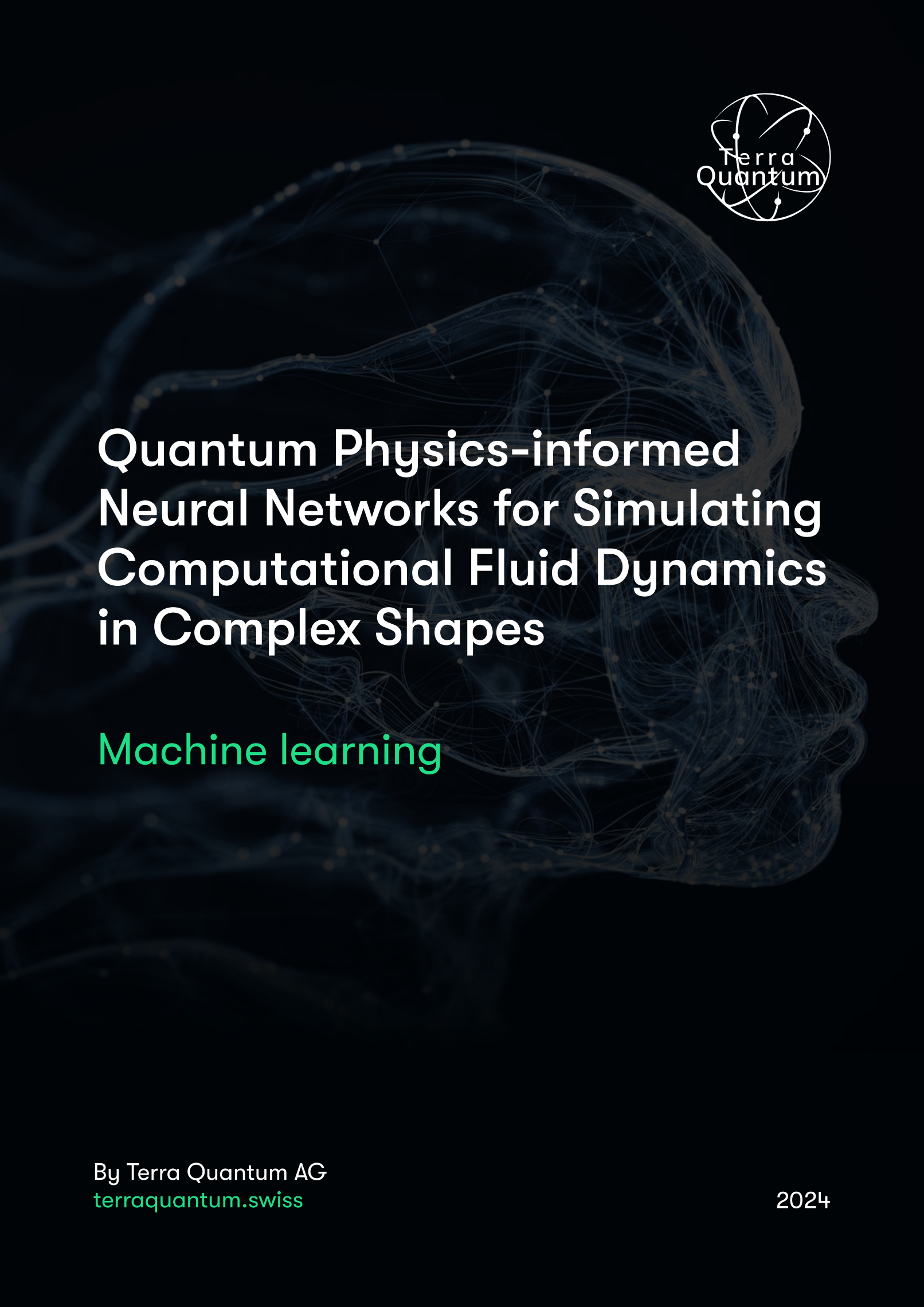


Quantum Physics-informed Neural Networks for Simulating Computational Fluid Dynamics in Complex Shapes

Machine learning

By Terra Quantum AG
terraquantum.swiss

2024

MACHINE LEARNING
Science and Technology

PURPOSE-LED PUBLISHING™

PAPER • OPEN ACCESS

Hybrid quantum physics-informed neural networks for simulating computational fluid dynamics in complex shapes

To cite this article: Alexandr Sedykh *et al* 2024 *Mach. Learn.: Sci. Technol.* **5** 025045

View the article online for updates and enhancements.

You may also like

- Accelerating physics-informed neural network based 1D arc simulation by meta learning
Linlin Zhong, Bingyu Wu and Yifan Wang
- Spectrally adapted physics-informed neural networks for solving unbounded domain problems
Mingtao Xia, Lucas Böttcher and Tom Chou
- Physics-informed neural network for turbulent flow reconstruction in composite porous-fluid systems
Seohee Jang, Mohammad Jadidi, Saleh Rezaeiravesh et al.

This content was downloaded from IP address 195.110.95.68 on 24/09/2024 at 16:10

IOP Publishing Mach. Learn.: Sci. Technol. 5 (2024) 025045 https://doi.org/10.1088/2632-2153/ad43b2

MACHINE LEARNING
Science and Technology

CrossMark

OPEN ACCESS

RECEIVED
17 January 2024

REVISED
27 March 2024

ACCEPTED FOR PUBLICATION
25 April 2024

PUBLISHED
22 May 2024

Original Content from this work may be used under the terms of the Creative Commons Attribution 4.0 licence.

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

PAPER

Hybrid quantum physics-informed neural networks for simulating computational fluid dynamics in complex shapes

Alexandr Sedykh[1], **Maninadh Podapaka**[2], **Asel Sagingalieva**[1], **Karan Pinto**[1], **Markus Pflitsch**[1] and **Alexey Melnikov**[1,*]

[1] Terra Quantum AG, 9000 St. Gallen, Switzerland
[2] Evonik Operations GmbH, 63450 Hanau-Wolfgang, Germany
[*] Author to whom any correspondence should be addressed.

E-mail: alexey@melnikov.info and ame@terraquantum.swiss

Keywords: quantum machine learning, physics-informed neural networks, computational fluid dynamics, 3D simulation, Navier–Stokes equations, hybrid quantum neural networks

Abstract

Finding the distribution of the velocities and pressures of a fluid by solving the Navier–Stokes equations is a principal task in the chemical, energy, and pharmaceutical industries, as well as in mechanical engineering and in design of pipeline systems. With existing solvers, such as OpenFOAM and Ansys, simulations of fluid dynamics in intricate geometries are computationally expensive and require re-simulation whenever the geometric parameters or the initial and boundary conditions are altered. Physics-informed neural networks (PINNs) are a promising tool for simulating fluid flows in complex geometries, as they can adapt to changes in the geometry and mesh definitions, allowing for generalization across fluid parameters and transfer learning across different shapes. We present a hybrid quantum PINN (HQPINN) that simulates laminar fluid flow in 3D *Y*-shaped mixers. Our approach combines the expressive power of a quantum model with the flexibility of a PINN, resulting in a 21% higher accuracy compared to a purely classical neural network. Our findings highlight the potential of machine learning approaches, and in particular HQPINN, for complex shape optimization tasks in computational fluid dynamics. By improving the accuracy of fluid simulations in complex geometries, our research using hybrid quantum models contributes to the development of more efficient and reliable fluid dynamics solvers.

1. Introduction

Computational fluid dynamics (CFD) solvers are primarily used to find the distribution of the velocity vector, $\boldsymbol{v}$, and pressure, p, of a fluid (or several fluids) given the initial conditions (e.g. initial velocity profile) and the geometrical domain where the fluid flows [1, 2]. To do this, it is necessary to solve a system of differential equations [3] called the Navier–Stokes (NS) equations that govern the fluid flow [4]. A well-established approach is to use numerical CFD solvers from several vendors, as well as publicly accessible alternatives, such as OpenFOAM [5] or Ansys [6]. These solvers discretize a given fluid volume into several small parts known as cells [7], where it is easier to get an approximate solution and then join the solutions of all the cells to get a complete distribution of the pressure and velocity over the entire geometrical domain.

While this is a rather crude explanation of how CFD solvers work, discretizing a large domain into smaller pieces accurately captures one of their main working principles [8]. The runtime of the computation and the accuracy of the solution both sensitively depend on the fineness of discretization, with finer grids taking longer but giving more accurate solutions. Furthermore, any changes to the geometrical parameters necessitate the creation of a new mesh and a new simulation. This process consumes both time and resources since one has to remesh and rerun the simulation every time a geometrical parameter is altered [9].

We propose a workflow employing physics-informed neural networks (PINNs) [10] to escape the need to restart the simulations whenever a geometrical property is changed completely. A PINN is a new promising tool for solving all kinds of parameterized partial differential equations (PDEs) [9], as it does not require

© 2024 The Author(s). Published by IOP Publishing Ltd

many prior assumptions, linearization or local time-stepping. One defines an architecture of the neural network (number of neurons, layers, etc) and then embeds physical laws and boundary conditions into it via constructing an appropriate loss function, so the prior task immediately travels to the domain of optimization problems.

For the classical solver, in the case of a parameterized geometric domain problem, getting accurate predictions for new modified shapes requires a complete program restart, even if the geometry has changed slightly. In the case of a PINN, to overcome this difficulty, one can use the transfer learning method [11] (section 3.6), which allows a model previously trained on some geometry to be trained on a slightly modified geometry without the need for a complete reset.

Also, using a trained PINN, it is easy to obtain a solution for other parameters of the PDE equation (e.g. kinematic viscosity in the NS equation [4], thermal conductivity in the heat equation, etc) with no additional training or restart of the neural network, but in the case of traditional solvers, restarts cannot be avoided.

One of the features that makes PINNs appealing is that they suffer less from the curse of dimensionality. Finite discretization of a d-dimensional cube with N points along each axis would require N^d points for a traditional solver. In other words, the complexity of the problem grows exponentially as the sampling size d increases. Using a neural network, however, one can define a $\mathbb{R}^d \to \mathbb{R}$ mapping (in case of just one target feature) with some weight parameters. Research on the topic suggests that the amount of weights/complexity of a problem in such neural networks grows polynomially with the input dimension d [12, 13]. This theoretical foundation alone allows PINNs to be a competitive alternative to solvers.

It is worth noting that although a PINN does not require N^d points for inference, it does require many points for training. There exist a variety of sampling methods, such as latin hypercube sampling [14], Sobol sequences [15], etc which can be used to improve a PINN's convergence on training [16]. However, a simple static grid of points is used in this work for the purpose of simplicity.

Classical machine learning can benefit substantially from quantum technologies. In [17], quantum computing is used in a similar problem setting. The performance of the current classical models is constrained by the high computing requirements. Quantum computing models can improve the learning process of existing classical models [18–24], allowing for better target function prediction accuracy with fewer iterations [25]. In many industries, including the pharmaceutical [26, 27], aerospace [28], automotive [29], logistics [30] and financial [31–35] sector quantum technologies can provide unique advantages over classical computing. Many traditionally important machine learning domains are also getting potential benefits from utilizing quantum technologies, e.g. in image processing [36–39] and natural language processing [40–43]. Solving nonlinear differential equations is also an application area for quantum algorithms that use differentiable quantum circuits [44, 45] and quantum kernels [46].

Recent developments in automatic differentiation enable us to compute the exact derivatives of any order of a PINN, so there is no need to use finite differences or any other approximate differentiation techniques. It, therefore, seems that we do not require a discretized mesh over the computational domain. However, we still need a collection of points from the problem's domain to train and evaluate a PINN. For a PINN to provide an accurate solution for a fluid dynamics problem, it is important to have a high expressivity (ability to learn solutions for a large variety of, possibly complex, problems). Fortunately, expressivity is a known strength of quantum computers [47–49]. Furthermore, quantum circuits are differentiable, meaning their derivatives can be calculated analytically, which is essential for noisy intermediate-scale quantum (NISQ) devices.

In this article, we propose a hybrid quantum PINN (HQPINN) shown in figure 1 to solve the NS equations with a steady flow in a 3D Y-shape mixer. The general principles and loss function of PINN workflow are described in section 2. The problem description, including the geometrical details, is presented in section 3 while in sections 3.5 and 3.7, we describe classical and hybrid PINNs in detail. Section 3.1 explains the intricacies of PINN's training process and simulation results. A transfer learning approach, applied to PINNs, is presented in section 3.6. Conclusions and further plans are described in section 4.

2. PINNs for solving PDEs

PINNs were originally introduced in [10]. The main idea is to use a neural network—usually a feedforward neural network like a multilayer perceptron—as a trial function for a PDE's solution. Let us consider an abstract PDE:

$$\mathcal{D}\left[f(\boldsymbol{r},t);\lambda\right]=0, \tag{1}$$

where $\Omega \subset \mathbb{R}^d$ is the computational domain, $\boldsymbol{r} \in \Omega$ is a coordinate vector, $t \in \mathbb{R}$ is time, $\mathcal{D}$ is a nonlinear differential operator with λ standing in for the physical parameters of the fluid and $f(\boldsymbol{r},t)$ is a solution function.

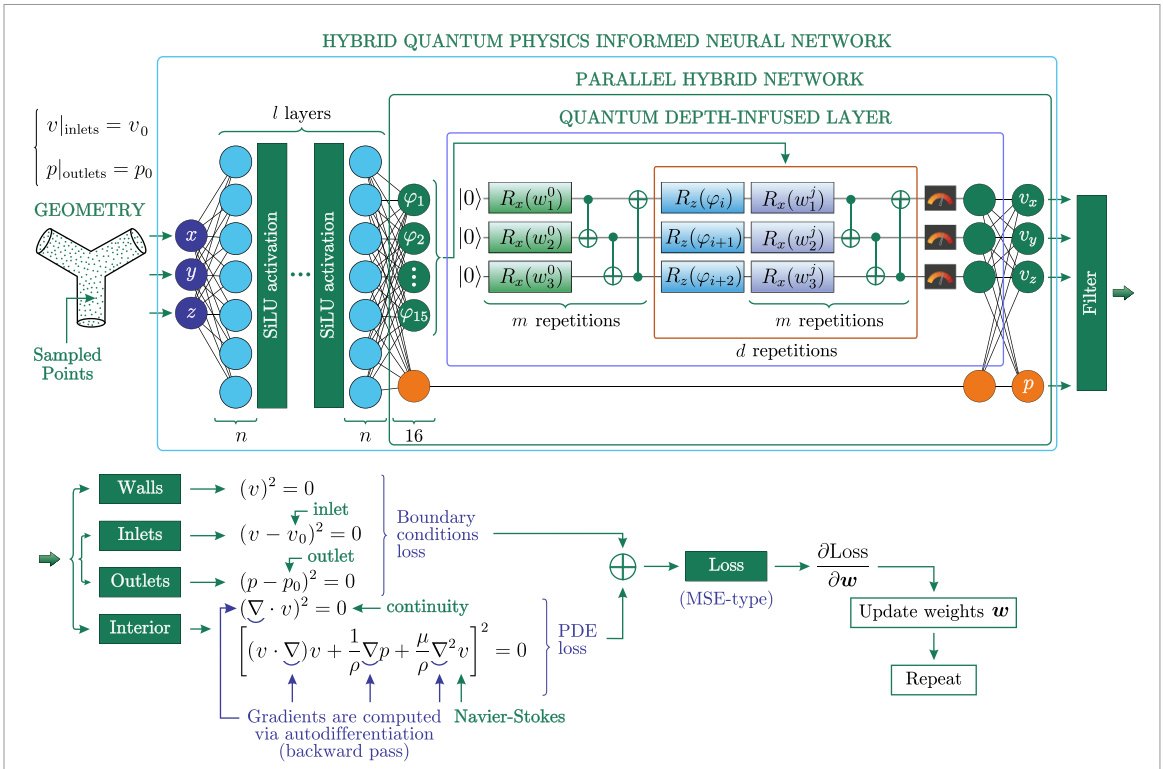


Figure 1. Scheme for the implementation of the HQPINN's training process and its architecture. HQPINN takes the (x, y, z) coordinates of the points, sampled in the geometrical domain (purple circles), and yields $(\boldsymbol{v}, p)$ as output, where $\boldsymbol{v}$ is the velocity vector (green circles) and p is the pressure (orange circle). The neural network itself begins with a classical part, which is a multilayer perceptron with l layers of n neurons. Then, there is a parallel hybrid network, which consists of a Quantum depth-infused layer (a variational quantum circuit) and a multilayer perceptron. A 'Filter' mechanism divides input points into four groups—fluid domain, walls, inlets, and outlets—each with its specific constraint (a boundary condition or a PDE). The error for each constraint is then calculated and incorporated into the total loss, which is then minimized via gradient descent to refine the model's predictions. The process iteratively adjusts the network's weights using full-batch gradient descent, ensuring that every point in the geometrical domain contributes to the whole solution that respects both the fluid dynamics within the domain and the conditions at its boundaries. The training process is described in section 3.1.

Let us consider a neural network $u(\boldsymbol{r}, t)$ that takes coordinates, $\boldsymbol{r}$, and time, t, as input and yields some real value (e.g. the pressure of a liquid at this coordinate at this particular moment).

We can evaluate $u(\boldsymbol{r}, t)$ at any point in the computational domain via a forward pass and compute its derivatives (of any order) $\partial_t^n u(\boldsymbol{r}, t)$, $\partial_{\boldsymbol{r}}^n u(\boldsymbol{r}, t)$ through backpropagation [50]. Therefore, we could substitute $f(\boldsymbol{r}, t) = u(\boldsymbol{r}, t)$ and try to learn the correct solution for the PDE via common machine learning gradient optimization methods [51] (e.g. gradient descent).

This approach is inspired firstly by the ability to calculate the *exact* derivatives of a neural network via autodifferentiation [52] and secondly by neural networks being universal function approximators [53].

The loss, $\mathcal{L}$, that the PINN tries to minimize is defined as

$$\mathcal{L} = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{BC}}, \tag{2}$$

where $\mathcal{L}_{\text{BC}}$ is the boundary condition loss and $\mathcal{L}_{\text{PDE}}$ is the PDE loss.

The boundary condition loss is responsible for satisfying the boundary conditions of the problem (e.g. a fixed pressure on the outlet of a pipe). For any field value, u, let us consider a Dirichlet (fixed-type) boundary condition [54]

$$u(\boldsymbol{r}, t)\,|_{\boldsymbol{r}\in B} = u_0(\boldsymbol{r}, t), \tag{3}$$

where $u_0(\boldsymbol{r}, t)$ is a boundary condition and $B \subset \mathbb{R}^d$ is the region where a boundary condition is applied.

If $u(\boldsymbol{r}, t)$ is a neural network function (see section 2), the boundary condition loss is calculated in a mean-squared error (MSE) manner:

$$\mathcal{L}_{\text{BC}} = \langle (u(\boldsymbol{r}, t) - u_0(\boldsymbol{r}, t))^2 \rangle_B, \tag{4}$$

where $\langle \cdot \rangle_B$ denotes averaging over all the data points $\boldsymbol{r} \in B$ that have this boundary condition.

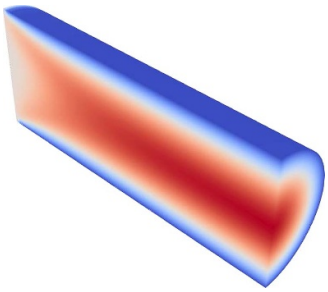


Figure 2. Isometric view of a slice of the cylinder. Color denotes the velocity magnitude for clarity.

The PDE loss is responsible for solving the governing PDE. If we have an abstract PDE (1) and a neural network function $u(\boldsymbol{r},t)$, substituting $f(\boldsymbol{r},t)=u(\boldsymbol{r},t)$ and calculating the MSE of the PDE gives:

$$\mathcal{L}_{\text{PDE}}=\langle(\mathcal{D}\left[u\left(\boldsymbol{r},t\right);\lambda\right])^2\rangle_{\Omega}, \tag{5}$$

where $\langle\cdot\rangle_{\Omega}$ means averaging over all the data points in the domain of the PDE.

3. Simulations

In this work, we consider the *steady* (i.e. time-independent) flow of an *incompressible* fluid in 3D without any external forces.

The NS equation (6) and the continuity equation (7) describe this scenario as follows:

$$-\left(\boldsymbol{v}\cdot\nabla\right)\boldsymbol{v}+\nu\Delta\boldsymbol{v}-\frac{1}{\rho}\nabla p=0, \tag{6}$$

$$\nabla\cdot\boldsymbol{v}=0, \tag{7}$$

where $\boldsymbol{v}(\boldsymbol{r})$ is the velocity vector, $p(\boldsymbol{r})$ is the pressure, ν is the kinematic viscosity and ρ is the fluid density. The PDE parameters ν and ρ were previously referred to as λ. For each of the 4 PDEs (3 projections of vector equation (6) and 1 scalar equation (7)), the $\mathcal{L}_{\text{PDE}}$ is calculated separately and then summed up.

3.1. Training PINNs

The geometry is a collection of points organized in a *.csv* file. It is split into four groups: fluid domain, walls, inlets, and outlets. The fluid domain is the domain in which the NS equations are solved, i.e. where the fluid flows. The other three groups have boundary conditions described in section 2. While untrained, PINN produces some random distribution of velocities and pressures. These values and their gradients are substituted into the corresponding NS equations and boundary conditions. With every iteration, the weights of the neural network are updated to minimize the error in the governing equations, and our solution becomes more and more accurate.

The training iteration is simple: the point cloud is passed through the PINN, the MSE loss is calculated (getting it requires taking gradients of $(\boldsymbol{v},p)$ at each point of the geometry), the gradient of the loss with respect to the weights is taken and the parameters are updated.

To visualize the training outcomes of the neural network, we used ParaView [55], and the simulation results are shown in figure 4(1) for loss and velocity distribution.

3.2. Cylinder flow simulation

At first, we used a simple 3D cylinder flow as a baseline solution for classical PINN. We also validated PINN's predictions by comparing them to OpenFOAM's solution.

In this simulation, we have no-slip boundary condition on the walls, fixed velocity of $v_0=10$ mm s^{-1} on the inlet, and fixed zero pressure $p=0$ Pa on the outlet. The fluid is water with standard density $\rho=1$ g cm^{-3} and $\nu=1$ mm^2 s^{-1} kinematic viscosity. The cylinder has radius of 2 mm and height of 10 mm (figure 2). Reynolds number for such flow is 10, so it can be considered laminar.

The training was done on a single NVIDIA A100 GPU for 20 000 iterations with an L-BFGS optimizer. For its training, the model uses 25 000 points inside the cylinder, which are taken straight from the mesh nodes used in OpenFOAM's simulation. PINN represented the solver's solution quite well: the relative error of velocity magnitude averaged over the whole cylinder is 1.2%, and the mean relative error of pressure is

Table 1. Pressure p and velocity magnitude $\|\boldsymbol{v}\|$ relative errors (averaged over the whole geometrical domain) between PINN predicted solution and reference OpenFOAM solution for different values of kinematic viscosity ν. The values, which model did not train on, are put in bold.

ν, mm^2 s^{-1}	p rel. error, %	$\|\boldsymbol{v}\|$ rel. error, %
1	10.0	4.7
1.5	2.0	2.8
2	2.7	1.5
2.5	5.9	3.6
3	8.5	5.8
3.5	10.5	7.7
4	12.3	9.4
5	15.1	12.3
10	21.0	24.6

0.8%. Refer to figure 3 for ground truth (OpenFOAM) and prediction (PINN) field values, as well as the distribution of relative error across the geometry.

The only error spikes both for pressure and velocity fields are located near the inlet edge of the pipe, where the uniform velocity profile of 10 mm s^{-1} abruptly changes to 0 due to non-slip boundary condition on walls, which is somewhat unphysical. This can be corrected by making the inlet velocity profile parabolic instead of uniform, but we decided to stick to a simple problem statement for the baseline solution.

3.3. Generalization of PINNs

To check if our baseline model possesses any generalization capabilities, we trained the same model as in section 3.2, but now incorporating 4 input parameters: the x, y, z coordinates and the kinematic viscosity ν, which was held constant at 1 mm^2 s^{-1} for the aforementioned model. Given that we employ a full-batch strategy, where the optimizer processes the gradient of all points in the geometry simultaneously, such a modification significantly increases the memory requirements for storing the total gradient. Consequently, we limited our training set to a set of ν values, $\{1, 2, 3, 4\}$ mm^2 s^{-1}. Nonetheless, we believe this range is sufficient to explore the generalization capabilities of the classical PINN.

Following the same training regime as in section 3.2, we present the results on the test data in table 1. The results indicate that the model generalizes well across the range of ν values it trained on.

However, when the model attempts to extrapolate solutions for ν values outside its training range, the quality of the solutions begins to degrade rapidly. More specifically, the solution tends to converge to the trivial case of $\boldsymbol{v} = 0$ as it approaches the end of the pipe. This phenomenon will be revisited and discussed further in section 3.4, where we attempt simulations on more complex geometries than the current one.

3.4. Y-shaped mixer flow simulation

This time, we try to simulate the flow of liquid in a Y-shaped mixer consisting of three tubes (figure 4). The mixing fluids are identical and have parameters $\rho = 1.0$ kg m^{-3} and $\nu = 1.0$ m^2 s^{-1}.

The imposed boundary conditions are as follows:

- no-slip BC on walls: $\boldsymbol{v}(\boldsymbol{r})|_{\text{walls}} = 0$,
- fixed velocity profile on inlets: $\boldsymbol{v}(\boldsymbol{r})|_{\text{inlets}} = \boldsymbol{v}_0(\boldsymbol{r})$,
- fixed pressure on outlets: $p(\boldsymbol{r})|_{\text{outlets}} = p_0$,

where, $\boldsymbol{v}_0(\boldsymbol{r})$ is a parabolic velocity profile on each inlet and $p_0(\boldsymbol{r}) = 0$.

The PINN was trained via full-batch gradient descent with the Adam optimizer [56] for 1000 epochs and then with the L-BFGS optimizer for 100 epochs until the gradients vanished and training became impossible. After the training, the PINN managed to learn a non-trivial downward flow at the beginning of both pipes. On the edges of these pipes, the velocities become zero, as they should, due to no-slip wall boundary conditions. However, further down the mixer, the solution degenerates to zero, so it does not even reach the mixing point. This fact should at least violate the continuity equation because there is an inward flow without matching outward flow. This problem is primarily caused by gradient vanishing inherent to PINN models.

In an attempt to overcome the challenge, we introduce an HQPINN model in 3.7, which shows better results in terms of PDE and boundary conditions satisfaction. We also research potential ways of making a generalizable PINN in 3.2 by employing a transfer learning method.

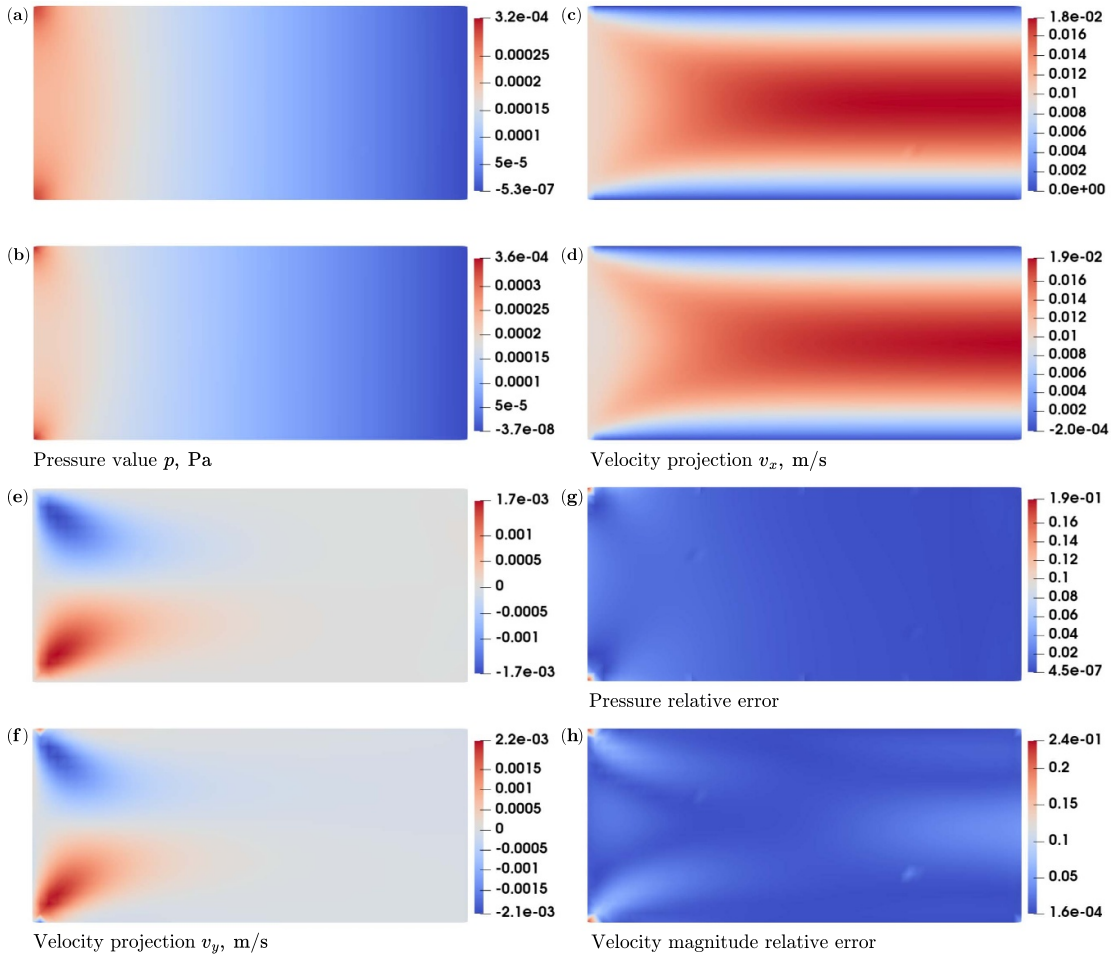


Figure 3. (a), (c) and (e) Ground truth pressure and velocity projections. (b), (d) and (f) PINN predicted pressure and velocity projections. (g) and (h) Relative pressure and velocity magnitude errors between PINN and ground truth (absolute error divided by maximum pressure or velocity magnitude across the cylinder).

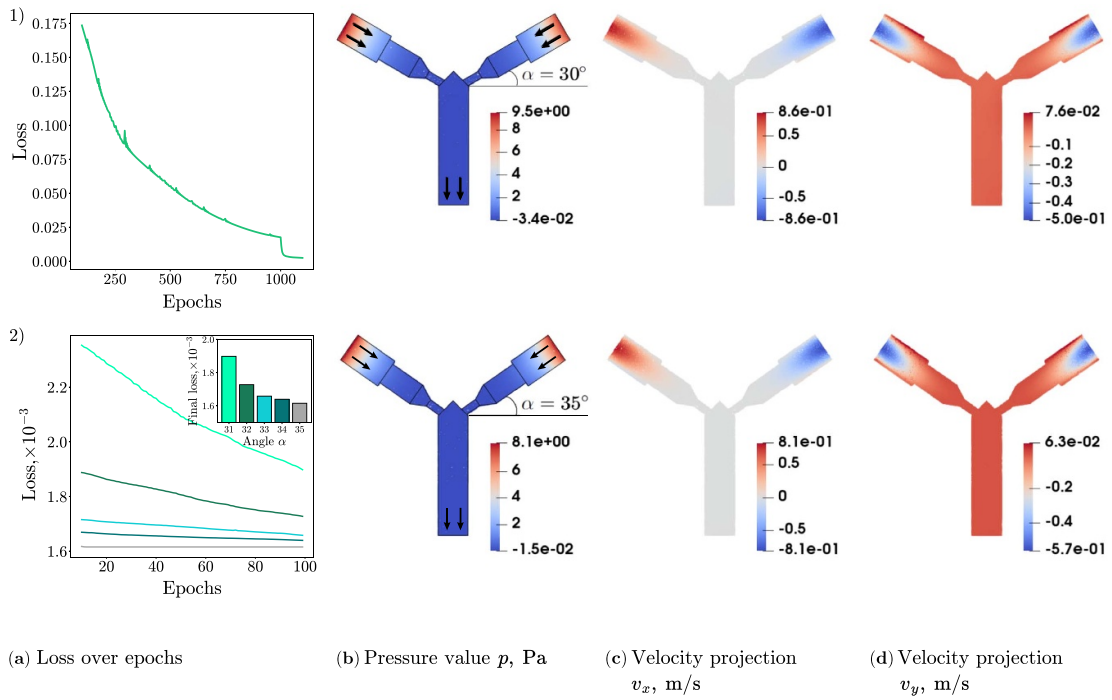
3.5. PINN architecture

In this section, we provide details on the PINN's architecture. The core of the PINN is a neural network whose architecture is a multilayer perceptron with several fully connected layers. As shown in figure 1, the first layer consists of 3 neurons (since the problem is 3D), then there are $l = 5$ hidden layers with n neurons, where $n = 128$ for the cylinder flow and $n = 64$ for the Y-shaped mixer. For the classical PINN, the 'Parallel Hybrid Network' box is replaced with one fully connected layer $n \to 4$. There is no quantum layer in the classical case, so the output goes straight into the filter. Between adjacent layers, there is a sigmoid linear unit activation function [57]. The PINN takes the (x, y, z) coordinates as inputs and yields $(\boldsymbol{v}, p)$ as its output, where $\boldsymbol{v}$ is the velocity vector with three components and p is the pressure.

3.6. Transfer learning

Transfer learning is a powerful method of using the knowledge and experience of one model that has been pretrained on one problem to solve another problem [11, 58]. It is extremely useful because it means that a second model does not have to be trained from scratch. This is especially helpful for fluid modeling, where selecting the most appropriate geometrical hyperparameters would otherwise lead to the simulations being rerun many times.

For transfer learning, we used a model from the previous section as a base, which had $\alpha_0 = 30°$, where α is the angle between the right pipe and the x axis (see figure 4). Then, for each $\alpha = \{31°, 32°, 33°, 34°, 35°\}$, we tried to obtain the solution, each time using the previously trained model as an initializer. For example, to transfer learn from 31° to 32°, we used the 31° model as a base and so on. Each iteration is trained for 100 epochs with L-BFGS. Figure 4 shows that the PINN adapts well to changes in the value of α. That is, our hypothesis was correct: with PINNs, one does not need to completely rerun the simulation on a parameter change, transfer learning from the base model will suffice.



(a) Loss over epochs (b) Pressure value p, Pa (c) Velocity projection v_x, m/s (d) Velocity projection v_y, m/s

Figure 4. (1) Classical model. (a) Loss curve. The classical PINN managed to learn a solution near the entry point well. However, it vanishes to zero further down the mixer. (2) Transfer learned models. (a) Loss curve. The transfer-learned models trained better with each iteration, surpassing the original model. (b)–(d) Distributions of the fluid pressure and velocities for the last model with $\alpha = 35°$.

3.7. HQPINN

Quantum machine learning was shown to be applicable to solving differential equations [44–46]. Here, we introduce a hybrid quantum neural network called HQPINN and compare it against its classical counterpart, classical PINN. As shown in figure 1, the architecture of HQPINN is comprised of classical fully connected layers, specifically a multilayer perceptron, coupled with a parallel hybrid network [23]. The latter is a unique intertwining of a quantum depth-infused layer [26, 48] and a classical layer. Interestingly, the first 15 units ($\phi_1, \ldots, \phi_{15}$, depicted in green) of the layer are dedicated to the quantum layer, whereas the information from one orange unit proceeds along the classical pathway. This parallel structure enables simultaneous information processing, thereby enhancing the efficiency of the learning process.

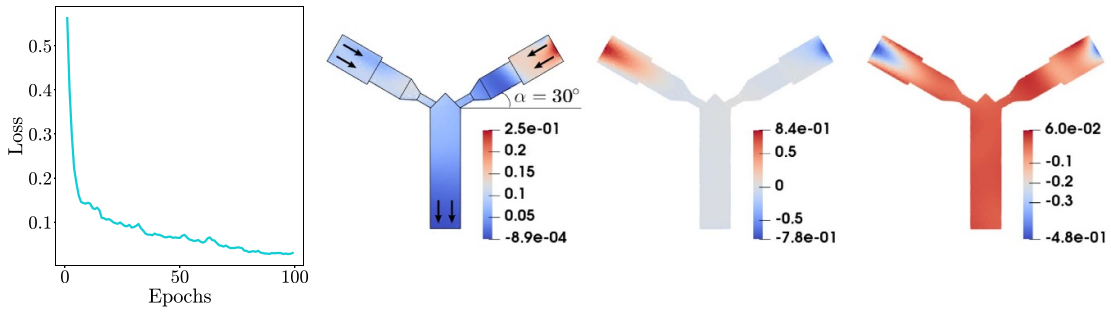
Regarding the quantum depth-infused layer, it is implemented as a variational quantum circuit (VQC), a promising strategy for navigating the complexities of the NISQ epoch [59]. The NISQ era is characterized by quantum devices with a limited number of qubits that cannot yet achieve error correction, making strategies like VQCs particularly relevant [26, 60, 61].

The capacity of HQPINN to direct diverse segments of the input vector toward either the quantum or classical part of the network equips PINNs with an enhanced ability to process and learn from various patterns more efficiently. For instance, some patterns may be optimally processed by the quantum network, while others might be better suited for the classical network. This flexibility in processing contributes to the robust learning capabilities of HQPINN.

3.7.1. Quantum depth-infused layer

Transitioning into the building blocks of our model, the quantum depth-infused layer takes center stage. Quantum gates are the basic building blocks for any quantum circuit, including those used for machine learning. Quantum gates come in single-qubit (e.g. rotation gate $R_y(\theta)$, gate that plays a central role in quantum machine learning) and multiple-qubit gates (e.g. CNOT) and modulate the state of qubits to perform computations. The $R_y(\theta)$ gate rotates the qubit around the y-axis of the Bloch sphere by an angle θ, while the two-qubit CNOT gate changes the state of one qubit based on the current state of another qubit. Gates can be fixed, which means they perform fixed calculations, such as the Hadamard gate, or they can be variable, such as the rotation gate that depends on the rotation angle and may perform computations with tunable parameters.

To extract the results, qubits are measured and projected onto a specific basis, and the expected value is calculated. When using the σ_z Pauli matrix observable, the expected value of the measurement is represented



(a) Loss over epochs (b) Pressure value p, Pa (c) Velocity projection v_x, m/s (d) Velocity projection v_y, m/s

Figure 5. Hybrid quantum model. (a) Loss curve. (b)–(d) Distribution of the fluid pressure and velocities for the HQPINN at ($\alpha = 30°$). Similarly to the classical model, a non-trivial solution near the entry point is present. However, there is an asymmetry between the left and the right pipe, which should not be there. This could be an effect of the data encoding strategy.

as: $\langle\psi|\,\sigma_z\,|\psi\rangle$, with ψ signifying the wave function that depicts the current state of our quantum system. For a more detailed understanding of quantum circuits, including logic gates and measurements, standard quantum computing textbooks such as [62] offer a comprehensive guide.

To refine the functioning of the network we propose, the initial phase of data processing is performed on a classical computer. Subsequently, these data are integrated into the quantum gate parameters in an encoding layer (blue gates) repeated $d = 5$ times, forming part of the quantum depth-infused layer followed by the variational layer (green gates). Number of repetitions of the variational layer $m = 2$. As the algorithm develops, these gate parameters in the variational layer are dynamically adjusted. Finally, at the measurement level, the qubits are quantified, leading to a series of classical bits as the output.

Conceptually, a quantum algorithm resembles a black box that receives classical information as input and yields classical information as output. The objective here is to fine-tune the variational parameters such that the measurement outcome most accurately reflects the prediction function. In essence, this parameter optimization is akin to optimizing the weights in a classical neural network, thereby effectively training the quantum depth-infused layer.

3.7.2. Training HQPINN

The HQPINN consists of the classical PINN with weights pre-initialized from the previous stage (as described in section 3.5), a parallel hybrid network and a fully-connected layer at the end.

The training process of the hybrid PINN does not differ from that of the classical PINN except in the following ways. Firstly, all calculations are done on the classical simulator of quantum hardware, the QMware server [63], which has recently been shown to be quite good for running hybrid algorithms [64].

Secondly, how does one backpropagate through the quantum circuit layer? The answer is to use the 'adjoint differentiation' method, introduced in [65], which helps to compute derivatives of a VQC on a classical simulator efficiently.

This time the model was trained for 100 epochs using mini-batch gradient descent with the Adam optimizer (figure 5). The mini-batch strategy was employed due to the very low training speed of quantum circuits, as they train on a CPU. We will then compare this model with a purely classical one, with the same architecture from section 3.5, but this time trained only with mini-batch Adam. All learning hyperparameters (learning rate, scheduler parameters, batch size) are shared between the quantum and classical models. Comparing the two, figure 6 shows that the quantum model outperforms the classical one in terms of the loss value by 21%. As the loss function for PINNs directly corresponds to PDE and BC satisfaction, it implies that HQPINN has achieved better physical accuracy before the gradient vanish. However, the mini-batch training strategy for the HQPINN is far from ideal and stems from the large training time of simulated quantum circuits. Proper hardware backend for highly parallel GPU-accelerated quantum computing could greatly extend the HQPINN advantage.

Additionally, the question of computational demands scaling is of importance. For the CFD examples considered in the paper, we used a three-qubit circuit, which is not computationally demanding to execute on a classical computer. However, with each additional qubit, the runtime of the quantum circuit execution will approximately double. That means, according to the benchmarks [64], the runtime will become significant beyond 20 qubits circuits. Nonetheless, with the development of quantum computers, the scaling is expected to be more favorable as the runtime on a quantum chip would stay constant in case the circuit

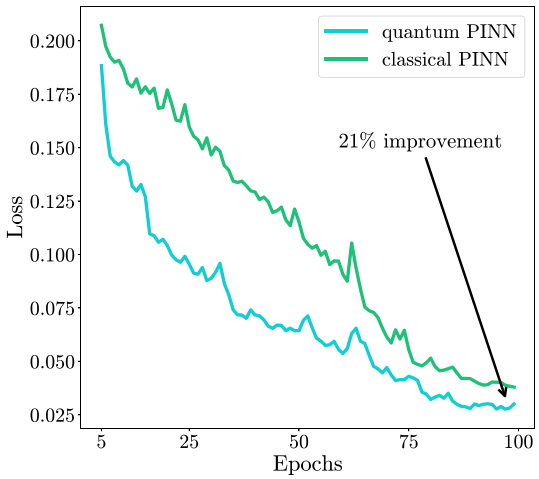


Figure 6. Losses for the classical and hybrid quantum PINNs. HQPINN outperforms the classical PINN in terms of loss value due to higher expressivity.

depth is fixed. Therefore, we believe that for more complex shapes, the simulation would require more data and, hence, larger HQPINN models require the need of quantum computers for simulation.

4. Discussion

In our investigation, we pursued two distinct goals aimed at advancing the capabilities of PINNs through the integration of quantum computing methodologies. The first objective focused on enhancing the performance of classical PINNs in solving PDEs within 3D geometries through the adoption of HQPINNs. This endeavor was motivated by the recognized success of classical PINNs in 2D problem settings [10] and the existing gap in their application to more complex 3D scenarios. The significance of this goal lies in its potential to broaden the applicability of PINNs to a wider range of scientific and engineering problems characterized by three-dimensional spaces.

Our second goal was to investigate the presence and feasibility of transfer learning in classical PINNs when applied to 3D geometries. This exploration is crucial for understanding the ability of PINNs to generalize across different geometrical configurations and physical problems, potentially enabling more efficient re-training processes on novel problems and geometries. The overarching aim here is to enhance the model's versatility and reduce computational costs associated with the training of neural networks for new tasks.

Our results in applying HQPINNs to 3D PDE problems demonstrate a notable improvement in reducing PDE loss, marking a significant step forward in our first objective. Specifically, we observed a 21% reduction in loss when comparing the performance of HQPINNs against their purely classical counterparts. This quantitative improvement highlights the enhanced expressiveness and computational efficiency brought about by the integration of quantum layers, suggesting that quantum computing elements can indeed augment the capability of PINNs in handling the intricacies of 3D problems. However, it is important to recognize that despite these advancements, achieving an optimal solution for 3D PDEs remains a challenging endeavor. The observed loss reduction, while substantial, does not resolve the complexities of 3D geometrical problem solving, indicating the need for further refinement and exploration of the HQPINN framework.

In the realm of transfer learning, our exploration yielded encouraging signs that classical PINNs possess an inherent capability to adapt to variations in a geometrical shape, when theangle of a Y-shaped mixer was changed. This qualitative finding is instrumental in demonstrating the potential for PINNs to be applied in shape optimization tasks and other applications requiring flexibility across different geometrical configurations. However, the journey towards realizing full transfer learning and generalization capabilities in PINNs is still in its early stages. Our initial successes serve as a foundation for further research, underscoring the necessity for continued development and investigation into the mechanisms that enable effective transfer learning within PINNs.

In developing our HQPINNs, we have also considered recent advances in other quantum and quantum-inspired methods for solving nonlinear equations, such as based on the matrix product states approach [66] and variational quantum algorithms for nonlinear problems [44, 67]. While our approach primarily focuses on handling complex boundary conditions, the integration of these quantum-inspired

methods and variational quantum algorithms could further improve the performance and accuracy of our proposed quantum PINN model.

Collectively, our findings contribute valuable insights into the potential of hybrid quantum neural networks and the prospects for transfer learning in solving complex physical problems. As we continue to push the boundaries of what is achievable with PINNs, our future efforts will focus on refining these models to approach accurate full-scale CFD simulation in complex 3D shapes.

The plan includes exploring better architectures for the quantum PINN, investigating their impact on expressiveness, generalizability and optimization landscape, and trying data-driven approaches. Entirely different networks, such as neural operators [68, 69] and graph neural networks [70, 71], could also be considered in a quantum setting and enhanced with quantum circuits.

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

ORCID iDs

Alexandr Sedykh https://orcid.org/0009-0003-8431-0873
Asel Sagingalieva https://orcid.org/0009-0009-3931-3702
Alexey Melnikov https://orcid.org/0000-0002-5033-4063

References

[1] Zawawi M H, Saleha A, Salwa A, Hassan N H, Zahari N M, Ramli M Z and Muda Z C 2018 A review: fundamentals of computational fluid dynamics (CFD) *AIP Conf. Proc.* **2030** 020252
[2] Anderson J D and Wendt J 1995 *Computational Fluid Dynamics* vol 206 (Springer)
[3] Finlay Simmons G 1972 Differential equations with applications and historical notes *Differential Equations With Applications and Historical Notes* (McGraw-Hill)
[4] Katz A J 2009 *Meshless Methods for Computational Fluid Dynamics* (Stanford University)
[5] OpenFOAM 2022 (available at: www.openfoam.com/)
[6] Ansys Engineering Simulation Software 2022 (available at: www.ansys.com/)
[7] Marić T, Kotheb D B and Bothe D 2020 Unstructured un-split geometrical volume-of-fluid methods—a review *J. Comput. Phys.* **420** 109695
[8] Marić T, Marschall H and Bothe D 2013 voFoam—a geometrical volume of fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using OpenFOAM (arXiv:1305.3417)
[9] Cai S, Mao Z, Wang Z, Yin M and Em Karniadakis G 2022 Physics-informed neural networks (PINNs) for fluid mechanics: a review *Acta Mech. Sin.* **37** 1–12
[10] Raissi M, Perdikaris P and Karniadakis G E 2019 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
[11] Neyshabur B, Sedghi H and Zhang C 2020 What is being transferred in transfer learning? (arXiv:2008.11687)
[12] Hutzenthaler M, Jentzen A, Kruse T and Anh Nguyen T 2020 A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations *SN Partial Differ. Equ. Appl.* **1** 1–34
[13] Grohs P, Hornung F, Jentzen A and Von Wurstemberger P 2018 A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations (arXiv:1809.02362)
[14] Stein M 1987 Large sample properties of simulations using latin hypercube sampling *Technometrics* **29** 143–51
[15] Sobol' I M, Asotsky D, Kreinin A and Kucherenko S 2011 Construction and comparison of high-dimensional Sobol'generators *Wilmott* **2011** 64–79
[16] Zubov K *et al* 2021 Neuralpde: automating physics-informed neural networks (PINNs) with error approximations (arXiv:2107.09443)
[17] Gaitan F 2020 Finding flows of a Navier–Stokes fluid through quantum computing *npj Quantum Inf.* **6** 1–6
[18] Dunjko V and Briegel H J 2018 Machine learning & artificial intelligence in the quantum domain: a review of recent progress *Rep. Prog. Phys.* **81** 074001
[19] Melnikov A, Kordzanganeh M, Alodjants A and Lee R-K 2023 Quantum machine learning: from physics to software engineering *Adv. Phys.* X **8** 2165452
[20] Neven H, Denchev V S, Rose G and Macready W G 2012 QBoost: large scale classifier training with adiabatic quantum optimization *Proc. Asian Conf. Mach. Learn.* (*Proc. Machine Learning Research* vol 25) ed C H H Steven and W Buntine (PMLR) pp 333–48
[21] Rebentrost P, Mohseni M and Lloyd S 2014 Quantum support vector machine for big data classification *Phys. Rev. Lett.* **113** 130503
[22] Saggio V *et al* 2021 Experimental quantum speed-up in reinforcement learning agents *Nature* **591** 229–33
[23] Kordzanganeh M, Kosichkina D and Melnikov A 2023 Parallel hybrid networks: an interplay between quantum and classical neural networks *Intell. Comput.* **2** 0028
[24] Kurkin A, Hegemann J, Kordzanganeh M and Melnikov A 2023 Forecasting the steam mass flow in a powerplant using the parallel hybrid network (arXiv:2307.09483)
[25] Perelshtein M, Sagingalieva A, Pinto K, Shete V, Pakhomchik A, Melnikov A, Neukart F, Gesek G, Melnikov A and Vinokur V 2022 Practical application-specific advantage through hybrid quantum computing (arXiv:2205.04858)
[26] Sagingalieva A, Kordzanganeh M, Kenbayev N, Kosichkina D, Tomashuk T and Melnikov A 2023 Hybrid quantum neural network for drug response prediction *Cancers* **15** 2705
[27] Gircha A I, Boev A S, Avchaciov K, Fedichev P O and Fedorov A K 2021 Training a discrete variational autoencoder for generative chemistry and drug design on a quantum annealer (arXiv:2108.11644)

[28] Rainjonneau S et al 2023 Quantum algorithms applied to satellite mission planning for Earth observation IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 16 7062–75
[29] Sagingalieva A et al 2023 Hybrid quantum ResNet for car classification and its hyperparameter optimization Quantum Mach. Intell. 5 38
[30] Haboury N, Kordzanganeh M, Schmitt S, Joshi A, Tokarev I, Abdallah L, Kurkin A, Kyriacou B and Melnikov A 2023 A supervised hybrid quantum machine learning solution to the emergency escape routing problem (arXiv:2307.15682)
[31] Alcazar J, Leyton-Ortega V and Perdomo-Ortiz A 2020 Classical versus quantum models in machine learning: insights from a finance application Mach. Learn.: Sci. Technol. 1 035003
[32] Coyle B, Henderson M, Chan Jin Le J, Kumar N, Paini M and Kashefi E 2021 Quantum versus classical generative modelling in finance Quantum Sci. Technol. 6 024013
[33] Pistoia M et al 2021 Quantum machine learning for finance (arXiv:2109.04298)
[34] Emmanoulopoulos D and Dimoska S 2022 Quantum machine learning in finance: time series forecasting (arXiv:2202.00599)
[35] Cherrat E A et al 2023 Quantum deep hedging (arXiv:2303.16585)
[36] Senokosov A, Sedykh A, Sagingalieva A, Kyriacou B and Melnikov A 2024 Quantum machine learning for image classification Mach. Learn.: Sci. Technol. 5 015040
[37] Li W, Chu P-C, Liu G-Z, Tian Y-B, Qiu T-H and Wang S-M 2022 An image classification algorithm based on hybrid quantum classical convolutional neural network Quantum Eng. 2022 1–9
[38] Naumov A, Melnikov A, Abronin V, Oxanichenko F, Izmailov K, Pflitsch M, Melnikov A and Perelshtein M 2023 Tetra-AML: automatic machine learning via tensor networks (arXiv:2303.16214)
[39] Riaz F, Abdulla S, Suzuki H, Ganguly S, Deo R C and Hopkins S 2023 Accurate image multi-class classification neural network model with quantum entanglement approach Sensors 23 2753
[40] Hong Z, Wang J, Qu X, Zhao C, Tao W and Xiao J 2022 QSpeech: low-qubit quantum speech application toolkit (arXiv:2205.13221)
[41] Lorenz R, Pearson A, Meichanetzidis K, Kartsaklis D and Coecke B 2021 QNLP in practice: running compositional models of meaning on a quantum computer (arXiv:2102.12846)
[42] Coecke B, de Felice G, Meichanetzidis K and Toumi A 2020 Foundations for near-term quantum natural language processing (arXiv:2012.03755)
[43] Meichanetzidis K, Toumi A, de Felice G and Coecke B 2020 Grammar-aware question-answering on quantum computers (arXiv:2012.03756)
[44] Kyriienko O, Paine A E and Elfving V E 2021 Solving nonlinear differential equations with differentiable quantum circuits Phys. Rev. A 103 052416
[45] Paine A E, Elfving V E and Kyriienko O 2023 Physics-informed quantum machine learning: solving nonlinear differential equations in latent spaces without costly grid evaluations (arXiv:2308.01827)
[46] Paine A E, Elfving V E and Kyriienko O 2023 Quantum kernel methods for solving regression problems and differential equations Phys. Rev. A 107 032428
[47] Kordzanganeh M, Sekatski P, Fedichkin L and Melnikov A 2023 An exponentially-growing family of universal quantum circuits Mach. Learn.: Sci. Technol. 4 035036
[48] Schuld M, Sweke R and Meyer J J 2021 Effect of data encoding on the expressive power of variational quantum-machine-learning models Phys. Rev. A 103 032430
[49] Schuld M, Bocharov A, Svore K M and Wiebe N 2020 Circuit-centric quantum classifiers Phys. Rev. A 101 032308
[50] Rumelhart D E, Hinton G E and Williams R J 1986 Learning representations by back-propagating errors Nature 323 533–6
[51] Ruder S 2016 An overview of gradient descent optimization algorithms (arXiv:1609.04747)
[52] Gunes Baydin A, Pearlmutter B A, Andreyevich Radul A and Siskind J M 2018 Automatic differentiation in machine learning: a survey J. Mach. Learn. Res. 18 1–43
[53] Hornik K, Stinchcombe M B and White H L 1989 Multilayer feedforward networks are universal approximators Neural Netw. 2 359–66
[54] Greenshields C and Weller H 2022 Notes on Computational Fluid Dynamics: General Principles (CFD Direct Ltd)
[55] ParaView - Open-source, multi-platform data analysis and visualization application 2022 (available at: www.paraview.org/)
[56] Kingma D P and Ba J 2014 Adam: a method for stochastic optimization (arXiv:1412.6980)
[57] Elfwing S, Uchibe E and Doya K 2018 Sigmoid-weighted linear units for neural network function approximation in reinforcement learning Neural Netw. 107 3–11
[58] Mari A, Bromley T R, Izaac J, Schuld M and Killoran N 2020 Transfer learning in hybrid classical-quantum neural networks Quantum 4 340
[59] Preskill J 2018 Quantum computing in the NISQ era and beyond Quantum 2 79
[60] Zhao C and Gao X-S 2019 QDNN: DNN with quantum neural network layers (arXiv:1912.12660)
[61] Dou T, Wang K, Zhou Z, Yan S and Cui W 2021 An unsupervised feature learning for quantum-classical convolutional network with applications to fault detection 2021 40th Chinese Control Conf. (CCC) (IEEE) pp 6351–5
[62] Nielsen M A and Chuang I 2002 Quantum Computation and Quantum Information (American Association of Physics Teachers)
[63] QMware Cloud 2022 Hybrid quantum computing (available at: https://qm-ware.com/)
[64] Kordzanganeh M, Buchberger M, Kyriacou B, Povolotskii M, Fischer W, Kurkin A, Somogyi W, Sagingalieva A, Pflitsch M and Melnikov A 2023 Benchmarking simulated and physical quantum processing units using quantum and hybrid algorithms Adv. Quantum Technol. 6 2300043
[65] Jones T and Gacon J 2020 Efficient calculation of gradients in classical simulations of variational quantum algorithms (arXiv:2009.02823)
[66] Gourianov N, Lubasch M, Dolgov S, van den Berg Q Y, Babaee H, Givi P, Kiffner M and Jaksch D 2022 A quantum-inspired approach to exploit turbulence structures Nat. Comput. Sci. 2 30–37
[67] Lubasch M, Joo J, Moinier P, Kiffner M and Jaksch D 2020 Variational quantum algorithms for nonlinear problems Phys. Rev. A 101 010301
[68] Kovachki N et al 2021 Neural operator: learning maps between function spaces (arXiv:2108.08481)
[69] Li Z et al 2020 Fourier neural operator for parametric partial differential equations (arXiv:2010.08895)
[70] Pfaff T, Fortunato M, Sanchez-Gonzalez A and Battaglia P W 2021 Learning mesh-based simulation with graph networks Int. Conf. on Learning Representations
[71] Sanchez-Gonzalez A, Godwin J, Pfaff T, Ying R, Leskovec J and Battaglia P 2020 Learning to simulate complex physics with graph networks Int. Conf. on Machine Learning (PMLR) pp 8459–68