CEAS Aeronautical Journal
https://doi.org/10.1007/s13272-024-00774-2

ORIGINAL PAPER

Solving transport equations on quantum computers—potential and limitations of physics-informed quantum circuits

Pia Siegl[1] · Simon Wassing[2] · Dirk Markus Mieth[1] · Stefan Langer[2] · Philipp Bekemeyer[2]

Received: 16 February 2024 / Revised: 30 August 2024 / Accepted: 16 September 2024

Abstract
Quantum circuits with trainable parameters, paired with classical optimization routines can be used as machine learning models. The recently popularized physics-informed neural network (PINN) approach is a machine learning algorithm that solves differential equations by incorporating them into a loss function. Being a mesh-free method, it is a promising approach for computational fluid dynamics. The question arises whether the properties of quantum circuits can be leveraged for a quantum physics-informed machine learning model. In this study, we compare the classical PINN-ansatz and its quantum analog, which we name the physics-informed quantum circuit (PIQC). The PIQC simulations are performed on a noise-free quantum computing simulator. Studying various differential equations, we compare expressivity, accuracy and convergence properties. We find that one-dimensional problems, such as the linear transport of a Gaussian-pulse or Burgers' equation, allow a successful approximation with the classical and the quantum ansatz. For these examples, the PIQC overall performs similarly to PINN and converges more consistently and for Burgers' equations even faster. While this is promising, the chosen quantum circuit approach struggles to approximate discontinuous solutions which the classical PINN-ansatz can represent. Based on this comparison, we extrapolate that the required number of qubits for solving two-dimensional problems in aerodynamics may already be available in the next few years. However, the acceleration potential is currently unclear and challenges like noisy circuits and approximations of discontinuous solutions have to be overcome.

Keywords Quantum machine learning · Physics informed neural network · Variational quantum circuit · Differential equations

1 Introduction

Computational fluid dynamics (CFD) is an essential tool for many applications in science and industry. Conventional methods, such as finite volume and finite element, rely on the discretization of time and space and require an increasing number of degrees of freedom for high Reynolds number flows. To this day, the application of scale resolving simulations (like direct numerical simulation and large eddy simulation) is infeasible in an industrial aerospace context due to the billions of points that are required to obtain accurate physical models of turbulent flows [1]. Quantum computing is a potentially disruptive technology that may be able to accelerate previously expensive computational tasks, as solving non-linear differential equations [2–5].

In this context, it is crucial to develop and assess novel simulation algorithms which are compatible with faulty quantum hardware. To make a step in this direction, in this work, we consider one approach, namely the physics-informed quantum circuits and draw the comparison to its classical variant, the physics informed neural network.

1.1 Physics-informed neural networks

Supervised deep learning has become ubiquitous for applications, such as computer vision and natural language processing. In computational physics, however, purely data-driven models are oftentimes unsuitable because they may not respect physical laws, e.g. due to noisy data. Ideally, we

Pia Siegl, Simon Wassing have contributed equally to this work.

✉ Pia Siegl
pia.siegl@dlr.de

1 Institute of Software Methods for Product Virtualization, German Aerospace Center (DLR), Zwickauerstraße 46, 01069 Dresden, Germany

2 Institute for Aerodynamics and Flow Technology, German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Braunschweig, Germany

want to leverage the physical knowledge available in the form of partial differential equations (PDEs), to enhance a machine learning model. One method to reach this goal, is the physics-informed neural network (PINN). This method approximates the solution to a differential equations with a neural network. The differential equation is incorporated into a loss function used for training and the required derivatives are calculated using automatic differentiation. The initial and boundary conditions are typically incorporated into the loss function via additional loss terms. Therefore, PINNs reformulate boundary value problems as optimization problems. Here, the neural network acts as a global continuous ansatz for the solution which distinguishes the method from classical methods, based on discretization in space and time. PINNs can be used as a deep learning-based PDE solver for forward problems. This is the purpose for which they are used in this work. The training of PINNs is typically significantly slower than solving the PDE with a classical solver. However, contrary to classical solvers, they offer the advantage that additional solution data (e.g. from measurements or other simulations) can be incorporated into the loss. This makes them applicable to tackle certain inverse problems [6–8]. Furthermore, they may be used to solve parametric forward problems by approximating the solution in a continuous parameter space. Once the network is trained, it can be evaluated quickly and solutions for different parameter values can be retrieved. Originally proposed in the 90s by works of Dissanayake and Phan-Thien [9] and later Lagaris [10], the approach has recently gained substantial interest after a series of papers by Raissi et al. [6, 11, 12] where the method is applied to a variety of forward and inverse problems, coining the term *physics-informed neural network*. Subsequently, numerous variants and extensions have been proposed and the method was shown to be effective in a wide variety of scenarios. For a thorough overview the interested reader is referred to [13]. In particular, PINNs have been applied to fluid dynamics [14]. So far, the complexity of analyzed problems is limited to simple, two-dimensional flows. However, initial steps towards higher complexity problems with relevance for aerospace science are made. In the following, some notable examples are listed. Jin et al. [15] solve the incompressible Navier-Stokes equations for laminar flows and turbulent channel flows at Reynolds number $Re = 1000$. Eivazi et al. [16] solve the incompressible Reynolds-averaged-Navier-Stokes equations for zero and adverse pressure gradient boundary layers and a NACA 0012 airfoil. Raissi et al. [8] reconstruct full flow fields for Navier-Stokes equations based on noisy measurements of a passive scalar which is advected by the flow. The compressible Euler equations have been considered for supersonic two-dimensional cases such as an oblique shock wave or expansion wave for forward [17] and inverse problems [7, 17]. Wassing et al. [18] show parametric PINN simulations of the compressible Euler equations for sub- and supersonic flows in two dimensions.

1.2 Variational quantum algorithms

Next to machine learning, quantum computing is a further promising future technology. In the context of fluid dynamics we often face the challenge of non-linearity of the governing equations and non-hermicity of diffusion [19] hindering a straight forward implementation in terms of quantum gates. However, various ideas on how to tackle these problems were developed and are currently investigated. We will shortly outline some key resources. For a more detailed discussion, we refer the interested reader to [19]. Many legacy CFD methods, like finite volumes or finite elements, rely on a discretization of the spatial fields allowing for a linearization of the equations. This linear formulation enables the use of quantum linear solvers like the Harrow-Hassidim-Lloyd algorithm [20] and its improvements [21, 22]. With this techniques the scaling of a quantum finite element routine [23] was studied and a finite difference scheme solving the wave equation [24] implemented. Moreover, Lattice–Boltzmann routines are under active consideration [25–27] and Carleman-Linearization seems to be a possible tool [28]. Furthermore, quantum computers seem to be well suited for probability density function methods and ensemble simulations [29] as those require linear differential equations to be solved.

Most of the aforementioned approaches require a large number of qubits and error-corrected operations. Hence, they are not yet applicable for real use-cases on quantum computing hardware which is still prone to noise and limited in the number of available qubits and connectivity [30]. First doubts if the mentioned approaches designed for fault-tolerant computers allow for quantum advantages are raised [31].

In this early stage of quantum computing, promising candidates are variational quantum algorithms (VQA) that combine trainable quantum circuits with classical optimization routines [32–35]. Following the variational approach, Syamlal et al. published a preprint with first evidence for beneficial scaling of a variational quantum lattice boltzmann method [36] for CFD applications. Quantum Reservoir Computing is under consideration to capture thermal convective [37] and turbulent flows [38]. Further works introduced VQAs that focus on solving linear [39, 40] and non-linear [3, 4] differential equations. In this context, Lubasch et al. introduced a variational approach using amplitude encoding and finite differences that allows to solve non-linear differential equations. The method relies on the paradigm of tensor networks, which showed to be suitable for flow simulations [41, 42]. This approach was extended to treat different boundary conditions [43] and space-time methods [44]. VQAs are further an important building block for quantum machine learning (QML) algorithms and researchers study the quantum potential in nearly all areas of classical machine learning (ML). For example, there are experimental studies on quantum convolutional ML [45] and quantum reinforcement learning, as well as theoretical work, trying to identify generalization bounds [46] and quantify quantum circuit expressivity [47–49].

Kyriienko et al. proposed the quantum version [4] of the physics-informed neural network algorithm [12] for solving PDEs. They show the general feasibility to solve one-dimensional non-linear differential equations by training the parameters of a quantum circuit with a loss function that incorporates the differential equations. To this aim, they compare different handling strategies of the input encoding and the boundary treatment. Using a hardware-efficient ansatz together with different feature maps, they study the performance on two different differential equations with continuous solutions. In following works, they expand their understanding of the feature maps and their application to stochastic PDEs [50]. While they show the general ability of the quantum approach in [4], they do not perform any comparison to classical PINNs and they leave the question of current limits of the quantum approach open.

Our work performs a direct comparison of the classical and quantum approach, for different relevant PDEs of different complexity. We use a different circuit ansatz that relies on data reuploading [51] as it allows us to solve a broader variety of problems. In analogy to PINNs, we name the quantum approach physics-informed quantum circuit (PIQC) throughout this work. While our PIQC-ansatz is able to approximate continuous solutions to all considered differential equations, both the ansatz presented in [4, 51] fail to capture discontinuous solutions. Our work illustrates when PIQCs show a comparable or even beneficial behaviour over classical PINNs. Furthermore, it pin-points that the presented ansatzes are incapable of finding flow solutions with discontinuities, an important phenomena in many industry-relevant flow applications. Many studies that claim advantage of QML and classical ML are currently criticized [52, 53] for using data which is down-sampled beyond recognition and choosing particularly bad-performing architectures of the classical neural networks. The focus of our work is an empirical study as well. However, we choose our problems as broad as possible to allow for a fair comparison. We further use neural network architectures, that work well for PINNs and do not hide results that are not beneficial for the PIQC.

During the publishing process of this work, Paine et al. released a preprint on physics-informed quantum machine learning, with a different procedure of information encoding [54].

1.3 Scope and structure of this work

This work is devoted to the evaluation and comparison between PINNs and PIQC. We share experimental experiences comparing the convergence speed and accuracy. Furthermore, current limitations are discussed, occurring due to the missing availability of quantum computers with large qubit numbers and the infeasibility of simulating them. This paper is structured as follows. First, we will give an introduction into physics-informed neural networks and variational quantum algorithms. Next, we will present the differential equations studied. First an ordinary differential equation (ODE), second a linear transport equations, and third a non-linear Burgers' equations. The last problem, i.e. the transport of a shock, is only well captured by the PINN, showcasing the limitations of the currently used quantum circuit ansatz.

2 Physics-informed neural networks

Physics-informed neural networks are a deep learning-based method using classical neural networks for differential problems such as solving partial differential equations. PINNs incorporate differential terms into the loss function which is minimized during the training of the neural network.

Let us consider a one-dimensional initial-boundary value problem:

$$\begin{aligned}\frac{\partial u}{\partial t}-\mathcal{D}(u) &=0 \quad (x,t)\in\Omega\times(0,T)\\ \mathcal{I}(u(x,0),x) &=0 \quad x\in\Omega\\ \mathcal{B}(u(x,t),x,t) &=0 \quad (x,t)\in\partial\Omega\times(0,T)\end{aligned} \tag{1}$$

for an unknown solution $u(x,t)$ in space $x\in\Omega$ and in the time interval $t\in[0,T)$. Here, $\mathcal{D}$ is a general differential operator and $\mathcal{I}$ and $\mathcal{B}$ are initial and boundary conditions, respectively. To approximate a solution $u(x,t)$ with a PINN, we use a neural network $\hat{u}_\theta(x,t)$ as the global ansatz function $\hat{u}_\theta(x,t)\approx u(x,t)$. The neural network acts as a parametric ansatz function for the solution. Since neural networks are universal function approximators [55], a sufficiently large network is theoretically able to capture continuous solutions at arbitrary precision. Empirical result show that even discontinuous functions can be approximated [12]. However, to find such good approximations, one has to find a global optimum of a non-convex optimization problem to determine the parameters of the network, which in general can be an NP-hard problem [56].

A fully connected feed-forward neural network is obtained by a repeated composition of parametric vector-valued linear functions and non-linear activation functions. It can be expressed as follows:

$$
\begin{aligned}
&\hat{u}(\boldsymbol{r}) : \mathbb{R}^{N_0} \longrightarrow \mathbb{R}^{N_M} \\
&\hat{u}_j \equiv x_j^M, \quad j = 1 \dots N_M \\
&\boldsymbol{r} = \left(x_1^0, x_2^0, \dots, x_{N_0}^0\right) \\
&x_j^k = \sigma\left(\sum_{i=1}^{N_{k-1}} w_{i,j}^{k-1} x_i^{k-1} - b_j\right), \\
&j = 1 \dots N_k, \quad k = 1 \dots M.
\end{aligned} \tag{2}
$$

Each composition of one parametric linear function and the non-linear activation function $\sigma(\cdot)$ is called a layer. N_k is the dimension of the k-th layer. The input vector $\boldsymbol{r} = (x_1^0, x_2^0, \dots, x_{N_0}^0)$ and the output vector x_j^M determine the dimensions of the domain and codomain of the network. All other layers are called *hidden* layers and the network has a depth of M layers. The values of each layer k only depends on the function values of the previous layer $k-1$ and w and b are the parameters of the network, called weights and biases. Since this architecture is inspired by brain neurons, the individual outputs x_j^k at each layer are often called *neurons*. For an in-depth explanation of classical neural network architectures, consider for example [57]. For physics-informed neural networks, a typical activation function is the hyperbolic tangent $\sigma(\cdot) = \tanh(\cdot)$.

The optimization or training of the network weights and biases requires the construction of a loss function $\mathcal{L}$. For regression problems, this loss is most commonly the mean squared error between network predictions and target data. However, for physics-informed neural networks, we construct a composite loss function which consists of three terms:

$$
\mathcal{L} = \mathcal{L}_{\mathrm{Res}} + \lambda_{\mathrm{I}}\mathcal{L}_{\mathrm{I}} + \lambda_{\mathrm{B}}\mathcal{L}_{\mathrm{B}}. \tag{3}
$$

The first term on the right hand side is the residual loss. It directly incorporates the differential equation and its magnitude measures the residual of the predicted solution,

$$
\begin{aligned}
\mathcal{L}_{\mathrm{Res}} = \frac{1}{N_{\mathrm{Res}}} \sum_{i=1}^{N_{\mathrm{Res}}} \left(\frac{\partial \hat{u}_\theta}{\partial t}(x_i, t_i) - \mathcal{D}(\hat{u}_\theta(x_i, t_i)) \right)^2, \\
x_i \in \Omega \,; t_i \in (0, T).
\end{aligned} \tag{4}
$$

Backpropagation methods use reverse mode automatic differentiation to calculate derivatives of a loss function $\mathcal{L}$ with respect to parameters of the network. Similarly, we can use reverse mode automatic differentiation to calculate derivatives of the network output $\hat{u}_\theta$ with respect to its input $\boldsymbol{r}$. This essentially exploits the chain rule of derivatives in a hardware-efficient manner to calculate the derivatives of a particular evaluation of the network. Firstly, the network is evaluated and the prediction $\hat{u}_\theta$ is obtained. The operations, performed on the inputs by the neural network to obtain the prediction, are stored in a computational graph. Subsequently, by stepping back through the network from the last to the first layer, we can calculate the desired derivatives by iteratively applying the chain rule, one layer at a time using the computational graph. For more details, see for example [57]. In practice, we can use third-party libraries such as PyTorch [58] to implement these routines in a straightforward manner.

The second and third term of the loss function penalize differences between the predicted solution and the given initial condition and boundary condition, respectively,

$$
\mathcal{L}_{\mathrm{I}} = \frac{1}{N_{\mathrm{I}}} \sum_{i=1}^{N_{\mathrm{I}}} \mathcal{I}(\hat{u}_\theta(x_i, 0))^2, \quad x_i \in \Omega, \tag{5}
$$

$$
\begin{aligned}
\mathcal{L}_{\mathrm{B}} = \frac{1}{N_{\mathrm{B}}} \sum_{i=1}^{N_{\mathrm{B}}} \mathcal{B}(\hat{u}_\theta(x_i, t_i))^2, \\
x_i \in \partial\Omega \,; t_i \in (0, T).
\end{aligned} \tag{6}
$$

Note that these loss terms are required to avoid convergence to a trivial solution of the differential equation. For all following experiments, we set the scalar weighting factors $\lambda_{\mathrm{I}} = \lambda_{\mathrm{B}} = 1$. The training points (x_i, t_i) inside of the domain and on the boundary can be randomly sampled. Thus, we can train the model using only information of the solution given by the boundary conditions and without additional solution data in the interior of the domain. Using the loss function Eq. (3) we can formulate the optimization problem for the trainable parameters as:

$$
\arg \min_{\theta \in \mathbb{R}^{n_{\mathrm{p}}}} \left(\mathcal{L}(\hat{u}_\theta)\right), \tag{7}
$$

where $\boldsymbol{\theta}$ is the vector containing all trainable parameters of the neural network (weights and biases in Eq. (2)) and n_{p} is the number of parameters in the network. A schematic explanation of the PINN method is shown in Fig. 1.

3 Variational quantum algorithms

A variational quantum algorithm (VQA) is a hybrid quantum-classical approach that consists of a quantum circuit with trainable parameters and an optimization routine that runs on a classical computer.

In the following, we will shortly deepen the idea of this hybrid quantum-classical approach.

The VQA can be split into a quantum part and a classical part. Let us first consider the classical neural network-based approach in Fig. 1. For the VQA approach, the neural network in this picture is replaced by a quantum circuits which also acts as the global ansatz function. We use the measurement results of the quantum circuit to define the function value u. The computation of the loss, as well as the

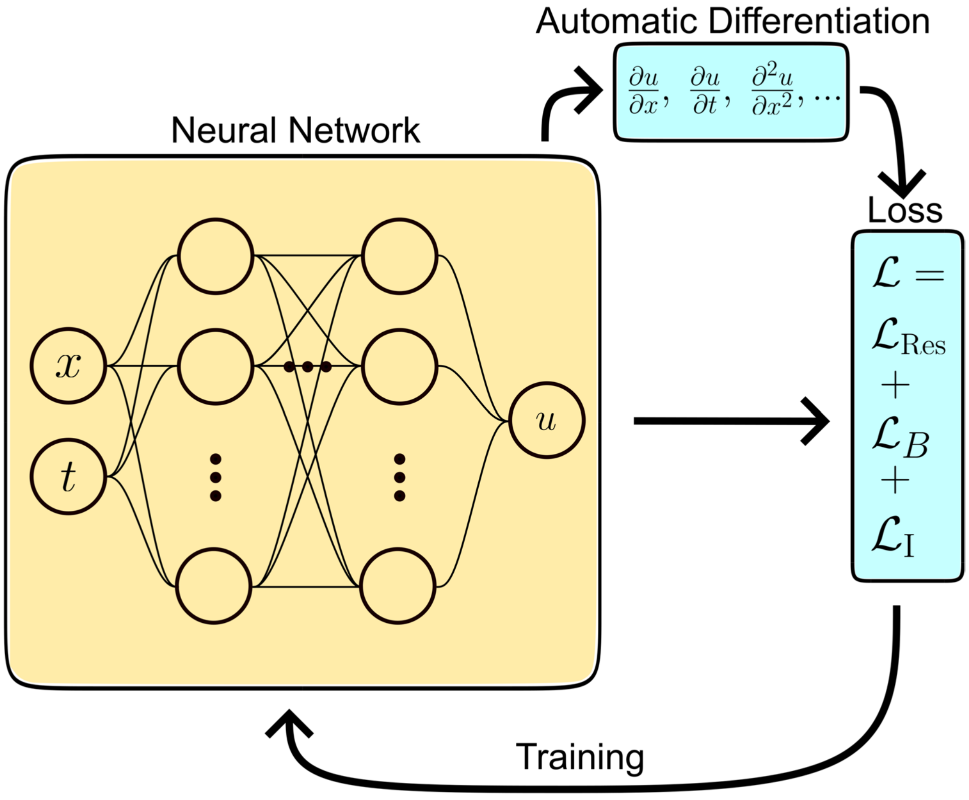
Automatic Differentiation
$\frac{\partial u}{\partial x}, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial x^2}, \dots$
Neural Network
x
t
u
Loss
$\mathcal{L} = \mathcal{L}_{\mathrm{Res}} + \mathcal{L}_B + \mathcal{L}_{\mathrm{I}}$
Training

Fig. 1 Schematic depiction of the PINN approach. The neural network is trained using a composite loss function which measures the agreement of the neural network with the residual and initial and boundary conditions. Partial derivatives are calculated using automatic differentiation

optimization routine remain on the classical computer and work equivalently to the classical PINNs. To find the optimal set of trainable parameters for the PINN and PIQC, we rely on gradient-based optimization. To make use of these routines, the quantum circuit needs to be differentiable so that gradients of the loss with respect to the trainable parameters inside of the circuit can be calculated.

For all gates and measurements used throughout this work, it is well known how to compute the derivatives of the quantum circuit. On real quantum hardware, this is possible, with the help of the parameter-shift rule [32, 59] and its generalizations [60, 61]. In this work, using noise-free simulators of quantum hardware, the gradients were computed analytically. The differentiability of the circuit is further required to compute the residual needed for the loss function in Eq. (3) which in our examples includes derivatives of the form $\frac{\partial u}{\partial x}$, $\frac{\partial u}{\partial t}$ and $\frac{\partial^2 u}{\partial x^2}$. The used quantum circuits consist of three main building block (see Fig. 2). First, there is an encoding layer for feeding in the inputs, second, the trainable layer aimed to be optimized, and last, the measurement layer, necessary to extract the output-data from the quantum computer. For simplicity, we will introduce each of these building blocks separately. In practice, they can occur multiple times.

3.1 Rotational gates

Rotational gates play an important role for data encoding, as well as for the trainable layers of the VQC. They are parameterized single-qubit operations that implement a rotation of the single-qubit state on the Bloch sphere [62]. The rotational gates, which are unitary operations and implement a rotation around the respective axis, can be expressed in matrix representation as

$$\begin{aligned} R_x(\alpha) &= \begin{pmatrix} \cos\alpha/2 & -i\sin\alpha/2 \\ -i\sin\alpha/2 & \cos\alpha/2 \end{pmatrix}, \\ R_y(\alpha) &= \begin{pmatrix} \cos\alpha/2 & -\sin\alpha/2 \\ \sin\alpha/2 & \cos\alpha/2 \end{pmatrix}, \\ R_z(\alpha) &= \begin{pmatrix} \exp -i\alpha/2 & 0 \\ 0 & \exp i\alpha/2 \end{pmatrix}. \end{aligned} \tag{8}$$

The angle $\alpha \in \mathbb{R}$ can take the value of an input datum or act as a trainable parameter.

3.2 Data encoding

In general, there are various possibilities for how to encode information into a gate-based quantum computer. Here, we will restrict ourselves to the rotational encoding. A detailed description of other important encoding schemes as bit-encoding or amplitude encoding can be found in [63]. We

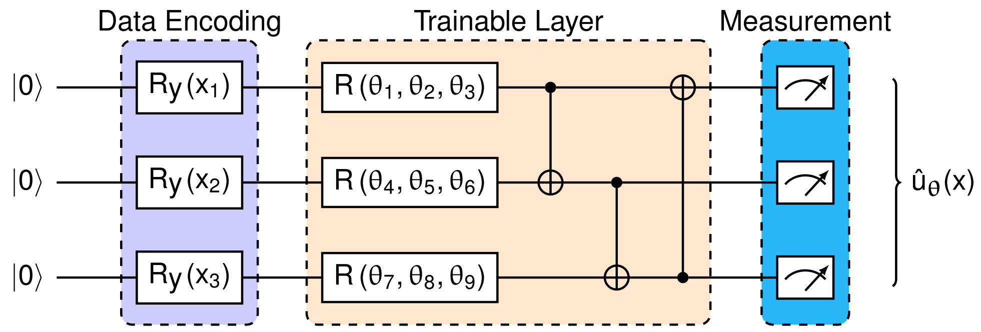

Fig. 2 Example of a variational quantum circuit with three qubits. It consists of three main parts. First, the encoding (purple), where input information $\boldsymbol{x}$ is encoded via rotation gates. Second, the trainable layer, where rotation gates contain trainable parameters $\boldsymbol{\theta}$. Their values are to be determined by solving the optimization problem Eq. (7). The trainable layer further contains two-qubit gates that create entanglement. Third, there is the measurement procedure. Here, we consider a Pauli measurement in the z-basis on each qubit. The measured results are summed up to the model approximation $\hat{u}_{\theta(x)}$

use the rotation gates to encode one real number per qubit and gate. For example, applying $R_y(x)$ on the state $|0\rangle$ yields

$$R_y(x)|0\rangle = \cos(x/2)|0\rangle - \sin(x/2)|1\rangle. \tag{9}$$

This allows for an efficient data-encoding without expensive state preparation routines. In combination with Pauli-measurements it further ensures the differentiability of the quantum circuit. The rotational encoding is well suited for problems with small input dimension.

3.3 Trainable layers

Rotation gates are also used in the trainable layer, shown in Fig. 2. Here, the gate $R(\theta_i, \theta_j, \theta_k) = R_x(\theta_i)R_y(\theta_j)R_z(\theta_k)$ refers to a rotation around all three axes in the Bloch sphere. In the trainable layers, the rotational angles are the trainable parameters θ. Hence, they are determined by solving the optimization problem Eq. (7) with a classical training routine. Additionally, two-qubit gates like the CNOT gate are used to create entanglement. This enhances the expressivity of the quantum circuit.

3.4 Measurement and output

Measurement is required to extract the information of interest into a format that we can process. Here, we perform a Pauli measurement in the z-basis on each qubit and sum over these values to obtain our approximation

$$\hat{u}_{\theta}(\boldsymbol{x}) = \sum_i \langle S_i^z \rangle. \tag{10}$$

The index i refers to the i-th qubit.

3.5 Quantum circuit ansatz

For real applications, the quantum circuit is usually more complex than the simplified version shown in Fig. 2. There are various possibilities to increase the expressivity, i.e. enhance the number and the complexity of functions the circuit can represent. One possibility to increase the expressivity is the ansatz presented by Kyriienko in [4], which repeats the trainable layer multiple times. They furthermore apply a feature map on the input values, to increase the complexity.

A different ansatz was presented by Schuld et al. in [51]. Their quantum circuit consists of alternating repetitions of encoding layers and trainable layers, as shown in Fig. 3. In comparison to the simple ansatz in Fig. 2 and the ansatz presented by Kyriienko in [4], this ansatz starts and ends with a trainable layer. Schuld et al. show that their quantum models can be written as partial fourier series in the data [51]. The accessible range of fourier frequencies in the model increases with increasing number of qubits and/or layers.

With this alternating ansatz we were able to approximate a broader variety of functions, than with the ansatz presented by Kyriienko in [4], see Appendix 6.3 for more details. For all results shown in the following, we used the alternating circuit ansatz introduced by Schuld et al.

4 Experiment

In this section, we will show a comparison between training a classical PINN and a PIQC by means of various examples. In all experiments, the PINN and the PIQC share the same cost-function, training routine and are implemented in the same Software framework SMARTy [64]. The quantum computations are implemented by using Pennylane [65] and performed on a simulator. Studying the effect of noise

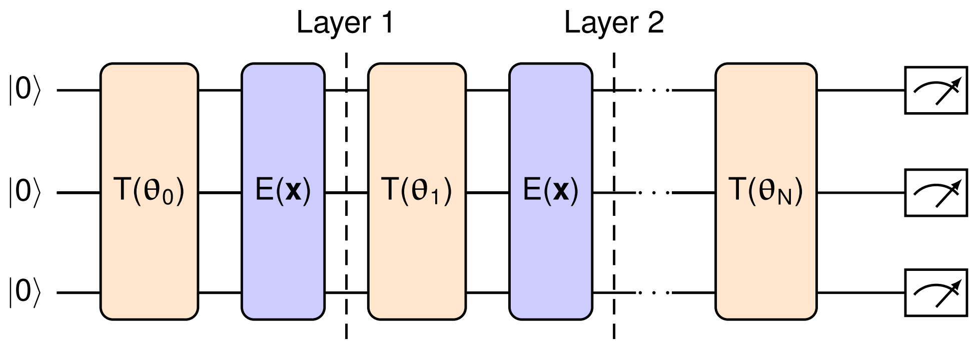

Fig. 3 Circuit ansatz used for the performed PIQC-experiments with varying number of qubits and layers. Each layer consists of a trainable (orange), and an encoding (purple) part. The repeated encoding leads to a higher expressivity of the quantum circuit. This ansatz was proposed and analyzed in [51]

and errors is beyond the scope of the current work and will be addressed in the future. Simulators (as well as currently available real quantum hardware) only allow to study circuits with a very limited number of qubits and layers. This also limits the size and complexity of problems, that can be treated. Thus, we decide to focus on simple problems and extensive parameter studies for the design of the circuits/networks dimensions. This allows for the simulation of many different circuit shapes in parallel. The used values can be found in Tab. 2.

4.1 Problems

In this paper, we study four different problems with increasing complexity and known analytical solution. *Problem I* is a simple ODE

$$\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = 1, \tag{11}$$

in the domain $x \in [-1, 1]$ and boundary conditions $u(-1) = u(1) = 0$, with a square function as analytical solution

$$u(x) = \frac{1}{2}\left(x^2 - 1\right). \tag{12}$$

Second, we study the linear transport equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \tag{13}$$

where the analytical solution is

$$u(x,t) = I(x - ct) \tag{14}$$

and I is the initial condition. As *Problem II*, we select a Gauss pulse as the initial condition:

$$I(x) = \exp(-x^2/0.1), \tag{15}$$

with periodic boundary conditions in the domain $x \times t \in [-1, 1] \times [0, 0.5]$ and a transport velocity $c = 0.5$.

Next, as *problem III* we study the one dimensional Burgers' equation which is a non-linear PDE and a frequently used test case for fluid flow applications

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}. \tag{16}$$

We consider the domain $x \times t \in [-1, 1] \times [0, 1]$ and a viscosity of $\nu = 0.01$. One can show that an analytical solution to this equation is given by [66]

$$u(x,t) = \frac{\frac{x}{t+1}}{1 + \sqrt{\frac{t+1}{t_0}}\exp\left(\frac{x^2}{4\nu(t+1)}\right)}, \tag{17}$$

with $t_0 = \exp 1/(8\nu)$. The initial and boundary conditions follow from the analytical solution. *Problem I–III* share the advantage that the function to be approximated, is continuous and smooth. In real fluid fields, however, we need to handle complex situations, like the chaotic nature of turbulence and the appearances of shocks. In this part, we study the capability of the PIQC and the PINN to approximate a shock. Therefore, for *problem IV*, we again consider the linear transport Eq. (13) with a step-function as the initial condition

$$I(x) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0, \end{cases} \tag{18}$$

which models a fully-developed shock. For the boundary we choose Dirichlet conditions $u(x = -1, t) = 0,\ u(x = 1, t) = 1$ and as the transport velocity $c = 0.5$.

4.2 Optimization

To optimize the parameters of the neural network and the quantum circuit, we use the low memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newton algorithm (L-BFGS) [67]. Empirically, we have observed most success with this algorithm for optimizing PIQCs compared to first-order stochastic gradient methods. This may be due to the fact that, as a quasi-Newton method, the algorithm requires comparatively less iterations than first-order methods. For the PIQC we are severely limited in the number of iterations, due to the significant cost of simulating the quantum circuit. In addition, L-BFGS is also a popular choice for PINN optimization, especially for fine-tuning models after an initial training stage with stochastic gradient descend variants (see for example [7, 15, 17, 18]). For a fair comparison we use the implementation of PyTorch [58] for both approaches with the same hyperparameters (see the Appendix 6.2 or further details).

4.3 Quantum circuit and neural network shapes

The performance of the PINNs and PIQC on a specific problem depends on the dimensions of the selected ansatz function because it determines the number of trainable parameters. Hence, the dimensions of the parametric ansatz determine the expressivity. Since the fundamental nature of the parametric functions is different, we compare the results based on the number of trainable parameters, which can easily be determined on both cases.

For the neural network, we fix the number of neurons per hidden layer $N_k = N_{\text{hidden}}$ for all layers. We then repeat each training run for different numbers of neurons per hidden layer N_{hidden} and different numbers of layers M. In addition, since the performance of PINNs can be inconsistent, depending on the random initialization of the trainable parameters at the beginning of the training, we repeat each run with different random initializations of the parameters. This gives a rough estimate of how consistently PINN performs. For the quantum circuit, we choose various layouts in terms of number of qubits and layers. The selected layouts for the PINN and the PIQC are depicted in Figs. 11, 12, 13 and 14. Due to the computational effort of the PIQC simulation (see Appendix 6.2.1 for more details), the runs are not repeated.

4.4 Studied quantities

For the comparison, we are interested in the convergence speed and the overall accuracy for each differential equation. For the accuracy we consider the mean absolute error (MAE):

$$\varepsilon = \frac{1}{N_{\text{val}}} \sum_{i=1}^{N_{\text{val}}} \left| \hat{u}_\theta(x_i, t_i) - u(x_i, t_i) \right|, \tag{19}$$

where N_{val} is the number of validation points, $\hat{u}_\theta(x, t)$ is the approximated solution of the PINN or the PIQC, and $u(x, t)$ is the analytical solution. To assess the accuracy, we look at the ε values which is reached after a fixed amount of epochs $n_{\text{epoch}} = 50$. For the convergence speed, we select a certain value for ε^{th} for each problem. Then we count the number of epochs n_{epochs} needed, to reach the threshold $\varepsilon < \varepsilon^{\text{th}}$ (c.f. Table 2). Note that the set of validation points is different from the set of training point.

4.5 Results

In the following, we compare the approximation performance of the PINN and the PIQC for problems I–IV.

4.6 Number of training epochs

We start with comparing the convergence speed for *problem I* and *problem II*, where the solution can be approximated well by both the quantum circuit and the neural network. For both problems we observe that the PINN and PIQC are able to reach the designated accuracies ε^{th} within 50 epochs, as long as the number of parameters is sufficient (see Figs. 4, 5). In that case, both methods generally require less than 10 epochs for *problem I* and *II*. For the PINN we observe that it may perform rather inconsistently over a wide range of parameter numbers. For *problem II*, even for more than 300 parameters, the network is occasionally unable to converge. Looking at the runs that fail, we see that this is usually the case if the number of neurons per layer is low. Even a high number of layers can not make up for a layer width of less than five neurons. For the PIQC on the other hand, we see that as long as a certain number of parameters is available (about 25 for *problem I* and about 120 for *problem II*), it is able to approximate the solution well. Overall, both methods perform similarly and can find reasonably accurate solutions at similar numbers parameters as long as the neural network is not too narrow.

For *problem III* for the PINN method, we see that it performs extremely inconsistently for the entire parameter range and is unable to reliably reach the designated ε^{th} in less than 50 epochs, as shown in Fig. 6. PINNs require a larger number of training epochs for this particular problem. For the PIQC method however, we see that for sufficiently large circuits with more than 300 parameters, the method converges again reliably and for more than 400 parameters it takes less than 10 epochs to converge to ε^{th}. For both, *problem II* and *III*, the PIQC shows the much clearer trend of the number of required epochs than the PINN. While low-parameter runs do not converge at all within 50 epochs, we see a steady decrease of the number of epochs with increasing number of trainable parameters after a critical size is reached.

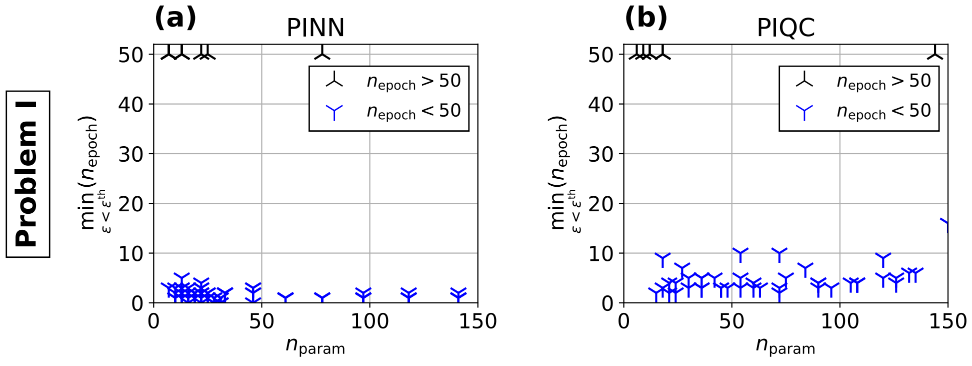

Fig. 4 The number of training epochs n_{epoch} required to decrease the mean absolute error below $\varepsilon^{\text{th}} = 10^{-3}$ in dependence of the number of trainable parameters n_{param}. Trained is **a** a classical PINN and **b** a PIQC to find a solution to the ODE Eq. (11). Both approaches quickly converge over the whole range of n_{param}, but the PINN needs slightly less epochs to reach ε^{th}

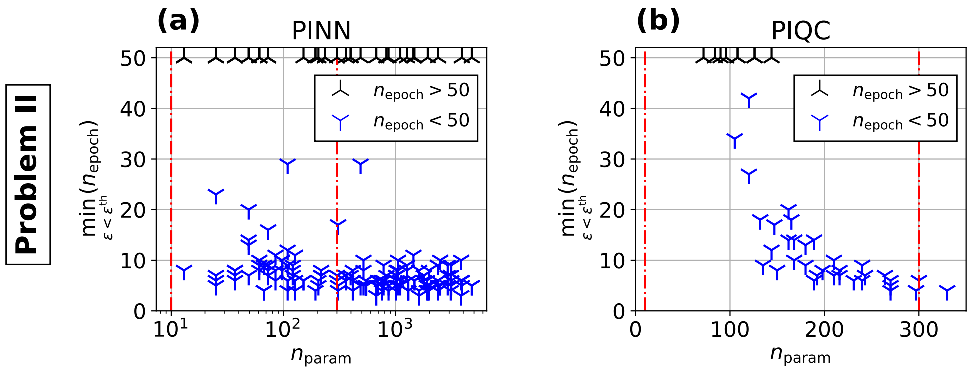

Fig. 5 The number of training epochs n_{epoch} required to decrease the mean absolute error below $\varepsilon^{\text{th}} = 3 \times 10^{-3}$ in dependence of the number of trainable parameters n_{param}. Trained is **a** a classical PINN and **b** a PIQC to find a solution to the transport equation of a Gauss pulse Eq. (13) and Eq. (15). The red vertical lines are positioned at $n_{\text{param}} = 10, 300$ and help to visualize the different range and scaling of n_{param} for both approaches. For *problem II*, the PINN shows a fast convergence for most runs but also training inconsistencies over the whole range of n_{param}. The PIQC instead requires a minimal amount of $n_{\text{param}} \approx 150$ to converge but then shows a clear dependence of n_{epoch} on n_{param}

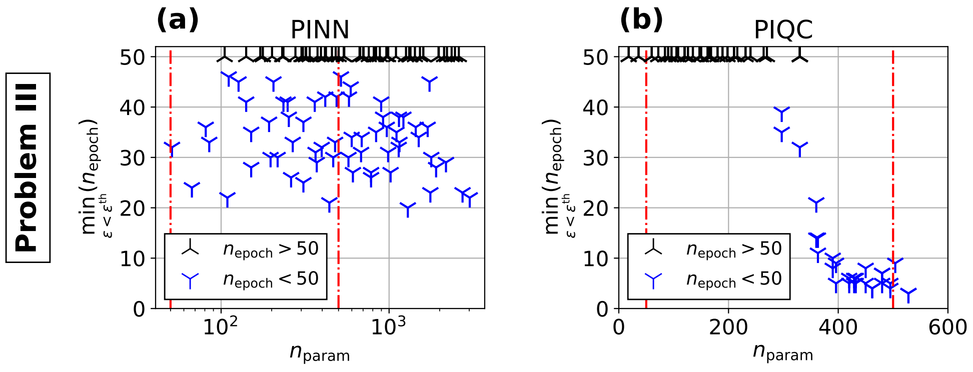

Fig. 6 The number of training epochs n_{epoch} required to decrease the mean absolute error below $\varepsilon^{\text{th}} = 5 \times 10^{-3}$ in dependence of the number of trainable parameters n_{param}. Trained is **a** a classical PINN and **b** a PIQC to find a solution to Burgers' equation (16). The PINN shows converging and non-converging runs over the whole range of parameters without a visible dependence on n_{param}. Instead, the PIQC does show a clear dependence of n_{epoch} on n_{param} and allows for a faster convergence for $n_{\text{param}} > 380$

4.7 Accuracy

Next, we compare the accuracy of both approaches after a limited number of 50 L-BFGS iterations (see Figs. 7, 8). For *problems I* and *II* we generally observe no significant changes in loss and validation errors after more than 30 epochs. Therefore, this comparison gives an assessment of the accuracy that can be reached with both approaches for the given number of training points. The test is again repeated for various neural network and quantum circuit dimensions. Each PINN network shape run is repeated with random initializations (five times for problem I and three times for problem II).

For *problem I*, the PIQC and PINN both achieve an accuracy of $\varepsilon \approx 10^{-5}$ for the best runs. The PIQC shows to have an optimal size for $n_{\text{epoch}} = 25 - 60$. For more trainable parameters, the requires number of epochs does increase again. The PINN seems to be less dependent on the number of trainable parameters. It shows similar values over the whole range of n_{param}. For *problem II* we see similar accuracies for both approaches. But in this case, the PIQC is more consistent and we observe no random training instabilities. This is in contrast to the PINN which is more prone to random training instability for narrow networks. For realistic usage of PINNs one would reduce the initial step size (the learning rate) to avoid such instabilities. This would however result in slower convergence for the PINN.

For *problem III* no accuracy plot has been created, as each epoch of the high-parameter runs is computationally very expensive in terms of wall-clock time (see Appendix 6.2.1 for details). Hence, not all runs were trained up to 50 epochs, which would be required to allow for a fair comparison. For each problem, an example of the approximation with the corresponding reached accuracy is given in Appendix 6.1.

4.8 Representing shocks

Problem IV allows us to study the capacity to represent shocks, with PINNs and PIQCs. In Fig. 9 the approximation of the step-function with a PINN and a PIQC is depicted. The PIQC with $n_{\text{param}} = 480$ is not able to accurately represent the step. Instead, it shows an oscillating behaviour around the shock. Meanwhile, the PINN with a comparable amount of training parameters is capable to approximate the step and reaches error values of $\varepsilon = 1 \times 10^{-3}$. The PINN shows superior results for the entire range of $n_{\text{param}} \in [0, 4771]$ compared to the PIQC. A total of five of the analyzed 60 PINN runs did not converge, which can be attributed to the relatively high learning rate that is used. The PIQC shows an improvement in the solution for an increased circuit size. However, the simulations cost limits the possible size and the number of training epochs.

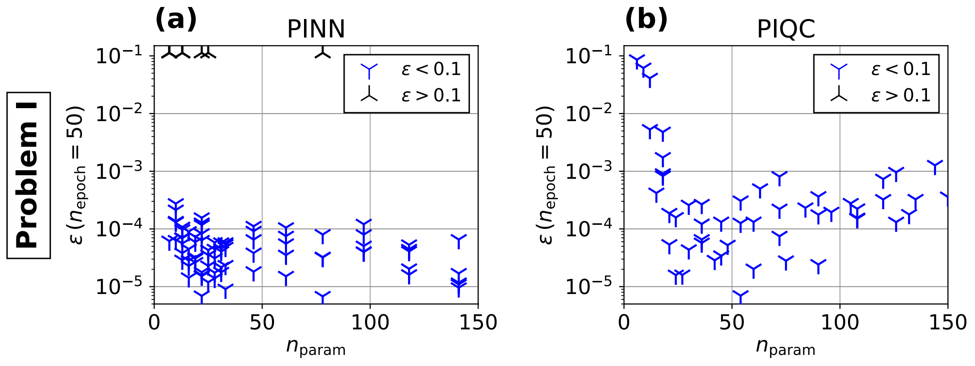

Fig. 7 Mean absolute error ε in dependence of the number of trainable parameters n_{param}. Trained is **a** a PINN and **b** a PIQC for 50 epochs to approximate the solution of the ODE in Eq. (11). The PINN reaches accuracies between $\varepsilon = 10^{-5} - 10^{-4}$ for most runs. The PIQC shows a broader range of $\varepsilon = 10^{-5} - 10^{-3}$ for most runs, the best accuracy is reached around $n_{\text{param}} = 50$

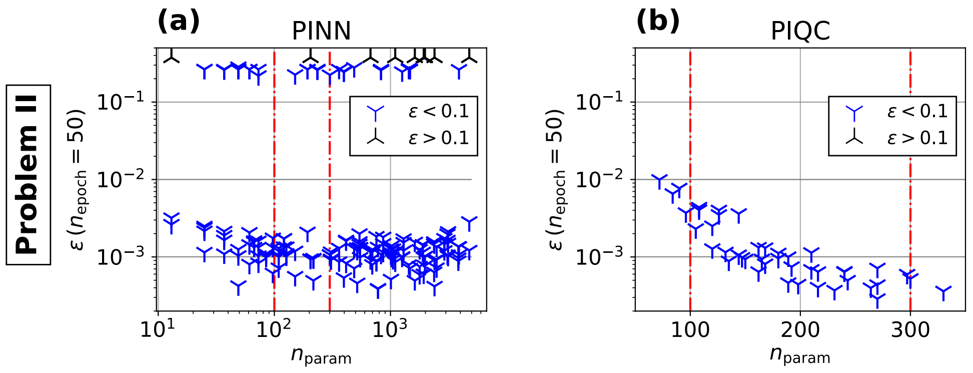

Fig. 8 Mean absolute error ε in dependence of the number of trainable parameters n_{param}. Trained is **a** a PINN and **b** a PIQC for 50 epochs to approximate the solution of the transport of a Gauss pulse Eq. (11) and Eq. (15). When the PINN converges, most runs reach accuracies around $\varepsilon = 10^{-3}$. However, as seen already in Fig. 5 not all runs converge. The PIQC shows a steady decrease of ε with n_{param}, reaching accuracies better than $\varepsilon = 10^{-3}$ for $n_{\text{param}} > 200$

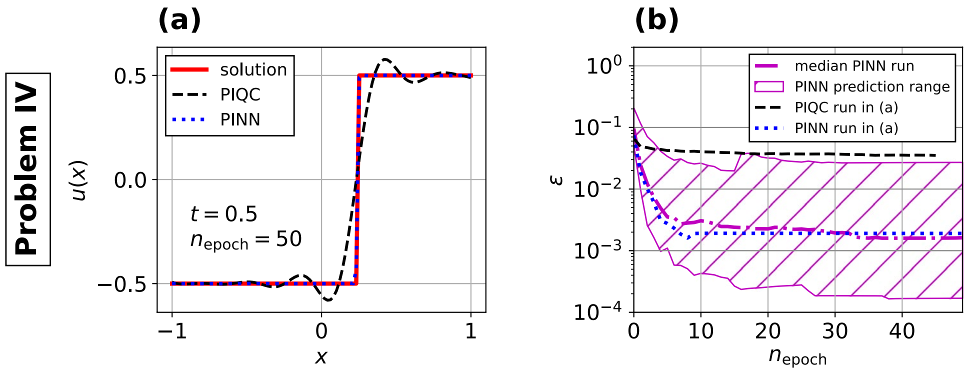

Fig. 9 **a** Approximation of the transported shock with a PINN and a PIQC after training for 50 epochs. Trained are a PIQC with $n_{\text{param}} = 480$ and a PINN with $n_{\text{param}} = 415$. While the PINN approximates the step nicely, the PIQC exhibits oscillations around the step position. **b** History of ε during training for the PIQC and PINN runs in (**a**) as well as the median PINN prediction and the prediction range (0.1 to 0.9 quantile) over all analyzed network shapes (c.f. Fig. 14). Five of the 60 runs did not converge or diverged. Hence, we consider the median and quantile predictions to account for outliers. The PINN performs significantly better than the PIQC. The median PINN reaches an accuracy between $10^{-3} - 10^{-2}$ if trained for 50 epochs

4.9 Discussion

Overall, our results show that for smooth problems the PIQC approach indeed performs similar to classical PINN methods on simple PDEs in terms of convergence speed and accuracy. For certain problems, as the Transport of the Gauss Pulse and the Burgers' equation, we even observe more consistent convergence and for the Burgers' equations a faster convergence speed given enough training parameters. This gives a first indication that the PIQC method may indeed be a reasonable candidate for solving PDEs on quantum hardware. However, it has to be clearly stated that the presented results are nothing more than an initial assessment of the approach. At this point of time, with quantum computing in its infancy, the complexity of possible experiments is severely limited. Due to computational effort (see Appendix 6.2.1 for more details) that is required for the simulation of quantum circuits on classical hardware, the extend of the current experiments is restricted in multiple ways. Firstly, the PIQC can only be trained for very few epochs within a reasonable simulation time for the considered problems. For more complex problems (see for example [15, 16, 18]), classical PINNs are usually trained with variants of stochastic gradient descent such as ADAM for thousands of epochs followed by a second training phase with a quasi-Newton optimizer such as L-BFGS. Secondly, our experiments were limited in the number of training points that the model is evaluated on. Lastly, we can only consider small models with parameter numbers on the order of $n_{\text{param}} \approx 10^2$. State of the art PINN models typically require thousand or even ten-thousand of parameters.

Moreover, we observe that the currently used circuit ansatz seems less suitable to approximate discontinous solutions.

For a linear advection equation, the PINN is able to capture the discontinuity without any additional measures while the PIQC approach fails as evident by Fig. 9. Hence, we conclude that the native shock-capturing capabilities of PINNs are superior to the chosen PIQC ansatz. However, for non-linear conservation laws such as the inviscid Burgers' equation or Euler equations, the PINN method requires additional measures [17, 18, 68, 69] such as adaptive point distributions or artificial viscosity to accurately capture shocks. These shock capturing methods might also be applicable to the PIQC approach. For now it is unclear how PIQCs behave for large amount of epochs, and whether further convergence after initial training plateaus occur, as often observed for PINNs.

Due to the current restrictions on our quantum circuit simulations, we are unable to reliably asses the performance of the PIQC approach on more complex problems. Assuming that the shown trend of similar performance at similar numbers of parameters extends even to higher complexity problems, we can make a rough estimate for the circuit sizes that are required to solve such problems. We consider two publications which used PINNs to solve two-dimensional aerodynamic flows. Eivazi et al. [16] solve the incompressible Reynolds-averaged Navier Stokes equations using networks with 8 layers and 20 neurons per layer which results in 3063 trainable parameters. Wassing et al. [18] solve the compressible Euler equations in a parametric formulation using a maximum of 8 layers of 40 Neurons. This results in 11764 trainable parameters. The number of parameters n_{param} in the presented circuit ansatz can be computed as $n_{\text{param}} = 3 \cdot (n_{\text{layers}} + 1) \cdot n_{\text{qubits}}$, where n_{layers}, n_{qubits} is the number of layers and qubits respectively. Hence, the number of trainable parameters can be increased by either increasing the number of qubits and/or the number of layers. Assuming that PIQCs would require a similar number of parameters, we can estimate that this would require for example a circuitof of 30 qubits and 33 layers to reach 3036 trainable parameters and of 60 qubits and 65 layers for 11764 trainable parameters. If we double the number of qubits, the number of layers can be approximately halved. For even more complex, industry-relevant problems like three dimensional, high Reynolds number flows, the requirements in terms of circuit size would be even further increased. However, taking into account that Kim et al. [70] present first successfully experiments on a 127 qubit quantum computer from IBM, the required circuit sizes for more complex problems will probably be in reach within the next years. Furthermore, due to the fact that the state space of quantum circuits grows exponentially with the number of qubits, it may be possible to approximate more complex problems with relatively few qubits and thus fewer parameters than for the classical equivalent.

Besides the current limit on circuit size, we assumed ideal quantum circuits for our experiments. Current generation quantum computers are prone to noise which imposes a limit on the circuit depth and is expected to further reduce accuracy.

5 Conclusion and outlook

In this work, we compared empirically the convergence speed and accuracy of PINNs and PIQCs on the basis of various differential equations. Starting with an ODE, we increased the complexity by looking at the linear transport equation, and by finally studying the non-linear viscous Burgers' equation. For *problem I-III* the solution could be represented with both, PINN and PIQCs. For problem I and II we observe convergence to the selected accuracy threshold $\varepsilon^{\mathrm{th}}$ in less than 10 epochs for both models, given sufficiently large networks/quantum circuits. For the PINN however, we observe occasional instability due to the relatively large optimization step size that we selected for both models. In comparison, the PIQC performs consistently well. For problem III the PIQC converges far more reliably and faster in terms of the number of epochs than the PINN given a sufficient number of parameters. It would be interesting to study, whether this trend of faster convergence of the PIQC persists or even increases for more complex problems with smooth solutions. In terms of accuracy, we observe similar precision for the best runs of both models on problem I and II. Again the PINN models may perform inconsistently and randomly fail for certain shapes and initializations, mostly for narrow network shapes. The PIQC was unable to approximate *problem IV*, i.e. the transport of a shock with the current circuit ansatz. This is in contrast to the classical PINN which is able to find more accurate approximations of the solution for all analyzed network shapes. It is necessary, to further study the impact of the circuit ansatz, to allow for an approximation of discontinuous problems with the PIQC. An interesting next step to tackle the shock-capturing possibilities, would be to further investigate the data-reuploding ansatz in this work, combined with the feature map approach [4, 50]. Including expressivity measures [47–49] and analytical methods for comparison [52, 71], could deepen the understanding of the impact of the circuit ansatz. The ultimate aim of using quantum computers for solving partial differential equations is to reduce the required time for numerical simulations. Therefore, future investigations will also need to address the question, whether the PIQC approach can result in reduced wall clock times, compared to classical computing. An important next step is to port the algorithm on real quantum hardware. First, this allows us to study the impact of noise and errors on its success and second enables a direct comparison between PINNs and PIQC with respect to wall clock times and energy consumption per run.

Appendix

Approximations

In Fig. 10, the analytical solution of *problem I–III* is depicted as well as exemplary approximations with PINN and PIQC. For *problem I* we used a PINN with 8 neurons and 1 layer and a PIQC with 2 qubits and 3 layers. *Problem II* is solved with a a PINN with 15 neurons and 2 layers and a PIQC with 8 qubits and 10 layers and *problem III* with a PINN with 11 neurons and 4 layers and a PIQC with 10 qubits and 15 layers. The reached accuracies of these runs are given in Table 1. Both approaches are capable to approximate the solutions of all three problems.

Training details

In the following we provide details on the training values, as number of epochs, number of training points and the chosen value of $\varepsilon^{\mathrm{th}}$ for all problems, see Table 2. We further introduce the design of experiment, i.e. the number of neurons/qubits and the number of layers for the experiments see Figs. 11, 12, 13 and 14. For *problem III*, not all possible combinations of number of qubits and layers where computed in the considered range, as each simulation in the high-parameter regime is time intensive (see 6.2.1 for time examples). For all PyTorch operations, double-precision floating point numbers were used for all PINN experiments, while floating point precision was used for the PIQC experiments to reduce the wall-clock time.

The training points were sampled, using the low-discrepancy Halton sequence to obtain a uniform distribution

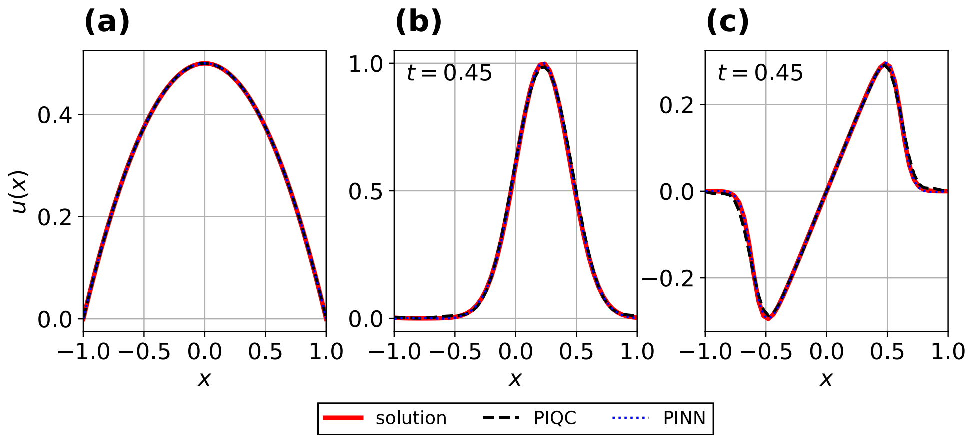

Fig. 10 Comparison of analytical solutions and exemplary PINN and PIQC predictions for **a** *problem I*, **b** *problem II* and **c** *problem III*. Both approaches are capable to capture the solutions of all three problems

Table 1 Accuracy of the approximation with the PINN or the PIQC after training for 50 epochs. The runs where performed with a PINN with *problem I*) 8 neurons, 1 layer, *problem II*) 15 neurons, 2 layers and *problem III*) 11 neurons, 4 layers; and a PIQC with *problem I*) 2 qubits and 3 layers, *problem II*) 8 qubits and 10 layers and *problem III* 10 qubits and 15 layers

	PINN	PIQC
Problem I	2.29×10^{-5}	1.59×10^{-5}
Problem II	1.07×10^{-3}	3.99×10^{-4}
Problem III	1.41×10^{-3}	2.68×10^{-3}

Table 2 Information on the training variables of *problems I–IV*

	ε^{th}	n_{training}	$n_{\text{validation}}$	max n_{epoch}
Problem I	1×10^{-3}	30	30	50
Problem II	3×10^{-3}	100	100	50
Problem III	5×10^{-3}	200	100	50
Problem IV	6×10^{-3}	200	100	50

Given is the threshold mean absolute error ε^{th} used for the convergence experiments, the number of training points n_{training}, the number of validation points $n_{\text{validation}}$ and the maximal number of epochs max n_{epoch}, used for the accuracy experiments

of quasi-random points [72]. The validation points are equidistant.

The PyTorch [58] implementation of the L-BFGS optimization algorithm [67] was used. In terms of hyperparameters of the optimizer, we mainly applied the default parameters suggested by PyTorch. However, to decrease computational efforts for problem III and IV we decreased the history size. This has shown to be possible without a significant loss in accuracy. Furthermore, we disabled the termination limits for these problems (i.e. set them to a negative value). For all four individual problems, the specified hyperparameters were used both by the PINN and the PIQC. An overview of the optimizer parameters is shown in Table 3.

Computational Effort

The classical simulation of a general quantum computer is only feasible for small numbers of qubits. The size of the vector representing the quantum state, as well as the matrices representing the operations, increase exponentially with the number of qubits. Here, we give some examples of the computation times of certain runs. Note, that this is

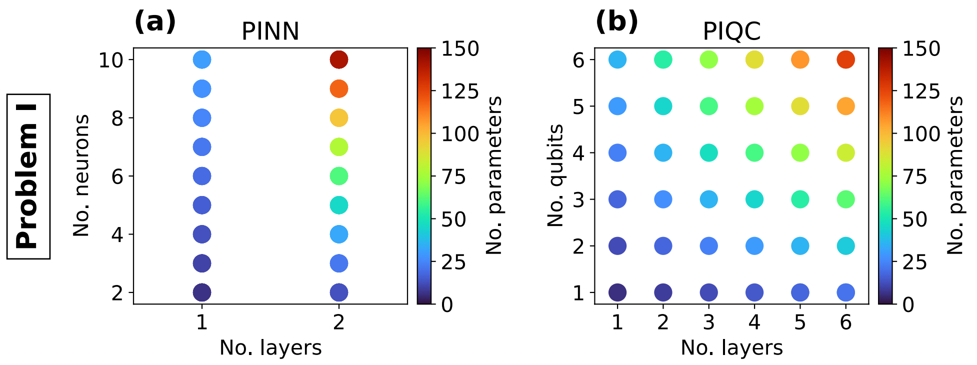

Fig. 11 Summary of analyzed neural network dimensions (**a**) and quantum circuit dimensions (**b**) for ODE (*problem I*)

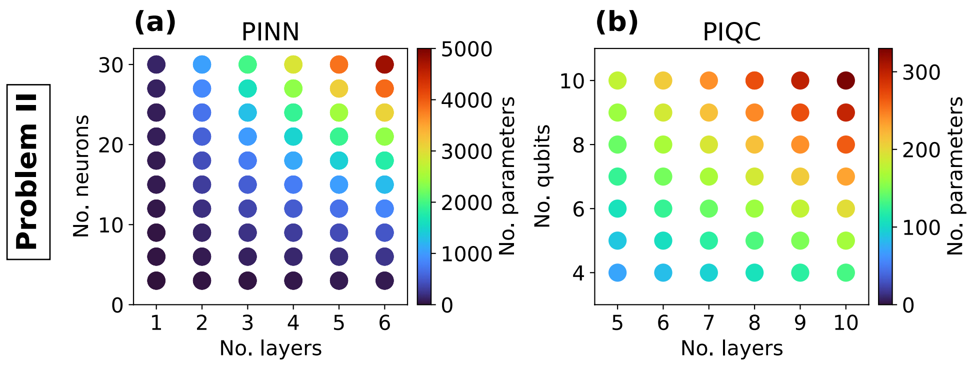

Fig. 12 Summary of analyzed neural network dimensions (**a**) and quantum circuit dimensions (**b**) for linear transport equation (*problem II*)

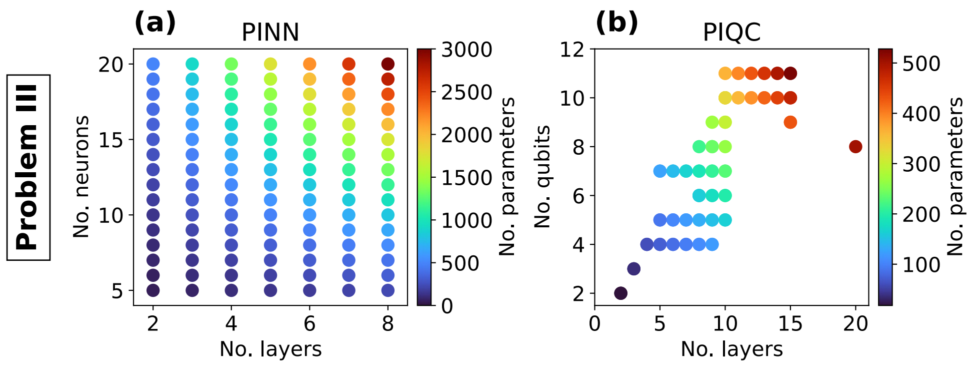

Fig. 13 Summary of analyzed neural network dimensions (**a**) and quantum circuit dimensions (**b**) for Burgers' equation (*problem III*)

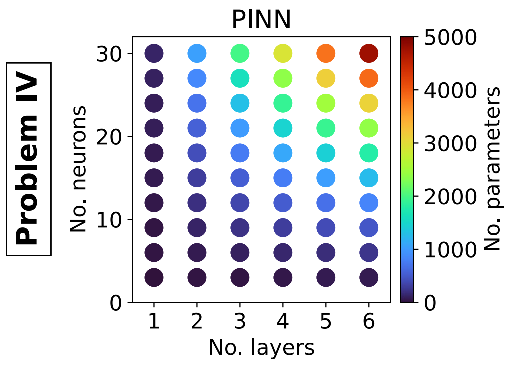

Fig. 14 Summary of analyzed neural network dimensions (**a**) and quantum circuit dimensions (**b**) for linear transport equation (problem IV)

Table 3 Hyperparameters of the PyTorch L-BFGS optimizer that were used during training

	Problem I	Problem II	Problem III	Problem IV
`lr`	1	1	1	1
`max_iter`	20	20	10	40
`max_eval`	25	25	12	50
`tolerance_grad`	10^{-7}	10^{-7}	−1	−1
`tolerance_change`	10^{-9}	10^{-9}	−1	−1
`history_size`	100	100	10	40
`line_search_fn`	`None`	`None`	`None`	`None`

For further details about each parameter, please consult the PyTorch documentation https://pytorch.org/docs/stable/index.html

not a detailed study on simulation time but only meant to allow the reader to develop a rough idea of the computational effort. Naturally, all training values, like the number of training points and the circuit size influence the duration of one epoch. Each quantum simulation was run on DLR's high performance computing cluster CARA, consisting of 2x AMD EPYC 7601 (32 cores; 2,2 GHz) processors with 128 GB DDR4 (2666 MHz) RAM per node [73]. In Table 4, the training time per epoch for various sample runs is given.

As expected, we see that an increased number of qubits and layers increase the computing time. Fine-tuning the simulation framework would allow for certain limited simulation speed-ups. For example, we did not implement parallelization. However, the overall problem to simulate large quantum circuits can not be circumvented.

In comparison, the classical PINN runs take between $4 \cdot 10^{-3}$ s for one epoch on *problem I* and about 0.5 s for

Table 4 Average wallclock time for the PIQC per epoch for various runs on a simulator

	n_{qubits}	n_{layers}	n_{param}	Average Wall Clock Time per Epoch
Problem I	2	2	18	45 s (*)
Problem I	3	4	45	2 min, 05 s (*)
Problem I	4	6	84	4 min, 12 s (*)
Problem II	4	5	72	26 min, 46 s
Problem II	4	8	108	41 min, 53 s
Problem II	8	8	216	1h, 55 min, 39 s (*)
Problem III	6	8	162	2 h, 05 min, 24 s
Problem III	6	10	162	2 h, 24 min, 45 s
Problem III	10	10	330	6h, 54 min, 1 s
Problem III	10	15	480	22h, 40 min, 03 s

Considered are *problems I–III* with different circuit shapes, defined by the number of qubits n_{qubits} and layers n_{layers}

one epoch on *problem III*. For all PINN runs an Intel i7-9700 desktop CPU has been used.

Ansatz comparison

There are many degrees of freedom in the design of the quantum ansatz. It is for example possible to use various patterns in the entanglement layers, different rotation gates, different structural repetitions of the single layers and different feature maps on the inputs. Studying the impact of all those choices is beyond the scope of the current work where we focus on the comparison with the classical PINN. We chose the ansatz presented by Schuld et al. [51] as it allowed a successful approximation of all solutions of *problem I–III* without requesting individual hyperparameter searches for each single problem. In the original work of Kyriienko et al. [4] they did not use data-reuploading but instead introduced different feature maps to enhance the expressivity. They achieved the best accuracies for all presented use-cases applying the *Tower Chebychev Feature Map*.

Figure 15 shows a comparison between the *Fourier ansatz* [51] and the *TChebychev ansatz* for *problem II* and *problem III* using the training parameters given in Appendix 6.2. For both use cases and different n_{param} the *Fourier ansatz* reaches a lower final error ϵ. All runs are trained for 50 epochs and do not exhibit a significant change in the training loss and *epsilon* in the second half of the training process. We find that the *Fourier ansatz* reaches significantly better accuracies than the the *TChebychev ansatz* for the considered examples and training parameters. Both approaches fail to approximate *problem IV*. In future work, we aim to explore whether a careful combination of data-reuploading and feature maps enables to approximate the step-function of *problem IV*.

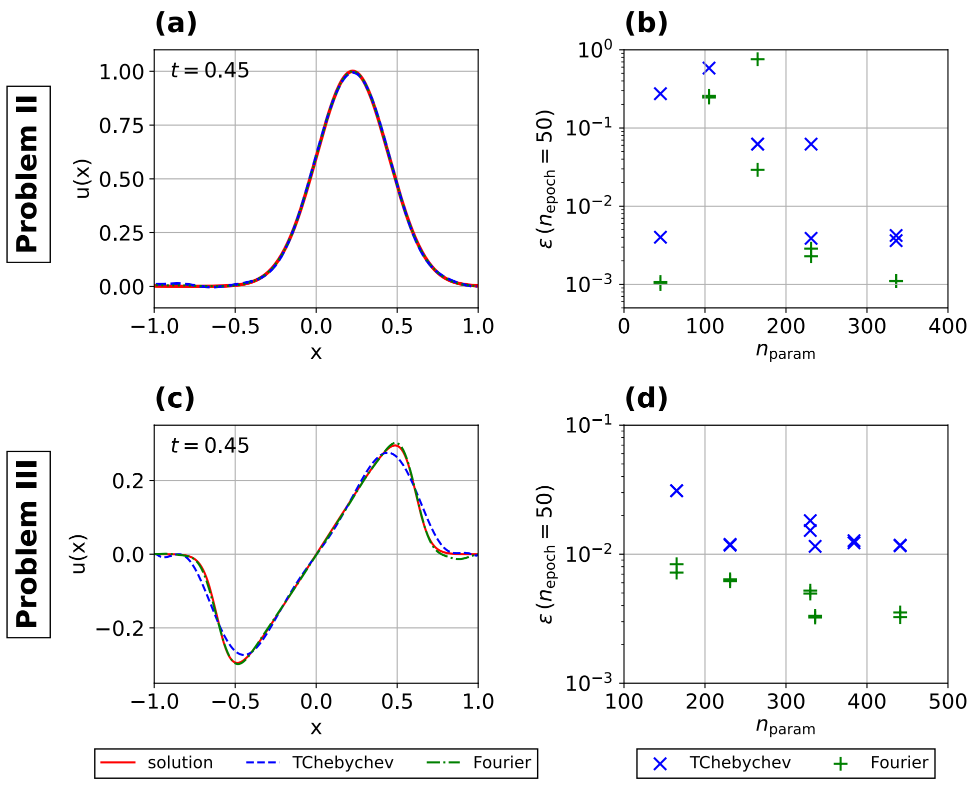

Fig. 15 Ansatz comparison between *Fourier ansatz* presented in [51] and the *Tower Chebychev ansatz* introduced in [4] for *problem II* and *problem III*. In **a** we compare the solution (Gauss pulse) of both ansatzes using 7 qubits and 15 layers. Panel **b** depicts the error over the number of parameters. Except for one run, where the *Fourier ansatz* does not converge, the *Fourier ansatz* reaches better accuracies. **c** Shows the solution of the Burgers equation of one run with 7 qubits and 20 layers. The comparison for *problem III* for different number of parameters is shown in (**d**). The *Fourier ansatz* reaches better accuracies for all runs

We do not claim that the ansatz presented in [4] does not allow for similarly good results. However, the *Fourier ansatz* seems less prone to subtleties in the hyperparameter space and does not require the search of a well suited problem-specific feature map. Instead, we found that using the *Fourier ansatz* we are able to work with the same hyperparameters for all considered use cases, leaving only the number of qubits and layers as degrees of freedom. This enables the straightforward comparison with the classical PINN presented in this work.

Acknowledgements The authors gratefully acknowledge the scientific support and HPC resources provided by the German Aerospace Center (DLR). The HPC system CARA is partially funded by "Saxon State Ministry for Economic Affairs, Labour and Transport" and "Federal Ministry for Economic Affairs and Climate Action". This work was carried out as part of the German Aerospace Center's (DLR) internal project "Machine Learning and Quantum Computing - Digitalization of Aircraft Development 2.0". This project was partly made possible by the DLR Quantum Computing Initiative and the Federal Ministry for Economic Affairs and Climate Action; *qci.dlr.de/projects/toquaflics*

Author Contributions P.S. and S.W. contributed equally to this work. The detailed contributions are as follows. All authors conceptualized this work. D.M.M., P.S., and S.W. developed the theory P.S., S.W and D.M.M. implemented the code for the PINN and PIQC experiments. P.S. performed the PIQC and S.W. the PINN simulations. P.S. and S.W. wrote the manuscript. D.M.M, P.B. and S.L. acquired project funding. S.L. coordinated the project. P.B. presented the results at the congress. All authors contributed to revisions of the manuscript

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability Data sets generated during the current study are available from the corresponding author on reasonable request.

Declarations

conflicts of interest The authors declare no conflicts of interest.

References

1. Abbas-Bayoumi, A., Becker, K.: An industrial view on numerical simulation for aircraft aerodynamic design. J. Math. Ind. **1**, 10 (2011). https://doi.org/10.1186/2190-5983-1-10
2. Lloyd, S., Palma, G.D., Gokler, C., Kiani, B., Liu, Z.-W., Marvian, M., Tennie, F., Palmer, T.: Quantum algorithm for nonlinear differential equations (2020). https://doi.org/10.48550/arXiv.2011.06571
3. Lubasch, M., Joo, J., Moinier, P., Kiffner, M., Jaksch, D.: Variational quantum algorithms for nonlinear problems. Phys. Rev. A **101**, 010301 (2020). https://doi.org/10.1103/PhysRevA.101.010301
4. Kyriienko, O., Paine, A.E., Elfving, V.E.: Solving nonlinear differential equations with differentiable quantum circuits. Phys. Rev. A **103**, 052416 (2021). https://doi.org/10.1103/PhysRevA.103.052416
5. Jaksch, D., Givi, P., Daley, A.J., Rung, T.: Variational quantum algorithms for computational fluid dynamics. AIAA J. **61**(5), 1885–1894 (2023). https://doi.org/10.2514/1.J062426
6. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics informed deep learning (Part I): data-driven solutions of nonlinear partial differential equations. arXiv (2017). https://doi.org/10.48550/arXiv.1711.10561
7. Jagtap, A.D., Mao, Z., Adams, N., Karniadakis, G.E.: Physics-informed neural networks for inverse problems in supersonic flows. J. Comput. Phys. **466**, 111402 (2022). https://doi.org/10.1016/j.jcp.2022.111402
8. Raissi, M., Yazdani, A., Karniadakis, G.E.: Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations. Science (New York, N.Y.) **367**(6481), 1026–1030 (2020). https://doi.org/10.1126/science.aaw4741
9. Dissanayake, M.W.M.G., Phan-Thien, N.: Neural-network-based approximations for solving partial differential equations. Commun. Numer. Methods Eng. **10**(3), 195–201 (1994). https://doi.org/10.1002/cnm.1640100303
10. Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw. **9**(5), 987–1000 (1998). https://doi.org/10.1109/72.712178
11. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics informed deep learning (Part II): data-driven discovery of nonlinear partial differential equations. arXiv (2017). https://doi.org/10.48550/arXiv.1711.10566
12. 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). https://doi.org/10.1016/j.jcp.2018.10.045
13. 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. (2022). https://doi.org/10.1007/s10915-022-01939-z
14. Cai, S., Mao, Z., Wang, Z., Yin, M., Karniadakis, G.E.: Physics-informed neural networks (PINNs) for fluid mechanics: a review. Acta. Mech. Sin. **37**(12), 1727–1738 (2021). https://doi.org/10.1007/s10409-021-01148-1
15. Jin, X., Cai, S., Li, H., Karniadakis, G.E.: NSFnets (Navier-Stokes flow nets): physics-informed neural networks for the incompressible Navier-Stokes equations. J. Comput. Phys. **426**, 109951 (2021). https://doi.org/10.1016/j.jcp.2020.109951
16. Eivazi, H., Tahani, M., Schlatter, P., Vinuesa, R.: Physics-informed neural networks for solving Reynolds-averaged Navier-Stokes equations. Phys. Fluids **34**(7), 075117 (2022). https://doi.org/10.1063/5.0095270
17. Mao, Z., Jagtap, A.D., Karniadakis, G.E.: Physics-informed neural networks for high-speed flows. Comput. Methods Appl. Mech. Eng. **360**, 112789 (2020). https://doi.org/10.1016/j.cma.2019.112789
18. Wassing, S., Langer, S., Bekemeyer, P.: Physics-informed neural networks for parametric compressible Euler equations. Comput. Fluids **270**, 106164 (2024). https://doi.org/10.1016/j.compfluid.2023.106164

19. Succi, S., Itani, W., Sreenivasan, K., Steijl, R.: Quantum computing for fluids: where do we stand? Europhys. Lett. **144**(1), 10001 (2023). https://doi.org/10.1209/0295-5075/acfdc7
20. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. **103**, 150502 (2009). https://doi.org/10.1103/PhysRevLett.103.150502
21. Ambainis, A.: Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations (2010). https://arxiv.org/abs/1010.4458
22. Childs, A.M., Kothari, R., Somma, R.D.: Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. SIAM J. Comput. **46**(6), 1920–1950 (2017). https://doi.org/10.1137/16M1087072
23. Montanaro, A., Pallister, S.: Quantum algorithms and the finite element method. Phys. Rev. A **93**, 032324 (2016). https://doi.org/10.1103/PhysRevA.93.032324
24. Costa, P.C.S., Jordan, S., Ostrander, A.: Quantum algorithm for simulating the wave equation. Phys. Rev. A **99**, 012323 (2019). https://doi.org/10.1103/PhysRevA.99.012323
25. Schalkers, M.A., Möller, M.: Efficient and fail-safe collisionless quantum Boltzmann method (2022) arXiv:2211.14269
26. Schalkers, M.A., Möller, M.: On the importance of data encoding in quantum Boltzmann methods. Quantum Inform. Process. (2023). https://doi.org/10.1007/s11128-023-04216-6
27. Budinski, L.: Quantum algorithm for the advection-diffusion equation simulated with the lattice Boltzmann method. Quantum Inf. Process. **20**(2), 57 (2021). https://doi.org/10.1007/s11128-021-02996-3
28. Itani, W., Sreenivasan, K.R., Succi, S.: Quantum Carleman lattice boltzmann simulation of fluids (2023). https://arxiv.org/abs/2301.05762
29. Succi, S., Itani, W., Sanavio, C., Sreenivasan, K.R., Steijl, R.: Ensemble fluid simulations on quantum computers. Comput. Fluids **270**, 106148 (2024). https://doi.org/10.1016/j.compfluid.2023.106148
30. Ichikawa, T., Hakoshima, H., Inui, K., Ito, K., Matsuda, R., Mitarai, K., Miyamoto, K., Mizukami, W., Mizuta, K., Mori, T., Nakano, Y., Nakayama, A., Okada, K.N., Sugimoto, T., Takahira, S., Takemori, N., Tsukano, S., Ueda, H., Watanabe, R., Yoshida, Y., Fujii, K.: Current numbers of qubits and their uses. Nat. Rev. Phys. **6**(6), 345–347 (2024). https://doi.org/10.1038/s42254-024-00725-0
31. Penuel, J., Katabarwa, A., Johnson, P.D., Farquhar, C., Cao, Y., Garrett, M.C.: Feasibility of accelerating incompressible computational fluid dynamics simulations with fault-tolerant quantum computers (2024). https://arxiv.org/abs/2406.06323
32. Mitarai, K., Negoro, M., Kitagawa, M., Fujii, K.: Quantum circuit learning. Phys. Rev. A **98**, 032309 (2018). https://doi.org/10.1103/PhysRevA.98.032309
33. Benedetti, M., Lloyd, E., Sack, S., Fiorentini, M.: Parameterized quantum circuits as machine learning models. Quantum Sci. Technol. **4**(4), 043001 (2019). https://doi.org/10.1088/2058-9565/ab4eb5
34. Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S.C., Endo, S., Fujii, K., McClean, J.R., Mitarai, K., Yuan, X., Cincio, L., Coles, P.J.: Variational quantum algorithms. Nat. Rev. Phys. **3**(9), 625–644 (2021). https://doi.org/10.1038/s42254-021-00348-9
35. Cerezo, M., Verdon, G., Huang, H.-Y., Cincio, L., Coles, P.J.: Challenges and opportunities in quantum machine learning. Nat. Comput. Sci. **2**(9), 567–576 (2022). https://doi.org/10.1038/s43588-022-00311-3
36. Syamlal, M., Copen, C., Takahashi, M., Hall, B.: Computational fluid dynamics on quantum computers (2024). https://arxiv.org/abs/2406.18749
37. Pfeffer, P., Heyder, F., Schumacher, J.: Hybrid quantum-classical reservoir computing of thermal convection flow. Phys. Rev. Res. **4**, 033176 (2022). https://doi.org/10.1103/PhysRevResearch.4.033176
38. Pfeffer, P., Heyder, F., Schumacher, J.: Reduced-order modeling of two-dimensional turbulent Rayleigh-Bénard flow by hybrid quantum-classical reservoir computing. Phys. Rev. Res. **5**, 043242 (2023). https://doi.org/10.1103/PhysRevResearch.5.043242
39. Bravo-Prieto, C., LaRose, R., Cerezo, M., Subasi, Y., Cincio, L., Coles, P.J.: Variational quantum linear solver. Quantum **15**, 10 (2023). https://doi.org/10.22331/q-2023-11-22-1188
40. Demirdjian, R., Gunlycke, D., Reynolds, C.A., Doyle, J.D., Tafur, S.: Variational quantum solutions to the advection-diffusion equation for applications in fluid dynamics. Quantum Inf. Process. **21**(9), 322 (2022). https://doi.org/10.1007/s11128-022-03667-7
41. Gourianov, N., Lubasch, M., Dolgov, S., Berg, Q.Y., Babaee, H., Givi, P., Kiffner, M., Jaksch, D.: A quantum-inspired approach to exploit turbulence structures. Nature Comput. Sci. **2**(1), 30–37 (2022). https://doi.org/10.1038/s43588-021-00181-1
42. Kiffner, M., Jaksch, D.: Tensor network reduced order models for wall-bounded flows. Phys. Rev. Fluids **8**, 124101 (2023). https://doi.org/10.1103/PhysRevFluids.8.124101
43. Over, P., Bengoechea, S., Rung, T., Clerici, F., Scandurra, L., Villiers, E., Jaksch, D.: Boundary treatment for variational quantum simulations of partial differential equations on quantum computers (2024). https://arxiv.org/abs/2402.18619
44. Pool, A.J., Somoza, A.D., Keever, C.M., Lubasch, M., Horstmann, B.: Nonlinear dynamics as a ground-state solution on quantum computers (2024). https://arxiv.org/abs/2403.16791
45. Cong, I., Choi, S., Lukin, M.D.: Quantum convolutional neural networks. Nat. Phys. **15**(12), 1273–1278 (2019). https://doi.org/10.1038/s41567-019-0648-8
46. Caro, M.C., Huang, H.-Y., Cerezo, M., Sharma, K., Sornborger, A., Cincio, L., Coles, P.J.: Generalization in quantum machine learning from few training data. Nat. Commun. **13**(1), 4919 (2022). https://doi.org/10.1038/s41467-022-32550-3
47. Sim, S., Johnson, P.D., Aspuru-Guzik, A.: Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. Adv. Quantum Technol. **2**(12), 1900070 (2019). https://doi.org/10.1002/qute.201900070
48. Haug, T., Bharti, K., Kim, M.S.: Capacity and quantum geometry of parametrized quantum circuits. PRX Quantum **2**, 040309 (2021). https://doi.org/10.1103/PRXQuantum.2.040309
49. Chen, C.-C., Watabe, M., Shiba, K., Sogabe, M., Sakamoto, K., Sogabe, T.: On the expressibility and overfitting of quantum circuit learning. ACM Trans. Quantum Computi. (2021). https://doi.org/10.1145/3466797
50. Williams, C.A., Paine, A.E., Wu, H.-Y., Elfving, V.E., Kyriienko, O.: Quantum Chebyshev transform: mapping, embedding, learning and sampling distributions (2023). https://arxiv.org/abs/2306.17026
51. Schuld, M., Sweke, R., Meyer, J.J.: Effect of data encoding on the expressive power of variational quantum-machine-learning models. Phys. Rev. A **103**, 032430 (2021). https://doi.org/10.1103/PhysRevA.103.032430
52. Mingard, C., Pointing, J., London, C., Nam, Y., Louis, A.A.: Exploiting the equivalence between quantum neural networks and perceptrons (2024). https://arxiv.org/abs/2407.04371
53. Bowles, J., Ahmed, S., Schuld, M.: Better than classical? The subtle art of benchmarking quantum machine learning models (2024). https://arxiv.org/abs/2403.07059
54. Paine, A.E., Elfving, V.E., Kyriienko, O.: Physics-informed quantum machine learning: solving nonlinear differential equations in latent spaces without costly grid evaluations (2023). https://doi.org/10.48550/arXiv.2308.01827
55. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8

56. Danilova, M., Dvurechensky, P., Gasnikov, A., Gorbunov, E., Guminov, S., Kamzolov, D., Shibaev, I.: Recent theoretical advances in non-convex optimization. In: Nikeghbali, A., Pardalos, P.M., Raigorodskii, A.M., Rassias, M.T. (eds.) High-Dimensional Optimization and Probability. Springer Optimization and Its Applications, vol. 191, pp. 79–163. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-00832-0_3
57. Bishop, C.M.: Pattern Recognition and Machine Learning. Information science and statistics. Springer, New York, NY (2006).
58. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: an imperative style, high-performance deep learning library (2019).https://doi.org/10.48550/arXiv.1912.01703
59. Schuld, M., Bergholm, V., Gogolin, C., Izaac, J., Killoran, N.: Evaluating analytic gradients on quantum hardware. Phys. Rev. A **99**, 032331 (2019). https://doi.org/10.1103/PhysRevA.99.032331
60. Kyriienko, O., Elfving, V.E.: Generalized quantum circuit differentiation rules. Phys. Rev. A **104**, 052417 (2021). https://doi.org/10.1103/PhysRevA.104.052417
61. Wierichs, D., Izaac, J., Wang, C., Lin, C.Y.-Y.: General parameter-shift rules for quantum gradients. Quantum **6**, 677 (2022). https://doi.org/10.22331/q-2022-03-30-677
62. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press (2010). https://doi.org/10.1017/CBO9780511976667
63. Schuld, M., Petruccione, F.: Machine Learning with Quantum Computers. Springer (2021). https://doi.org/10.1007/978-3-030-83098-4
64. Bekemeyer, P., Bertram, A., Chaves, D.A.H., Ribeiro, M.D., Garbo, A., Kiener, A., Sabater, C., Stradtner, M., Wassing, S., Widhalm, M., Goertz, S., Jaeckel, F., Hoppe, R., Hoffmann, N.: Data-driven aerodynamic modeling using the DLR SMARTy toolbox. https://doi.org/10.2514/6.2022-3899
65. Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Ahmed, S., Ajith, V., Alam, M.S., Alonso-Linaje, G., AkashNarayanan, B., Asadi, A., Arrazola, J.M., Azad, U., Banning, S., Blank, C., Bromley, T.R., Cordier, B.A., Ceroni, J., Delgado, A., Matteo, O.D., Dusko, A., Garg, T., Guala, D., Hayes, A., Hill, R., Ijaz, A., Isacsson, T., Ittah, D., Jahangiri, S., Jain, P., Jiang, E., Khandelwal, A., Kottmann, K., Lang, R.A., Lee, C., Loke, T., Lowe, A., McKiernan, K., Meyer, J.J., Montañez-Barrera, J.A., Moyard, R., Niu, Z., O'Riordan, L.J., Oud, S., Panigrahi, A., Park, C.-Y., Polatajko, D., Quesada, N., Roberts, C., Sá, N., Schoch, I., Shi, B., Shu, S., Sim, S., Singh, A., Strandberg, I., Soni, J., Száva, A., Thabet, S., Vargas-Hernández, R.A., Vincent, T., Vitucci, N., Weber, M., Wierichs, D., Wiersema, R., Willmann, M., Wong, V., Zhang, S., Killoran, N.: PennyLane: Automatic differentiation of hybrid quantum-classical computations (2022). https://doi.org/10.48550/arXiv.1811.04968
66. San, O., Maulik, R., Ahmed, M.: An artificial neural network framework for reduced order modeling of transient flows. Commun. Nonlinear Sci. Numer. Simul. **77**, 271–287 (2019). https://doi.org/10.1016/j.cnsns.2019.04.025
67. Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Math. Program. **45**(1–3), 503–528 (1989). https://doi.org/10.1007/BF01589116
68. Fuks, O., Tchelepi, H.A.: Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. J. Mach. Learn. Model. Comput. **1**(1), 19–37 (2020). https://doi.org/10.1615/JMachLearnModelComput.2020033905
69. Coutinho, E.J.R., Dall'Aqua, M., McClenny, L., Zhong, M., Braga-Neto, U., Gildin, E.: Physics-informed neural networks with adaptive localized artificial viscosity. J. Comput. Phys. **489**, 112265 (2023). https://doi.org/10.1016/j.jcp.2023.112265
70. Kim, Y., Eddins, A., Anand, S., Wei, K.X., Berg, E., Rosenblatt, S., Nayfeh, H., Wu, Y., Zaletel, M., Temme, K., Kandala, A.: Evidence for the utility of quantum computing before fault tolerance. Nature **618**(7965), 500–505 (2023). https://doi.org/10.1038/s41586-023-06096-3
71. Schuld, M.: Supervised quantum machine learning models are kernel methods (2021). https://arxiv.org/abs/2101.11020
72. Halton, J.H.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. Numer. Math. **2**(1), 84–90 (1960). https://doi.org/10.1007/BF01386213
73. CARA. https://www.dlr.de/de/forschung-und-transfer/forschungsinfrastruktur/grossforschungsanlagen/hpc-cluster/cara. Accessed 12 Sep 2023