply computed from the probabilities of $|0, \mathbf{e}_j\rangle$ and $|1, \mathbf{e}_j\rangle$ for all needed j.

If the input dimension is larger than the output dimension ($m > n$), the tomography circuit is still the same, but only the information of bottom m qubits are finally considered. In other words, the $\mathbf{e}_j$ in Eq. (1) refers to jth unary state for the bottom m qubits. Consequently, the output $\sum_i W_{ij}x_i$ is restricted to size m.

2.1.5 Summary and remarks

In conclusion, the structure of a complete quantum layer is shown in Fig. 1. The number of qubits needed is $n+1$ for $\mathbf{W} \in \mathbb{R}^{n\times n}$, in which bottom n qubits are used to store information and perform operations, while the top 1 qubit is included for tomography purpose. Sequentially, we implement data loading, pyramidal circuit and tomography, and thus complete the matrix multiplication in quantum space.

Complexity. Quantum layers can accelerate the feedforward pass, achieving a complexity of $O(n/\delta^2)$. Here, δ is the threshold for the tomography error. The complexities of other components of quantum layers are shown in Table 1.

Table 1: **Complexity of each step of a quantum layer.** Here, n is the input dimension, and δ represents the threshold for the tomography error.

Operation	Doading input data	Quantum pyramidal circuit	Extracting output
Complexity	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n/\delta^2)$

2.2 Quantum orthogonal neural network

By integrating multiple quantum layers, we can construct a quantum orthogonal neural network (QOrthoNN). The input vector goes through a linear transformation in quantum space and is then measured and convert to classical space (Fig. 2). Although not shown in the diagram, we add bias and apply non-linear transform thereafter. We proceed to the next layer and perform similar process. The sequence can be repeated several times until we reach the last layer, which consists solely of a classical linear transform. The dimension and norm of the quantum neural network output of is determined by the output layer, giving that former quantum layers always constrain the norm of processed vector to be 1.

The comparison between the classical orthogonal neural network and the standard neural network is presented in Table 2. By "classical orthogonal neural network" (OrthoNN), we refer to a classical neural network that adopts the same mathematical formulation as QOrthoNN. This network is designed to facilitate the training of QOrthoNN, which will be further explained in Section 2.4. OrthoNN benefits from the properties of orthogonality, such as improved accuracy and better convergence during training [50, 51], while maintaining the same asymptotic running time as a standard neural network. While OrthoNN and standard neural network both have a quadratic dependency on the input dimension n for the forward pass, the QOrthoNN only requires a linear dependency, achieving a quadratic improvement in the input dimension. This reduction in computational complexity is particularly beneficial in scenarios where the input dimension is large and frequent evaluations are required.

Table 2: **Comparison of complexity for three networks.** n and δ represent the input dimension and threshold for the tomography error, respectively.

Algorithm	Feedforward pass	Weight matrix update
Quantum orthogonal neural network (QOrthoNN) [46]	$\mathcal{O}(n/\delta^2)$	–
Classical orthogonal neural network (OrthoNN) [46]	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Standard neural network	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$

2.3 Quantum DeepONet

DeepONet is a neural network architecture that aims to learn operators mapping between two infinite-dimensional function spaces. The most popular application of DeepONet is solving PDEs. Our goal is often to predict functions satisfying the PDEs under varying conditions, which could

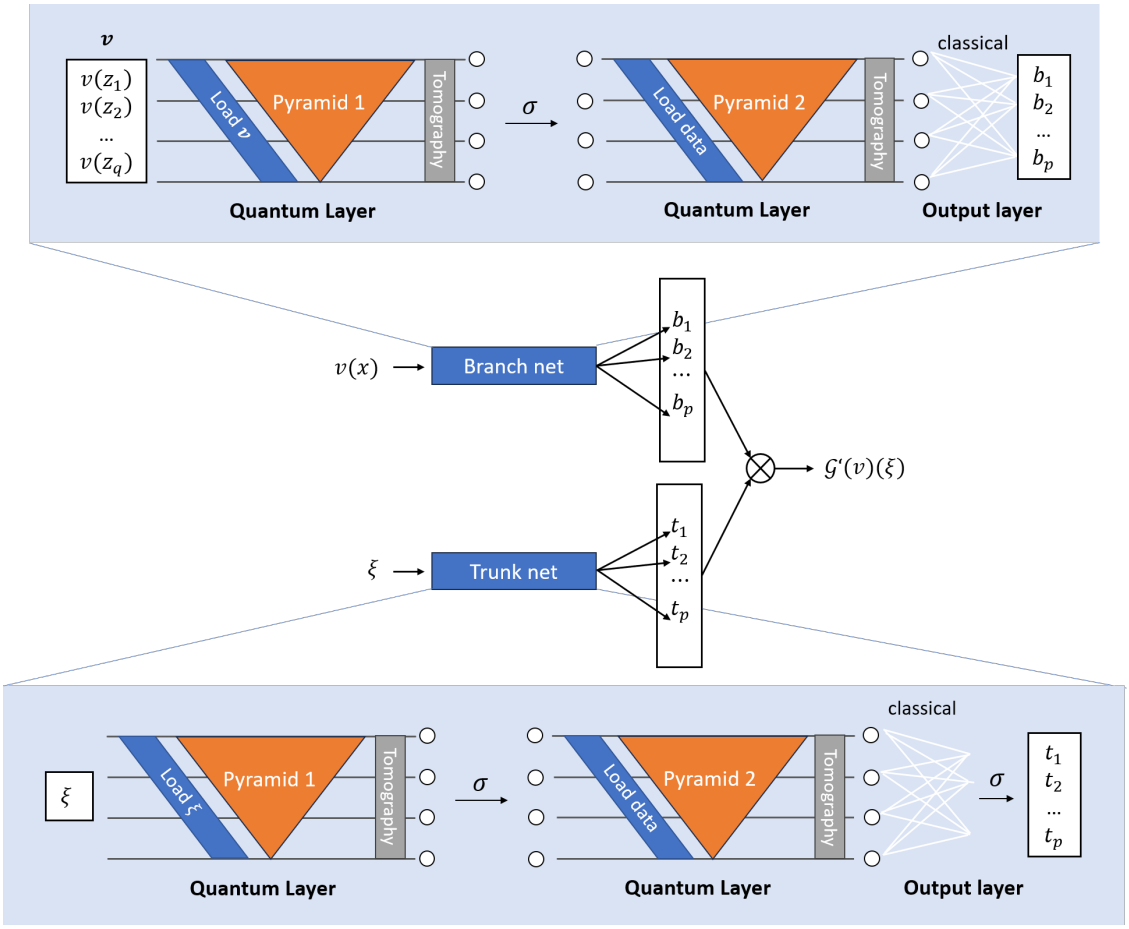

Figure 2: **Architecture of quantum DeepONet.** DeepONet consists of two subnetworks: the branch net and the trunk net. In quantum DeepONet, we replace these with QOrthoNN, which is composed of several quantum layers arranged sequentially. The nonlinear operations are performed on classical computers.

be the initial conditions, boundary conditions or coefficient fields of the PDEs. We define the input function $v \in \mathcal{V}$ over the domain $D \subset \mathbb{R}^d$ as

$$v : D \ni x \mapsto v(x) \in \mathbb{R},$$

and similarly, we define the output function $u \in \mathcal{U}$ over $D' \subset \mathbb{R}^{d'}$, which is described as

$$u : D' \ni \xi \mapsto u(\xi) \in \mathbb{R}.$$

Suppose $\mathcal{V}$ and $\mathcal{U}$ are Banach spaces, and consider a parametric PDE taking the form

$$\mathcal{N}(v, u) = 0,$$

where $\mathcal{N}$ is a differential operator. The mapping between the input function space $\mathcal{V}$ and output function space $\mathcal{U}$ is defined the operator:

$$\mathcal{G} : \mathcal{V} \ni v \mapsto u \in \mathcal{U}.$$

DeepONet, therefore, is used to approximate $\mathcal{G}$.

A DeepONet includes a branch net and trunk net, each with an equivalent number of output neuron, denoted by p. The branch and trunk nets can adopt arbitrary architectures, like fully connected neural network (FNN), convolutional neural network (CNN), recurrent neural network (RNN), and residual neural network (ResNet). A diagrammatic representation of DeepONet is illustrated in the center of Fig. 2. The branch network receives the input function evaluated at a discrete set of points $\{z_1, z_2, \ldots, z_q\}$, represented by $[v(z_1), v(z_2), \ldots, v(z_q)]$. The trunk net is fed with the location ξ at which the output function is evaluated, which can include both time and space coordinates. The outputs of the branch and trunk networks are denoted by $[b_1(v), b_2(v), \ldots, b_p(v)]$ and $[t_1(\xi), t_2(\xi), \ldots, t_p(\xi)]$. Thus, the final output of DeepONet is the sum of the dot product of the branch and trunk network outputs and a bias $b_0 \in \mathbb{R}$, y expressed as

$$\mathcal{G}'_\theta(v)(\xi) = \sum_{k=1}^{p} b_k(v) t_k(\xi) + b_0,$$

where $\mathcal{G}'$ denotes the learned approximation of operator $\mathcal{G}$, and θ is the trainable parameter of the network.

In this work, we propose a modification to the DeepONet framework by replacing the conventional branch and trunk networks with QOrthoNN (Fig. 2). We refer to the resulting model as quantum DeepONet.

2.4 Training quantum DeepONet

Up to this point, we have introduced QOrthoNN and the quantum DeepONet, but we have not yet discussed the training process of these quantum networks. Adapting the backpropagation scheme from Ref. [46] for the pyramidal circuit, we train the network on classical computers, utilizing a classical orthogonal neural network (OrthoNN) that shares the same mathematical expression as QOthoNN. After training, we substitute the angular parameters of the RBS gates in the quantum circuits with trained paramters during the evaluation phase. It is during this evaluation phase on quantum computers that we anticipate significant acceleration benefits.

For data-driven training of quantum DeepONet, we sample N distinct input functions $\{v^{(i)}\}_{i=1}^{N}$ from $\mathcal{V}$, and Q locations $\{\xi_j^{(i)}\}_{j=1}^{Q}$ for each input function $v^{(i)}$ as the inputs of training dataset. The

corresponding solution $\mathcal{G}(v^{(i)})(\xi_j^{(i)})$ is taken as the label of training dataset. The loss of DeepONet can therefore be expressed as

$$\mathcal{L}_{\text{operator}}(\theta) = \frac{1}{NQ} \sum_{i=1}^{N} \sum_{j=1}^{Q} \left| \mathcal{G}'_\theta(v^{(i)})(\xi_j^{(i)}) - \mathcal{G}(v^{(i)})(\xi_j^{(i)}) \right|^2. \tag{2}$$

To summarize, the workflow of our quantum method is divided into three distinct phases:

- Training quantum DeepONet on classical computer;
- Transferring of parameters to quantum layer;
- Execution on quantum computer or simulator for evaluation.

2.5 Quantum physics-informed DeepONet

We further introduce physics-informed loss term during training,

$$\mathcal{L}_{\text{physics}}(\boldsymbol{\theta}) = \frac{1}{NQ} \sum_{i=1}^{N} \sum_{j=1}^{Q} \left| \mathcal{N}\left(v^{(i)}, \mathcal{G}'_\theta(v^{(i)})(\xi_j^{(i)})\right) \right|^2.$$

The total loss function is therefore

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\text{physics}}(\boldsymbol{\theta}) + \mathcal{L}_{\text{operator}}(\boldsymbol{\theta}),$$

where $\mathcal{L}_{\text{operator}}$ has the same definition as Eq. (2). In PI-DeepONet, $\mathcal{L}_{\text{operator}}$ only includes the initial conditions and boundary conditions. We name such architecture as quantum physics-informed DeepONet (QPI-DeepONet). By introducing the physics information into our network, we can reduce the demand of data and even train the network in the absence of solution input-output pairs. In evaluation stage, QPI-DeepONet follows the same procedure as ordinary quantum DeepONet.

In some cases, we can embed boundary conditions into the network architecture, known as hard constrain [52]. For example, to enforce Dirichlet BCs $\mathcal{G}_\theta(v)(\xi) = g(\xi)$ for $\xi \in \Gamma_D$, we can construct the quantum DeepONet output as

$$\mathcal{G}''_\theta(v)(\xi) = g(\xi) + \ell(\xi)\mathcal{G}'_\theta(v)(\xi),$$

where $\mathcal{G}'_\theta(v)(\xi)$ is the output of vanilla quantum DeepONet, and $\ell(\xi)$ satisfy

$$\begin{cases} \ell(\xi) = 0, & \xi \in \Gamma_D, \\ \ell(\xi) > 0, & \text{otherwise.} \end{cases}$$

For periodic boundary condition, e.g., $\mathcal{G}(v)(\xi)$ is periodic with respect to ξ of the period P in 1D, we can directly substitute trunk input ξ with Fourier basis

$$\{1, \cos(\omega\xi), \sin(\omega\xi), \cos(2\omega\xi), \sin(2\omega\xi), \ldots\}$$

with $\omega = 2\pi/P$.

The branch inputs of DeepONet are often high-dimensional to include the information of input functions, especially for less smooth v, more sensors are needed [13]. Here we apply principal component analysis (PCA) to reduce input dimension [53].

3 Ideal quantum simulation results

To demonstrate the efficacy of our method, we first use QOrthoNN to approximate certain functions (Section 3.1). Subsequently, we move to the application of quantum DeepONet on learning ODE and PDE problems, including the antiderivative operator (Section 3.2), advection equation (Section 3.3), and Burgers' equation (Section 3.4). Finally, we test QPI-DeepONet using antiderivative operator and Poisson's equation (Section 3.5).

We implement the classical training by using the library DeepXDE [8]. After classical training on OrthoNN, we extract the weights and biases and construct a quantum version incorporating quantum layers, applying Qiskit [54] for quantum simulation. It is important to note that, in this section, we adopt an idealized scenario during quantum simulation. This approach exclude any quantum and statistical noise, aiming to assess the theoretical accuracy and performance of the quantum model. The hyperparameters of neural networks and L^2 relative errors of different examples are summarized in Table 3. The code of all examples are published in GitHub (https://github.com/lu-group/quantum-deeponet).

Table 3: **The default parameters and test error for different examples of quantum DeepONet.** For quantum DeepONet, the first number in the "Depth" column is the depth of branch net, and the second number is the depth of trunk net. The same for the "Activation" column.

Example	Depth	Width	Activation	Learning rate	Iteration	Error
§3.1 Function 1	3	3	Tanh	0.0001	5×10^4	0.15%
§3.1 Function 2	4	10	ReLU	0.0005	4×10^4	1.49%
§3.2 Antiderivative ($l = 1.0$)	[2,2]	10	ReLU, ReLU	0.001	3×10^4	0.49%
§3.2 Antiderivative ($l = 0.5$)	[2,2]	20	ReLU, ReLU	0.001	3×10^4	0.84%
§3.3 Advection	[7,7]	21	SiLU, SiLU	0.0005	4×10^4	2.25%
§3.4 Burgers'	[6,6]	20	SiLU, SiLU	0.0005	3×10^4	1.38%

3.1 Function approximation

In this section, we adopt two functions to test the accuracy of QOrthoNN. We first consider a function

$$\text{Function 1:} \qquad f(x) = \frac{1}{1 + 25x^2}, \quad x \in [-1, 1]$$

and approximate it using OrthoNN and the normalization mentioned in Section 2.1.2. We choose 80 points for training and 100 points for testing, where x is uniformly sampled in $[-1, 1]$. Specially, for this example, we use tanh activation function to circumvent the "dying ReLU" problem [55], which is particularly relevant here given the small width of the network.

For this function 1, we can achieve a small L^2 relative error of 0.149% for testing set after training classically. Following classical training, we construct QOrthoNN using the pyramid quantum circuit according to the classically training parameter. The ideal quantum simulation yields an error identical to classical training: 0.149% (Fig. 3A). Essentially, OrthoNN and QOrthoNN are the same neural network, differing only in their prediction methods—one is executed on a classical computer, while the other is run on a quantum simulator.

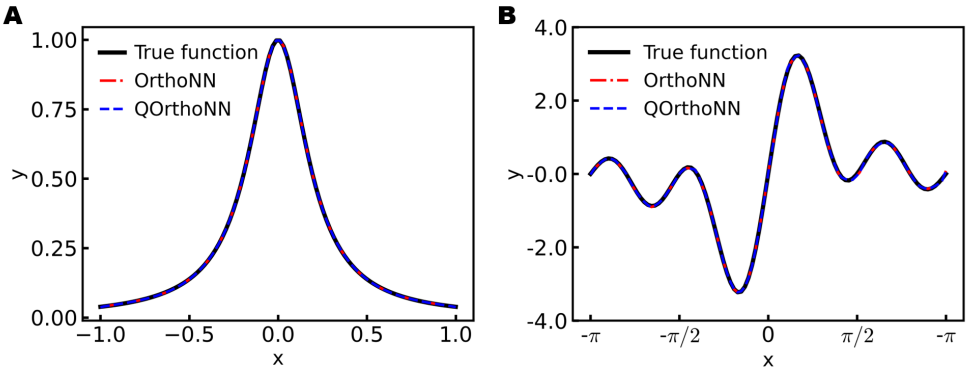

Figure 3: **Quantum simulation result of function predictions.** The black, red and blue lines represent the reference solution, classical prediction of OrthoNN, and ideal quantum simulation result of QOrthoNN, respectively. (**A**) Results for $f(x) = 1/(1 + 25x^2)$. (**B**) Results for $f(x) = \sum_{k=1}^{4} \sin(kx)$.

Then, we consider a more complex case for function approximation:

$$\text{Function 2:} \qquad f(x) = \sum_{k=1}^{4} \sin(kx), \quad x \in [-\pi, \pi].$$

We use 200 training points, and 100 testing points. Three quantum layers and an output layer with a width of 10 are adopted in the training. The testing error reaches a low relative error of 1.49% (Fig. 3B), highlighting the proficiency of classical orthogonal FNN and QOrthoNN. The ideal quantum simulation result is also 1.49%.

3.2 Antiderivative operator

Next we exam quantum DeepONet. We begin with an antiderivate operator:

$$\frac{du(x)}{dx} = v(x), \quad x \in [0, 1], \tag{3}$$

with initial condition $u(0) = 0$. Here, we aim to learn the operator

$$\mathcal{G} : v \to u.$$

To generate the input function $v(x)$, we use Gaussian Random Field (GRF):

$$v \sim \mathcal{G}(0, k_l(x_1, x_2)),$$

where $k_l(x_i, x_j) = \exp(-\frac{d(x_i,x_j)^2}{2l^2})$ denotes the radial basis function (RBF) kernel. In this context, $d(\cdot, \cdot)$ is the Euclidean distance between two points and l represents the length scale of the kernel, which modulates the smoothness of the generated function. Specifically, an increase of the value of l leads to a smoother generated function. Therefore, we can adjust l depending on our desired level of function's complexity.

In this example, we explore two scenarios with different length scales: $l = 1.0$ and $l = 0.5$, corresponding to different size of the network during training. We achieved small errors of 0.49% and 0.84% in theses two senarios, respetively (see Table 3).

3.3 Advection Equation

Consider the 1D advection equation:

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0, \quad x \in [0, 1], \ t \in [0, 1],$$

with initial condition $u(x, 0) = u_0(x)$ and periodic boundary condition. Our objective is to learn the operator that maps $u_0(x)$ to the solution $u(x, t)$:

$$\mathcal{G} : u_0(x) \mapsto u(x, t).$$

The initial condition $u_0(x)$ is sampled from GRF with Exp-Sine-Squared kernel, formulated as

$$k(x_i, x_j) = \exp\Big(-\frac{2\sin^2(\pi d(x_i, x_j)/p)}{l^2}\Big).$$

Here, p is the periodicity of the kernel and is set to 1. We choose $l = 1.5$ and derive the ground truth using the analytical solution $u(x, t) = u_0(x - t)$. For branch input $u_0(x)$, 20 sensors are uniformly placed (see one example in Fig. 4A left). Regarding trunk input, we employ a grid of 50×50 points, covering the range of x and t. We implement the ResNet architecture in both branch and trunk nets, which has a formulation of $\mathbf{x}' = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{x}$ for each layer. This approach effectively mitigate the issue of gradient vanishing during training. The final test error of classical prediction reaches 2.25%. Ideal simulation of quantum DeepONet yeilds the same error: 2.25%. Fig. 4A provides an example of illustrating the ground truth, predictions of quantum DeepONet and the absolute error between them.

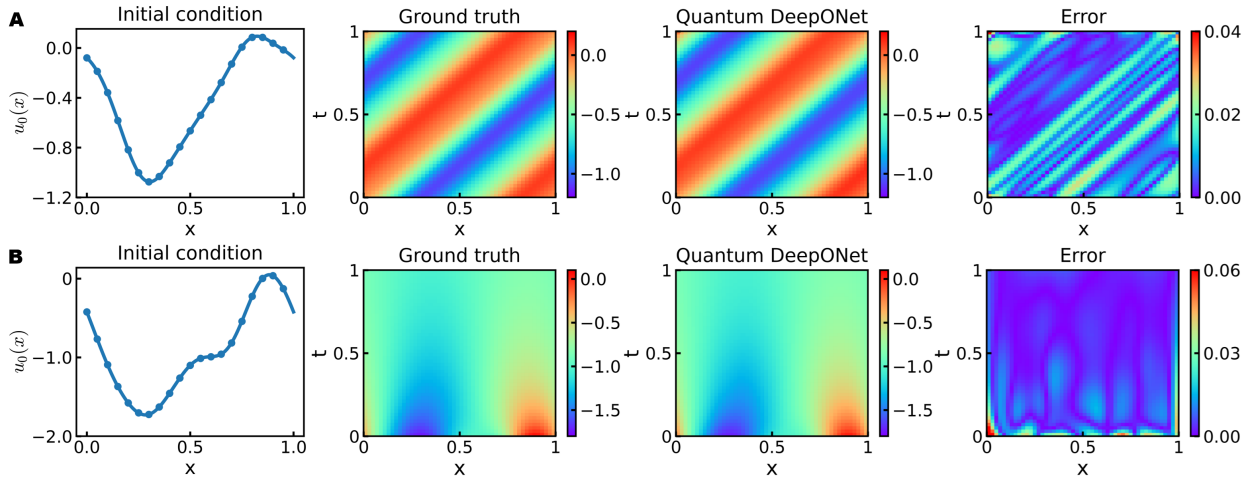

Figure 4: **Examples of quantum DeepONet prediction for two PDEs.** (**A**) Advection equation. (**B**) Burgers' equation.

3.4 Burgers' Equation

Based on the linear advection equation example, we further examine the non-linear 1D Burgers' equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in [0, 1], \ t \in [0, 1],$$

with initial condition $u_0(x)$ and periodic boundary condition, where $\nu = 0.05$ is the viscosity. We aim to learn the mapping from $u_0(x)$ to the solution $u(x,t)$. Additionally, recognizing the periodic nature of the output function $u(x,t)$, instead of directly input ξ for trunk net, we expand it to $[\xi, \cos(2\pi\xi), \sin(2\pi\xi), \cos(4\pi\xi), \sin(4\pi\xi)]$. This modification ensures that the output $G(u_0)(\xi)$ also exhibits periodicity. Other neural network settings are the same as Section 3.3, except for the depth and width. The relative error is 1.38%. The ideal quantum simulation result is the same: 1.38%. An example of the ground truth and prediction of quantum DeepONet is shown in Fig. 4B.

3.5 Quantum physics-informed DeepONet

In this section, we further show that our quantum DeepONet can also be trained without labeled data. We choose antiderivative euqation in Eq. (3) for comparison with data driven case in Section 3.2. Additionally, 1D Poisson's equation

$$\frac{\partial^2 u}{\partial x^2} = v(x),$$

with zero Dirichlet boundary condition is also considered for demonstration. The branch inputs in both cases are the $v(x)$ in equations, which is generated by GRF with RBF kernel. To facilitate training and keep PDE residual within reasonable range, in Poisson's equation, we multiply generated GRF with a factor of 10 and take the enlarged function as input sample. The boundary condition is hard constrained using corresponding neural network architecture as mentioned in Section 2.5. Zero coordinate shift algorithm [56] is utilized to reduce GPU memory consumption and training time. During the training, the number of input samples is 10000 with batch size = 2000. Adam optimizing is used with 2×10^5 iteration. The PDE residual is evaluated at 100 uniformly distributed points in $[0, 1]$. During the training, we employed PCA with original dimension 100. The training result is shown in Table. 4. And ideal quantum simulation results also agree well with classical training results for all of these examples, shown "L^2 relative error" column.

We also conducted experiments without any dimension reduction techniques. For antiderivative with initial condition $l = 1$, using branch and trunk net with depth of 3 and width of 10, the test L^2 relative error is 4.05%. For comparison, using the same hyperparameters with PCA, which projects original 100 dimensions down to 10, resulted in an error of 0.76%. We believe this difference is due to the critical dependency of QPI-DeepONet on the sampling of input sensors. PCA enables us to incoporate more information within limited input dimension. The limitations of current quantum devices compel us to use narrower neural networks, leading to sparse sampling of the branch input. As derivatives are taken with respect to the inputs, QPI-DeepONet is more sensitive to the input data.

Table 4: **Hyperparameters and training results of QPI-DeepONet for two PDEs with various input function complexity.**

Example	Number of PCs	Depth	Width	L^2 relative error
Antiderivative ($l = 1$)	10	[3,3]	20	0.76%
Antiderivative ($l = 0.5$)	10	[4,4]	20	1.21%
Antiderivative ($l = 0.2$)	19	[5,5]	20	1.91%
Poisson's ($l = 1$)	10	[3,3]	20	0.95%
Poisson's ($l = 0.5$)	10	[5,5]	20	1.55%
Poisson's ($l = 0.2$)	19	[7,7]	20	2.31%

4 Effects of noise

Quantum noise is a major obstacle for the practicality of a quantum algorithm in the noisy intermediate-scale quantum (NISQ) era. It emerges from various sources, including the imperfect implementation of quantum operators, undesired environmental or qubit interactions, and erroneous state preparation or measurement. During the execution of a quantum circuit, the accumulated errors produced by the noise can destroy any information we intend to obtain. Meanwhile, the inaccuracy resulting from the finite number of measurements can affect the error level and complexity of the neural network, making it unavoidable in the discussion of the feasibility of our work on near-term quantum computers. Thus, in Sections 4.1 and 4.2, we first provide a theoretical analysis of the effects of a well-known noise channel, depolarizing noise, and finite-sampling noise on the single RBS gate and tomography outputs, respectively. Then, we demonstrate our noisy simulation results of quantum DeepONet under both types of noise, as well as a more comprehensive noise model emulating a real IBM quantum computer in Section 4.3.

4.1 Depolarizing noise on a RBS gate

The depolarizing noise is a widely adapted noise channel in analyzing the effects of quantum noise on variational quantum circuits [41, 57, 42, 58]. We provide a closer look at how depolarizing noise affects a QOrthoNN consisting of a single RBS gate and the influence of the parameter value of the gate on the level of error induced by the noise. Note that the statevector representation becomes insufficient to depict the quantum system under the influence of quantum noise. Hence, we utilize a density matrix to describe a quantum state and present a brief introduction to its definition and computation in Appendix B. The specific type of noise in our interest is depolarizing noise. The n-qubit depolarizing channel has the expression [59]

$$\mathcal{E}(\rho) = (1-\lambda)\rho + \lambda \frac{I^{(2^r)}}{2^r}, \tag{4}$$

where ρ is an arbitrary r-qubit density matrix and $I^{(2^r)}$ is a 2^r-by-2^r identity matrix. Specifically, in a 2-qubit case, Eq. (4) is equivalent to

$$\begin{aligned}\mathcal{E}(\rho) &= (1-\lambda)\rho + \lambda \frac{I^{(4)}}{4} \\ &= (1-\lambda)\rho + \frac{\lambda}{16} \sum_{i,j\in[4]} (L_i \otimes L_j)\rho(L_i \otimes L_j),\end{aligned} \tag{5}$$

where $L = \{X, Y, Z, I\}$ is the set of Pauli matrices and the 2-by-2 identity matrix I. In other words, the effect of 2-qubit depolarizing noise means there is $1 - 15\lambda/16$ chance that the state ρ remains unaffected and an equal chance to have each of 15 different kinds of 2-qubit Pauli noise happens on the state ρ. To further show the influence of a noisy RBS gate, we consider the noise model where a noiseless RBS gate is first applied on the state ρ, then a depolarizing channel follows, and the resultant state is ρ'. In particular, we have

$$\begin{aligned}\rho' &= \mathcal{E}\left(U_{RBS}\rho U_{RBS}^\dagger\right) \\ &= (1-\lambda)\left(U_{RBS}\rho U_{RBS}^\dagger\right) + \frac{\lambda}{16}\sum_{i,j\in[4]} (L_i \otimes L_j)\left(U_{RBS}\rho U_{RBS}^\dagger\right)(L_i \otimes L_j).\end{aligned} \tag{6}$$

However, the difference between ρ and ρ' is not in our interest since only the 2^{nd}, and the 3^{rd} elements of the diagonals of ρ and ρ' contain the information we need. So, we put the further discussion into a QOrthoNN scenario.

Define normalized input vector is $\mathbf{x} = (x_1 \quad x_2)^T \in \mathbb{R}^2$, $x_1^2 + x_2^2 = 1$, and the vector after the linear transformation is $\mathbf{y} = W\mathbf{x}$. Let $\mathbf{y}'$ denote the noisy version of $\mathbf{y}$ due to the depolarizing noise in a RBS gate and $\cdot^{\circ 2}$ represent the Hadamard (element-wise) square. Because we only encode the entry values of $\mathbf{x}$ and $\mathbf{y}$ on the coefficients of $|01\rangle$ and $|10\rangle$, the vector $\mathbf{y}^{\circ 2}$ is the vector consists of the 2^{nd} and 3^{rd} elements of the diagonal of $U_{RBS}\rho U_{RBS}^\dagger$ and $\left(\mathbf{y}^{\circ 2}\right)'$ is the vector of 2^{nd} and 3^{rd} elements of the diagonal of ρ'.

Define function $\text{diag} : \mathbb{R}^{r\times r} \to \mathbb{R}^r$ extract the diagonal of a matrix into a vector. In this case, the density matrix ρ is

$$\rho = \begin{pmatrix} 0 \\ x_1 \\ x_2 \\ 0 \end{pmatrix} (0 \quad x_1 \quad x_2 \quad 0) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & x_1^2 & x_1x_2 & 0 \\ 0 & x_1x_2 & x_2^2 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{7}$$

and

$$\mathbf{y}^{\circ 2} = \begin{pmatrix} y_1^2 \\ y_2^2 \end{pmatrix} = \begin{pmatrix} \text{diag}\left(U_{RBS}\rho U_{RBS}^\dagger\right)_2 \\ \text{diag}\left(U_{RBS}\rho U_{RBS}^\dagger\right)_3 \end{pmatrix} = \begin{pmatrix} (x_1\cos\theta + x_2\sin\theta)^2 \\ (-x_1\sin\theta + x_2\cos\theta)^2 \end{pmatrix}.$$

Combining Eqs. (6) and (7), the noisy output is

$$\begin{aligned} \left(\mathbf{y}^{\circ 2}\right)' &= \begin{pmatrix} (y_1')^2 \\ (y_2')^2 \end{pmatrix} = \begin{pmatrix} \text{diag}\,(\rho')_2 \\ \text{diag}\,(\rho')_3 \end{pmatrix} \\ &= \frac{1}{4}\begin{pmatrix} \lambda\left(-x_1\sin\theta + x_2\cos\theta\right)^2 - 3\lambda\left(x_1\cos\theta + x_2\sin\theta\right)^2 + 4\left(x_1\cos\theta + x_2\sin\theta\right)^2 \\ 4\left(-x_1\sin\theta + x_2\cos\theta\right)^2 - 3\lambda\left(-x_1\sin\theta + x_2\cos\theta\right)^2 + \lambda\left(x_1\cos\theta + x_2\sin\theta\right)^2 \end{pmatrix}. \end{aligned}$$

Determining the sign of each entry of $\mathbf{y}$ and $\mathbf{y}'$ requires an additional tomography step, as shown in Fig. 1, which could have noise on itself. For simplicity, we only compute L^2 relative error of $|\mathbf{y}|$,

$$\begin{aligned} \frac{\||\mathbf{y}| - |\mathbf{y}'|\|_2}{\||\mathbf{y}|\|_2} &= \||\mathbf{y}| - |\mathbf{y}'|\|_2 = \sqrt{(|y_1| - |y_1'|)^2 + (|y_2| - |y_2'|)^2} \\ &= \frac{1}{2}\sqrt{\begin{array}{l}\left(\sqrt{-3\lambda\left(x_1\sin\theta - x_2\cos\theta\right)^2 + \lambda\left(x_1\cos\theta + x_2\sin\theta\right)^2 + 4\left(x_1\sin\theta - x_2\cos\theta\right)^2} - 2\left|x_1\sin\theta - x_2\cos\theta\right|\right)^2 \\ + \left(\sqrt{\lambda\left(x_1\sin\theta - x_2\cos\theta\right)^2 - 3\lambda\left(x_1\cos\theta + x_2\sin\theta\right)^2 + 4\left(x_1\cos\theta + x_2\sin\theta\right)^2} - 2\left|x_1\cos\theta + x_2\sin\theta\right|\right)^2 \end{array}}, \end{aligned} \tag{8}$$

as $|y|$ is a normalized vector. The reverse triangle inequality guarantees the L^2 relative error of $\mathbf{y}$ is always lower-bounded by that of $|\mathbf{y}|$

$$\frac{\||\mathbf{y}| - |\mathbf{y}'|\|_2}{\||\mathbf{y}|\|_2} = \sqrt{(|y_1| - |y_1'|)^2 + (|y_2| - |y_2'|)^2} \le \sqrt{(y_1 - y_1')^2 + (y_2 - y_2')^2} = \|\mathbf{y} - \mathbf{y}'\|_2 = \frac{\|\mathbf{y} - \mathbf{y}'\|_2}{\|\mathbf{y}\|_2}.$$

Also, Eq. (8) is a periodic function with respect to θ in a period of $\pi/2$. The value of λ controls the amplitude of the function. We numerically illustrate this expression in Fig. 5 with $x_1 = x_2 = 1/\sqrt{2}$ for $\lambda = 0.1$ and $\lambda = 0.05$. To show that our computation is consistent with the noise model in Qiskit Aer, we also provide the estimations from the samples in the Qiskit Aer simulator with simulated depolarizing noise models. Each data point in the simulated case in Fig. 5 is the average of 100,000 samples from the simulator. Note that $\mathbf{y}$ and $\mathbf{y}'$ are non-negative for all λ and θ in Fig. 5, so the plot is for the L^2 relative error of $\mathbf{y}$ instead of just $|\mathbf{y}|$.

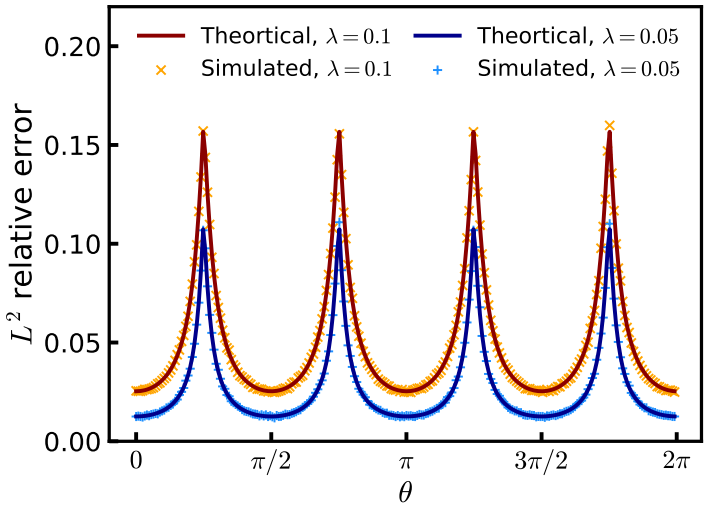

Figure 5: **Errors of the output vector, y, due to the 2-qubit depolarizing noise on a single RBS gate as a function of the angle of the RBS gate, θ.** The initial state in the circuit is $\left[0,\, 1/\sqrt{2},\, 1/\sqrt{2},\, 0\right]^T$. Each simulated data point is averaged from 100,000 samples in the Qiskit Aer simulator with a simulated depolarizing noise model.

4.2 Finite-sampling noise in tomography

In the tomography step, the probabilities $\Pr[0, \mathbf{e}_j]$ and $\Pr[1, \mathbf{e}_j]$, for $j \in \{1, ..., r\}$, are estimated from the frequencies of measurement outcomes, where n is the input vector dimension, m is the output vector dimension, and $r = \max(m, n)$. This results in an additional error on the estimation of output vector $\mathbf{y}$ caused by the finite number of measurements (shots). Let $q^{(0,j)}$ be the probability of measuring $|0, \mathbf{e}_j\rangle$ in tomography layer and $\hat{q}^{(0,j)}$ is the estimated value from N_{shot} shots. We want to estimate the size of finite-sampling error by first calculating the standard deviation of $\hat{q}^{(0,j)}$.

Let $Z_k^{(0,j)}$ be a Bernoulli random variable

$$Z_k^{(0,j)} = \begin{cases} 1 & \text{, if measures } |0, \mathbf{e}_j\rangle \text{ in } k^{th} \text{ shot with probability } q^{(0,j)} \\ 0 & \text{, otherwise} \end{cases}$$

and $S^{(0,j)} = \sum_{k=1}^{N_{shot}} Z_k^{(0,j)}$ is a Binomial random variable. Thus, we have

$$\hat{q}^{(0,j)} = \frac{S^{(0,j)}}{N_{shot}}.$$

Since the variance of $S^{(0,j)}$ is $\mathrm{Var}\left[S^{(0,j)}\right] = N_{shot} q^{(0,j)} \left(1 - q^{(0,j)}\right)$, we have

$$\mathrm{Var}\left[\hat{q}^{(0,j)}\right] = \mathrm{Var}\left[\frac{S^{(0,j)}}{N_{shot}}\right] = \frac{q^{(0,j)}\left(1 - q^{(0,j)}\right)}{N_{shot}}. \tag{9}$$

Similarly, define $\hat{q}^{(1,j)}$ be the probability of measuring $|1, \mathbf{e}_j\rangle$ in tomography layer and $\hat{q}^{(1,j)}$ is its

estimation from N_{shot} shots. We can obtain the variance of $\hat{q}^{(1,j)}$ similar to Eq. (9). It is clear that

$$\begin{aligned}
\text{Cov}\left[\hat{q}^{(0,j)}, \hat{q}^{(1,j)}\right] &= \frac{1}{N_{shot}^2} \sum_{k=1}^{N_{shot}} \sum_{l=1}^{N_{shot}} \text{E}\left[Z_k^{(0,j)} Z_l^{(1,j)}\right] - \text{E}\left[\frac{S^{(0,j)}}{N_{shot}}\right] \text{E}\left[\frac{S^{(1,j)}}{N_{shot}}\right] \\
&= \frac{1}{N_{shot}^2} N_{shot}^2 q^{(0,j)} q^{(1,j)} - q^{(0,j)} q^{(1,j)} \\
&= 0.
\end{aligned}$$

With Eq. (1), the standard deviation of the estimated y_j is

$$\begin{aligned}
\text{Std}\left[y_i\right] &= \text{Std}\left[\sqrt{r}(\text{Pr}[0, \mathbf{e}_i] - \text{Pr}[1, \mathbf{e}_j])\right] \\
&= \sqrt{r}\sqrt{\text{Var}\left[\hat{q}^{(0,j)}\right] + \text{Var}\left[\hat{q}^{(1,j)}\right] - 2\text{Cov}\left[\hat{q}^{(0,j)}, \hat{q}^{(1,j)}\right]} \\
&= \frac{\sqrt{r}}{\sqrt{N_{shot}}}\sqrt{q^{(0,j)}\left(1 - q^{(0,j)}\right) + q^{(1,j)}\left(1 - q^{(1,j)}\right)} \\
&\propto \frac{\sqrt{r}}{\sqrt{N_{shot}}},
\end{aligned} \tag{10}$$

where $q^{(0,j)}, q^{(1,j)} \in [0, 1]$. In conclusion, the finite-sampling error on the estimation of output vector $\mathbf{y} \in \mathbb{R}^r$ is proportional to $N_{shot}^{-0.5}$ when r is relatively small compare to N_{shot}.

4.3 Noisy simulation results

In our subsequent research, we adapt two types of noise models to assess the accuracy of quantum layers under noisy conditions. The first approach, named as simplified noise model, incorporate only 1-qubit and 2-qubit depolarizing noise channels on all basis gates. The goal of using this model is to examine the noise resilience of QOrthoNN circuits and the effects of our error mitigation technique on this extensively researched noise channel. In the experiments, we select several different values for 1-qubit noise parameter λ, as in Eq. (4), and set the 2-qubit noise parameter $\lambda' := 0.8\lambda$. This is to guarantee both noise channels have the same error rate. Recall Eq. (4), 1-qubit depolarizing noise channel has the expression

$$\mathcal{E}_{dep}(\rho) = (1 - \lambda)\rho + \lambda\frac{I}{2} = \left(1 - \frac{3}{4}\lambda\right)\rho + \frac{\lambda}{4}\left(X\rho X + Y\rho Y + Z\rho Z\right).$$

So the error-free probability is $1 - \frac{3}{4}\lambda$. We can also see the error-free probability for a 2-qubit depolarizing noise channel is $1 - \frac{15}{16}\lambda'$ in Eq. (5). Thus, setting $\lambda' = 0.8\lambda$ makes two probabilities equal. In our experiments, we choose the values of λ from 0 to 2×10^{-3} since the gate error rates on real IBMQ quantum computers are in the similar scale, as shown in Table 5.

Table 5: **1-qubit basis gate error rates among all qubits on selected IBMQ quantum computers (data collected on May 21, 2024 [60]).**

	ibm_osaka	ibm_brisbane	ibm_sherbrooke	ibm_torino
Average	1.37×10^{-3}	6.29×10^{-4}	2.07×10^{-4}	1.53×10^{-3}
Median	2.68×10^{-4}	2.38×10^{-4}	5.08×10^{-4}	3.52×10^{-4}

While the first approach aims to an direct and intuitive evaluation on the accuracy of QOrthoNN under a noisy environment, depolarizing noise is insufficient to fully reflect the noise in real quantum

computers and the 2-qubit gates usually have less fidelity than 1-qubit gates [61, 62, 63]. To fill this gap, we also carry experiments with the second approach: the backend-noise model from Qiskit Aer [54, 64]. The backend-noise model is in composite of

- measurement noise: emulated by classical 1-qubit bit-flip error in the measurement;
- gate noise: emulated by the combination of 1-qubit depolarizing error and thermal relaxation error, while the 2-qubit error operator is the tensor product of 1-qubit error operators.

The parameters of backend-noise model comes from the regular benchmarking tests performed by the device vendor. By comparing these models, we can identify the feasibility of our quantum neural network and provide benchmarks for the improvement of near-term quantum computers.

We use the same error mitigation method in [46], where only unary measurement outcomes are kept and all the other non-unary outcomes are discarded. This technique is a benefit of unary encoding. The effect of this error mitigation method will be shown in Section 4.3.2.

4.3.1 Function approximation

To demonstrate the impact of quantum noise, we first choose the most simple example of function approximation $f(x) = 1/(1 + 25x^2), x \in [-1, 1]$ mentioned in Section 3.1 function 1. All of the following results are calculated in a Qiskit simulator.

We investigate the impact of finite-sampling error by varying the number of shots, N_{shot}, i.e., how many times we do the measurement to reconstruct the quantum state. The error with respect to true function value decrease when we increase number of shots (Fig. 6A). In this example, when number of shots reaches 10^8, shots-based simulation result is close to ideal simulation result. We further analysed the error between shots-based and ideal simulation, which is exactly the finite-sampling error (Fig. 6B). The finite-sampling error is proportional to $N_{shot}^{-0.5}$, which fits perfectly with Eq. (10).

We further included depolarizing error to estimate the affect of quantum gate noise. Recalling Eq. (4), by adjusting the value of λ, we can determine the necessary capabilities that future quantum computers must achieve to maintain a reasonable error margin. When λ is within $[0, 2 \times 10^{-4}]$, the error increases almost linearly with λ (Fig. 6C). When we expand the range to $[0, 2 \times 10^{-3}]$ (Fig. 6), the error increases non-linearly and reaches a plateau at approximately $\lambda = 10^{-3}$.

In order to simulate the performance of our quantum neural network on real quantum computer, we adapt the backend noise model in Qiskit. Here, we choose IBM_brisbane backend, loading the corresponding noise parameters for simulation. The error turns out to be 14.4%, suggesting some more sophisticated error mitigation methods are needed. Since the scale of error is already too large in the simplest QOrthoNN experiment, the backend-noise model will not be tested in further experiments.

4.3.2 Antiderivative operator

The impact of quantum noise on quantum DeepONet is also investigated using the antiderivative operator example (Section. 3.2) when $l = 1.0$ (Fig. 7). Specifically, the finite-sampling error follows a zero-mean distribution (Fig. 7A), and the scale of which is proportional to $N_{shot}^{-0.5}$, as expected in Eq. (10). When depolarizing quantum noise is considered, the error mitigation method discussed at the beginning of Section. 4 can be applied. Although error mitigation helps eliminate undesired results caused by quantum noise, it also reduces the number of shots that are ultimately usable. It's obvious that

$$\text{useful shots} \approx C \times \text{total shots},$$

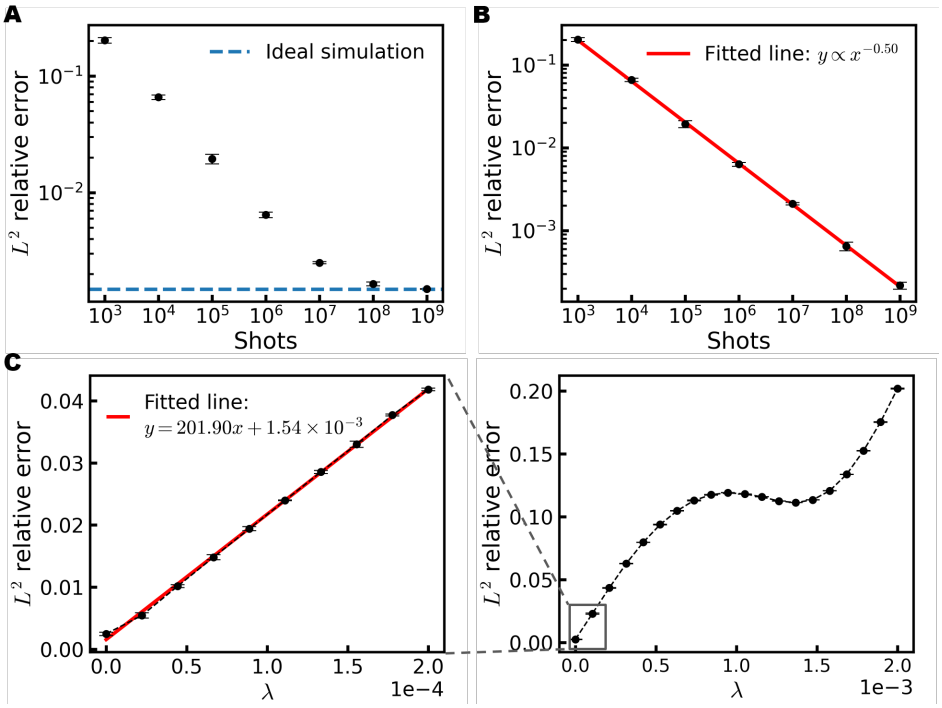

Figure 6: **Effect of quantum noise on function approximation example of $f(x) = 1/(1 + 25x^2)$.** (**A** and **B**) Finite-sampling noise at different number of shots. (A) L^2 error between shots-based results and true function with different shots, compared with ideal simulation. (B) L^2 error between shots-based and ideal simulation. (**C**) Depolarizing noise model for different depolarizing parameter. In both cases, number of shots is set to be 10^7.

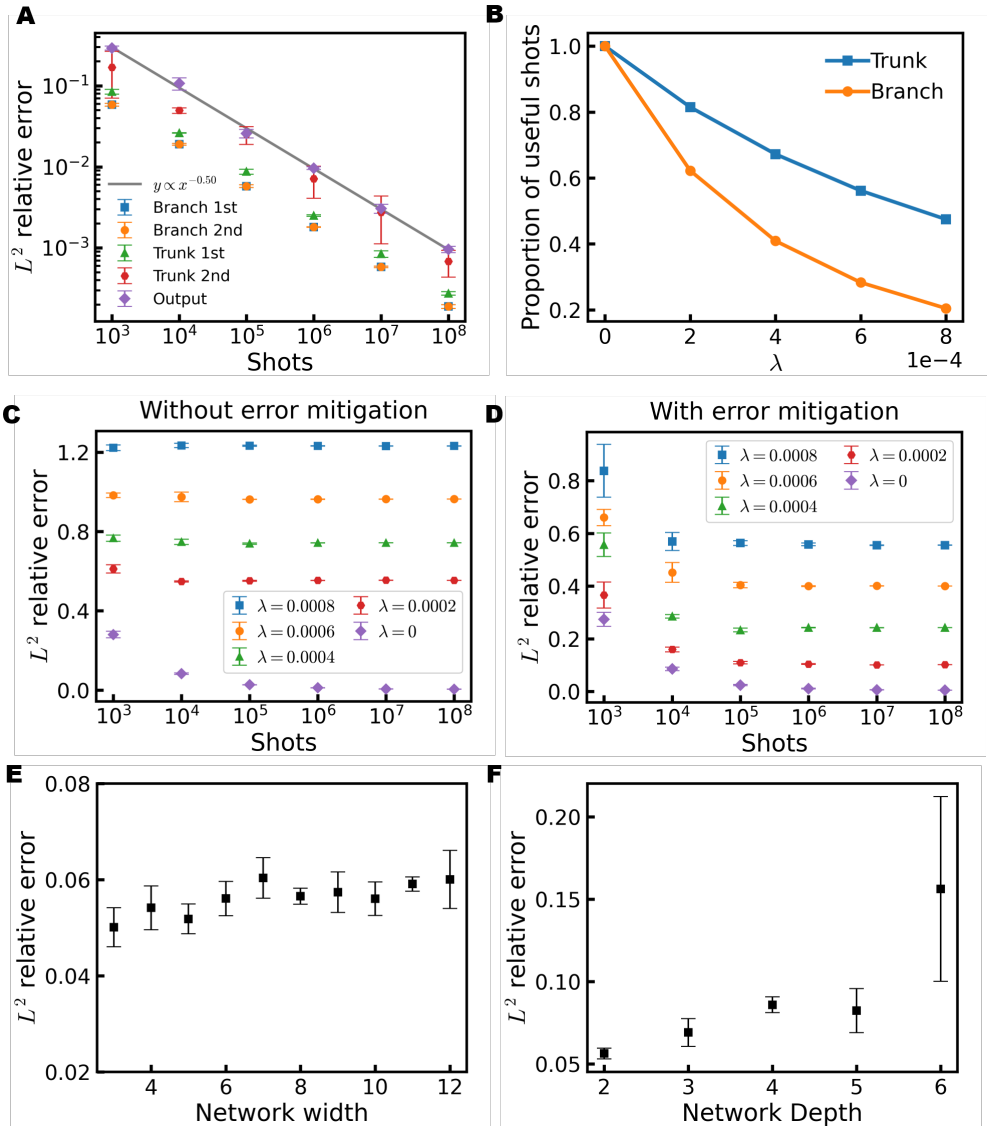

Figure 7: **Effect of noise on quantum DeepONet for the example of antidetivative operator.** (**A**) L^2 relative error between finite shots and infinite shots results for different layers. (**B**) Proportion of useful shots in total shots at different depolarizing noise level λ when implementing error mitigation. (**C** and **D**) L^2 error between simulation results at different depolarizing level λ and true solution. (C) Error mitigation is used. (D) Error mitigation is disabled. (**E** and **F**) The error for different neural network size. We set $\lambda = 10^{-4}$ for all gates and fixed the number of shots at 10^7. For each neural network sizes, we performed classical training 5 times until the test error is reduced to 3%. Each training run is quantumly simulated 3 times. The average and uncertainty of these noisy simulation results were then calculated. (E) The network depth of both the branch and trunk nets is fixed at 5, while the width of both is varied simultaneously. (F) The network width of both the branch and trunk nets is fixed at 10, while the depths of both networks are varied simultaneously.

with $C = 1.0$ when $\lambda = 0$. The parameter C decreases as λ increases (Fig. 7B) because higher levels of noise produce more unreasonable results.

Our problem-specific error mitigation scheme significantly reduces the error in noise cases with both finite-sampling and depolarizing noise, comparing Fig. 7C with D. In Fig. 7C, where we do not use error mitigation and directly accept all of the unary and non-unary results, the error barely reduce as number of shots increases. This is due to the fact that, by contracting the scales of errors in Figs. 7A and 7D, the finite-sampling error is relatively insignificant under the influence of depolarizing error. Therefore, increasing shots, which only reduce the finite-sampling error, does not work well. With error mitigation, due to the lowest overall error level, the decreasing of finite-sampling is more obvious in the plots (Fig. 7D).

We also investigated how the network size can affect the error of noisy model (Fig. 7E and F). For each neural network size, we performed classical training 5 times. The networks were trained uniformly until the test error was reduced to 3%. For each training run we quantum simulated 3 times. The parameters of simulations included 10^7 shots and $\lambda = 10^{-4}$ for depolarizing noise. It is important to note that even though the test error remained the same across classical training runs, the noisy simulation results varied. We believe this variation arises because the network converge to different parameter values in each training run, leading to different levels of error due to depolarizing noise. This observation aligns with our discussion in Section 4.1 about how the parameters of RBS gates influence the magnitude of errors. By comparing the two plots, we conclude that the error increases almost exponentially with increasing network depth. In contrast, when only the width is increased, the error shows minimal growth within our experiment range. Therefore, quantum DeepONet shows resilience to noise with respect to network width . Therefore, in practice, to minimize quantum noise, it is advisable to opt for wider rather than deeper neural networks.

5 Conclusions

We proposed Quantum DeepONet, which can be both data driven and physics-informed. Experimental results was conducted to confirm that Quantum DeepONet perform efficiently in solving different PDEs. We further considered the impact of quantum and finite-sampling noise in simulation, and benchmarked the noise level and corresponding accuracy.

There are a few limitations in our current implementation. Based on the unary encoding, the Quantum DeepONet currently could not handle large network width due to the limitation on the number of qubits and connectivities in the existing quantum devices, and the in-effectiveness of simulation on classical computers.

However, although such demand on the number of qubits can be greatly reduced by giving up the unary encoding, the rising cost of data loading and data tomography resulting from this change will require further analysis. On the other hand, in the noise simulation, both tested noise models do not include coherent noise and non-local noise such as cross-talk. The effects of a more complicated and realistic noise model is needed to examine the noise resilience of our design.

Additionally, our future work will explore extending Quantum DeepONet to accommodate more complex architectures [65, 66], which will allow us to address a broader range of applications and increase the model's utility.

Acknowledgments

This work was supported by the U.S. Department of Energy Office of Advanced Scientific Computing Research under Grant No. DE-SC0022953 and the U.S. National Science Foundation under Grant No. DMS-2347833.

A Model of quantum computing

Our work utilizes the quantum circuit model. It is an analogy to the classical circuit where a series of gates are conducted to perform computation. For a basic quantum circuit, there are three components: an initial quantum state, a series of quantum gates, and measurements. The initial state stores the initial information, which is then changed by the sequence of quantum gates. After the computation, the state is measured to get classical bits as the final outputs. The unit of quantum information is a qubit, analogizing to a bit in classical information.

In most of our work, we use statevector representation for quantum states. That is, an n-qubit quantum state is a vector in $\mathbb{C}^{2^n}$. Such a quantum state is often written as a linear combination of basis states. For example, a general 1-qubit state $|\psi\rangle$ is

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

where the notation $|\cdot\rangle$ represents a statevector, basis state $|0\rangle$ is $[1\ 0]^T$, basis state $|1\rangle$ is $[0\ 1]^T$, and $|\alpha|^2 + |\beta|^2 = 1$ for complex numbers α and β. So, when we measure the state $|\psi\rangle$, there is $|\alpha|^2$ chance to obtain a classical bit 0 and $|\beta|^2$ chance to obtain a classical bit 1. If neither α nor β is 0, then the quantum state is in the superposition of state $|0\rangle$ and $|1\rangle$. Similarly, a 2-qubit state is a linear combination of 2-qubit bases and the squared norms of coefficients sum to 1. The 2-qubit basis states are $|00\rangle = |0\rangle \otimes |0\rangle$, $|01\rangle = |0\rangle \otimes |1\rangle$, $|10\rangle = |1\rangle \otimes |0\rangle$, and $|11\rangle = |1\rangle \otimes |1\rangle$, where the operator $\otimes$ is the Kronecker product. If a 2-qubit state cannot be factored into the tensor product of two 1-qubit states, then this 2-qubit state is entangled.

An n-qubit quantum logic gate is a 2^n-by-2^n unitary matrix. Several common 1-qubit gates are

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix},$$
$$R_x(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, R_y(\theta) = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}, R_z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix},$$

and two widely used 2-qubit controlled gates are

$$CNOT = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X \text{ and } CZ = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes Z,$$

where I is the 2-by-2 identity matrix.

If an 1-qubit gates U apply on the first qubit of a 2-qubit state $|\chi\rangle$ and another 1-qubit gate V apply on the second qubit simultaneously, the resultant computation is $(U \otimes V)\,|\chi\rangle$. Based on the gate definitions introduced above, an implementation of $U_{RBS}(\theta)$ according to [46] is shown in Fig. 8. It can be verified that

$$U_{RBS}(\theta) = [H \otimes H]CZ[R_y(\theta) \otimes R_y(-\theta)]CZ[H \otimes H].$$

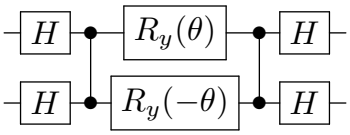

Figure 8: An implementation of $U_{RBS}(\theta)$ according to Ref. [46], where the symbol of two connected dots between H and R_y gates is the CZ gate.

B Quantum states in the density-matrix representation

A density matrix represents a quantum state in quantum information, providing a more general description than the statevector. In quantum computing, density matrices often come when the discussion includes quantum noise because quantum noise, such as the depolarizing channel in Section 4.1, can result in non-unitary evolution. The resultant quantum system may have p_k probability in the state $|\psi_k\rangle$ for multiple different indices k, making a single statevector insufficient to depict it. To express this system in a density matrix ρ, we have

$$\rho = \sum_k p_k \left|\psi_k\right\rangle \left\langle\psi_k\right| .$$

where $\sum_k p_k = 1$. Thus, the density matrix ρ is trace-one, Hermitian, and positive semidefinite [59]. The state evolution governed by the unitary operator U is computed by

$$\rho \xrightarrow{U} U\rho U^\dagger .$$

The non-unitary evolution can be described similarly to the depolarizing noise channel Section 4.1.

References

[1] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[2] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490, 2016.

[3] Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[4] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[5] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64:525–545, 2019.

[6] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

[7] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[8] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228, 2021.

[9] Yuyao Chen, Lu Lu, George Em Karniadakis, and Luca Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics express*, 28(8):11618–11633, 2020.

[10] Mitchell Daneker, Shengze Cai, Ying Qian, Eric Myzelev, Arsh Kumbhat, He Li, and Lu Lu. Transfer learning on physics-informed neural networks for tracking the hemodynamics in the evolving false lumen of dissected aorta. *Nexus*, 1(2), 2024.

[11] Mitchell Daneker, Zhen Zhang, George Em Karniadakis, and Lu Lu. Systems biology: Identifiability analysis and parameter identification via systems-biology-informed neural networks. In *Computational Modeling of Signaling Networks*, pages 87–105. Springer, 2023.

[12] Benjamin Fan, Edward Qiao, Anran Jiao, Zhouzhou Gu, Wenhao Li, and Lu Lu. Deep learning for solving and estimating dynamic macro-finance models. *arXiv preprint arXiv:2305.09783*, 2023.

[13] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

[14] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021.

[15] Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[16] Zijie Li, Kazem Meidani, and Amir Barati Farimani. Transformer for partial differential equations' operator learning. *arXiv preprint arXiv:2205.13671*, 2022.

[17] Zhongkai Hao, Zhengyi Wang, Hang Su, Chengyang Ying, Yinpeng Dong, Songming Liu, Ze Cheng, Jian Song, and Jun Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023.

[18] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[19] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40):eabi8605, 2021.

[20] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *ACM/JMS Journal of Data Science*, 1(3):1–27, 2024.

[21] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *Journal of Machine Learning Research*, 24(388):1–26, 2023.

[22] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE transactions on neural networks*, 6(4):911–917, 1995.

[23] Lizuo Liu and Wei Cai. Multiscale deeponet for nonlinear operators in oscillatory function spaces for building seismic wave responses. *arXiv preprint arXiv:2111.04860*, 2021.

[24] Pengzhan Jin, Shuai Meng, and Lu Lu. Mionet: Learning multiple-input operators via tensor product. *SIAM Journal on Scientific Computing*, 44(6):A3490–A3514, 2022.

[25] Min Zhu, Shihang Feng, Youzuo Lin, and Lu Lu. Fourier-deeponet: Fourier-enhanced deep operator networks for full waveform inversion with improved accuracy, generalizability, and robustness. *Computer Methods in Applied Mechanics and Engineering*, 416:116300, 2023.

[26] Zhongyi Jiang, Min Zhu, Dongzhuo Li, Qiuzi Li, Yanhua O Yuan, and Lu Lu. Fourier-mionet: Fourier-enhanced multiple-input neural operators for multiphase modeling of geological carbon sequestration. *arXiv preprint arXiv:2303.04778*, 2023.

[27] Shunyuan Mao, Ruobing Dong, Lu Lu, Kwang Moo Yi, Sifan Wang, and Paris Perdikaris. Ppdonet: Deep operator networks for fast prediction of steady-state solutions in disk–planet systems. *The Astrophysical Journal Letters*, 950(2):L12, 2023.

[28] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[29] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[30] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, 2021.

[31] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018.

[32] Ashish Kapoor, Nathan Wiebe, and Krysta Svore. Quantum perceptron models. *Advances in neural information processing systems*, 29, 2016.

[33] Vedran Dunjko, Jacob M Taylor, and Hans J Briegel. Quantum-enhanced machine learning. *Physical review letters*, 117(13):130501, 2016.

[34] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009.

[35] Junyu Liu, Minzhao Liu, Jin-Peng Liu, Ziyu Ye, Yunfei Wang, Yuri Alexeev, Jens Eisert, and Liang Jiang. Towards provably efficient quantum algorithms for large-scale machine-learning models. *Nature Communications*, 15(1):434, 2024.

[36] Nathan Wiebe, Daniel Braun, and Seth Lloyd. Quantum algorithm for data fitting. *Physical review letters*, 109(5):050505, 2012.

[37] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Physical Review X*, 8(2):021050, 2018.

[38] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature physics*, 10(9):631–633, 2014.

[39] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[40] Nishant Jain, Jonas Landman, Natansh Mathur, and Iordanis Kerenidis. Quantum fourier networks for solving parametric pdes. *Quantum Science and Technology*, 2023.

[41] Samson Wang, Enrico Fontana, Marco Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J Coles. Noise-induced barren plateaus in variational quantum algorithms. *Nature communications*, 12(1):6961, 2021.

[42] Enrico Fontana, M Cerezo, Andrew Arrasmith, Ivan Rungger, and Patrick J Coles. Non-trivial symmetries in quantum landscapes and their resilience to quantum noise. *Quantum*, 6:804, 2022.

[43] Supanut Thanasilp, Samson Wang, Nhat Anh Nghiem, Patrick Coles, and Marco Cerezo. Subtleties in the trainability of quantum machine learning models. *Quantum Machine Intelligence*, 5(1):21, 2023.

[44] Marco Schumann, Frank K Wilhelm, and Alessandro Ciani. Emergence of noise-induced barren plateaus in arbitrary layered noise models. *arXiv preprint arXiv:2310.08405*, 2023.

[45] Martin Larocca, Supanut Thanasilp, Samson Wang, Kunal Sharma, Jacob Biamonte, Patrick J Coles, Lukasz Cincio, Jarrod R McClean, Zoë Holmes, and M Cerezo. A review of barren plateaus in variational quantum computing. *arXiv preprint arXiv:2405.00781*, 2024.

[46] Jonas Landman, Natansh Mathur, Yun Yvonna Li, Martin Strahm, Skander Kazdaghli, Anupam Prakash, and Iordanis Kerenidis. Quantum Methods for Neural Networks and Application to Medical Image Classification. *Quantum*, 6:881, December 2022.

[47] Scott Aaronson. Read the fine print. *Nature Physics*, 11(4):291–293, 2015.

[48] Hsin-Yuan Huang, Richard Kueng, and John Preskill. Predicting many properties of a quantum system from very few measurements. *Nature Physics*, 16(10):1050–1057, 2020.

[49] Joran van Apeldoorn, Arjan Cornelissen, András Gilyén, and Giacomo Nannicini. Quantum tomography using state-preparation unitaries. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1265–1318. SIAM, 2023.

[50] Nitin Bansal, Xiaohan Chen, and Zhangyang Wang. Can we gain more from orthogonality regularizations in training deep cnns? In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4266–4276, Red Hook, NY, USA, 2018. Curran Associates Inc.

[51] Shuai Li, Kui Jia, Yuxin Wen, Tongliang Liu, and Dacheng Tao. Orthogonal deep neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 43(4):1352–1368, 2019.

[52] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

[53] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M. Stuart. Model Reduction And Neural Networks For Parametric PDEs. *The SMAI Journal of computational mathematics*, 7:121–157, 2021.

[54] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.

[55] Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

[56] Kuangdai Leng, Mallikarjun Shankar, and Jeyan Thiyagalingam. Zero coordinate shift: Whetted automatic differentiation for physics-informed operator learning. *Journal of Computational Physics*, page 112904, 2024.

[57] Daniel Stilck França and Raul Garcia-Patron. Limitations of optimization algorithms on noisy quantum devices. *Nature Physics*, 17(11):1221–1227, 2021.

[58] Diego García-Martín, Martin Larocca, and Marco Cerezo. Effects of noise on the overparametrization of quantum neural networks. *Physical Review Research*, 6(1):013295, 2024.

[59] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[60] IBM. IBM Quantum Platform. https://quantum.ibm.com/, May 2024.

[61] Kenneth Wright, Kristin M Beck, Sea Debnath, JM Amini, Y Nam, N Grzesiak, J-S Chen, NC Pisenti, M Chmielewski, C Collins, et al. Benchmarking an 11-qubit quantum computer. *Nature communications*, 10(1):5464, 2019.

[62] Konstantinos Georgopoulos, Clive Emary, and Paolo Zuliani. Modeling and simulating the noisy behavior of near-term quantum computers. *Physical Review A*, 104(6):062432, 2021.

[63] Mirko Amico, Helena Zhang, Petar Jurcevic, Lev S Bishop, Paul Nation, Andrew Wack, and David C McKay. Defining best practices for quantum benchmarks. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 1, pages 692–702. IEEE, 2023.

[64] Samudra Dasgupta and Travis Humble. Impact of unreliable devices on stability of quantum computations. *ACM Transactions on Quantum Computing*, 2024.

[65] Shengze Cai, Zhicheng Wang, Lu Lu, Tamer A Zaki, and George Em Karniadakis. Deepm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *Journal of Computational Physics*, 436:110296, 2021.

[66] Lu Lu, Raphaël Pestourie, Steven G Johnson, and Giuseppe Romano. Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. *Physical Review Research*, 4(2):023210, 2022.