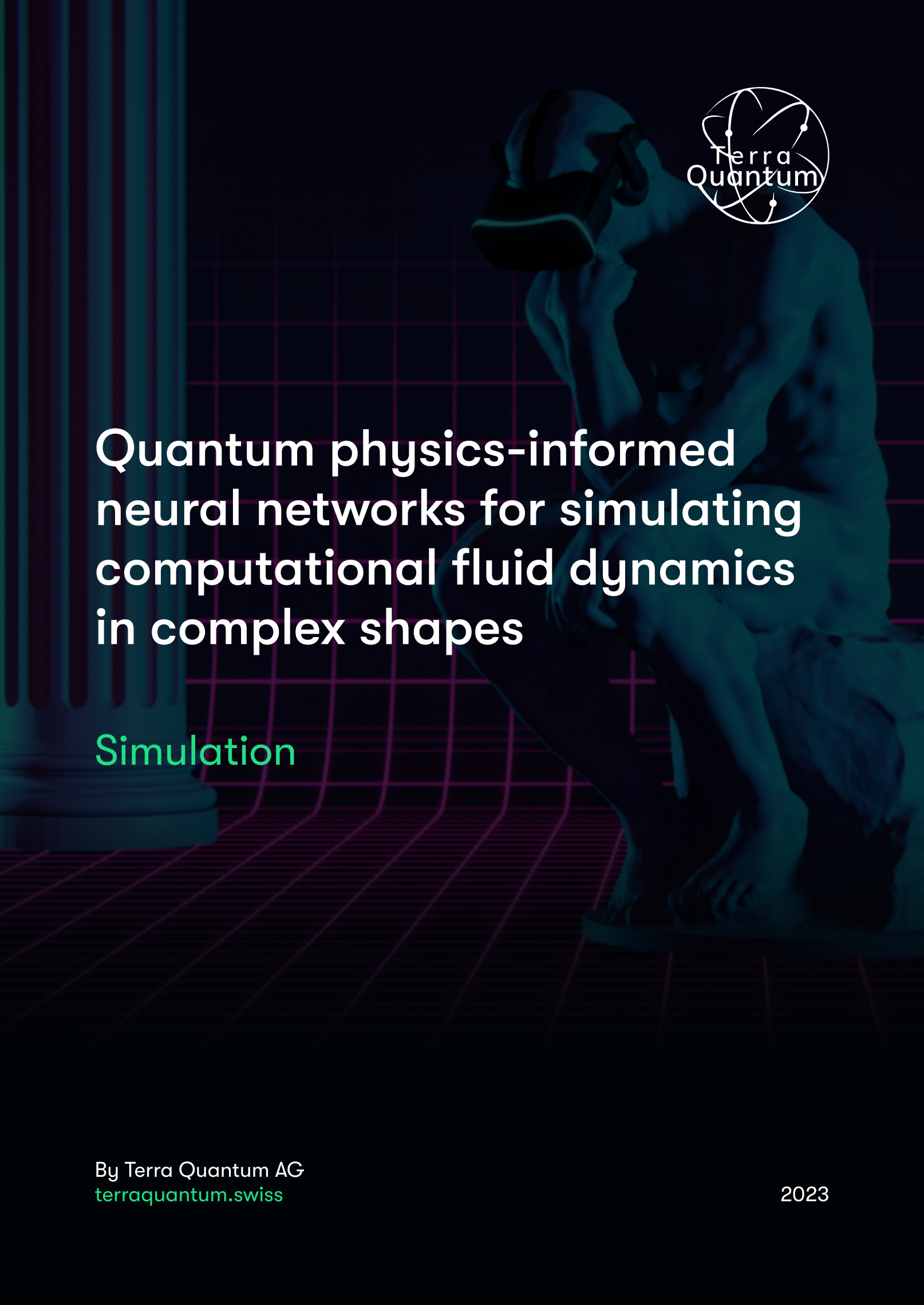


Quantum physics-informed neural networks for simulating computational fluid dynamics in complex shapes

Simulation

By Terra Quantum AG
terraquantum.swiss

2023

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

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

[1] *Terra Quantum AG, 9000 St. Gallen, Switzerland*
[2] *Evonik Industries AG, 45128 Essen, Germany*

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 the 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 different shapes. We present a hybrid quantum physics-informed neural network that simulates laminar fluid flows 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 quantum PINNs, for complex shape optimization tasks in computational fluid dynamics. By improving the accuracy of fluid simulations in complex geometries, our research using quantum PINNs contributes to the development of more efficient and reliable fluid dynamics solvers.

I. INTRODUCTION

Computational fluid dynamics (CFD) solvers are primarily used to find the distribution of the velocity vector, , and pressure, p, of a fluid (or several fluids) given the initial conditions (e.g. initial velocity profile) and the geometrical domain in which 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 actually work, the idea of 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 completely restart the simulations whenever a geometrical property is changed. A PINN is a new promising tool for solving all kinds of parameterized partial differential equations (PDEs) [9], as it does not require 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 restart of the program, 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] (Sec. III A 2), 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 which 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 [18–23], allowing for better target function prediction accuracy with fewer iterations [24]. In many industries, including the pharmaceutical [25, 26], space [27], automotive [28], and financial [29–33] 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 [34–37] and natural language processing [38–41].

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 [42]. Furthermore, quantum circuits are differentiable, which means that their derivatives can be calculated analytically, which is essential on noisy intermediate-scale quantum devices.

In this article, we propose a quantum PINN to solve the NS equations with a steady flow in 3D Y-shape mixer. The general principles and loss function of PINN workflow are described in Sec. II. The problem description, including the geometrical details, is presented in Sec. III while in Sec. III A and Sec. III B, we describe classical and quantum PINNs in detail. Sec. III A 1 explains the intricacies of PINN's training process and simulation results. A transfer learning approach, applied to PINNs, is presented in Sec. III A 2. Conclusions and further plans are described in Sec. IV.

II. PHYSICS-INFORMED NEURAL NETWORKS FOR SOLVING PARTIAL DIFFERENTIAL EQUATIONS

Physics-informed neural networks (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:

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

where $\Omega \subset \mathbb{R}^d$ is the computational domain, $r \in \Omega$ 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 $f(r,t)$ is a solution function.

Let us consider a neural network $u(r,t)$ that takes coordinates, 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 of time).

We can evaluate $u(r,t)$ at any point in the computational domain via a forward pass and compute its derivatives (of any order) $\partial_t^n u(r,t)$, $\partial_r^n u(r,t)$ through backpropagation [43]. Therefore, we could just substitute $f(r,t) = u(r,t)$ and try to learn the correct solution for the PDE via common machine learning gradient optimization methods [44] (e.g., gradient descent).

This approach is inspired firstly by the ability to calculate the *exact* derivatives of a neural network via autodifferentiation [45] and secondly by neural networks being universal function approximators [46].

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 partial differential equation 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 [47]

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

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

If $u(r,t)$ is a neural network function (see Sec. II), the boundary condition loss is calculated in a mean-squared error (MSE) manner:

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

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

The PDE loss is responsible for solving the governing PDE. If we have an abstract PDE (1) and a neural network function $u(r,t)$, substituting $f(r,t) = u(r,t)$ and calculating the mean-squared error of the PDE gives:

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

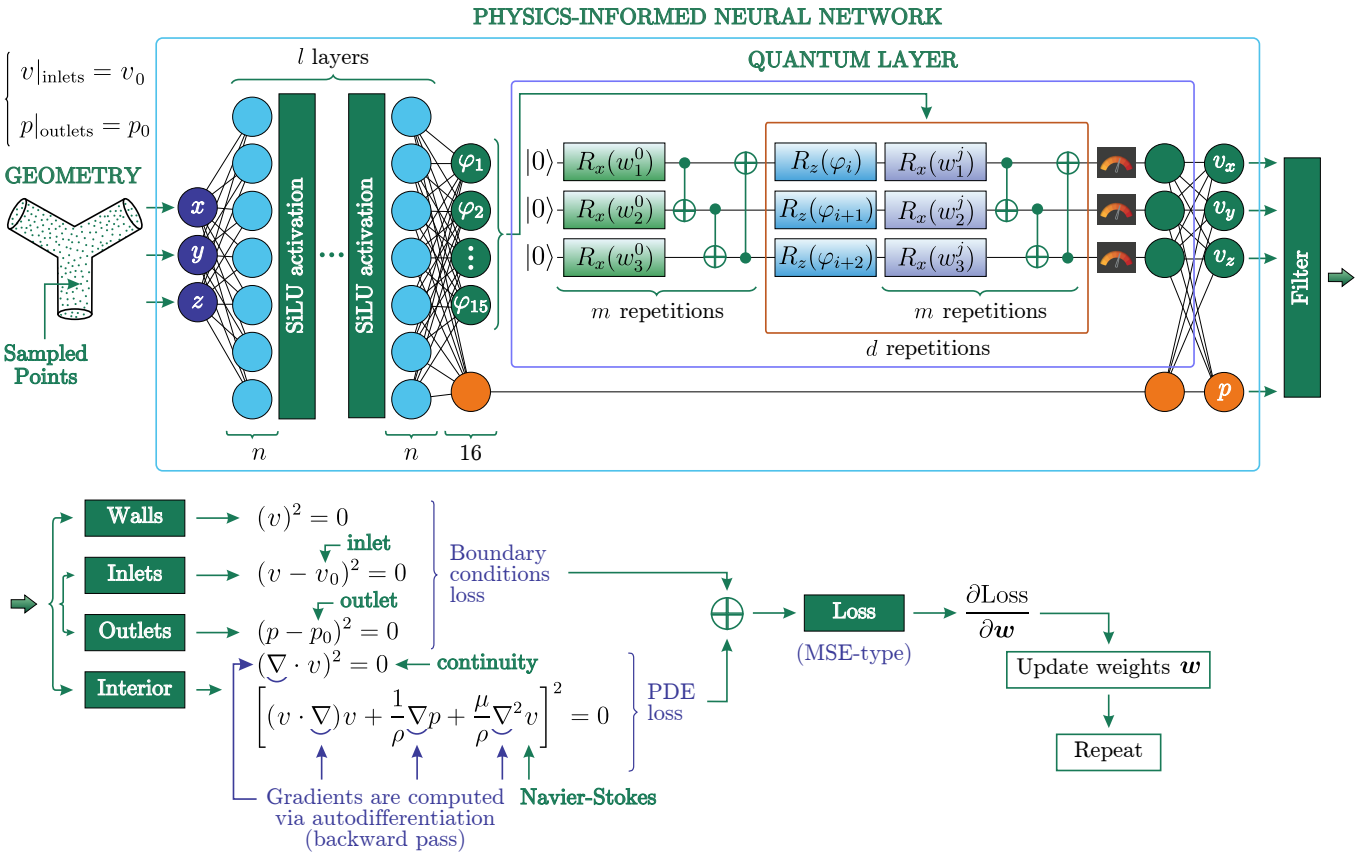


FIG. 1. Scheme for the implementation of the hybrid PINN's training process and its architecture. PINN takes the (x, y, z) coordinates of the points, sampled in the geometrical domain, and yields (v, p) as output, where v is the velocity vector and p is the pressure. The neural network itself begins with a classical part, which is a multilayer perceptron with l layers of n neurons. Then there is a quantum layer (a variational quantum circuit). "Filter" divides input points into 4 groups, where each group has its own constraint (a boundary condition or a PDE). The error of each constraint is calculated and added to the total loss, which is minimized via gradient descent. The training process is described in Sec. III A 1.

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

III. 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 equations (6) and the continuity equation (7) describe this scenario as follows:

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

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

where $v(r)$ is the velocity vector, $p(r)$ 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.

Our task is to simulate the flow of liquid in a Y-shaped mixer consisting of three tubes (Fig. 2). The mixing fluids are identical and have parameters $\rho = 1.0$ kg/m^3 and $\nu = 1.0$ m^2/s.

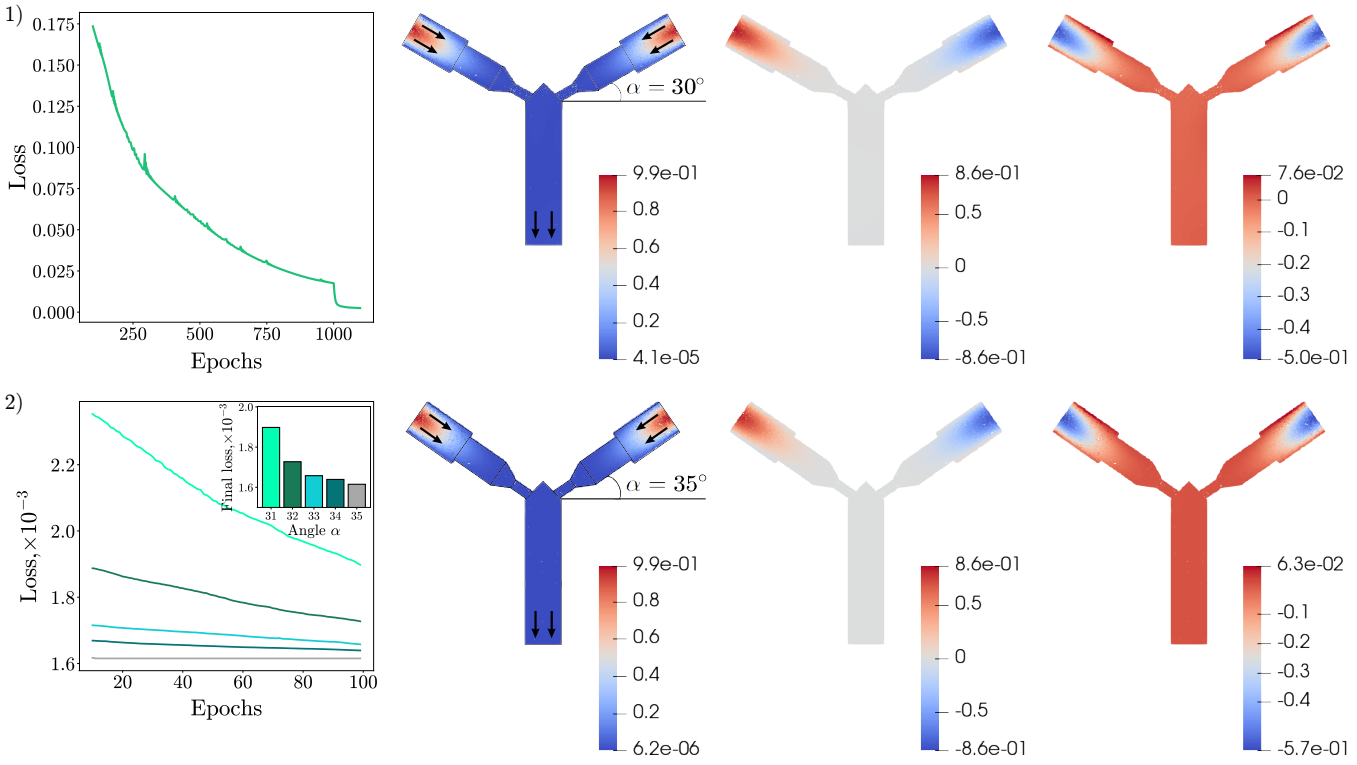
The imposed boundary conditions are as follows:

- no-slip BC on walls: $v(r)|_{\text{walls}} = 0$,
- fixed velocity profile on inlets: $v(r)|_{\text{inlets}} = v_0(r)$,
- fixed pressure on outlets: $p(r)|_{\text{outlets}} = p_0$,

where, in this work, $v_0(r)$ are parabolic velocity profiles on each inlet and $p_0(r) = 0$.

A. Classical PINN

In this section, we provide details on the PINN's architecture, training process, and simulation results. The PINN is a neural network whose architecture is a multilayer perceptron with several fully connected layers. As shown in Fig. 1, the first layer consists of 3 neurons (since



(a) Loss over epochs (b) Absolute velocity value $|\mathbf{v}|$, m/s (c) Velocity projection v_x, m/s (d) Velocity projection v_y, m/s

FIG. 2. 1) Classical model. (a) Loss curve. The classical PINN managed to learn 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, c, d) Distributions of the fluid velocity for the last model with $\alpha = 35°$.

the problem is 3D), then there are $l = 5$ hidden layers with $n = 64$ neurons, and the penultimate layer has 16 neurons. For the classical PINN, the "quantum layer" box is replaced with one fully-connected layer $16 \to 4$. 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 (SiLU) activation function [48]. The PINN takes the (x, y, z) coordinates as inputs and yields (v, p) as its output, where v is the velocity vector with three components and p is the pressure.

1. *Training*

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 Sec. II.

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 PINN is trained via full-batch gradient descent with the Adam optimizer [49] for 1000 epochs, and then with L-BFGS optimizer [50] for 100 epochs. All points from the geometrical domain are processed at once, so one step of the optimizer equals one epoch. The training iteration is simple: the point cloud is passed through the PINN, the MSE loss is calculated (getting it requires taking gradients of (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.

Classical PINN training (1100 total epochs) took 5 minutes on single NVIDIA A100 GPU. To visualize the training outcomes of the neural network, we used ParaView [51] and the simulation results are shown in Fig. 2 (1) for loss and velocity distribution.

After the training, the PINN manages 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 degener-

ates 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, but the model still treats this situation as an optimum and refuses to propagate the solution further. Possible reasons for this could be a mismatch between the scales of the different loss terms (continuity, NS and BC) in the total loss, the limited expressivity of the underlying neural network, the points sampling strategy or the choice of the optimizer. Still, we stick to our simple workflow and use this incomplete solution as a reference to further explore the properties of the PINN without overcomplicating the problem.

2. *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]. 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 the crucial issue of 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 Fig. 2). Then, for each $\alpha = \{31°, 32°, 33°, 34°, 35°\}$, we tried to obtain the solution, each time using the previously trained model as a 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. Fig. 2 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.

B. Hybrid Quantum PINN

Quantum machine learning may help to improve the performance of the classical PINN. We propose a quantum PINN, which is a hybrid quantum-classical neural network. In the Noisy Intermediate-Scale Quantum (NISQ) era, when quantum computers do not have access to a quantum memory or a large number of qubits, the algorithms that are known to be superior to classical ones cannot be implemented. Heurisitc quantum algorithms, however, have less stringent requirements for their implementation. Here, we use a variational quantum circuit (VQC) approach, which is one of the most promising NISQ algorithms today. It combines the best ideas of classical and quantum machine learning [25, 52–55].

At the moment, using only VQCs is not enough to solve large problems, especially for industrial tasks, so several more classical neural layers are used with a VQC (hence the name, "hybrid"). Also, the use of a large number of qubits and layer repetitions leads to the barren plateau problem [56, 57].

As an overview of the workflow, preliminary data processing takes place on a classical computer. This data is then encoded into the parameters of the quantum gates of an encoding layer of the VQC, followed by a variational layer. As the algorithm progresses, the gate parameters in the variational layer of the VQC are varied and finally, in the measurement layer of the VQC the qubits are measured, producing a set of classical bits as the output.

A quantum algorithm can be represented as a kind of black box into which classical information enters, classical information comes out. The goal is to choose such variational parameters so that the measurement result is as accurate as possible in terms of the prediction function. In this way, finding the optimal gate parameters is like finding the optimal weights in a classical neural network. Hence, we refer to this as training the VQC.

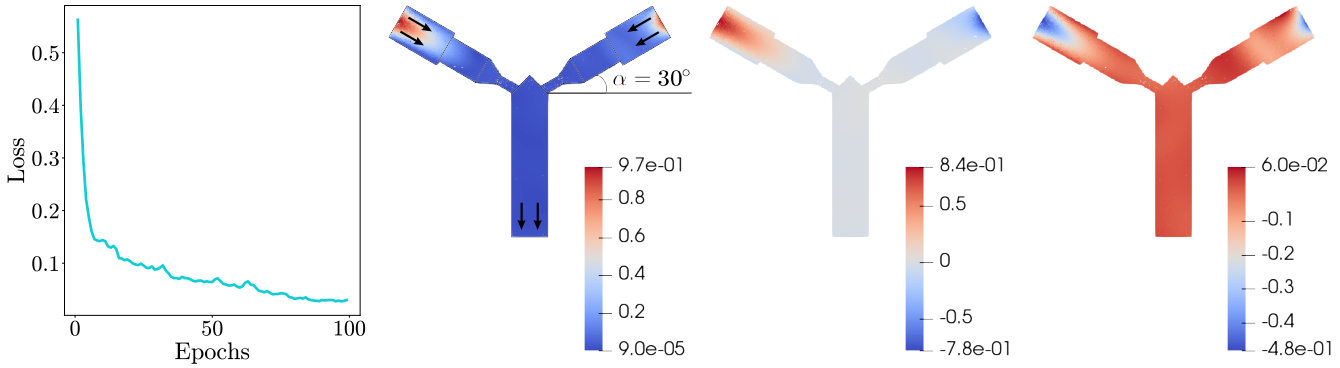
1. *Variational Quantum Circuit*

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 get the results, the qubits are measured (i.e., projected onto some basis) and the expected value is calculated. If we use the σ_z Pauli matrix observable, the expected value of that measurement is expressed as: $\langle\psi|\,\sigma_z\,|\psi\rangle$, where ψ is the wave function which describes the current state of our quantum system. An introduction to quantum circuits, including logic gates and measurements, can be found in standard quantum computing textbooks, such as [58].

2. *Training*

The hybrid PINN consists of the classical PINN with weights preinitialized from the previous stage (as described in Sec. III A), a VQC and a fully-connected layer at the end. The VQC's design is inspired by [59].



(a) Loss over epochs (b) Absolute velocity value $|\mathbf{v}|$, m/s (c) Velocity projection v_x, m/s (d) Velocity projection v_y, m/s

FIG. 3. Hybrid quantum model. (a) Loss curve. (b, c, d) Distribution of the fluid velocities for the hybrid PINN 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.

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 [60], which has recently been shown to be quite good for running hybrid algorithms [61].

Secondly, how does one backpropagate through the quantum circuit layer? The answer is to use the "adjoint differentiation" method, introduced in [62], which helps to efficiently compute derivatives of a VQC on a classical simulator.

This time the model was trained for 100 epochs using mini-batch gradient descent with the Adam optimizer (Fig. 3). The mini-batch strategy was employed due to the very low speed of training of quantum circuits, as they train on a CPU. We will then compare this model with a purely classical one, with exactly the same architecture from Sec. III A, 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 model. Comparing the two, Fig. 4 shows that the quantum model outperforms the classical one in terms of the loss value by 21%.

We tried to do inference of our model on real quantum processing units (QPUs) Lucy [63] and Rigetti's Aspen-M-3 [64]. That means, we have chosen some points from the geometrical domain and predicted velocity and pressure fields for them. On comparison, results from both QPUs greatly differ from the simulator ones. Most likely, these errors are caused by noise and decoherence, which are hard to mitigate. Also, particular QPUs work better with circuits that take into account their particular topology and gate implementations. We did not consider such limitations, as the primary tests of our quantum circuit were conducted on a simulator.

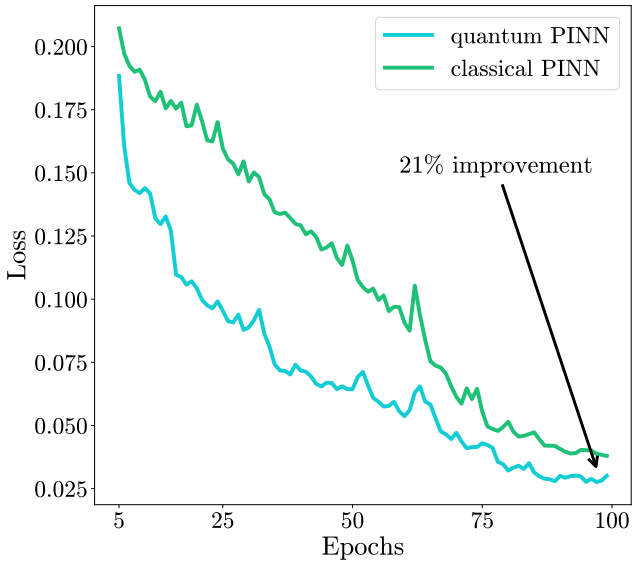


FIG. 4. Losses for the classical and hybrid quantum PINNs. Hybrid quantum PINN outperforms the classical PINN in terms of loss value due to higher expressivity.

IV. DISCUSSION

In this work we examine a recently introduced [10] way of solving computational fluid simulation tasks, PINNs. This approach is based on substituting the solution function of the problem with a neural network of an arbitrary architecture (a multi-layer perceptron is used in this work) and optimizing trainable parameters via gradient descent (Adam, L-BFGS). In this problem setting, the loss function becomes a combination of the residual error of corresponding PDE and boundary conditions. We try to use this approach to solve a laminar mixing problem in a 3D Y-shaped geometrical domain and achieve plausible solution in the upper part of the mixer. The solution,

however, vanishes to zero as it goes lower through the mixer. The reason behind such behaviour could be insufficient expressivity of the model, static point sampling or even the choice of the optimizers.

We then explore a transfer learning technique with application to a classical PINN, which shows that the PINN is actually capable of generalizing an existing solution to a weakly changed geometrical domain with little additional training. This enables PINNs to be very helpful in shape optimization tasks because traditional solvers lack this feature and demand a full restart on any parameter change. Also, with each step of the angle α, we see that the PINN achieves an even lower loss than before. It may be that this step-by-step learning helps the PINN to mitigate overfitting to a particular geometrical domain of the problem.

Afterwards, we introduce a new PINN framework – a quantum PINN. We build it on top of a classical fully-conntected PINN by adding a quantum circuit with trainable gates. We explore the potential of this model to solve the NS equations and show its superiority to purely classical network in terms of the loss value on 21%, which could mean that the chosen quantum circuit increases the PINN's expressiveness.

The future plan includes exploring better architectures for the quantum PINN, investigating their impact on expressiveness, generalizability and optimization landscape, as well as trying out data-driven approaches. Entirely different networks, such as neural operators [65, 66] and graph neural networks [67, 68], could also be considered in a quantum setting and enhanced with quantum circuits.

[1] Mohd Hafiz Zawawi, A Saleha, A Salwa, NH Hassan, Nazirul Mubin Zahari, Mohd Zakwan Ramli, and Zakaria Che Muda. A review: Fundamentals of computational fluid dynamics (CFD). In *AIP conference proceedings*. AIP Publishing LLC, 2018.

[2] John David Anderson and John Wendt. *Computational fluid dynamics*, volume 206. Springer, 1995.

[3] George Finlay Simmons. Differential equations with applications and historical notes. In *Differential Equations With Applications and Historical Notes*. McGraw-Hill, 1972.

[4] Aaron Jon Katz. *Meshless methods for computational fluid dynamics*. Stanford University, 2009.

[5] OpenFOAM. https://www.openfoam.com/, 2022.

[6] Ansys. https://www.ansys.com/, 2022.

[7] Tomislav Marić, Douglas B. Kotheb, and Dieter Bothe. Unstructured un-split geometrical volume-of-fluid methods - A review. *Journal of Computational Physics*, 420, 2020.

[8] Tomislav Marić, Holger Marschall, and Dieter Bothe. vofoam - a geometrical volume of fluid algorithm on arbitrary unstructured meshes with local dynamic adaptive mesh refinement using OpenFOAM. *arXiv preprint arXiv: 1305.3417*, 2013.

[9] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, pages 1–12, 2022.

[10] 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.

[11] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *arXiv preprint arXiv:2008.11687*, 2020.

[12] Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, and Tuan Anh Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *SN partial differential equations and applications*, 1:1–34, 2020.

[13] Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe Von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. *arXiv preprint arXiv:1809.02362*, 2018.

[14] Michael Stein. Large sample properties of simulations using latin hypercube sampling. *Technometrics*, 29(2):143–151, 1987.

[15] Ilya M Sobol', Danil Asotsky, Alexander Kreinin, and Sergei Kucherenko. Construction and comparison of high-dimensional Sobol'generators. *Wilmott*, 2011(56):64–79, 2011.

[16] Kirill Zubov, Zoe McCarthy, Yingbo Ma, Francesco Calisto, Valerio Pagliarino, Simone Azeglio, Luca Bottero, Emmanuel Luján, Valentin Sulzer, Ashutosh Bharambe, et al. Neuralpde: Automating physics-informed neural networks (PINNs) with error approximations. *arXiv preprint arXiv:2107.09443*, 2021.

[17] Frank Gaitan. Finding flows of a Navier–Stokes fluid through quantum computing. *npj Quantum Information*, 6(1):1–6, 2020.

[18] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 81(7):074001, 2018.

[19] Alexey Melnikov, Mohammad Kordzanganeh, Alexander Alodjants, and Ray-Kuang Lee. Quantum machine learning: from physics to software engineering. *Advances in Physics: X*, 8(1):2165452, 2023.

[20] Hartmut Neven, Vasil S. Denchev, Geordie Rose, and William G. Macready. QBoost: Large scale classifier training with adiabatic quantum optimization. In Steven C. H. Hoi and Wray Buntine, editors, *Proc. Asian Conf. Mach. Learn.*, volume 25 of *Proceedings of Machine Learning Research*, pages 333–348. PMLR, 2012.

[21] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical Review Letters*, 113:130503, Sep 2014.

[22] Valeria Saggio, Beate E Asenbeck, Arne Hamann, Teodor Strömberg, Peter Schiansky, Vedran Dunjko, Nicolai Friis, Nicholas C Harris, Michael Hochberg, Dirk Englund, et al. Experimental quantum speed-up in rein-

forcement learning agents. *Nature*, 591(7849):229–233, 2021.

[23] Mohammad Kordzanganeh, Daria Kosichkina, and Alexey Melnikov. Parallel hybrid networks: an interplay between quantum and classical neural networks. *arXiv preprint arXiv:2303.03227*, 2023.

[24] Michael Perelshtein, Asel Sagingalieva, Karan Pinto, Vishal Shete, Alexey Pakhomchik, Artem Melnikov, Florian Neukart, Georg Gesek, Alexey Melnikov, and Valerii Vinokur. Practical application-specific advantage through hybrid quantum computing. *arXiv preprint arXiv:2205.04858*, 2022.

[25] Asel Sagingalieva, Mohammad Kordzanganeh, Nurbolat Kenbayev, Daria Kosichkina, Tatiana Tomashuk, and Alexey Melnikov. Hybrid quantum neural network for drug response prediction. *arXiv preprint arXiv:2211.05777*, 2022.

[26] A. I. Gircha, A. S. Boev, K. Avchaciov, P. O. Fedichev, and A. K. Fedorov. Training a discrete variational autoencoder for generative chemistry and drug design on a quantum annealer. *arXiv:2108.11644*, 2021.

[27] Serge Rainjonneau, Igor Tokarev, Sergei Iudin, Saaketh Rayaprolu, Karan Pinto, Daria Lemtiuzhnikova, Miras Koblan, Egor Barashov, Mohammad Kordzanganeh, Markus Pflitsch, et al. Quantum algorithms applied to satellite mission planning for Earth observation. *arXiv preprint arXiv:2302.07181*, 2023.

[28] Asel Sagingalieva, Andrii Kurkin, Artem Melnikov, Daniil Kuhmistrov, Michael Perelshtein, Alexey Melnikov, Andrea Skolik, and David Von Dollen. Hyperparameter optimization of hybrid quantum neural networks for car classification. *arXiv preprint arXiv:2205.04878*, 2022.

[29] Javier Alcazar, Vicente Leyton-Ortega, and Alejandro Perdomo-Ortiz. Classical versus quantum models in machine learning: insights from a finance application. *Machine Learning: Science and Technology*, 1(3):035003, 2020.

[30] Brian Coyle, Maxwell Henderson, Justin Chan Jin Le, Niraj Kumar, Marco Paini, and Elham Kashefi. Quantum versus classical generative modelling in finance. *Quantum Science and Technology*, 6(2):024013, 2021.

[31] Marco Pistoia, Syed Farhan Ahmad, Akshay Ajagekar, Alexander Buts, Shouvanik Chakrabarti, Dylan Herman, Shaohan Hu, Andrew Jena, Pierre Minssen, Pradeep Niroula, Arthur Rattew, Yue Sun, and Romina Yalovetzky. Quantum Machine Learning for Finance. *arXiv preprint arXiv:2109.04298*, 2021.

[32] Dimitrios Emmanoulopoulos and Sofija Dimoska. Quantum Machine Learning in Finance: Time Series Forecasting. *arXiv preprint arXiv:2202.00599*, 2022.

[33] El Amine Cherrat, Snehal Raj, Iordanis Kerenidis, Abhishek Shekhar, Ben Wood, Jon Dee, Shouvanik Chakrabarti, Richard Chen, Dylan Herman, Shaohan Hu, Pierre Minssen, Ruslan Shaydulin, Yue Sun, Romina Yalovetzky, and Marco Pistoia. Quantum Deep Hedging. *arXiv preprint arXiv:2303.16585*, 2023.

[34] Arsenii Senokosov, Alexander Sedykh, Asel Sagingalieva, and Alexey Melnikov. Quantum machine learning for image classification. *arXiv preprint arXiv:2304.09224*, 2023.

[35] Wei Li, Peng-Cheng Chu, Guang-Zhe Liu, Yan-Bing Tian, Tian-Hui Qiu, and Shu-Mei Wang. An Image Classification Algorithm Based on Hybrid Quantum Classical Convolutional Neural Network. *Quantum Engineering*, 2022:1–9, 2022.

[36] A Naumov, Ar Melnikov, V Abronin, F Oxanichenko, K Izmailov, M Pflitsch, A Melnikov, and M Perelshtein. Tetra-AML: Automatic machine learning via tensor networks. *arXiv preprint arXiv:2303.16214*, 2023.

[37] Farina Riaz, Shahab Abdulla, Hajime Suzuki, Srinjoy Ganguly, Ravinesh C. Deo, and Susan Hopkins. Accurate Image Multi-Class Classification Neural Network Model with Quantum Entanglement Approach. *Sensors*, 23(5):2753, 2023.

[38] Zhenhou Hong, Jianzong Wang, Xiaoyang Qu, Chendong Zhao, Wei Tao, and Jing Xiao. QSpeech: Low-Qubit Quantum Speech Application Toolkit. *arXiv preprint arXiv:2205.13221*, 2022.

[39] Robin Lorenz, Anna Pearson, Konstantinos Meichanetzidis, Dimitri Kartsaklis, and Bob Coecke. QNLP in Practice: Running Compositional Models of Meaning on a Quantum Computer. *arXiv preprint arXiv:2102.12846*, 2021.

[40] Bob Coecke, Giovanni de Felice, Konstantinos Meichanetzidis, and Alexis Toumi. Foundations for Near-Term Quantum Natural Language Processing. *arXiv preprint arXiv:2012.03755*, 2020.

[41] Konstantinos Meichanetzidis, Alexis Toumi, Giovanni de Felice, and Bob Coecke. Grammar-Aware Question-Answering on Quantum Computers. *arXiv preprint arXiv:2012.03756*, 2020.

[42] Oleksandr Kyriienko, Annie E Paine, and Vincent E Elfving. Solving nonlinear differential equations with differentiable quantum circuits. *Physical Review A*, 103(5):052416, 2021.

[43] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[44] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv: 1609.04747*, 2016.

[45] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[46] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert L. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[47] C Greenshields and H Weller. Notes on computational fluid dynamics: General principles. *CFD Direct Ltd.: Reading, UK*, 2022.

[48] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018.

[49] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[50] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[51] ParaView. https://www.paraview.org/, 2022.

[52] Andrea Mari, Thomas R. Bromley, Josh Izaac, Maria Schuld, and Nathan Killoran. Transfer learning in hybrid classical-quantum neural networks. *Quantum*, 4:340, 2020.

[53] Chen Zhao and Xiao-Shan Gao. QDNN: DNN

with quantum neural network layers. *arXiv preprint arXiv:1912.12660*, 2019.

[54] Tong Dou, Kaiwei Wang, Zhenwei Zhou, Shilu Yan, and Wei Cui. An unsupervised feature learning for quantum-classical convolutional network with applications to fault detection. In *2021 40th Chinese Control Conference (CCC)*, pages 6351–6355. IEEE, 2021.

[55] Mohammad Kordzanganeh, Pavel Sekatski, Leonid Fedichkin, and Alexey Melnikov. An exponentially-growing family of universal quantum circuits. *arXiv preprint arXiv:2212.00736*, 2022.

[56] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1):4812, 2018.

[57] Marco Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1):1–12, 2021.

[58] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[59] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Physical Review A*, 103(3):032430, 2021.

[60] QMware. https://qm-ware.com/, 2022.

[61] Mohammad Kordzanganeh, Markus Buchberger, Maxim Povolotskii, Wilhelm Fischer, Andrii Kurkin, Wilfrid Somogyi, Asel Sagingalieva, Markus Pflitsch, and Alexey Melnikov. Benchmarking simulated and physical quantum processing units using quantum and hybrid algorithms. *arXiv preprint arXiv:2211.15631*, 2022.

[62] Tyson Jones and Julien Gacon. Efficient calculation of gradients in classical simulations of variational quantum algorithms. *arXiv preprint arXiv:2009.02823*, 2020.

[63] Lucy. https://oxfordquantumcircuits.com/, 2022.

[64] Rigetti. https://qcs.rigetti.com/, 2022.

[65] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, et al. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

[66] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, et al. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[67] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

[68] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.