Solving Advection-Reaction-Diffusion Equations with Quantum Trainable Embedding Physics-Informed Neural Networks

Ban Q. Tran[1,2(✉)], Hai T. D. Pham[3], and Susan Mengel[1]

[1] Department of Computer Science, Texas Tech University, Lubbock, TX 79416, USA
susan.mengel@ttu.edu
[2] Department of Computing Fundamentals, FPT University, Hanoi 10000, Vietnam
bantran@ttu.edu, bantq3@fe.edu.vn
[3] SAP Innovation Lab, FPT University, Hanoi 10000, Vietnam

Abstract. Multi-layer neural networks are the cornerstone of deep learning architectures and are applied across numerous critical fields. One area currently focusing on leveraging the strengths of neural networks is solving partial differential equations for physical simulations. Furthermore, the emergence of quantum computing technology has provided powerful support to classical neural networks, enhancing the quality of these models. In this research, we introduce a PDE solver based on a hybrid quantum neural network having a quantum learnable embedding combined with a classical optimizer. We demonstrate the effectiveness of this approximate solver through three case studies: the Heat, Reaction-Diffusion, and Navier-Stokes equations. Our results show that the hybrid classical-quantum solver achieves a deeper and smoother reduction in the loss function, with no signs of overfitting when compared to classical solvers.

Keywords: Quantum Machine Learning · Quantum Neural Networks · Quantum Physics-Informed Neural Networks · Hybrid Quantum-Classical Computing

1 Introduction

Quantum computing [1], with its unique properties, such as superposition, entanglement, and interference, has enabled revolutionary algorithms like Grover's and Shor's, and has demonstrated clear superiority over classical algorithms designed for the same purposes. Meanwhile, neural network architectures, which underpin deep learning, are surpassing classical machine learning algorithms and are at the forefront of many current technological trends [2]. Currently, one of the most potent application areas for neural networks is the development of partial differential equation (PDE) solvers [3,4]. These solvers, physics-informed neural

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16758, pp. 365–376, 2027.
https://doi.org/10.1007/978-3-032-30497-1_22

networks (PINN) demonstrate significant effectiveness in solving nonlinear and highly stiff PDEs [5]. The intersection of quantum computing and classical neural networks has led to powerful new solvers known as quantum physics-informed neural networks (QPINNs) [6], which exploit the strengths of both to make solving PDEs more efficient. Although QPINN solvers, with VQC circuits as their core, have demonstrated relatively good performance compared to PINN, the issue of embedding remains one of the primary obstacles hindering quantum advantage [7]. To address this issue, research has shown that using a neural network to learn the parameters θ of the main variational quantum circuits (VQCs) prior to encoding can improve the performance of the QPINN model [8]. Based on this idea, we hypothesize that applying a quantum neural network (QNN) will also improve the learning capability of the VQC circuit, similar to what classical neural networks have achieved. In the following section of the paper, we will introduce the architecture and mathematical foundation of the Quantum Neural Networks - Quantum Physics-Informed Neural Networks (QNN-QPINN) solver. In the experimental section, we evaluate the performance of this solver using three case studies: Heat, Reaction-diffusion, and Navier-Stokes PDEs.

2 Methodology

2.1 Physics-Informed Neural Networks

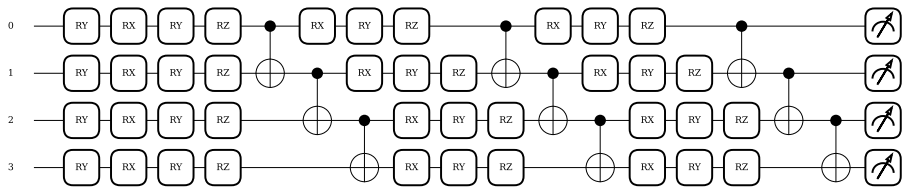

Fig. 1. The VQC quantum circuit.

The general architecture of a PINN [5] is similar to other classical neural network architectures where a PINN consists of a feedforward neural network (FNN) block with multiple layers, where each layer contains a set of neurons. This FNN takes spatial and temporal inputs (x, y, t) and predicts solution variables (u, v, p). Automatic differentiation is employed to compute derivatives, which are then used to calculate the loss function. The loss function of a PINN (Formula 1) is unique in that it incorporates losses from the PDE residuals, boundary conditions, and initial conditions. An optimizer is used to update the network parameters to minimize the loss and ensure that physical laws are preserved. The formula for the PINN loss function is as follows:

$$\mathcal{L}(\phi) = \mathcal{L}_{\text{PDE}}(\phi) + \sum_{j} \lambda_j \, \mathcal{L}_{\text{BC},j}(\phi) + \lambda_{\text{IC}} \, \mathcal{L}_{\text{IC}}(\phi), \tag{1}$$

The input data (Φ) for the model training is a computational domain, which is discretely sampled into collocation points. These collocation points are divided into different groups: collocation points for each boundary and those located within the domain Ω. Each collocation point has a residual value, and inside the domain, these values correspond to the PDE residuals. At the boundaries, the collocation points are fixed values corresponding to specific boundary conditions, such as Dirichlet or Neumann conditions.

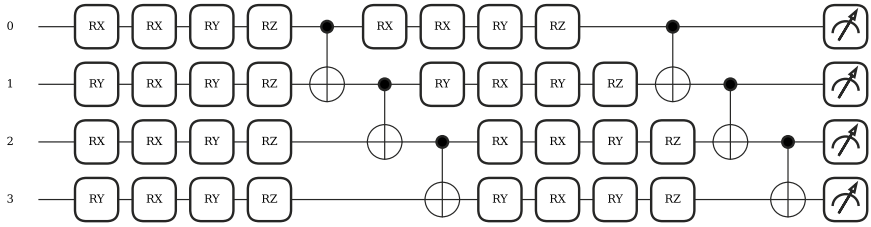

Fig. 2. The QNN embedding quantum circuit.

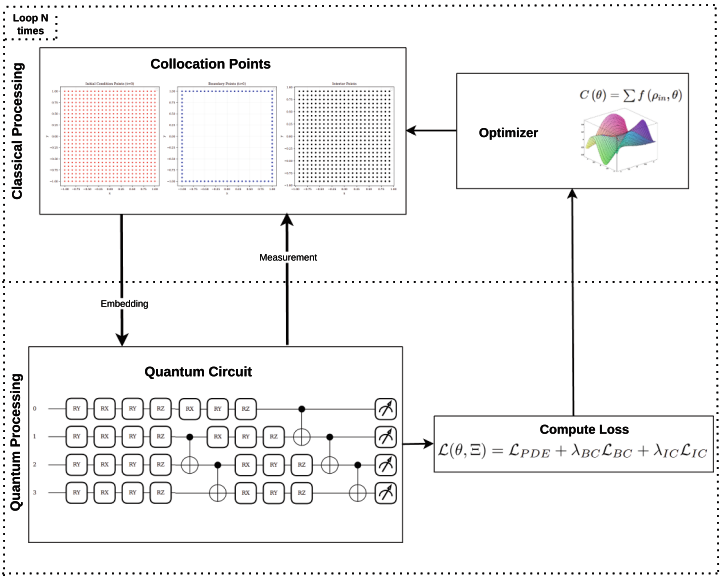

Fig. 3. The QNN-QPINN architecture.

2.2 Quantum Physics-Informed Neural Networks

Quantum physics-informed neural networks [6] are an extension of the traditional PINN model into the quantum computing environment. Instead of using classical artificial neural networks, QPINN utilizes variational quantum circuits to approximate the solutions of partial differential equations. Like PINNs, QPINNs do not only learn from data but also adhere to physical laws by directly integrating mathematical equations. A QPINN model consists of three main components: embedding (also known as a feature map), a variational ansatz circuit, and a physics-based loss function that uses automatic differentiation for hybrid classical optimizers or the parameter-shift rule for pure quantum optimizers. Figure 1 illustrates the VQC circuit of the QPINN architecture used in our study.

The variational unitary is structured as a series of L layers, defined by

$$U_{\text{var}}(\theta_{\text{var}}) = U_L(\theta_L) \cdots U_2(\theta_2)\, U_1(\theta_1) \tag{2}$$

In this architecture, each individual layer $U_\ell(\theta_\ell)$ is composed of a combination of both parameterized and fixed unitary operations: $U_\ell(\theta_\ell) \prod_m \exp!\big(- i, \theta_{\ell,m}, H_m\big), W_m$. Within this expression, H_m represents a set of constant Hermitian generators, while W_m refers to unitary operators that do not contain tunable parameters.

2.3 Quantum Trainable Embedding Physics-Informed Neural Networks

A trainable embedding - quantum physics-informed neural network (TE-QPINN) [8] is a QPINN architecture for solving PDEs that utilizes the idea of improving embedding capabilities by applying a neural network to learn the parameters θ fed into the VQC circuit. Before the embedding process, the spatial and temporal coordinates transform into a bounded range via the following expressions:

$$\tilde{x} = \mathcal{N}(x; x_{\min}, x_{\max}), \qquad \tilde{t} = \mathcal{N}(t; t_{\min}, t_{\max}),$$

In this context, $\mathcal{N}(\cdot)$ represents an affine scaling function that maps the raw input domains $[x_{\min}, x_{\max}]$ and $[t_{\min}, t_{\max}]$ onto the normalized interval $[-1, 1]$. By implementing this coordinate rescaling, we address potential discrepancies in magnitude between spatial and temporal variables, thereby enhancing the numerical robustness of the training phase. A comprehensive feature map is constructed by the embedding function.

$$\Gamma(x,t) = \big[\, \psi_1(\tilde{x},\tilde{t}),\ \ldots,\ \psi_{N_q}(\tilde{x},\tilde{t}) \,\big]^T,$$

In this formulation, the rotation angle for every individual i-th qubit is defined by the corresponding component $\psi_i(\tilde{x},\tilde{t})$. The transformation performed by the quantum encoder is governed by the specific values of the embedding angles.

$$U_{\text{enc}}(x,t;\theta_{\text{emb}}) = \bigotimes_{i=1}^{N_q} R_y\big(\psi_i(\tilde{x},\tilde{t},\theta_{\text{emb}})\big) \tag{3}$$

where $R_y(\cdot) = \exp(-i(\cdot)\sigma_y/2)$ represents a rotation operation on the y-axis for the i-th qubit, and N_q signifies the total qubit count.

The variational quantum state corresponding to a specific input (x, t) is generated by acting upon the encoded input with a universal variational circuit.

$$|\phi_{\text{var}}(x, t; \Theta)\rangle = U_{\text{var}}(\theta_{\text{var}})U_{\text{enc}}(x, t; \theta_{\text{emb}})|0\rangle^{\otimes N_q}. \tag{4}$$

By combining Formulas 2 and 3, concentration field predictions are determined through the following expectation values of species-dependent observables.

$$\tilde{C}_i(x, t, \Theta) = \left\langle \phi_{\text{var}}(x, t, \theta)\middle|\mathcal{O}_i\middle|\phi_{\text{var}}(x, t, \theta)\right\rangle \tag{5}$$

where the Hermitian observables, $\mathcal{O}_i$, are defined in relation to their associated concentration fields.

The training process (Formula 3) for the encoding parameters θ in the VQC is carried out in parallel with the main VQC circuit. Our proposed architecture essentially utilizes this trainable embedding concept, but replaces U_{emb} with a QNN architecture as shown in Fig. 2. The overall architecture of the proposed QNN-QPINN is described in detail in Fig. 3. The QNN-QPINN algorithm is described in Algorithm 1.

3 Numerical Results

In this section, we focus on interpreting the results from the model training experiments and the inference outcomes of the quantum neural network solver across various PDEs of increasing complexity.

3.1 Environments and Quantum Circuits

Due to constraints on quantum hardware in the NISQ era, specifically the lack of affordable fault-tolerant quantum computers for model training, we used classical computers with simulators to train our models. The simulation environment employed is the REmotely-managed Power Aware Computing Systems and Services (REPACSS) supercomputer at Texas Tech University [9], which utilizes GPU compute nodes equipped with 4 x NVIDIA H100 NVLs (94 GB HBM per GPU). Training models using classical computers is entirely feasible for systems with fewer than 20 qubits. Regarding the software, we utilized the Python programming language and Facebook's PyTorch deep learning library to implement the classical neural network models and the hybrid model optimizer [10]. To construct the quantum neural network architecture, we employed the PennyLane quantum machine learning library for its convenience, robustness, and support for diverse hardware in the post-training inference phase [11].

The quantum circuits forming the solver are constructed by applying single-qubit rotation gates (R_x, R_y, R_z) and CNOT entanglement gates to the input qubits. This grid-like quantum neural network architecture is known as a Variational Quantum Circuit (VQC), where the tunable parameters are the rotation

Algorithm 1. Training Algorithm for the QNN-QPINN Framework

Input:
Internal PDE operator, boundary operators $B_j[\cdot]$ and boundary data b_j
Initial condition $C_0(x)$
Collocation sets $\mathcal{S}_{\text{PDE}}, \{\mathcal{S}_{\text{BC},j}\}, \mathcal{S}_{\text{IC}}$
Loss weights $\{\lambda_j\}, \lambda_{\text{IC}}$
Equation parameters $\mathcal{P}_i$
Iterations N
Initialization:
Initialize embedding parameters θ_{emb}
Initialize variational circuit parameters θ_{var}
Normalize the input coordinates (x, t) to the scaled variables $(\tilde{x}, \tilde{t}) \in [-1, 1]^2$
Initialize optimizer for parameters $(\theta_{\text{emb}}, \theta_{\text{var}})$
while not converged **do**
    Forward evaluation
    for each collocation point (x^k, t^k) **do**
        Compute embedding angles $\psi(x^k, t^k; \theta_{\text{emb}})$
        Prepare encoded state $|\phi_{\text{enc}}\rangle = U_{\text{enc}}(\psi)|0\rangle^{\otimes N_q}$
        Apply variational circuit $|\phi_{\text{var}}\rangle = U_{\text{var}}(\theta_{\text{var}})\,|\phi_{\text{enc}}\rangle$
        Evaluate predicted fields $\tilde{C}_i(x^k, t^k) = \langle\phi_{\text{var}}|\mathcal{O}_i|\phi_{\text{var}}\rangle$
        Compute required spatial and temporal derivatives
    end for
    Loss evaluation
    Compute $\mathcal{L}_{\text{PDE}}, \mathcal{L}_{\text{BC}}, \mathcal{L}_{\text{IC}}$
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{PDE}} + \sum_j \lambda_j \mathcal{L}_{\text{BC},j} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}}$
    Gradient computation
    Compute $\nabla_{\theta_{\text{emb}}}\mathcal{L}$
    Compute $\nabla_{\theta_{\text{var}}}\mathcal{L}$
    Parameter update
    Update parameters $(\theta_{\text{emb}}, \theta_{\text{var}})$ using the LBFGS optimizer
end while
Output:
Save trained parameters $(\theta^\star_{\text{emb}}, \theta^\star_{\text{var}})$ and the model.

angles of the single-qubit gates. Figure 1 illustrates the quantum circuit of the solver with an input of 4 qubits and a depth of 3 layers, while Fig. 2 depicts the quantum circuit of the quantum neural networks (QNN) used to train the parameters for the primary VQC circuit.

3.2 Case Study 1: Heat Equation

In this section, we discuss the experimental results of the quantum-assisted learning model applied to the one-dimensional Heat equation, one of the most significant parabolic PDEs in physics. The Heat equation is a partial differential equation that describes the variation in temperature within a given region over time. It stands as one of the most fundamental equations in mathematics, physics, and engineering. Physically, it encapsulates the principle that heat flows from

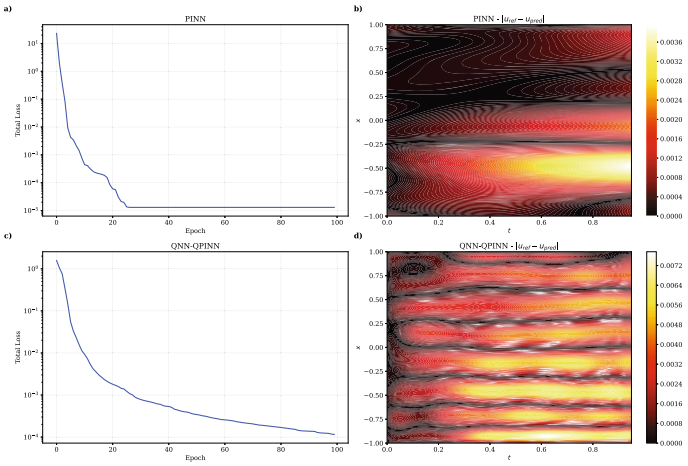

Fig. 4. Loss function and absolute error plots of PINN [5] and QNN-QPINN after training for 100 epochs. The quantum circuit uses 6 qubits and has a VQC depth of 10 layers. The embedding circuit of the QNN-QPINN uses 6 qubits and 5 layers. (a) Loss function of PINN (b), Absolute error between the PINN solution and the RK45 reference solution (c), Loss function of QNN-QPINN (d), Absolute error between the QNN-QPINN solution and the RK45 reference solution.

hotter to colder regions at a rate proportional to the temperature gradient. The equation consists of a derivative term representing the rate of change of temperature over time and a Laplace operator that describes the difference between the temperature at a specific point and its surrounding average. The thermal diffusivity coefficient, μ, depends on the material properties. In this case study, we have selected $\mu = 0.01/\pi$.

$$\frac{\partial u(t,x)}{\partial t} = \left(\frac{0.01}{\pi}\right) \frac{\partial^2 u(t,x)}{\partial x^2} \tag{6}$$

$$t \in [0, 0.95], \quad x \in [-1, 1], \quad u(0,x) = \sin(\pi x)$$

Figure 4 illustrates the loss function results and absolute error during the training process for both the PINN [5] and QNN-QPINN models over 100 epochs. The inset (a) in the upper-left corner for the PINN model shows that the model reaches convergence at approximately 22 epochs, after which it enters a saturation state for the remainder of the training process. The loss function curve exhibits a jagged pattern and a steep slope. In contrast, the loss function curve of the QNN-QPINN model demonstrates a steady and smooth decline throughout the 100 epochs and shows signs of continuing to decrease, implying that the model is still capable of further learning.

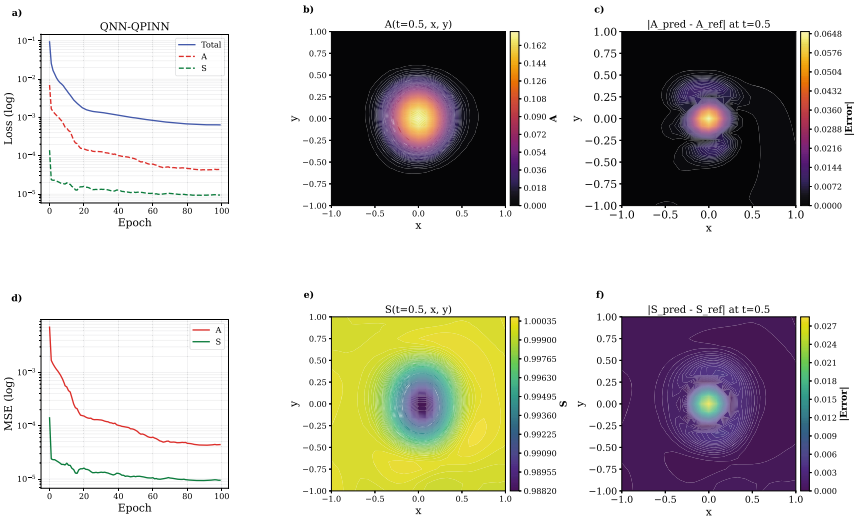

Fig. 5. Loss function plots, model predictions, and absolute error of the QNN-QPINN model compared to RK45. (a) Total loss and loss functions for activator A and substrate S. (d) MSE of activator A and substrate S. (b) and (e) Model predictions for A and S at time t = 0.5. (c) and (f) Absolute error for A and S at t = 0.5.

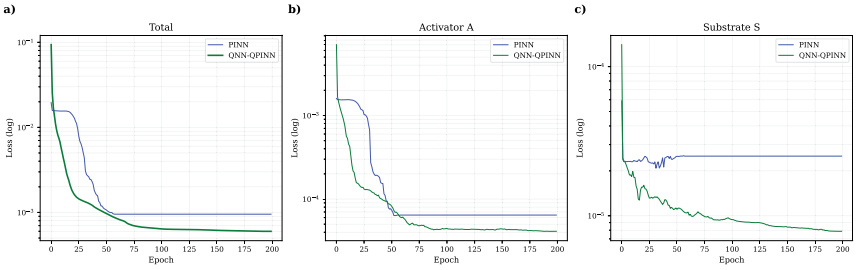

Fig. 6. Performance comparison plots during 200 training epochs between PINN [5] and QNN-QPINN for the two-dimensional reaction-diffusion equation, utilizing 4 qubits with a 10-layer VQC and a 5-layer QNN. (a) Total loss (b) Loss function for A (c) Loss function for S.

3.3 Case Study 2: Reaction-Diffusion Equation

The Reaction-Diffusion (RD) equation represents an evolutionary step beyond the Heat equation. It describes not only how a substance spreads through space but also how different substances interact and transform one another. In nature, this equation serves as a primary tool for explaining the formation of complex patterns, such as tiger stripes, leopard spots, or chemical spirals. These are often referred to as Turing patterns, a system of reacting and diffusing chemicals could spontaneously create such intricate structures. The two-dimensional Reaction-

Diffusion (RD) equation is expressed in its general form as follows:

$$\frac{\partial A}{\partial t}(t,x,y) = D_A \left(\frac{\partial^2 A}{\partial x^2}(t,x,y) + \frac{\partial^2 A}{\partial y^2}(t,x,y) \right) + k_1 A(t,x,y)^2 S(t,x,y) - k_2 A(t,x,y) \quad (7)$$

$$\frac{\partial S}{\partial t}(t,x,y) = D_S \left(\frac{\partial^2 S}{\partial x^2}(t,x,y) + \frac{\partial^2 S}{\partial y^2}(t,x,y) \right) - k_1 A(t,x,y)^2 S(t,x,y) + k_3$$

In this specific model, $A(t,x,y)$ represents the activator function, while $S(t,x,y)$ denotes the substrate function. The Laplace operator $\frac{\partial^2 A}{\partial x^2}(t,x,y) + \frac{\partial^2 A}{\partial y^2}(t,x,y)$ and $\frac{\partial^2 S}{\partial x^2}(t,x,y) + \frac{\partial^2 S}{\partial y^2}(t,x,y)$ represents the physical diffusion process. D_A and D_S are the diffusion coefficients for the activator A and substrate S, respectively. The three coefficients k_1, k_2, and k_3 are the reaction rate constants. In this case study, we have selected the parameters as $D_A = 10^{-5}$, $D_S = 2{\times}10^{-3}$, $k_1 = 1.0$, $k_2 = 1.0$, and $k_3 = 10^{-3}$.

Table 1. Performance comparison between PINN [5] and QNN-QPINN across different numbers of training epochs.

Number of epochs	PINN [5]	QNN-QPINN
100	9.53E−04	6.41E−04
150	9.53E−04	6.17E−04
200	3.82E−04	1.38E−05
250	1.56E−04	1.31E−05

Figure 5 and Fig. 6 illustrate the training results of the QNN-QPINN and provide a performance comparison against a standard PINN model featuring four layers with 32 neurons each [5]. The QNN-QPINN loss function exhibits more effective, smoother convergence across all three metrics: total loss, activator A loss, and substrate S loss. Furthermore, the predicted solutions for A and S at $t = 0.5$ in Fig. 5 show that the QNN-QPINN results are in excellent agreement with those obtained via the RK45 method. Table 1 presents the comparative performance of the PINN and QNN-QPINN models after being trained for 100, 150, 200, and 250 epochs, respectively. The results indicate that the QNN-QPINN consistently yields a lower loss function value compared to the PINN across all training durations.

3.4 Case Study 3: Navier-Stokes Equation

The Navier-Stokes equations are essentially Newton's second law applied to liquids and gases. It indicates that the change in a fluid particle's momentum is due to the forces acting on it, including pressure, viscosity, and external forces. This equation has applications in aircraft design, weather forecasting, and blood flow simulation in the body. This equation is a very complex nonlinear PDE that is

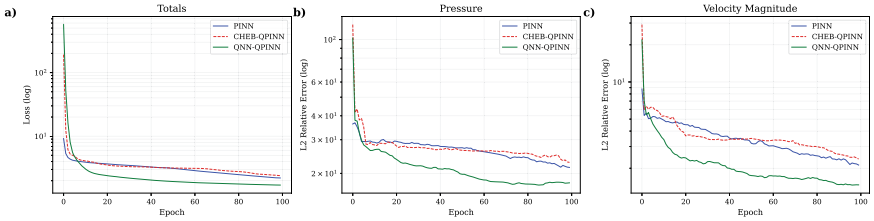

Fig. 7. Performance comparison plots of three models: PINN [5], Chebyshev-QPINN [6], and QNN-QPINN, when solving the Navier-Stokes equations. (a) Total loss (b) Pressure loss function (c) Velocity magnitude loss function.

difficult to solve with conventional solvers. The general form of the Navier-Stokes equation for incompressible fluids is often written as:

$$u\frac{\partial u}{\partial x} + \nu\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{8}$$

$$u\frac{\partial \nu}{\partial x} + \nu\frac{\partial \nu}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 \nu}{\partial x^2} + \frac{\partial^2 \nu}{\partial y^2}\right)$$

$$\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0$$

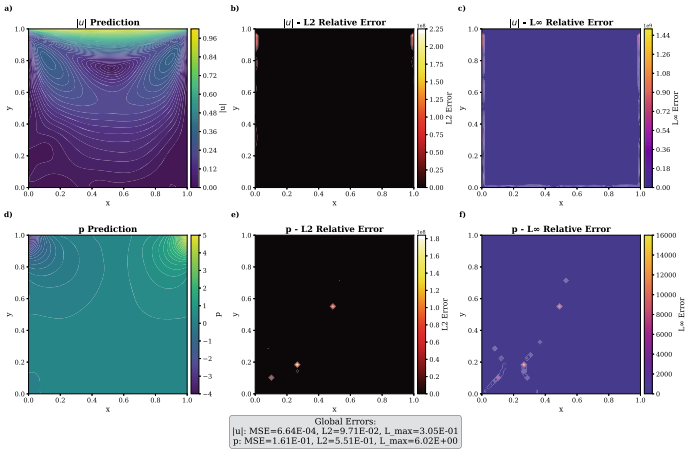

Fig. 8. Inference result plots of the QNN-QPINN model after training for 200 epochs. (a) and (d) Predictions for $|u|$ and p. (b) and (e) L2 relative error for $|u|$ and p. (c) and (f) L_∞ relative error for $|u|$ and p.

This formula is a set of three equations: two momentum equations for pressure and velocity, and one mass conservation law equation. The coefficient ν represents viscosity and ρ represents density. In this experiment, we choose $\nu = 0.1$

and $\rho = 1$, which yields a Reynolds number of 10; that is, the flow is stable and highly viscous, with fluid layers sliding smoothly over each other without vortices or disturbances. To test the effectiveness of learning the rotation angle parameter, we use an additional Chebyshev-QPINN model with a non-trained rotation angle parameter [6]. All three models are trained for 100 epochs. The results in Fig. 7 show that QNN-QPINN achieves the deepest loss reduction across total loss, pressure, and velocity magnitude. To test the quality of the QNN-QPINN model after training and to detect overfitting, we save the model and perform inference on new data.

Figure 8 shows the inference results along with the model's predictions and two metrics: L2 relative error and L infinity. The results show that the model provides solutions that closely match the RK45 reference solutions. This matching advantage is clearly demonstrated in the heatmap and in the error values of the two functions, pressure p and velocity u, in the global error.

4 Conclusions and Future Outlook

In this research, we introduce a QPINN architecture that utilizes a quantum neural network to learn the rotation angle parameters for the main variational quantum circuit of the QPINN. The model's performance is evaluated through three case studies involving partial differential equations: Heat, Reaction-Diffusion, and Navier-Stokes equations. Comparative results with the classical PINN model and the non-trainable parameter model, Chebyshev-QPINN, demonstrate that the QNN-QPINN achieves a superior loss function with significantly lower values and a consistent, smooth reduction without any signs of oscillation. This QNN-QPINN superior performance indicates that the model has learned effectively and does not exhibit signs of overfitting.

One of the primary limitations of QPINN models, in general and specifically the QNN-QPINN, is that training is more computationally intensive than that of classical PINNs. This training overhead stems from the fact that the training process currently relies on simulators to emulate quantum properties on classical hardware. As the number of qubits increases, the Hilbert space grows exponentially, leading to excessive computational resource consumption. However, this computational issue is expected to be fully resolved with the advent of fault-tolerant quantum computers in the near future.

Our study demonstrates the potential of PDE solvers that combine quantum and classical approaches, leveraging the strengths of quantum computing and machine learning in the field of simulation. These QPINN solvers, with their parameter-learning capabilities, are able to solve nonlinear and highly stiff PDEs, providing smoother solutions than conventional classical analytical methods. We will continue to enhance the capabilities of the QNN-QPINN solver by tackling more complex nonlinear PDEs. Concurrently, our research will focus on addressing the barren plateau phenomenon—a significant gradient-based challenge that arises as the quantum circuit depth increases. Mitigating these vanishing gradients is crucial for ensuring the scalability and trainability of our proposed framework in high-dimensional domains.

Data Availability Statement. The source code for this study is available to the corresponding author upon request.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2010)
2. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
3. Blechschmidt, J., Ernst, O.G.: Three ways to solve partial differential equations with neural networks–a review. GAMM-Mitteilungen **44**(2), e202100006 (2021)
4. Cuomo, S., Di Cola, V.S., Giampaolo, F., Rozza, G., Raissi, M., Piccialli, F.: Scientific machine learning through physics-informed neural networks: where we are and what's next. J. Sci. Comput. **92**(3), 88 (2022)
5. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019)
6. Kyriienko, O., Paine, A.E., Elfving, V.E.: Solving nonlinear differential equations with differentiable quantum circuits. Phys. Rev. A **103**(5), 052416 (2021)
7. Cerezo, M., Verdon, G., Huang, H.-Y., Cincio, L., Coles, P.J.: Challenges and opportunities in quantum machine learning. Nature Comput. Sci. **2**(9), 567–576 (2022)
8. Berger, S., Hosters, N., Möller, M.: Trainable embedding quantum physics informed neural networks for solving nonlinear PDEs. Sci. Rep. **15**(1), 18823 (2025)
9. REPACSS. REPACSS: High Performance Computing Center. https://www.repacss.org/. Accessed Mar 2026
10. Paszke, A., et al.: PyTorch (2019). http://pytorch.org. Accessed Mar 2026
11. Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., et al.: PennyLane: automatic differentiation of hybrid quantum-classical computations. arXiv preprint arXiv:1811.04968 (2018)