MACHINE LEARNING
Earth

Part of the
MACHINE LEARNING
Series

PAPER • OPEN ACCESS

Quantum neural networks for cloud cover parameterizations in climate models

To cite this article: Lorenzo Pastori *et al* 2026 *Mach. Learn.: Earth* **2** 015008

View the article online for updates and enhancements.

You may also like

- Bayesian optimization of hybrid quantum LSTM in a mixed model for precipitation forecasting
Yumin Dong and Huanxin Ding
- Quantum vs. classical: a comprehensive benchmark study for predicting time series with variational quantum machine learning
Tobias Fellner, David A Kreplin, Samuel Tovey et al.
- Physics-informed generative neural network: an application to troposphere temperature prediction
Zhihao Chen, Jie Gao, Weikai Wang et al.

This content was downloaded from IP address 129.247.247.239 on 19/03/2026 at 08:23

IOP Publishing Mach. Learn.: Earth 2 (2026) 015008 https://doi.org/10.1088/3049-4753/ae4981

MACHINE LEARNING
Earth

Check for updates

OPEN ACCESS

RECEIVED
10 October 2025

REVISED
25 January 2026

ACCEPTED FOR PUBLICATION
24 February 2026

PUBLISHED
18 March 2026

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

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

PAPER

Quantum neural networks for cloud cover parameterizations in climate models

Lorenzo Pastori[1,*], **Arthur Grundner**[1], **Veronika Eyring**[1,2] **and Mierk Schwabe**[1]

[1] Deutsches Zentrum für Luft- und Raumfahrt (DLR), Institut für Physik der Atmosphäre, Oberpfaffenhofen, Germany
[2] Institute of Environmental Physics (IUP), University of Bremen, Bremen, Germany
[*] Author to whom any correspondence should be addressed.

E-mail: lorenzo.pastori@dlr.de

Keywords: quantum machine learning, quantum neural networks, climate modeling, parameterizations, cloud cover

Abstract

Long-term climate projections require running global Earth system models on timescales of hundreds of years and have relatively coarse resolution (from 40 to 160 km in the horizontal) due to their high computational costs. Unresolved subgrid-scale processes, such as clouds, are described in a semi-empirical manner by so called parameterizations, which are a major source of uncertainty in climate projections. Machine learning (ML) models trained on short high-resolution climate simulations are promising candidates to replace conventional parameterizations. In this work, we take a step further and explore the potential of quantum ML, and in particular quantum neural networks (QNNs), to develop cloud cover parameterizations. QNNs differ from their classical counterparts, and their potentially high expressivity turns them into promising tools for accurate data-driven schemes to be used in climate models. Here we perform an extensive comparative analysis between several QNNs and classical neural networks (NNs), by training both on data coming from high-resolution simulations with the ICOsahedral Non-hydrostatic weather and climate model (ICON). Our results show that the overall performance of the investigated QNNs is comparable to that of classical NNs of similar size, i.e. with the same number of trainable parameters, with both approaches outperforming standard parameterizations used in climate models. Our study also includes an analysis of the generalization ability of the models as well as the geometrical properties of their optimization landscape. We furthermore investigate the effects of finite sampling noise, and show that the training and the predictions of the QNNs are stable even in this noisy setting. This work critically investigates the applicability of quantum ML to learn meaningful patterns in climate data, and is thus relevant for a broad range of problems within the climate modeling community.

1. Introduction

One of the requirements for improving mitigation and adaption strategies against climate change is the ability to perform reliable long-term climate projections using climate models. Climate models are numerical models that simulate the evolution of the various components of the Earth system [1, 2]. In a climate model, the equations governing the dynamics of the atmosphere are discretized on a grid with horizontal extension on the order of tens of kilometers, to enable ensembles of climate projections over several decades. Due to this relatively coarse horizontal resolution, current climate models are still affected by systematic biases compared to observations, despite continuous improvements [3, 4]. At these horizontal resolutions, important physical processes such as clouds, convection or turbulence cannot be resolved, and their effect must be re-introduced in the climate model in an approximate manner by so-called parameterizations [5–7]. The structural and parametric uncertainties in conventional parameterization schemes are a source of the aforementioned remaining biases [3, 8–10]. Machine learning (ML) models are promising candidates to replace conventional parameterizations [10–13]. Several

© 2026 The Author(s). Published by IOP Publishing Ltd

uses of ML for parameterizations have been proposed in the literature, with prominent applications to radiation [14–17], convection (using different approaches such as random forests [18], or neural networks (NNs) [19–22]) and cloud cover [23, 24]. One of the strategies in employing ML is to develop data-driven schemes trained on coarse-grained data coming from high-resolution climate simulations, where convection and clouds are more explicitly resolved. While enabling numerous improvements, the use of ML also introduces several challenges [12]. These include the need of complex yet trainable models to encompass various physical scenarios, the need of large amounts of training data, and the ability to generalize to unseen climate regimes [12].

In this work, we explore whether quantum ML (QML) [25–27] can yield novel data-driven approaches that help address these challenges. We address this question by focusing on a specific parameterization task, that of cloud cover, and analyzing a specific type of QML approach known as quantum NNs (QNNs) [28]. A QNN is a sequence of operations implemented on a real or simulated quantum computer, that depend on trainable parameters and define a trainable transformation of the data that is provided as input [28]. Due to the fundamentally different nature of how data is encoded and operations are implemented in quantum computers, QNNs constitute a different type of ansatz compared to classical NNs. Several theoretical works have highlighted their higher expressivity for certain tasks [29, 30] (i.e. their ability of representing classically intractable functions), their good generalization capability [31–34], and provided hints to their well-behaved optimization landscape which may result in a good trainability [35]. Our aim is to perform a thorough comparative analysis between quantum and classical NNs on the selected parameterization task, to assess whether QML models lend themselves to learning patterns in climate-specific data better than their classical counterparts.

Broadly, the field of QML expands in several directions, including supervised learning for regression and classification [28], generative modeling [36–39] and kernel methods [40, 41]. The number of QML applications to classical problems and datasets is rapidly growing (e.g. classical computer vision datasets [42, 43], high-energy physics and astrophysics datasets [44–46], clinical datasets [47, 48], anomaly detection [49], natural language processing [50, 51], to name a few), thus potentially increasing its scope beyond purely quantum-related problems. There is also growing interest in the application of quantum computing and QML in the context of weather and climate science (for instance, for solving the underlying differential equation [52–54], developing emulators [55], climate data analysis [56, 57], or addressing sustainability challenges [58, 59]; see also [60–62] for reviews on the topics). The application of QML to classical data is still in its infancy. This is due to the noise and limited size of current noisy intermediate-scale quantum (NISQ) devices [63], which limit the complexity of the algorithms that can be run, and the amount of classical data that can be effectively loaded on a quantum device [27]. Furthermore, simulating large-scale quantum algorithms on classical computers is known to be computationally hard [64]. The tests reported in the literature provide an empirical standpoint of the field, and it is still unclear how much quantum advantage can be distilled from QML in the long run. For instance, certain provably trainable QML models could be classically simulated [65–67], and for several experiments in the literature it is hard to disentangle quantum and classical contributions to the performance [68, 69]). Nevertheless, these empirical studies are important for understanding the applicability of QML, and our comparative analysis provides a thorough assessment for a highly relevant task in climate modeling.

The specific task we address here is the parameterization of cloud cover, that is, determining the cloudy fraction of a climate model's cell based on the cell's state variables. Accurately estimating the cloud cover from large-scale variables is a crucial task for climate models [70], as cloud cover is often used as input in subsequent parameterizations [71, 72]. Traditional parameterization schemes often diagnose cloud cover primarily as a function of relative humidity [73–76], a simplified approach that limits their ability to capture complex sub-grid variability [77]. Classical ML based on NNs has proven beneficial in improving the accuracy of cloud cover parameterizations [23] with respect to the aforementioned approaches. Building on these results, we analyze the performance of QNNs as data-driven cloud cover parameterization schemes and compare them with their classical counterparts. We perform numerical simulations training and evaluating QNNs and classical NNs on coarse-grained data coming from high-resolution simulations with the ICOsahedral Non-hydrostatic weather and climate model (ICON) [78], which were part of the DYnamics of the Atmospheric general circulation Modeled On Non-hydrostatic Domains (DYAMOND) project [79, 80]. The training data is described in section 2. Following [23], we train and evaluate the models offline, i.e. without coupling them with the dynamical core of the climate model. The quantum and classical models, described in section 3, are compared on several aspects:

- Prediction accuracy (section 4.1), i.e. performance metrics (cloud cover biases and mean deviations, horizontally and vertically resolved), for models with comparable size (number of trainable parameters).
- Generalization capabilities (section 4.2), i.e. the number of training data required to achieve a certain prediction accuracy, for models with comparable size.
- Trainability (section 4.3), i.e. the number of training epochs required for the classical or quantum model to reach its optimal configuration.

On these tests, our QNNs perform comparably to standard NNs, thus highlighting the applicability of QML to a specific, yet highly relevant, parameterization. Furthermore, since quantum devices require repeated measurement rounds, also called shots, to estimate the QNNs' outputs [81], we also investigate the effects of finite number of shots, in view of a practical implementation. We do so for both the training and the prediction stage of QNNs, and show that our QNNs can train stably even in presence of such finite sampling noise. This is shown in section 5. Our work highlights an important and timely use case of QML while also discussing its potential limitations, and is an important contribution the study of quantum or quantum-inspired improvements of climate projections.

2. The training data

The training data used in this work is obtained from global high-resolution ICON simulations, from the DYAMOND project [79, 80, 82]. These simulations offer an improved representation of clouds and convection compared to simulations at climate model resolutions [83]. They consist of 40 simulated days starting on the 1st August 2016, and 40 simulated days starting on the 20th January 2020, with a resolution of approximately 2.5 km in the horizontal, on a global domain with three-hourly outputs. In both cases, the first 10 days have been discarded as spin-up time of the simulation, to have training and testing datasets more closely representing physically realistic conditions.

Following [84] we define a high-resolution grid cell to be cloudy (cloud cover $= 1$) whenever a meaningful cloud condensate (cloud water or cloud ice) amount is detected (i.e. when specific cloud condensate content exceeds 10^{-6} kg kg^{-1}) and to otherwise be cloud-free (cloud cover $= 0$). Such a binary setting of cloud cover is much more sensible at the high horizontal and vertical resolution of the storm-resolving model simulations than at coarse climate model resolutions (since at these high resolutions clouds are much more likely to occupy entire grid cells).

The data is then coarse-grained to a horizontal resolution of approximately 80 km (corresponding to an R2B5 ICON grid, a typical climate model resolution), and vertically from 58 to 27 model levels below an altitude of 21 km, following [23, 24]. The coarse graining procedure followed here is based on calculating the mean value of the high-resolution state variables (including cloud cover) over the target (coarser) grid cells. We refer the reader to [23] (appendix A) for a detailed description of the coarse graining procedure and to [23, 24] where the corresponding coarse graining and processing scripts are provided. and The resulting cells have an area of approximately 6.4×10^3 km^2, and their vertical extension varies with the model level. The vertical resolution of the bottom-most cells is approximately 0.1 km, and increases to up to 2 km for the top layers, following the terrain-following hybrid sigma height grid of ICON [78]. After coarse-graining, cloud cover in a given cell can take any value between 0 and 1, representing the fraction of the cell that is occupied by clouds. Given that cloud cover cannot exist in the absence of cloud condensate, we remove from the dataset all the cells where the total amount of cloud condensate (per cell) is zero. This results in a dataset which is more balanced, i.e. where the cloud-free samples are less over-represented. The resulting distribution of cloud cover is shown in figure 9(g) in appendix A (the removal of condensate-free cells reduces the number of cloud-free cells by roughly one order of magnitude).

The coarse-grained state variables chosen as predictors for the cloud cover in the corresponding model cell are chosen among the following ones:

- q_v [kg kg^{-1}]: specific humidity,
- q_c [kg kg^{-1}]: specific cloud water content,
- q_i [kg kg^{-1}]: specific cloud ice content,
- T [K]: air temperature,
- p [Pa]: pressure,
- $h_w = \sqrt{u^2 + v^2}$ [m s^{-1}]: magnitude of horizontal wind component (with u and v being the zonal and meridional components, respectively),

- z_g [m]: geometric height at full level,
- ϕ [rad]: latitude.

The selection of these variables is based on previous works on ML-based cloud cover parameterizations [23, 24]. The distributions of these variables on the training dataset is shown in figure 9 in appendix A. These constitute the inputs to the QNNs described in the following sections. For every model cell, our QNNs diagnose the cell cloudiness based on (a subset of) the above cell's state variables.

3. QNNs as parameterization models

The approach we use to construct our QNN-based parameterization for cloud cover is summarized in figure 1. The data resulting from the coarse-graining procedure described above constitute the training dataset $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_i$, where $\boldsymbol{x}_i \in \mathbb{R}^N$ denotes the selected coarse-grained state variables used as inputs and $y_i \in \mathbb{R}$ the output corresponding to the coarse-grained cloud cover $\mathrm{clc}(\boldsymbol{x}_i)$, for the ith model cell (at a specific spatial location and point in time). These data are used for training our QNNs in a supervised manner, i.e. optimizing the QNNs' free parameters such that their output for a given input $\boldsymbol{x}_i$ approximates $\mathrm{clc}(\boldsymbol{x}_i)$ as closely as possible. QNNs are based on a parameterized quantum circuit (PQC) realizing the unitary operator $\hat{U}_{\boldsymbol{\theta}}(\boldsymbol{x})$, depending on the inputs $\boldsymbol{x}$ and on the union of all sets of trainable parameters $\boldsymbol{\theta} \in \mathbb{R}^D$. During optimization, all parameters in $\boldsymbol{\theta}$ are adjusted so that the distance between the QNN output $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $\mathrm{clc}(\boldsymbol{x})$ for all $\boldsymbol{x} \in \{\boldsymbol{x}_i\}_i$ is minimized. In the following, we discuss the details of our implementations of $\hat{U}_{\boldsymbol{\theta}}(\boldsymbol{x})$ and how the QNN output $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is constructed.

3.1. Architecture choice

A schematic visualization of the PQC forming the backbone of our QNNs is shown in figure 2. In this work, the number of qubits is equal to the number N of input features, i.e. the components of the vector $\boldsymbol{x}$. Before the application of the QNN circuit, all the qubits are initialized in the $|0\rangle$ state. The first step in the QNN development is the encoding of the classical inputs $\boldsymbol{x}$ in the state of the qubits. In our implementation, we encode the input features as angles of single-qubit x rotations, which results in the encoding layer

$$\hat{S}(\boldsymbol{x}) = \prod_{n=1}^{N} \mathrm{e}^{-\mathrm{i}\frac{x_n}{2}\hat{\sigma}_n^x}, \tag{1}$$

where x_n is the nth component of $\boldsymbol{x}$ and $\hat{\sigma}_n^x$ being the Pauli matrix generating the rotation of the nth qubit around the x axis. Noting that $\mathrm{e}^{-\mathrm{i}\frac{x_n}{2}\hat{\sigma}_n^x}|0\rangle = \cos(\frac{x_n}{2})|0\rangle - \mathrm{i}\sin(\frac{x_n}{2})|1\rangle$, it is easy to understand that the functions this encoding generates are trigonometric functions of the inputs x_n, and that applying $\hat{S}(\boldsymbol{x})$ to the qubits several times, i.e. re-uploading the data [85, 86], increases the number of Fourier frequencies that our model can capture [86, 87]. We refer the reader to appendix B for a more explicit discussion of the QNN functional form.

The second step in the QNN development is to introduce trainable parameters. We do so by introducing so called variational blocks, denoted with $\hat{V}(\boldsymbol{\vartheta}^{(k)})$ (with $k = 1, \ldots, n_{\mathrm{enc}}$), depending on trainable parameters $\boldsymbol{\vartheta}^{(k)}$, as elements of the PQC implementing the QNN. These interleave subsequent applications of $\hat{S}(\boldsymbol{x})$, which amount to n_{enc} data re-uploads. Additionally, to further increase the number of trainable parameters, additional n_{var} variational blocks $\hat{W}(\boldsymbol{\varphi}^{(\ell)})$ (with $\ell = 1, \ldots, n_{\mathrm{var}}$), depending on trainable parameters $\boldsymbol{\varphi}^{(\ell)}$, are applied at the end of the PQC. The specific form of the $\hat{V}(\boldsymbol{\vartheta}^{(k)})$ and $\hat{W}(\boldsymbol{\varphi}^{(\ell)})$ variational blocks is specified later on in this section. Importantly, these blocks contain entangling operations, i.e. circuit operations that correlate different qubits and therefore generate correlations among the different input features x_n. The resulting unitary operator describing the PQC reads as

$$\hat{U}_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x}) = \prod_{\ell=1}^{n_{\mathrm{var}}} \hat{W}\left(\boldsymbol{\varphi}^{(\ell)}\right) \prod_{k=1}^{n_{\mathrm{enc}}} \left(\hat{V}\left(\boldsymbol{\vartheta}^{(k)}\right)\hat{S}(\boldsymbol{x})\right), \tag{2}$$

where the subscripts $\boldsymbol{\vartheta}, \boldsymbol{\varphi}$ denote the dependence on the sets $\boldsymbol{\vartheta} = \{\boldsymbol{\vartheta}^{(k)}\}_k$ and $\boldsymbol{\varphi} = \{\boldsymbol{\varphi}^{(\ell)}\}_\ell$.

As third and last step of the QNN implementation, after the PQC computation, the expectation values

$$\langle \hat{\sigma}_n^z \rangle_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x}) = \langle 0 | \hat{U}^\dagger_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x})\, \hat{\sigma}_n^z\, \hat{U}_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x}) | 0 \rangle \tag{3}$$

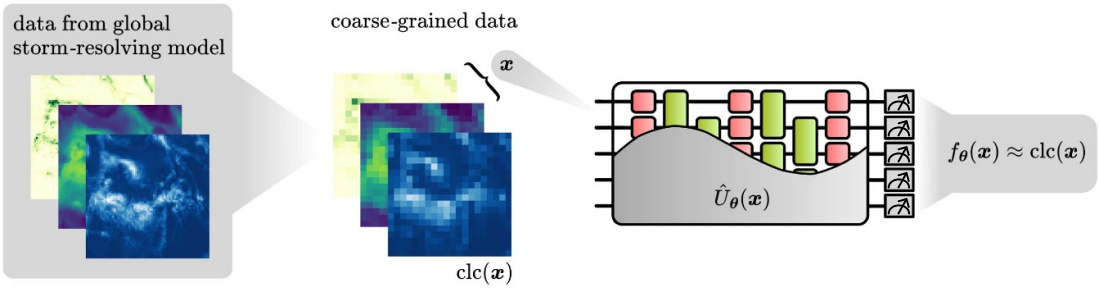

Figure 1. Schematics of our approach for developing a QNN-based parameterization. We coarse-grain high-resolution data to the target resolution, and construct a training dataset with coarse-grained state variables $\boldsymbol{x}$ as inputs and the corresponding coarse-grained cloud cover $\mathrm{clc}(\boldsymbol{x})$ as output. The dataset is used to train a QNN, i.e. to optimize the parameters $\boldsymbol{\theta}$ so that the QNN output $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ approximates $\mathrm{clc}(\boldsymbol{x})$.

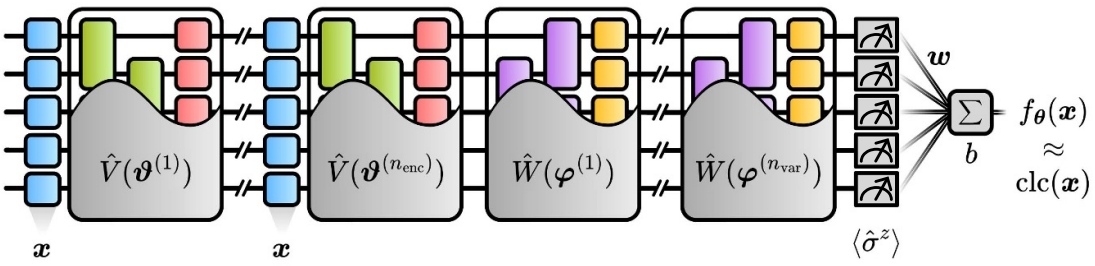

Figure 2. Schematics of our QNN architecture. The data $\boldsymbol{x}$ is uploaded n_{enc} times as angles of single-qubit rotations (blue boxes). In our implementation, each input feature is uploaded to the same qubit each time. These re-uploading gates are interleaved with variational blocks $\hat{V}(\boldsymbol{\vartheta}^{(k)})$ containing entangling gates and trainable parameters $\boldsymbol{\vartheta}^{(k)}$ $(k = 1, \ldots, n_{\mathrm{enc}})$. Afterwards, a sequence of n_{var} variational blocks $\hat{W}(\boldsymbol{\varphi}^{(\ell)})$ $(\ell = 1, \ldots, n_{\mathrm{var}})$ are applied. In the end, the expectation values of $\hat{\sigma}^z$ on all qubits are measured, and a weighted average of those is performed, with trainable weights $\boldsymbol{w}$ and a bias term b. The result $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ should approximate $\mathrm{clc}(\boldsymbol{x})$ after training the parameters $\boldsymbol{\theta} = \{\{\boldsymbol{\vartheta}^{(k)}\}_k, \{\boldsymbol{\varphi}^{(\ell)}\}_\ell, \boldsymbol{w}, b\}$.

are measured on the output state, and their weighted average with trainable weights $\boldsymbol{w}$ is computed, finally yielding the QNN prediction as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = b + \sum_{n=1}^{N} w_n \langle \hat{\sigma}_n^z \rangle_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x}) \ , \tag{4}$$

with w_n being the nth component of $\boldsymbol{w}$, b an additional trainable parameter (bias), and $\boldsymbol{\theta}$ summarizing all the parameters $\boldsymbol{\vartheta}$, $\boldsymbol{\varphi}$, $\boldsymbol{w}$, and b.

In this work, we focus on two architectures for the encoding and variational blocks in $\hat{U}_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x})$. We label the resulting QNN architectures with XYZ and ZZXY. For the XYZ circuit architecture, the encoding blocks take the following form

$$\begin{aligned}\hat{V}_{\mathrm{XYZ}}(\boldsymbol{\vartheta}) = \hat{R}_{yy}\left(\boldsymbol{\vartheta}_{(2N-1)\to(3N-3)}\right) \hat{R}_{xx}\left(\boldsymbol{\vartheta}_{N\to(2N-2)}\right) \\ \times \hat{R}_{zz}\left(\boldsymbol{\vartheta}_{1\to(N-1)}\right) \ ,\end{aligned} \tag{5}$$

where $\hat{R}_{\alpha\alpha}(\vartheta) = \prod_{n=1}^{N-1} \mathrm{e}^{-\mathrm{i}\frac{\vartheta_n}{2}\hat{\sigma}_n^\alpha \hat{\sigma}_{n+1}^\alpha}$, with $\hat{\sigma}_n^\alpha$ being the $\alpha = x, y, z$ Pauli matrix acting on the nth qubit, and $\boldsymbol{\vartheta}_{i\to j}$ denoting the slice of $\boldsymbol{\vartheta}$ from the ith to jth component. The variational blocks for XYZ read as

$$\begin{aligned}\hat{W}_{\mathrm{XYZ}}(\boldsymbol{\varphi}) = \hat{R}_x\left(\boldsymbol{\varphi}_{(3N-2)\to(4N-3)}\right) \\ \times \hat{R}_{yy}\left(\boldsymbol{\varphi}_{(2N-1)\to(3N-3)}\right) \\ \times \hat{R}_{xx}\left(\boldsymbol{\varphi}_{N\to(2N-2)}\right) \hat{R}_{zz}\left(\boldsymbol{\varphi}_{1\to(N-1)}\right) \ ,\end{aligned} \tag{6}$$

where $\hat{R}_\alpha(\varphi) = \prod_{n=1}^{N} \mathrm{e}^{-\mathrm{i}\frac{\varphi_n}{2}\hat{\sigma}_n^\alpha}$. For the ZZXY architecture the encoding and variational blocks take the following form

$$\hat{V}_{\mathrm{ZZXY}}(\boldsymbol{\vartheta}) = \hat{R}_y\left(\boldsymbol{\vartheta}_{N\to(2N-1)}\right) \hat{R}_{zz}\left(\boldsymbol{\vartheta}_{1\to(N-1)}\right) \ , \tag{7}$$

and

$$\hat{W}_{\mathrm{ZZXY}}(\boldsymbol{\varphi}) = \hat{R}_y\left(\boldsymbol{\varphi}_{2N\to(3N-1)}\right)\hat{R}_{zz}\left(\boldsymbol{\varphi}_{(N+1)\to(2N-1)}\right) \times \hat{R}_x\left(\boldsymbol{\varphi}_{1\to N}\right)\,. \tag{8}$$

We refer the reader to appendix C for a comparison with other investigated types of QNNs. In this work, the investigated QNNs are numerically simulated in Python using the Pennylane library [88] and optimized with JAX [89].

3.2. Input features and pre-processing

The components of the input vector $\boldsymbol{x}$ are chosen among the cell's state variables listed in section 2, namely specific humidity q_{v}, specific cloud water content q_{c}, specific cloud ice content q_{i}, temperature T, pressure p, horizontal wind h_{w}, geometric height at full level z_{g}, and latitude ϕ. Given the different magnitudes and distributions of these input features, it is necessary to suitably transform and re-scale them so that they can be encoded as angles in our PQCs. For the features T, p and z_{g} we found it sufficient to perform a min-max (linear) re-scaling within the interval $[0,\pi]$. For the features q_{v}, q_{c}, q_{i} and h_{w} the situation is slightly more complex, since their distribution is sharply peaked at 0 and contains long decaying tails, in particular for q_{c}, q_{i}. For these features we perform a non-linear transformation specifically constructed to (i) make the input feature distribution more uniform, so as to better distinguish from each other the values close to 0, and (ii) retain the input feature variability in the tails, which are associated with physical scenarios we are interested in distinguishing (such as deep convective regimes or extreme events). To achieve these objectives, we apply a logarithmic transform on these inputs, whose details are provided in appendix A. These input transformations are reminiscent of the cumulative distribution function of the features (which is known to map to uniform distributions). However the logarithmic dependence of our transformation functions on the input feature allows for retaining the variability in the tails of the input distribution, which otherwise would be all mapped to values very close to 1. The transformed features q_{v}, q_{c}, q_{i} and h_{w} are then multiplied by a factor π, and become approximately distributed in the interval $[0,\pi]$, so that they can be used as angles in the PQCs. To enable a proper comparison between quantum and classical NNs, the same input transformations are used in both cases.

As an additional pre-processing step further enhancing the performance of our networks, it is beneficial to learn a transformed version of cloud cover, i.e. to set the outputs y_i in the training set to $y_i = g(\mathrm{clc}(\boldsymbol{x}_i))$, with g denoting a suitable transformation function. The transformation function g is constructed to have the training outputs y_i approximately uniformly distributed in the interval $[0,1]$, given that the original cloud cover distribution is sharply peaked at 0 and 1, as we show in appendix A. An explicit expression of this transformation is given in appendix A.

3.3. Training QNNs

In an actual implementation of QNNs on quantum devices, the training of the parameters $\boldsymbol{\theta}$ is achieved via a quantum–classical feedback loop [90, 91]. At each iteration of this loop the QNN is run on the quantum device for given parameters $\boldsymbol{\theta}$ and the cost function is computed, and its value is used to propose new parameters to be used in the QNN at the next iteration. In our case, the computations of the QNN which would take place on a quantum device are simulated numerically. The cost function we minimize for training is the mean squared error (MSE) over the training dataset $\mathcal{D}$ of size $N_{\mathrm{train}} = |\mathcal{D}|$, computed as

$$\mathrm{MSE}_{\mathcal{D}}(\boldsymbol{\theta}) = \frac{1}{N_{\mathrm{train}}}\sum_{i=1}^{N_{\mathrm{train}}}\left(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i\right)^2\,, \tag{9}$$

with $y_i = g(\mathrm{clc}(\boldsymbol{x}_i))$. The parameters $\boldsymbol{\theta}$ are updated using gradient descent methods. Specifically, we use the Adam optimizer [92]. This requires the ability of calculating the gradients of the cost function with respect to the parameters efficiently, which for QNNs can be done using the parameter-shift rule [93, 94]. In our case all the gate generators are (one half times) Pauli operators multiplied by the variational parameters, and we can use the following form for the parameter-shift rule

$$\frac{\partial\langle\hat{O}\rangle_{\boldsymbol{\vartheta}}}{\partial\phi_j} = \frac{1}{2}\left(\langle\hat{O}\rangle_{\boldsymbol{\vartheta}+\frac{\pi}{2}\boldsymbol{e}_j} - \langle\hat{O}\rangle_{\boldsymbol{\vartheta}-\frac{\pi}{2}\boldsymbol{e}_j}\right)\,, \tag{10}$$

Table 1. Summary of the quantum (upper table) and classical (lower table) architectures used in the main text. For the quantum models, N refers to the number of qubits, corresponding to the number of features, and n_{enc} and n_{var} to the number of encoding and variational blocks in the PQC, respectively. For the classical models, the number of nodes in the hidden NN layers is shown (the first 'visible' layer is the input layer with N input nodes, and the last layer is the output layer with one node). Both classical NNs use tanh activations. D is the number of trainable parameters.

QNN	N	n_{enc}	n_{var}	D	Input features
$\text{XXY}^8_{5,3}$	8	5	3	201	$\{q_v, q_c, q_i, T, p, z_g, h_w, \phi\}$
$\text{ZZXY}^8_{2,7}$	8	2	7	200	$\{q_v, q_c, q_i, T, p, z_g, h_w, \phi\}$
$\text{XXY}^6_{4,2}$	6	4	2	109	$\{q_v, q_c, q_i, T, p, h_w\}$
$\text{ZZXY}^6_{2,5}$	6	2	5	114	$\{q_v, q_c, q_i, T, p, h_w\}$

NN	Hidden layers	D	Input features
$\text{NN}^8_{12,6,2}$	$12 \to 6 \to 2$	203	$\{q_v, q_c, q_i, T, p, z_g, h_w, \phi\}$
$\text{NN}^6_{8,3,7}$	$8 \to 3 \to 7$	119	$\{q_v, q_c, q_i, T, p, h_w\}$

with $\langle \hat{O} \rangle_{\boldsymbol{\vartheta}} = \langle 0 | \hat{U}^\dagger_{\boldsymbol{\vartheta}} \hat{O} \hat{U}_{\boldsymbol{\vartheta}} | 0 \rangle$, where $\hat{O}$ is a generic observable and $\hat{U}_{\boldsymbol{\vartheta}}$ is the unitary corresponding to the PQC with parameters $\boldsymbol{\vartheta}$ (analogous rules for the derivatives w.r.t. $\boldsymbol{\varphi}$ apply), where we dropped the possible dependence on the inputs $\boldsymbol{x}$. Applying this formula to our instances of PQCs, we obtain the gradients of the expectation values $\langle \hat{\sigma}^z_n \rangle_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x})$ and use them for computing the derivatives of equation (9).

4. Results in the noiseless regime

In this section we present the results from our QNNs in the absence of sampling noise, i.e. with the expectation values $\langle \hat{\sigma}^z_n \rangle_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x})$ evaluated to numerical precision. This allows us to more directly assess the learning and representational capabilities of our networks in the context of cloud cover parameterizations, and to make a better comparison with other existing classical schemes.

4.1. Evaluation of QNN predictions and comparison with classical schemes

We start by presenting the results of our QNNs trained and evaluated on the DYAMOND dataset, comparing them with classical methods such as classical feed-forward NNs and a traditionally used cloud cover parameterization scheme. To enable a fair comparison between the representational capability of QNNs and classical NNs, we restrict the architectures of the latter to a number of trainable parameters (approximately) equal to that of our QNNs. The specific details of the QNN and NN architectures are given in table 1 (with further details on the classical architectures in appendix D). For quantum and classical NNs, these design choices are optimized for reaching the best performance for a fixed number of trainable parameters. All compared models are given the same input features and are trained with the same number of training data for the same number of epochs. The influence of the number of training data on the performance of the networks and the analysis of the training dynamics is analyzed in the next sections. Here we focus on the evaluation of the predictions of the trained quantum and classical networks, while comparing them with a standard cloud cover scheme that sets the baseline of what is commonly used in climate models.

The quantum and classical network architectures presented in this section use the eight input features $\{q_v, q_c, q_i, T, p, z_g, h_w, \phi\}$ (corresponding to the $\text{XXY}^8_{5,3}$, $\text{ZZXY}^8_{2,7}$, and $\text{NN}^8_{12,6,2}$ models in table 1). All networks are trained for 200 epochs on $N_{\text{train}} = 2 \times 10^5$ training data from the coarse-grained DYAMOND dataset, with samples randomly selected at random locations in space and time. After training, we evaluate the networks on a testing dataset $\mathcal{D}_{\text{test}}$, extracted from the coarse-grained DYAMOND dataset. As for the training set, the testing samples are extracted at random locations in space and time (with care taken in not making them overlap with the training samples). We furthermore kept a selected day as additional testing set for investigating the spatial distribution of the biases (shown in figures 3(c)–(f), discussed later in this section). Besides the MSE defined in equation (9), another metric we use to evaluate the performance of our networks is the R^2 coefficient of determination defined as:

$$R^2 = 1 - \frac{\text{MSE}_{\mathcal{D}_{\text{test}}}}{\text{Var}_{\mathcal{D}_{\text{test}}}(\text{clc})}\,, \tag{11}$$

where $\text{Var}_{\mathcal{D}_{\text{test}}}(\text{clc})$ denotes the variance of the true cloud cover value over the testing dataset. Further evaluation metrics for our networks are discussed in appendix E. As a baseline to compare our QNNs to cloud cover schemes currently used in climate models, we use a version of the parameterization scheme

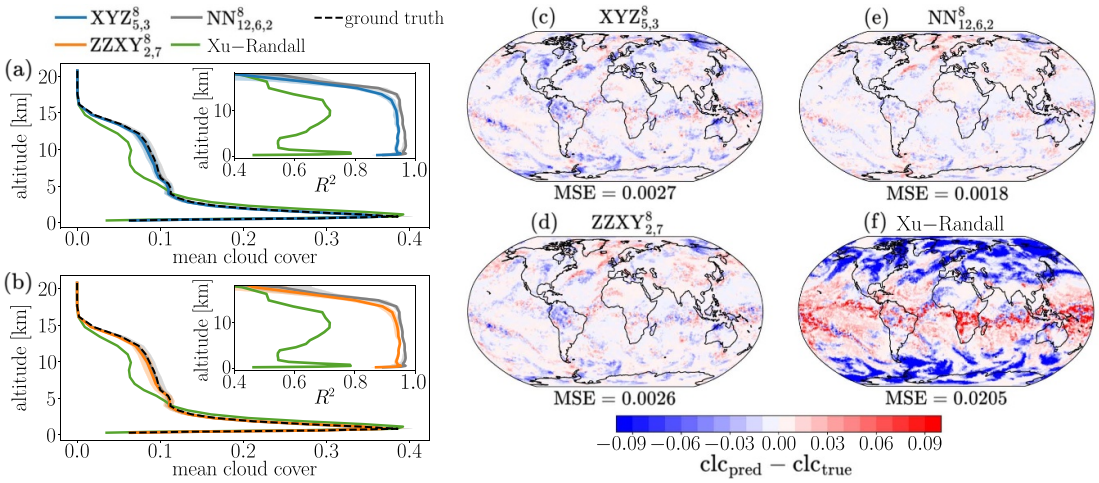

Figure 3. Evaluation of quantum neural networks (QNNs) compared to a classical NN and the Xu–Randall scheme. (a) and (b) show the vertical profiles of the mean cloud cover (i.e. the horizontally averaged cloud cover for each model layer) for the XYZ (blue) and ZZXY (orange) QNNs, respectively. These are compared with the predictions from a classical NN (grey), and from the Xu–Randall scheme (green). The insets show the corresponding vertically resolved R^2 coefficients. The shaded areas correspond to the spread (minimum to maximum) over an ensemble of 20 training instances (i.e. training the model on the same data but starting from different random initial parameters). Panels (c), (d), (e) and (f) show global maps of the difference between the predicted and the true vertically averaged cloud cover for a specific simulated day (29 February 2020, 15:00 UTC+00:00), for predictions from the XYZ QNN, ZZXY QNN, the classical NN and the Xu–Randall scheme, respectively. All networks are trained for 200 epochs with $N_{\text{train}} = 2 \times 10^5$ training data. Remark: the MSE and R^2 shown here are calculated on the non-transformed output, i.e. after applying the inverse output transform clc $= g^{-1}(f)$.

developed by Xu and Randall [75, 95], which is a semi-empirical scheme defined by:

$$\text{clc}_{\text{XR}} = \min\left\{1, \text{RH}\left(q_{\text{v}}, p, T\right)^{\beta}\left(1 - \text{e}^{-\alpha\left(q_{\text{c}}+q_{\text{i}}\right)}\right)\right\}, \tag{12}$$

where $\alpha = 4.034 \times 10^4$ and $\beta = 0.9942$ are parameters whose values we optimized (via standard MSE minimization) to reach the best performance over our training dataset. In the above equation, RH denotes the relative humidity which is calculated as:

$$\text{RH}\left(q_{\text{v}}, p, T\right) = \frac{p_{\text{v}}\left(q_{\text{v}}, p\right)}{p_{\text{s}}\left(T\right)} = \frac{p}{p_{\text{s}}\left(T\right)} \frac{q_{\text{v}}}{0.622 + 0.378\, q_{\text{v}}}, \tag{13}$$

where $p_{\text{v}}(q_{\text{v}}, p)$ is the water vapor pressure and $p_{\text{s}}(T)$ is the saturation vapor pressure (calculated according to [96]).

The QNN and NN predictions and their comparison with the Xu–Randall scheme are shown in figure 3. Looking at the vertical mean cloud cover profiles, we see that the quantum and classical NNs accurately predict the true profile (denoted by the dashed black line), while the Xu–Randall scheme exhibits visible biases throughout the whole vertical extent up to approximately 17 km, where the amount of condensate is so low that no cloud cover is diagnosed. Furthermore, quantum and classical networks are comparable also in terms of the prediction spread over the 20 training instances performed (where each instance corresponds to a training run starting from a randomly initialized parameter set). To better address the differences between the accuracy of QNNs and of classical NNs, we consider the vertically resolved R^2 coefficient in the insets. There we can see that, while both types of networks achieve a good prediction performance, the classical NN has a slightly better performance with a higher R^2 value of approximately 0.01 throughout the whole vertical extent. The drop in R^2 value for all architectures for altitudes above 15 km is due to the fact that at such high altitudes there is very low probability of observing clouds, meaning that the variance of cloud cover over the testing set at those altitudes is close to zero. Similar conclusions can be drawn from the global bias maps shown in panels (c), (d), (e) and (f) of figure 3. There, we show the bias of vertically averaged cloud cover with respect to the coarse-grained DYAMOND data for a selected day used for testing. As before, we see that the Xu–Randall scheme has the strongest prediction biases, whereas XYZ, ZZXY and NNs all achieve similar performances. In particular, we observe that all these networks show similar spatial distributions of the biases, with NNs achieving slightly lower MSE compared to the quantum architectures. We do not have a conclusive explanation for the slightly better performance of classical NNs over our QNNs for

our task. However, we find it plausible that the NNs used here as comparison (which are the best performing ones on our task—see appendix D for details on them) may be slightly better suited for our task, in terms of the functional properties they can access. More specifically, as mentioned in section 3.1 and further explained in appendix B, our QNNs can be seen as combinations of trigonometric functions with a number of frequencies (per input feature) bounded by the number of re-uploads. This functional form, although sufficient to reach a good accuracy overall, may be not enough to parameterize potentially sharp dependencies. In contrast, the NNs used here consist of sequential applications of functions of exponential nature (namely tanh functions, see appendix D for details). This makes them potentially better suited to capture strongly non-linear (in the sense of rapidly changing) functions. Let us for example focus on the modest negative bias in the snapshot of vertically averaged cloud cover over Amazonia in the QNNs predictions, a bias which is less pronounced in the case of classical NNs. This region is characterized by strong land-atmosphere coupling and frequent deep convection events with properties and occurrence that are peculiar to Amazonia and different from other rain forests such as the Congo [97]. These deep convective events are often characterized by rapid shallow-to-deep transitions [98, 99] governed by the evolution of free-tropospheric stability during the diurnal cycle [100]. From the perspective of QNN functional properties, the Amazonia bias observed here might be a hint of a slightly reduced representational performance of our QNNs compared with the classical NNs in this deep convective regime, likely characterized by strongly non-linear dependencies among state variables [98, 99]. However, a more thorough climatological analysis would be required to confirm this hypothesis.

We point out that this is not a universal deficiency of general QNNs. The representational performance of QNNs depend on the data encoding chosen, and one can adopt different and more flexible encoding strategies encompassing an increased set of frequencies [101], or different types of basis functions [102]. Investigating these options is subject of future work. Overall, our QNNs show very similar performance to the classical networks with both achieving a high prediction accuracy and moderate cloud cover biases, thus demonstrating that QML can yield suitable models to predict patterns in climate data. A further analysis of the QNN biases, e.g. investigating their vertical profiles, could help to detect the specific meteorological conditions where they struggle, and to devise strategies to improve them. In the next sections, we compare our QNNs with classical NNs on further aspects beyond the standard performance metrics.

4.2. Generalization in quantum and classical NNs

In this section, we address the in-distribution generalization ability of our QNNs and compare it to that of classical NNs. In-distribution generalization refers to whether a supervised learning model can generalize to unseen data extracted from the same distribution of training data. Here, our goal is to understand whether QNNs show better generalization abilities by requiring less training data to reach the same test performance. To this end, for both types of networks, we compute the scaling of the performance metrics (R^2 and MSE) with the size N_{train} of the training dataset, where the metrics are computed on a testing dataset $\mathcal{D}_{\text{test}}$ of size 2×10^5 following the same distribution as the training set. Here, we use slightly smaller networks compared to the previous section, to reduce the training runtime of the QNNs for gathering sufficient statistics for our training instances. Specifically, we use the architectures reported in table 1 with six input features. In the comparison, both R^2 and the MSE (figure 4) show a scaling with N_{train} that is very similar for the classical and quantum networks. The degradation of the prediction performance happens approximately at the same value of N_{train}, and furthermore no significant difference between our XYZ and ZZXY models can be seen. The MSE loss on the testing dataset follows approximately a $1/\sqrt{N_{\text{train}}}$ scaling for $N_{\text{train}} \geqslant D$ until saturation due to model deficiency starts to occur, which is consistent with the scaling laws reported in the works on generalization in QML [32, 33]. Classical and quantum NNs follow approximately the same scaling with the same pre-factors, and, consistently with the observations in the previous section, we see that for large values of N_{train} the classical NNs achieve a slightly better prediction performance.

4.3. Trainability from the perspective of information geometry

In this section, we analyze the training dynamics of our QNNs and NNs, and try to establish a connection with the geometrical properties of their optimization landscape, along the lines of [35]. The authors of [35] use tools from information geometry, in particular the Fisher information matrix (FIM), setting them in relation with the trainability of a model, and introduce a quantity called 'effective dimension', which they use to prove generalization bounds. They conduct numerical experiments comparing classical and quantum NNs, and empirically show that QNNs have a better behaved (i.e. more evenly spread) FIM spectrum and a higher effective dimension, which in their experiments results in a faster and more stable training compared to classical networks. Motivated by these empirical observations, here

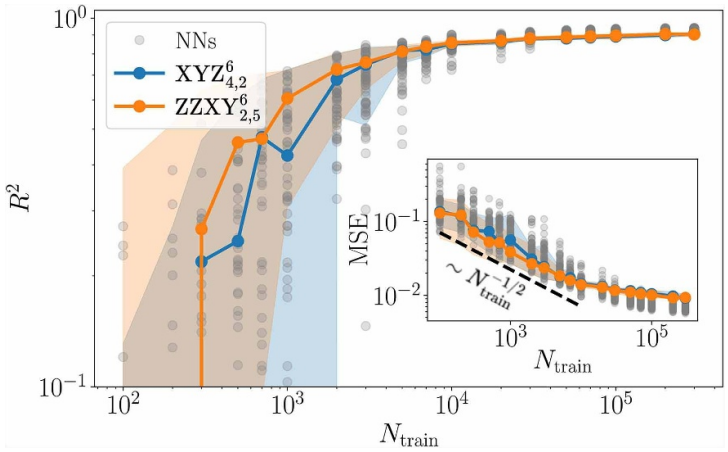

Figure 4. Scaling of network performance with size of the training set N_{train}. The grey points correspond to the tested classical NNs (the six architectures with six input features discussed in appendix D), with one point corresponding to a specific training instance for the given N_{train}, starting from randomly initialized parameters. The blue and the orange data correspond to the XYZ and the ZZXY QNNs, respectively. We plot 10 training instances per N_{train} value and architecture. For the QNNs, the dots indicate the mean over the training instances for a given N_{train}, while the shading denotes the spread (minimum to maximum). The main panel shows the R^2 score, the inset the MSE, both calculated over a test set of size 10^5. Remark: the MSE and R^2 shown here are calculated on the non-transformed output, i.e. after applying the inverse output transform clc $= g^{-1}(f)$.

we conduct a similar experiment. Specifically, for each quantum and classical network tested, we draw an ensemble of random parameter configurations (here consisting of 200 samples), and use it for constructing an ensemble of training instances on the one hand, and to calculate the FIM and the models effective dimension on the other hand. Before presenting the results, we briefly recap the definition and meaning of the FIM and effective dimension.

The FIM describes the geometrical properties of the parameter space of a parameterized model [103]. It is a central tool in the field of information geometry, as it defines a metric in the model space, i.e. the space of functions a parameterized model can represent [103–105]. While the FIM is typically defined for probabilistic models, its definition can be extended to the regression models studied here (see [106, 107] and appendix F for details). In this case, the elements of the FIM are computed as follows

$$\begin{aligned} F_{j,k}(\boldsymbol{\theta}) &= \mathbb{E}_{\boldsymbol{x}}\left[\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_j}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_k}\right] \\ &\approx \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_j}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_k}\,, \end{aligned} \tag{14}$$

with $\mathbb{E}_{\boldsymbol{x}}$ denoting the expected value over the input distribution, which is empirically approximated by averaging over the (training) dataset $\mathcal{D}$. The FIM F is a $D \times D$ positive semi-definite matrix. Giving rise to a metric in model space, the FIM encodes information on which directions in the parameter space lead to appreciable changes in the outputs of the model, and which directions are instead less influential or redundant. This information is reflected in the spectrum of the FIM: an evenly spread and non-zero spectrum indicates that all parameters are almost equally contributing to independent model changes, whereas the presence of small or zero eigenvalues means that the corresponding parameters (or linear combinations thereof) are irrelevant. Based on this observation, the FIM spectrum becomes a useful indication of the capacity of a model to effectively explore its parameter space, 'making use' of all its degrees of freedom. This capacity, according to [35], may also be beneficial during training. The (normalized) effective dimension introduced in [35] is constructed to capture this capacity, and is defined as

$$\hat{d}_{\text{eff}} = \frac{2\log\left(\frac{1}{V_\Theta}\int_\Theta \sqrt{\det\left(I_D + c_{N_{\text{data}}}\hat{F}(\boldsymbol{\theta})\right)}\,\mathrm{d}\boldsymbol{\theta}\right)}{D\log c_{N_{\text{data}}}}\,, \tag{15}$$

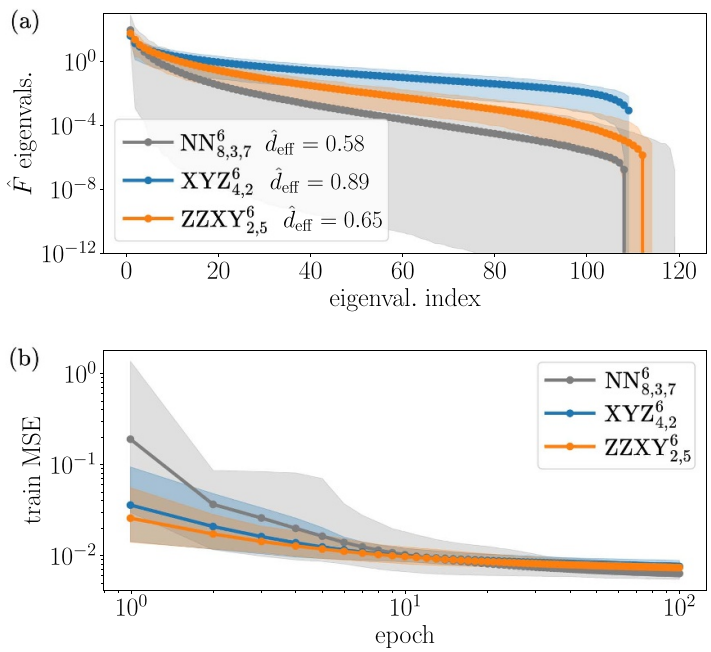

Figure 5. (a) Spectra of normalized FIM for classical NN (grey), and the XYZ (blue) and ZZXY (orange) QNNs. The dots correspond to the mean value over the 200 random parameter initializations, while the shading denotes the spread (minimum to maximum). To estimate the FIM we use 9×10^6 input data. The normalized effective dimension $\hat{d}_{\mathrm{eff}}$ is also reported in the legend. (b) Training curves for 200 training instances for classical NN (grey), XYZ (blue) and ZZXY (orange) QNNs. Here, $N_{\mathrm{train}} = 10^5$.

where $c_{N_{\mathrm{data}}} = \frac{N_{\mathrm{data}}}{2\pi \log N_{\mathrm{data}}}$ with $N_{\mathrm{data}} \equiv |\mathcal{D}|$ being the number of input data samples, D the number of parameters, and $\hat{F}(\boldsymbol{\theta})$, the normalized FIM, defined as

$$\hat{F}(\boldsymbol{\theta}) = \frac{D}{\frac{1}{V_\Theta}\int_\Theta \mathrm{tr}\left(F(\boldsymbol{\theta})\right)\mathrm{d}\boldsymbol{\theta}} F(\boldsymbol{\theta}) \ , \tag{16}$$

with $\frac{1}{V_\Theta}\int_\Theta \mathrm{tr}\big(F(\boldsymbol{\theta})\big)\mathrm{d}\boldsymbol{\theta} \approx \frac{1}{M}\sum_{m=1}^{M} \mathrm{tr}\big(F(\boldsymbol{\theta}_m)\big)$. The $\hat{d}_{\mathrm{eff}}$ is normalized to the dimensionality of the parameter space D, hence being bounded in $[0, 1]$. Furthermore, it is computed from averages over the parameter space, hence depending solely on the architecture choices and the input distribution.

Equipped with these definitions, we can now present the results of our experiments. From the spectra of the normalized FIM shown in figure 5(a) we can make two observations. First, on average the QNNs have larger FIM eigenvalues and a flatter spectrum, and second, the spread over the 200 parameter samples is larger for NNs compared to the QNNs. This reflects in the higher value of $\hat{d}_{\mathrm{eff}}$ for the QNNs, as reported in the legend. These different geometrical properties of quantum and classical NNs are consistent with those observed in [35], and therefore prompt us in investigating whether our QNNs also show a faster and more stable training for the problem at hand.

The results on the training dynamics of our networks are shown in panel (b) of figure 5. There, we observe that both quantum and classical architectures train to approximately the same value of the loss function, with the classical NN achieving slightly lower loss. The behavior of the training loss, in particular the speed at which the training has approximately converged, is very similar when comparing classical and quantum networks, unlike the observations in [35]. Therefore, for our test case we cannot pinpoint a strong relationship between the geometrical properties described before and the effectiveness of training (similar observations have been also reported in recent works [108, 109]). We refer the reader to appendix F for a further analysis of the relation between training dynamics and FIM, which also includes tests on other NNs and QNNs. We conclude this section by mentioning that the trainability aspects investigated here concern only the training dynamics of QNNs for a fixed number of qubits, and that in this work we are not addressing the problem of barren plateaus, i.e. the increasing occurrence of large and flat regions in the optimization landscape for larger number of qubits, expected to affect general QNN architectures [110–112].

5. Influence of shot noise

In a real quantum device, the expectation values $\langle\hat{\sigma}_n^z\rangle_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x})$ used in our QNNs would need to be estimated from a finite number of repeated measurements, also called shots, on the output quantum state. This introduces statistical fluctuations, i.e. shot noise, in the estimates for $\langle\hat{\sigma}_n^z\rangle_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}(\boldsymbol{x})$. In this section we analyze the effects of shot noise on the training and performance of our QNNs, and their dependence on the number n_{shots} of measurement shots. Furthermore, we present an application of the variance regularization technique introduced in [113] to our use case, and show that it can help to reduce the effects of shot noise in the training and subsequent inference stage of our QNN models.

5.1. Training and inference with shot noise

We now analyze the influence of shot noise on the training dynamics and on the prediction performance of our QNNs. To this end, it is instructive to recall how shot noise influences the predictions of our QNNs. The QNNs' predictions $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ are calculated from a weighted average of the expectation values $\langle\hat{\sigma}_n^z\rangle_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}$ which, in a realistic quantum device, would be estimated from a finite number n_{shots} of measured bit-strings as

$$\text{Est}_{n_{\text{shots}}}\left[\langle\hat{\sigma}_n^z\rangle_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}\right] = \frac{1}{n_{\text{shots}}}\sum_{s=1}^{n_{\text{shots}}} b_n^{(s)}\,, \tag{17}$$

with $b_n^{(s)} \in \{-1,+1\}$ denoting the nth 'bit' of the sth measured bit-string. Let us call $z_n \equiv \text{Est}_{n_{\text{shots}}}\left[\langle\hat{\sigma}_n^z\rangle_{\boldsymbol{\vartheta},\boldsymbol{\varphi}}\right]$ (we drop here the dependence on $\boldsymbol{x}$, $\boldsymbol{\vartheta}$ and $\boldsymbol{\varphi}$ for notational convenience). The z_n are random variables with covariance given by

$$\text{Cov}\left[z_m, z_n\right] = \frac{\langle\hat{\sigma}_m^z\hat{\sigma}_n^z\rangle - \langle\hat{\sigma}_m^z\rangle\langle\hat{\sigma}_n^z\rangle}{n_{\text{shots}}}\,. \tag{18}$$

Using this, we can calculate the variance of the final prediction as

$$\text{Var}\left[f_{\boldsymbol{\theta}}\left(\boldsymbol{x}\right)\right] = \sum_{m,n=1}^{N} w_m\, w_n\, \text{Cov}\left[z_m, z_n\right]\,, \tag{19}$$

with w_n being the nth trainable weight in $\boldsymbol{w}$ (see equation (4)). Every QNN evaluation for calculating both predictions and gradients (with the parameter-shift rule) is then affected by statistical fluctuations which in general negatively impact the training and prediction performance of the model.

We start by analyzing the training dynamics with a finite n_{shots} for the XYZ and ZZXY networks. The results are shown in panels (a) and (b) of figure 6, where we plot the training MSE during the training epochs for different n_{shots} (fixed throughout the training) for both the XYZ and ZZXY architectures, respectively. The different curves with the same color correspond to different training instances obtained by starting from the same initial parameter set, but with different shot noise realizations. For both architectures it is visible that for n_{shots} sufficiently large (i.e. on the order of 10^4) the training is successful, closely following the noiseless training curve. For smaller values of n_{shots}, the MSE is minimized in the initial iterations before the training becomes unstable, which likely happens when the gradients' magnitude becomes comparable to the noise in the gradients' evaluation. These results demonstrate that there is a regime where our QNNs train stably in the presence of shot noise, albeit it being relatively costly in the number of necessary circuit evaluations. We discuss in the next section a method targeted towards minimizing these costs.

Second, we analyze the performance of our QNNs on a testing dataset when varying n_{shots}. Specifically, we use the optimal network parameters configurations for both QNNs obtained after training in the noiseless regime, and test how a finite n_{shots} (in particular smaller than 10^4) affects their prediction performance. A useful metric we compute to understand the (average) impact of shot noise on the test set is the mean prediction variance (MPV), which is defined on a dataset $\mathcal{D}$ as

$$\text{MPV}_{\mathcal{D}}\left(\boldsymbol{\theta}\right) = \frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}_i\in\mathcal{D}} \text{Var}\left[f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_i\right)\right]\,. \tag{20}$$

The results are shown in panel (c) of figure 6, where we plot the R^2 and the MSE for different n_{shots} for the XYZ and ZZXY architectures. From these results it is clear that the quality of the predictions rapidly degrades for $n_{\text{shots}} < 10^4$ while it is stable for larger values. The slight difference between the two architectures can be attributed to the difference in the value of the MPV, which is indicated in the plot.

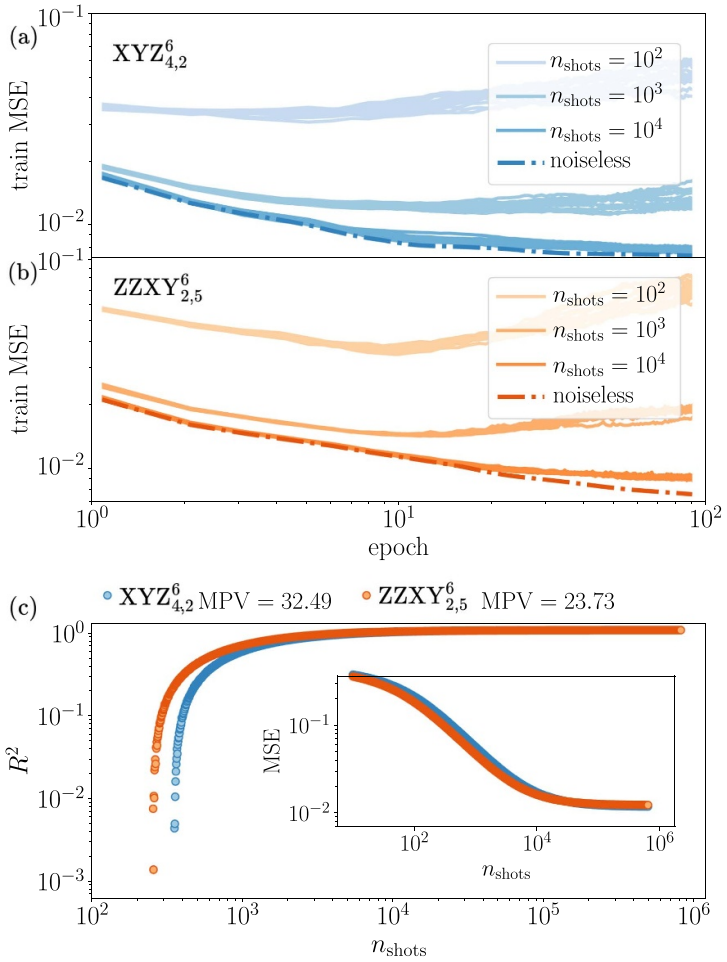

Figure 6. (a) and (b) Training curves for XYZ (blue) and ZZXY (orange) QNNs, respectively, for different values of n_{shots} used in the evaluation of predictions and gradients. The different solid lines correspond to different training instances performed starting from the same initial parameter configuration, but with different shot noise realizations. The dashed line corresponds to the noiseless limit, i.e. predictions and gradients evaluated to numerical precision. Here, $N_{\mathrm{train}} = 10^5$. (c) Test set performance of XYZ (blue) and ZZXY (orange) QNNs trained in the noiseless regime, against n_{shots} used at inference, for a test set of size 2×10^5. Remark: The MSE and R^2 shown here are calculated on the non-transformed output, i.e. after applying the inverse output transform $\mathrm{clc} = g^{-1}(f)$.

Specifically, the ZZXY architecture has a slightly lower MPV, which results in a lower number of shots needed to achieve a given accuracy. The MPV, capturing this difference, becomes therefore a crucial ingredient for the method presented in the next section.

5.2. Variance regularization

In this section, we discuss the variance regularization technique introduced in [113] and apply it to our QNNs for cloud cover. Variance regularization is a technique aimed at reducing the effects of measurement shot noise in the training and inference stage of a QML model. This is achieved by adding to the loss function (here the MSE) a term which is proportional to the variance of the output of the model, in order to simultaneously minimize both terms (if possible) and therefore obtain a model requiring fewer shots to be evaluated, both during and after training [113]. In our context, the loss function that we minimize in the training reads as

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}; \lambda) = \mathrm{MSE}_{\mathcal{D}}(\boldsymbol{\theta}) + \lambda \mathrm{MPV}_{\mathcal{D}}(\boldsymbol{\theta}) \ , \tag{21}$$

where $\mathrm{MPV}_{\mathcal{D}}(\boldsymbol{\theta})$ is defined in equation (20) and λ is a regularization parameter. In general, the evaluation of $\mathrm{MPV}_{\mathcal{D}}(\boldsymbol{\theta})$ requires measurements in addition to those performed for evaluating the model output. In our case however, we can estimate both MSE and MPV from the same set of measurement shots, since $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is constructed only from $\hat{\sigma}^z$ expectation values. Calculating the gradients of $\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}; \lambda)$ can be done with the same parameter-shift rule of equation (10), therefore this changing of the loss does not result in additional training overhead.

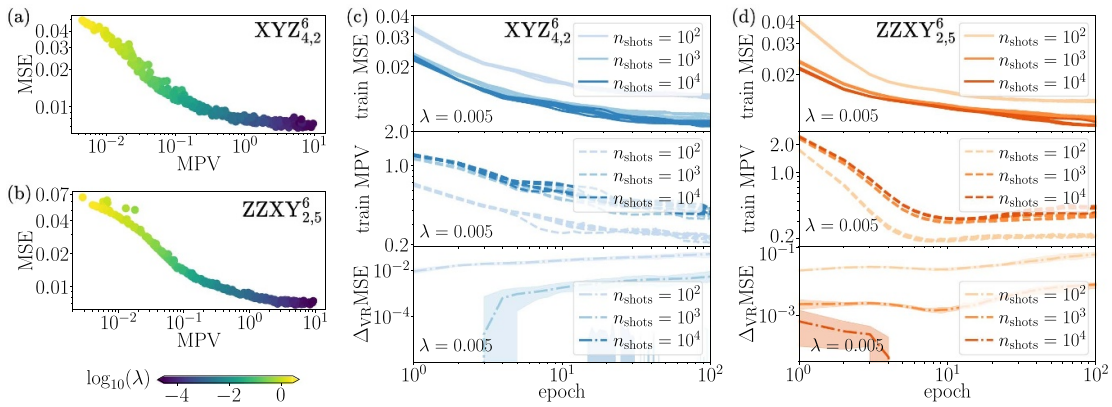

Figure 7. Variance regularization results. (a) and (b) Post-training mean squared error (MSE) and mean prediction variance (MPV) on testing set for XYZ and ZZXY QNNs, respectively. The color scale refers to the value of λ, the training is performed in the noiseless regime, with 2×10^5 training data for 150 epochs, and 2×10^5 testing data points are used. (c) and (d) Noisy training curves using variance regularization with $\lambda = 0.005$ for XYZ (c) and ZZXY (d) QNNs, respectively. The upper row (solid lines) shows the mean squared error (MSE), the middle row (dashed lines) shows the mean prediction variance (MPV). The different color shades correspond to different n_{shots} used, while the six different curves with the same colors correspond to six different training instances (from the same initial parameters, but different shot noise realizations). The lower row (dash-dotted line) shows the difference between the MSE without variance regularization and the MSE with variance regularization ($\Delta_{\mathrm{VR}}\mathrm{MSE} \equiv \mathrm{MSE}_{\mathrm{no\,var.\,reg.}} - \mathrm{MSE}_{\mathrm{var.\,reg.}}$). The shading correspond to the spread over the training instances.

We test this technique on our two QNN architectures in the same setting as in the previous section. The regularization parameter λ is chosen to be constant during training, although we remark here that [113] also proposes a dynamical regularization approach which may further improve the training. The results are shown in figure 7. Training the QNNs with variance regularization in the noiseless regime (figures 7(a) and (b)) shows that there exist values of λ for which the MPV can be reduced of more than one order of magnitude while still keeping the MSE low (i.e. below 10^{-2}). In other words, these results show that there are regions of the QNNs' parameter space where both the prediction error and the prediction variance can be low, and that variance regularization can effectively target them provided one chooses a suitable value for λ.

With these positive results, we move on to applying variance regularization with a finite n_{shots}, to assess whether a finite λ can help in stabilizing the training even when $n_{\mathrm{shots}} < 10^4$ (figures 7(c) and (d)). From both panels it is evident that a value of $\lambda = 0.005$ is already sufficient to stabilize the training even for a relatively low $n_{\mathrm{shots}} = 100$. For a better comparison with the case of no variance regularization, we also show the difference with respect to the training MSE in the previous section (figure 6(a)), in the lower row of panels (c) and (d). For $n_{\mathrm{shots}} < 10^4$, the difference $\Delta_{\mathrm{VR}}\mathrm{MSE} \equiv \mathrm{MSE}_{\mathrm{no\,var.\,reg.}} - \mathrm{MSE}_{\mathrm{var.\,reg.}}$ remains positive throughout the training, indicating a better training performance when using a finite λ. For a large number of shots $n_{\mathrm{shots}} \geqslant 10^4$, the difference is very small or negative, which indicates that in this regime adding the MPV to the loss contrasts the MSE minimization. For further minimizing the MSE one could continue the training potentially gradually decreasing λ, as mentioned above and done in [113]. Overall, the results presented here show that variance regularization is an effective approach for achieving a stable training of a QNN even with a moderate number of shots.

6. Discussion and outlook

In this paper we explore the potential of QNNs as parameterization schemes in climate models, focusing on the specific case of cloud cover parameterizations. With the goal of predicting cloud cover from coarse-grained state variables, we compare different QNN architectures with classical NNs of similar size (i.e. similar number of parameters) on several aspects: prediction accuracy, generalization ability and trainability. In all these aspects, the investigated QNNs show a similar overall performance to classical NNs, at least in absence of finite sampling noise. Our classical and quantum NNs show a higher prediction accuracy compared to standard parameterizations such as the Xu–Randall scheme which are operationally used in climate models (see also [23, 24] for a comparison of NNs with the Sundqvist scheme [74], operationally used in ICON-A [78]). Furthermore, our analysis shows that QNNs can be trained to learn a meaningful cloud cover even in the presence of finite sampling noise, which is an important

observation for their implementation on actual quantum devices. Overall, these observations are promising indications of the applicability of these models to learn patterns in climate data, while also highlighting potential architectural limitations that need to be addressed in future work. An important aspect concerns the QNNs representational capabilities and the functional form they can represent. In this work we used relatively standard QNN architectures based on angle encoding of the inputs, which are known to represent trigonometric functions with a number of accessible frequencies bounded by the number of re-uploads [86, 87]. For a limited number of re-uploads, this structure may be suboptimal for representing rapidly varying (highly non-linear) functions. The design of QNNs with a stronger inductive bias (i.e. a better suited functional form) requires further investigation of the specific meteorological conditions leading to these prediction deficits, such as those observed in this work over Amazonia, and likely involves more flexible input encoding strategies. For example, other strategies for accessing an exponentially large set of frequencies have been devised [101], and different types of basis functions, different than trigonometric ones, can be encoded [102], and potentially designed to fit the problem at hand.

Besides the application to cloud cover parameterizations, our work opens up several directions for further investigating the applicability of QML in climate models. We outline those in the following, while also discussing criticalities and potential limitations associated to them.

A first extension to our work would be to study the performance and learning behavior of the proposed architectures for problems of larger size, which would require moving away from the cell-based approach we took here, by considering also neighboring model cells (as done in [23] in the context of classical NNs). Furthermore, while we have performed a thorough analysis of possible QNN models, we are aware that our architecture search is not exhaustive. Notable interesting examples to investigate in future works could include quantum convolutional NNs [114] or QNNs with measurement feedforwards [115–118]. While increasing the size of the problem, and thus the number of qubits, may introduce trainability issues such as barren plateaus with increasing number of qubits [110–112], we note that there exist architectures that are immune to such problems [119], and parameter initialization strategies to mitigate them [120–123].

Our construction could be also tested on parameterization schemes other than cloud cover. Changing the parameterization scheme would likely result in a more complex learning task, which may also increase the chance of observing a separation between the quantum and classical ML algorithms for the considered dataset. In fact, it is to a large extent unknown what types of classical datasets are better suited to quantum learning models than to a classical ones, and extensive tests on several architectures and datasets as done here are very valuable to pinpoint features that may help answering this question.

Going beyond regression, another interesting extension would be to re-frame the parameterization learning problem in a probabilistic setting. Specifically, one could build quantum generative models [36–39] for learning the full probability distribution of the process considered (here cloud cover). Switching to generative modeling, a task that is to some extent more natural to quantum computing, may increase the chance of observing a separation between the quantum and classical models for the given problem [29].

Finally, another direction concerns the online implementation of the proposed QNNs, i.e. their coupling to the dynamical core of the climate model solving the Navier–Stokes equations. While QNNs consisting of a small number of qubits can be numerically simulated and thus in principle be readily used within a climate model, a larger number of qubits would require the coupling of a quantum device to the dynamical core. This quantum–classical coupling, especially at the level of high-performance computing facilities, poses challenges that are subject of active research (see, e.g. [124] for a recent review on the topic). Furthermore, given that parameterization schemes need to be run for every cell of the model at every time-step, such a coupling could become impractical in terms of computation runtime, given the high amount of data to be transferred between classical and quantum processors, and the need of running quantum computations multiple times to acquire predictions with sufficient accuracy. To address this limitation, a possibility would be to build classical surrogates [125–128] of the trained QNNs, to be then used online. In this way, the quantum device would be used only in the development stage of the parameterization, and our work lays a solid basis for developing such a workflow. Another possible direction are quantum-inspired methods such as tensor networks [129], which already find applications as (Q)ML models for data analysis [130–133], in classical data compression and loading [134–136], and generative modeling [137–139].

Acknowledgment

This project was made possible by the DLR Quantum Computing Initiative and the Federal Ministry for Research, Technology and Space; qci.dlr.de/projects/klim-qml. A G and V E were funded by the

European Research Council (ERC) Synergy Grant 'Understanding and Modelling the Earth System with Machine Learning (USMILE)' under the Horizon 2020 research and innovation programme (Grant Agreement No. 855187). V E was additionally supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) through the Gottfried Wilhelm Leibniz Prize awarded to Veronika Eyring (Reference No. EY 22/2-1). This work used resources of the Deutsches Klimarechenzentrum (DKRZ) granted by its Scientific Steering Committee (WLA) under project ID bd1179.

Data availability statement

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

Appendix A. Input and output distributions and transformations

In this appendix we discuss the input and output transformations that we applied to the DYAMOND data before feeding them to our classical and quantum models. For the input features, the idea behind the transformation is to make the feature distribution more uniform within a specified interval, and to still retain the input feature variability in the tails, which can be associated with physical scenarios we are interested in capturing. The transformation function we designed in this case reads as

$$h(x) = \frac{\log\left[1 + (\mathrm{e}-1)\left(\frac{x}{x_{\mathrm{high}}}\right)^{b}\right] - h_0\left(b, x_{\mathrm{low}}, x_{\mathrm{high}}\right)}{1 - h_0\left(b, x_{\mathrm{low}}, x_{\mathrm{high}}\right)}\,, \tag{22}$$

where $h_0(b, x_{\mathrm{low}}, x_{\mathrm{high}}) = \log\left[1 + (\mathrm{e}-1)\left(\frac{x_{\mathrm{low}}}{x_{\mathrm{high}}}\right)^{b}\right]$ and $x_{\mathrm{low}}, x_{\mathrm{high}}$ corresponding to the (approximate) minimum and maximum value of the given feature x estimated on the training dataset. We used the following parameters for the input features:

- specific humidity q_{v} [kg/kg]: $b = 0.25$, $x_{\mathrm{low}} = 10^{-7}$, $x_{\mathrm{high}} = 0.025$,
- cloud water q_{c} [kg/kg]: $b = 0.25$, $x_{\mathrm{low}} = 0$, $x_{\mathrm{high}} = 0.00145$,
- cloud ice q_{i} [kg/kg]: $b = 0.25$, $x_{\mathrm{low}} = 0$, $x_{\mathrm{high}} = 0.00055$,
- horizontal wind h_{w} [m/s]: $b = 0.5$, $x_{\mathrm{low}} = 0.0015$, $x_{\mathrm{high}} = 115.0$.

For all the features, these parameters have been manually selected to achieve a good balance between uniformity and distinguishability of the values in the tail of the resulting distribution. The so constructed transformations yield values approximately distributed in the interval $[0, 1]$. The remaining input features are transformed using a simple min-max scaling, and all features (including those listed above) are scaled within the interval $[0, \pi]$ (i.e. we multiplied the above $h(x)$ by a factor π). The transformations for q_{v}, q_{c} and q_{i} and the resulting histograms of the transformed values are shown in figure 8. In our experiments, the input features are then rescaled to the interval $[0, \pi]$.

Also the output transformation function $g(x)$ is constructed in order to have the training outputs (targets) in a more uniform distribution compared to the original one in the DYAMOND dataset, which in our case improved the performance of both our quantum and classical models. The transformation function is invertible, and reads as

$$g(x) = \frac{1}{2} + \frac{1}{\pi}\arcsin\left[2\left(\frac{\mathrm{e}^{bx^{a}} - 1}{\mathrm{e}^{b} - 1}\right)^{c} - 1\right] \tag{23}$$

with parameters $a = 1.29407913$, $b = -3.20011015$, $c = 0.70308237$, which have been chosen in order to have approximate uniformity. The transformation and the resulting transformed outputs histogram are shown in panels (g) and (h) of figure 8.

Appendix B. QNNs as Fourier models

In this appendix, we provide a brief description of the mapping between QNNs and Fourier models, which we hinted at in section 3.1. A full derivation can be found in [86, 87]. Our QNNs are of the form

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \langle 0|\hat{U}^{\dagger}_{\boldsymbol{\theta}}(\boldsymbol{x})\,\hat{M}\,\hat{U}_{\boldsymbol{\theta}}(\boldsymbol{x})\,|0\rangle\,, \tag{24}$$

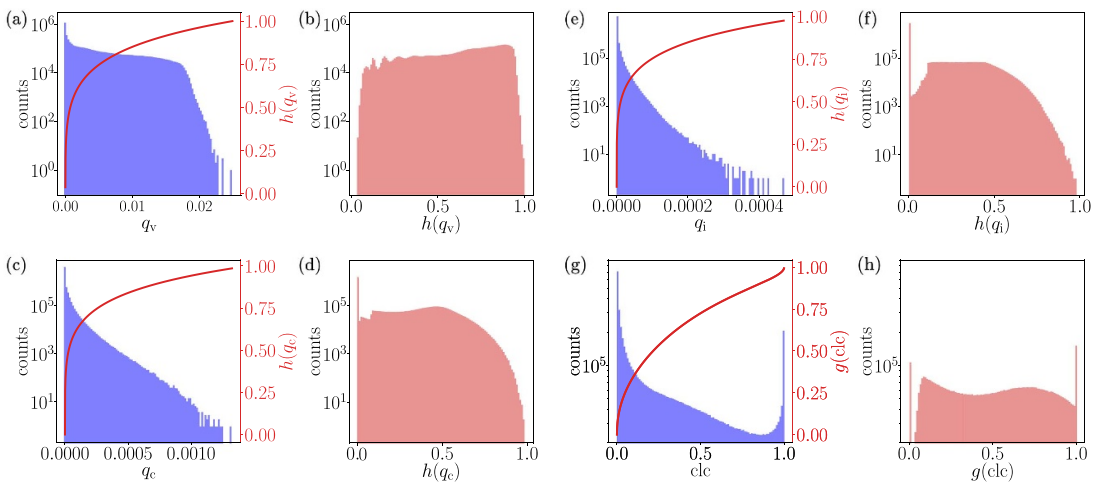

Figure 8. Input and output transformation functions and histograms. (a) Histogram of specific humidity q_v in the training dataset (blue) and transformation function $h(x)$ (red). (b) Histogram of the transformed $h(q_v)$. (c) Histogram of cloud water q_c in the training dataset (blue) and transformation function $h(x)$ (red). (d) Histogram of the transformed $h(q_c)$. (e) Histogram of cloud ice q_i in the training dataset (blue) and transformation function $h(x)$ (red). (f) Histogram of the transformed $h(q_i)$. (g) Histogram of cloud cover clc in the training dataset (blue) and transformation function $g(x)$ (red). (h) Histogram of the transformed $g(\mathrm{clc})$.

where $\hat{M}$ is a given observable (e.g. $\sum_{n=1}^{N} w_n \hat{\sigma}_n^z$ in the main text) and $\hat{U}_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the unitary operator describing the PQC (e.g. equation (2) in the main text). We consider the situation in the main text, i.e. that of angle encoding of the form $\hat{S}(\boldsymbol{x}) = \hat{R}_\alpha(\boldsymbol{\varphi}) = \prod_{n=1}^{N} \mathrm{e}^{-\mathrm{i}\frac{\varphi_n}{2}\hat{\sigma}_n^\alpha}$ $(\alpha = x, y, z)$, with n_{enc} layers of data re-uploading. In this situation, it is easy to show [86, 87] that the QNN output takes the following form:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{\boldsymbol{\omega}\in\Omega} c_{\boldsymbol{\omega}}(\boldsymbol{\theta})\, \mathrm{e}^{\mathrm{i}\boldsymbol{\omega}\cdot\boldsymbol{x}}\,, \tag{25}$$

i.e. a linear combination of trigonometric functions $\mathrm{e}^{\mathrm{i}\boldsymbol{\omega}\cdot\boldsymbol{x}}$ (here $\cdot$ denotes the scalar product of two real vectors), where the coefficients $c_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ carry the dependence on the trainable parameters $\boldsymbol{\theta}$ and satisfy $c_{\boldsymbol{\omega}}(\boldsymbol{\theta})^* = c_{-\boldsymbol{\omega}}(\boldsymbol{\theta})$ to have $f_{\boldsymbol{\theta}}(\boldsymbol{x}) \in \mathbb{R}$. The frequencies $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_N)$ available to the model depend on the encoding and the number of re-uploads, and we have

$$\omega_n \in \{-n_{\mathrm{enc}}, -n_{\mathrm{enc}}+1, \ldots, n_{\mathrm{enc}}-1, n_{\mathrm{enc}}\}\,, \tag{26}$$

for $n = 1, \ldots, N$. This means that the number of accessible frequencies to our QNN models, i.e. their Fourier expressivity, is bounded by the number of data re-uploads n_{enc}, and hence they are biased to accurately learning trigonometric functions with the corresponding number of Fourier modes.

Appendix C. Other QNN architectures tested

In this appendix we provide the details of the other QNN architectures tested in our work but not shown in the main text. The CNOT-PBC architecture contains as entangling gates CNOT gates arranged in a chain ring pattern. The encoding layer for this architecture is $\hat{S}_{\mathrm{CNOT-PBC}}(\boldsymbol{x}) = \hat{R}_x(\boldsymbol{x})$. The encoding blocks for the CNOT-PBC read as

$$\begin{aligned}\hat{V}_{\mathrm{CNOT-PBC}}(\boldsymbol{\vartheta}) = {} & \hat{R}_z\left(\boldsymbol{\vartheta}_{(N+1)\to 2N}\right) \\ & \times \hat{R}_y\left(\boldsymbol{\vartheta}_{1\to N}\right) \widehat{\mathrm{CNOT}}_{\mathrm{PBC}}\,,\end{aligned} \tag{27}$$

with $\widehat{\mathrm{CNOT}}_{\mathrm{PBC}} = \widehat{\mathrm{CNOT}}_{N,1} \prod_{n=1}^{N-1} \widehat{\mathrm{CNOT}}_{n,n+1}$, and the variational blocks as

$$\begin{aligned}\hat{W}_{\mathrm{CNOT-PBC}}(\boldsymbol{\varphi}) = {} & \hat{R}_x\left(\boldsymbol{\varphi}_{(2N+1)\to 3N}\right) \hat{R}_z\left(\boldsymbol{\varphi}_{(N+1)\to 2N}\right) \\ & \times \hat{R}_y\left(\boldsymbol{\varphi}_{1\to N}\right) \widehat{\mathrm{CNOT}}_{\mathrm{PBC}}\,.\end{aligned} \tag{28}$$

The CNOT-NN architecture contains as entangling gates CNOT gates acting between neighboring qubits in a chain. The encoding layer for this architecture is $\hat{S}_{\mathrm{CNOT-NN}}(\boldsymbol{x}) = \hat{R}_x(\boldsymbol{x})$. The encoding blocks for the CNOT-NN read as

$$\hat{V}_{\mathrm{CNOT-NN}}(\boldsymbol{\vartheta}) = \hat{R}_z\left(\boldsymbol{\vartheta}_{(N+1)\to 2N}\right)\hat{R}_y\left(\boldsymbol{\vartheta}_{1\to N}\right)\widehat{\mathrm{CNOT}}_{\mathrm{NN}}\,, \tag{29}$$

with $\widehat{\mathrm{CNOT}}_{\mathrm{NN}} = \prod_{n=1}^{N-1}\widehat{\mathrm{CNOT}}_{n,n+1}$, and the variational blocks as

$$\begin{aligned}\hat{W}_{\mathrm{CNOT-NN}}(\boldsymbol{\varphi}) = \hat{R}_x\left(\boldsymbol{\varphi}_{(2N+1)\to 3N}\right)\hat{R}_z\left(\boldsymbol{\varphi}_{(N+1)\to 2N}\right)\\ \times\hat{R}_y\left(\boldsymbol{\varphi}_{1\to N}\right)\widehat{\mathrm{CNOT}}_{\mathrm{NN}}\,.\end{aligned} \tag{30}$$

The IONS architecture contains as entangling operation one that is naturally implemented in trapped ions quantum simulators, generated by a long-range Hamiltonian of the form $\sum_{n=1}^{N-1}\sum_{m<n}\frac{\hat{\sigma}_n^x\hat{\sigma}_m^x}{m-n}$, coming from the laser coupling of the ions internal states to the center-of-mass vibrational mode of the ion chain. The IONS architecture takes as input the state $\prod_{n=1}^{N}\hat{\mathrm{H}}_n|0\rangle$ with $\hat{\mathrm{H}}_n$ being the Hadamard gate on qubit n. The encoding layer for this architecture is $\hat{S}_{\mathrm{IONS}}(\boldsymbol{x}) = \hat{R}_z(\boldsymbol{x})$. The encoding blocks for the IONS architecture read as

$$\hat{V}_{\mathrm{IONS}}(\boldsymbol{\vartheta}) = \hat{R}_y\left(\boldsymbol{\vartheta}_{2\to(N+1)}\right)\hat{U}_{\mathrm{IONS}}(\vartheta_1)\,, \tag{31}$$

with $\hat{U}_{\mathrm{IONS}}(\vartheta) = \exp(-\frac{\mathrm{i}\vartheta}{2}\sum_{n<m}\frac{\hat{\sigma}_n^x\hat{\sigma}_m^x}{m-n})$, and the variational blocks

$$\begin{aligned}\hat{W}_{\mathrm{IONS}}(\boldsymbol{\varphi}) = \hat{R}_y\left(\boldsymbol{\varphi}_{(2N+2)\to(3N+1)}\right)\hat{U}_{\mathrm{IONS}}\left(\varphi_{(2N+1)}\right)\\ \times\hat{R}_z\left(\boldsymbol{\varphi}_{(N+1)\to 2N}\right)\hat{R}_x\left(\boldsymbol{\varphi}_{1\to N}\right)\,.\end{aligned} \tag{32}$$

Additionally, we also experimented with augmenting the CNOT-PBC and CNOT-NN architectures with higher-order angle encoding layers of the form $\hat{S}_{\mathrm{HONE}}(\boldsymbol{x}) = \hat{R}_x(\boldsymbol{x}^{[2]})\hat{R}_x(\boldsymbol{x})$, with $\boldsymbol{x}^{[2]}$ being a vector with $(N-1)$ components which are computed from $\boldsymbol{x}$ as $x_m^{[2]} = \frac{x_m x_{m+1}}{2\pi}$.

Appendix D. Details on classical NNs tested

In this appendix we detail the structure of the classical NNs used as comparison for our QNNs. The NNs presented here are the result of an extensive architecture search with the constraint of keeping the number of parameters approximately equal to that of the QNNs discussed in the main text. We start from the networks taking as inputs the eight input features $\{q_{\mathrm{v}}, q_{\mathrm{c}}, q_{\mathrm{i}}, T, p, z_{\mathrm{g}}, h_{\mathrm{w}}, \phi\}$, and containing a number D of trainable parameters between 200 and 210. In the following lists, the number denotes the number of nodes in the given layer, and in parentheses we write the activation function used.

- 8 (inputs) → 10 (tanh) → 7 (tanh) → 4 (tanh) → 1 (linear—output).
- 8 (inputs) → 9 (tanh) → 4 (tanh) → 9 (tanh) → 4 (tanh) → 1 (linear—output).
- 8 (inputs) → 8 (tanh) → 8 (tanh) → 6 (tanh) → 1 (linear—output).
- 8 (inputs) → 12 (tanh) → 6 (tanh) → 2 (tanh) → 1 (linear—output): best performing, chosen for figure 3.
- 8 (inputs) → 10 (tanh) → 10 (tanh) → 1 (linear—output).
- 8 (inputs) → 8 (tanh) → 8 (tanh) → 4 (tanh) → 4 (tanh) → 1 (linear—output).

For the networks taking as inputs the six input features $\{q_{\mathrm{v}}, q_{\mathrm{c}}, q_{\mathrm{i}}, T, p, h_{\mathrm{w}}\}$, containing a number D of trainable parameters between 109 and 120, we used the following layouts.

- 6 (inputs) → 8 (tanh) → 3 (tanh) → 7 (tanh) → 1 (linear—output): best performing, chosen for figure 5.
- 6 (inputs) → 6 (tanh) → 5 (tanh) → 5 (tanh) → 1 (linear—output).
- 6 (inputs) → 7 (tanh) → 3 (tanh) → 7 (tanh) → 2 (tanh) → 1 (linear—output).
- 6 (inputs) → 8 (tanh) → 3 (leaky-ReLU) → 3 (tanh) → 2 (tanh) → 2 (leaky-ReLU) → 2 (tanh) → 1 (linear—output).
- 6 (inputs) → 10 (tanh) → 4 (tanh) → 1 (linear—output).
- 6 (inputs) → 5 (tanh) → 5 (tanh) → 4 (tanh) → 4 (tanh) → 1 (linear—output).

All the architectures in the last list are the ones also used for generating the plots in figures 4 and 11. For these architectures as well as for the quantum ones, the initial learning rate for the Adam optimizer is set to 0.001 and the batch size to 100, which are approximately optimal settings in both classical and quantum cases for our problem.

Appendix E. Other evaluation metrics

In this appendix we discuss further metrics that can be used to evaluate our quantum and classical networks beyond MSE and R^2. The additional metrics we compute measure the distance between the distributions of the model predictions and the distribution of cloud cover in a testing dataset. Specifically, we calculate the Hellinger distance:

$$\mathrm{HD}\,(P,Q) = \frac{1}{\sqrt{2}}\left\|\sqrt{P}-\sqrt{Q}\right\|_2 , \tag{33}$$

where P and Q are the predicted and the true (discretized) probability distributions for cloud cover, and $\|\cdot\|_2$ denotes the Euclidean norm. Additionally, we calculate the Wasserstein distance

$$\mathrm{WD}\,(P,Q) = \inf_{\Pi\in\Gamma(P,Q)} \mathbb{E}_{(x,y)\sim\Pi}\left(\|x-y\|\right) , \tag{34}$$

where $\Gamma(P,Q)$ is the set of all joint probability distributions that have P and Q as marginals (we calculate WD using the SciPy package in Python).

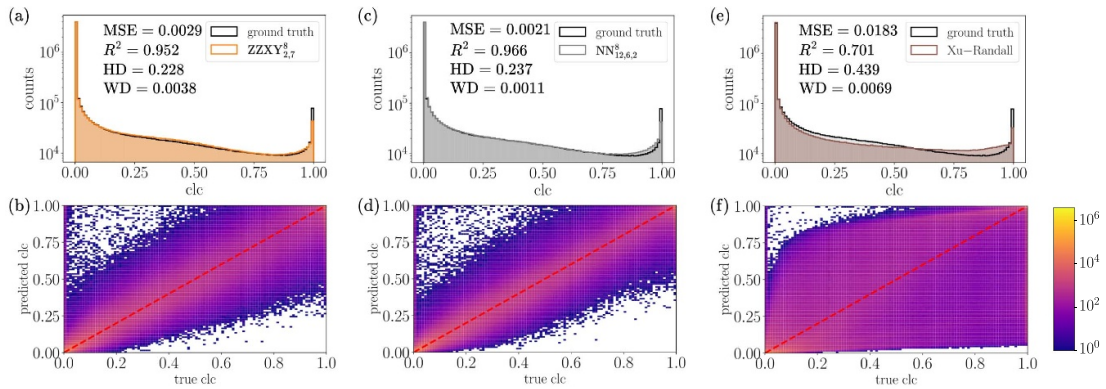

Figure 9. Output distributions. (a) Histogram of predicted (blue) and true (black) clc from ZZXY QNN. (b) Histogram of true vs. predicted clc for ZZXY QNN (logarithmic color scale). (c) Histogram of predicted (orange) and true (black) clc from classical NN. (d) Histogram of true vs. predicted clc for classical NN (logarithmic color scale). (e) Histogram of predicted (brown) and true (black) clc from Xu–Randall scheme. (f) Histogram of true vs. predicted clc for Xu–Randall scheme (logarithmic color scale). The histograms are obtained for a testing set of 6×10^6 data points. We report also the values of the mean squared error (MSE), R^2 coefficient, Hellinger distance (HD) and Wasserstein distance (WD) on the same testing set.

In figure 9 we show the histograms of the outputs from our ZZXY and NN architectures (the XYZ architecture has very similar performance and we do not show it here), compared with the Xu–Randall scheme, for the QNNs with eight input features used in section 4.1, and report also the values of the additional distance metrics just discussed. The histograms together with the additional metrics further confirm the observations made in the main text, namely, that the quantum and classical NNs perform comparably on the task at hand, with the classical ones being slightly better, while both outperform the Xu–Randall scheme. In figure 10 we show these metrics together with the MSE for all the QNNs tested in this work, including those not shown in the main text but described in appendix C. Here we can observe that all QNNs tested have comparable performance in the noiseless case, which is why we show only two types of QNNs in the main text.

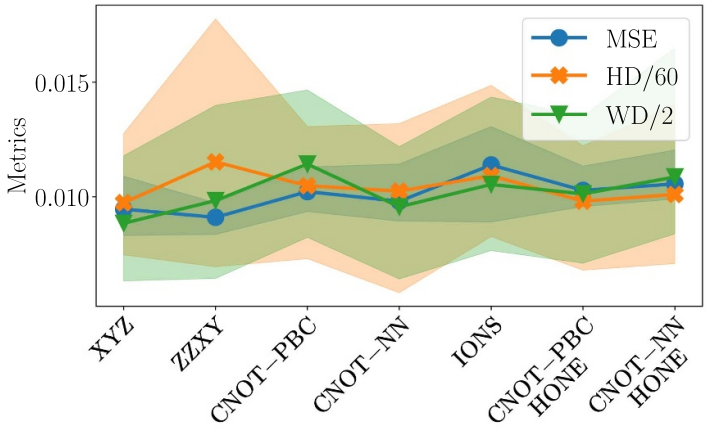

Figure 10. Evaluation metrics for the different QNNs tested: mean squared error (MSE, in blue), Hellinger distance (HD, in orange) and Wasserstein distance (WD, in green). HD and WD have been rescaled in order for all the metrics to have values in the same order of magnitude (scaling factor in the legend). The shaded area corresponds to the spread over 20 training instances. All QNNs for this plot have six input features $\{q_v, q_c, q_i, T, p, h_w\}$, the number of trainable parameters is between 110 and 120, and are trained with 2×10^5 training data for 150 epochs.

Appendix F. Details on Fisher information matrix (FIM) calculation and other trainability metrics

In this appendix, we give more details on the calculation of the FIM presented in the main text, and provide a further analysis of the training dynamics of our networks. We start by deriving the formula used for calculating the FIM in our case of regression with MSE loss function. For a statistical model $p(\boldsymbol{x}, y; \boldsymbol{\theta})$ the elements of the FIM are defined as [103–105]

$$F_{j,k}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p}\left[\frac{\partial \log p(\boldsymbol{x}, y; \boldsymbol{\theta})}{\partial \theta_j}\frac{\partial \log p(\boldsymbol{x}, y; \boldsymbol{\theta})}{\partial \theta_k}\right], \tag{35}$$

which can be rewritten as

$$F_{j,k}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p}\left[\frac{\partial \log p_{\boldsymbol{\theta}}(y|\boldsymbol{x})}{\partial \theta_j}\frac{\partial \log p_{\boldsymbol{\theta}}(y|\boldsymbol{x})}{\partial \theta_k}\right], \tag{36}$$

by noticing that $p(\boldsymbol{x}, y; \boldsymbol{\theta}) = p(\boldsymbol{x}) p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$, with $p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ being the model output probability conditioned on the input $\boldsymbol{x}$. The quantity $\ell(y, \boldsymbol{x}; \boldsymbol{\theta}) = -\log p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ corresponds to the negative log-likelihood, a typical loss function used for statistical models. In our case, with our networks outputting a deterministic value $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ the loss function corresponds to the MSE, which amounts to setting $p_{\boldsymbol{\theta}}(y|\boldsymbol{x}) = \mathcal{N}_{f_{\boldsymbol{\theta}}(\boldsymbol{x}),\sigma^2}(y)$ for a fictitious σ [106, 107]. Therefore, using $\partial_{\theta_j} p_{\boldsymbol{\theta}}(y|\boldsymbol{x}) = \sigma^{-2}(f_{\boldsymbol{\theta}}(\boldsymbol{x}) - y)\,\partial_{\theta_j} f_{\boldsymbol{\theta}}(\boldsymbol{x})$, we can rewrite the FIM as

$$\begin{aligned} F_{j,k}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}}\Bigg[\int \mathcal{N}_{f_{\boldsymbol{\theta}}(\boldsymbol{x}),\sigma^2}(y)\,(f_{\boldsymbol{\theta}}(\boldsymbol{x}) - y)^2\,\sigma^{-4} \\ \times \frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_j}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_k}\,\mathrm{d}y\Bigg]. \end{aligned} \tag{37}$$

Performing the Gaussian integration over y, we finally arrive at

$$F_{j,k}(\boldsymbol{\theta}) = \sigma^{-2}\,\mathbb{E}_{\boldsymbol{x}}\left[\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_j}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial \theta_k}\right], \tag{38}$$

which is proportional to the definition given in equation (14).

We now move on to discussing further aspects of the training dynamics of our networks beyond the training curves shown in figure 5 in the main text. To this end, we not only monitor the value of the MSE loss during training, but also track the dynamics of the parameters $\boldsymbol{\theta}$, viewing each training experiment starting from a random parameter configuration as a walker in the parameter space. We denote with $\boldsymbol{\theta}^{(m)}(t)$ $(m = 1, \ldots, M)$ the parameter configuration of the mth walker at a selected training step t.

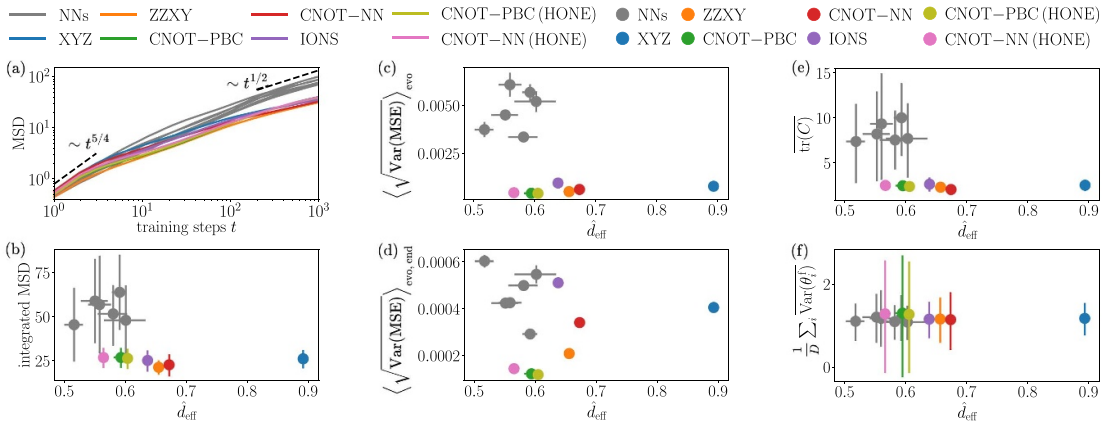

Figure 11. Analysis of training dynamics of classical and quantum networks. (a) Mean squared displacement (MSD) evolution during training. Here each step corresponds the evolution after 100 batch updates during the Adam optimization (we use 10^5 training samples, so each epoch consists of 1000 batch updates with a batch size of 100). The dashed lines are guides to the eye indicating specific slopes. (b) Integrated MSD (over time) against normalized effective dimension $\hat{d}_{\text{eff}}$. (c) Spread of MSE loss function averaged over all training epochs plotted against $\hat{d}_{\text{eff}}$. (d) Spread of MSE loss function averaged over the final 25 training epochs plotted against $\hat{d}_{\text{eff}}$. (e) Trace of parameter correlation matrix C averaged over all 200 training instances (walkers), plotted against $\hat{d}_{\text{eff}}$. (f) Mean spread of final parameters plotted against $\hat{d}_{\text{eff}}$. The error bars correspond to the estimated statistical uncertainty bootstrapped from the 200 random parameter realizations. All networks have six input features and the number of trainable parameters is between 109 and 120.

A quantity that is natural to analyze when looking at ensembles of random walkers is the mean squared displacement (MSD), defined as

$$\text{MSD}\,(t) = \overline{\|\boldsymbol{\theta}\,(t) - \boldsymbol{\theta}\,(0)\,\|^2} \equiv \frac{1}{M}\sum_{m=1}^{M} \|\boldsymbol{\theta}^{(m)}\,(t) - \boldsymbol{\theta}^{(m)}\,(0)\,\|^2\,, \tag{39}$$

see figure 11(a). We observe the same qualitative behavior for both classical and quantum networks, with the initial dynamics being faster (super-diffusive), then progressively slowing down to a sub-diffusive regime when the gradients have significantly decreased in magnitude. Interestingly, it appears that for the QNNs tested here this slow-down of the training dynamics happens faster compared to classical NNs. Attempting to correlate this observation with the geometrical properties of the models captured by the FIM, we consider the integrated MSD, $\frac{1}{T}\sum_{t=1}^{T}\text{MSD}(t)$ as function of the normalized effective dimension $\hat{d}_{\text{eff}}$ defined in equation (15) (figure 11(b)). While we still observe a difference between classical and quantum architectures, no clear correlation between the integrated MSD and the value of $\hat{d}_{\text{eff}}$ can be seen. The absence of a clear correlation between the training dynamics and $\hat{d}_{\text{eff}}$ can also be seen in panels (c) and (e). In panel (c) we plot the average of the spread of the MSE loss during training, which we compute as

$$\left\langle \sqrt{\text{Var}\,(\text{MSE})} \right\rangle_{\text{evo}} = \frac{1}{N_{\text{epochs}}}\sum_{j=1}^{N_{\text{epochs}}} \sqrt{\text{Var}\,(\text{MSE}\,(j))}\,, \tag{40}$$

where $\text{Var}(\text{MSE}(j))$ is the variance of the training MSE at epoch j, calculated over the different walkers. In panel (e) we show the trace of parameter correlation matrix C averaged over all M walkers, as a measure of the cumulative variance of the parameters during their evolution. Specifically, the parameters correlation matrix for the mth walker is calculated as

$$C^{(m)} = \frac{1}{T}\,\Theta^{(m)\,\top}\Theta^{(m)}\,, \tag{41}$$

where $\Theta^{(m)}$ is a $(T \times D)$ centered data matrix with elements $\Theta^{(m)}_{t,i} = \theta^{(m)}_j(t) - \frac{1}{T}\sum_t \theta^{(m)}_j(t)$. The eigenvectors of $C^{(m)}$ associated to the largest eigenvalues denote the directions along which the parameters have changed the most, and the associated eigenvalues their variance. Hence

$$\overline{\text{tr}\,(C)} \equiv \frac{1}{M}\sum_{m=1}^{M} \text{tr}\left(C^{(m)}\right) \tag{42}$$

gives an estimate of the parameters cumulative variance during the evolution. Despite the two quantities sharing similar behavior, from panels (c) and (e) no correlation with $\hat{d}_{\text{eff}}$ can be concluded. The same holds when focusing on only the final part of the training dynamics, as shown in panel (d), as well as when looking at the mean spread of the final parameters $\frac{1}{D}\sum_{i=1}^{D}\text{Var}(\theta_i^{\text{f}})$, with $\theta_i^{\text{f}} = \theta_i(T)$ and the variance taken over the walkers ensemble, which is shown in panel (f). Thus, to summarize, in our analysis we see no clear correlation between the geometrical properties captured by the FIM and the training dynamics of our networks.

In particular, we expect the effectiveness of training to be dependent not only on the model ability to effectively explore the parameter space, but also on how well the functions it can represent are suited to the problem at hand (i.e. its inductive bias), an aspect to which the FIM is agnostic. We refer the reader to [109] for a recent work on this topic. Hence, given the slightly better performance of NNs on our task, their structure may be more favorable for learning cloud cover, which in turn has a positive impact on the training dynamics.

ORCID iDs

Lorenzo Pastori 0000-0001-5882-8482
Arthur Grundner 0000-0002-3765-242X
Mierk Schwabe 0000-0001-6565-5890

References

[1] Jacobson M Z 2005 *Fundamentals of Atmospheric Modeling* (Cambridge University Press)
[2] Gettelman A and Rood R B 2016 *Demystifying Climate Models* (Springer)
[3] Sherwood S C, Bony S and Dufresne J-L 2014 *Nature* **505** 37–42
[4] Bock L, Lauer A, Schlund M, Barreiro M, Bellouin N, Jones C, Meehl G A, Predoi V, Roberts M J and Eyring V 2020 *J. Geophys. Res.* **125** e2019JD032321
[5] Stensrud D J 2007 *Parameterization Schemes: Keys to Understanding Numerical Weather Prediction Models* (Cambridge University Press)
[6] McFarlane N 2011 *WIREs Clim. Change* **2** 482–97
[7] Christensen H and Zanna L 2022 Parametrization in weather and climate models
[8] Randall D, Khairoutdinov M, Arakawa A and Grabowski W 2003 *Bull. Am. Meteorol. Soc.* **84** 1547–64
[9] Schneider T, Teixeira J, Bretherton C S, Brient F, Pressel K G, Schär C and Siebesma A P 2017 *Nat. Clim. Change* **7** 3–5
[10] Eyring V, Mishra V, Griffith G P, Chen L, Keenan T, Turetsky M R, Brown S, Jotzo F, Moore F C and van der Linden S 2021 *Nat. Clim. Change* **11** 279–85
[11] Gentine P, Eyring V and Beucler T 2021 Deep learning for the parametrization of subgrid processes in climate models *Deep Learning for the Earth Sciences* 2nd edn, ed G Camps-Valls, D Tuia, X X Zhu and M Reichstein (Wiley)
[12] Eyring V *et al* 2024 *Nat. Clim. Change* **14** 916–28
[13] Eyring V, Gentine P, Camps-Valls G, Lawrence D M and Reichstein M 2024 *Nat. Geosci.* **17** 963–71
[14] Chevallier F, Chéruy F, Scott N A and Chédin A 1998 *J. Appl. Meteorol.* **37** 1385–97
[15] Krasnopolsky V M, Fox-Rabinovitz M S and Chalikov D V 2005 *Mon. Weather Rev.* **133** 1370–83
[16] Lagerquist R, Turner D, Ebert-Uphoff I, Stewart J and Hagerty V 2021 *J. Atmos. Ocean. Technol.* **38** 1673–96
[17] Hafner K, Iglesias-Suarez F, Shamekh S, Gentine P, Giorgetta M A, Pincus R and Eyring V 2025 *J. Geophys. Res.* **2** e2024JH000501
[18] Yuval J and O'Gorman P A 2020 *Nat. Commun.* **11** 3295
[19] Rasp S, Pritchard M S and Gentine P 2018 *Proc. Natl Acad. Sci.* **115** 9684–9
[20] Gentine P, Pritchard M, Rasp S, Reinaudi G and Yacalis G 2018 *Geophys. Res. Lett.* **45** 5742–51
[21] Brenowitz N D and Bretherton C S 2019 *J. Adv. Modeling Earth Syst.* **11** 2728–44
[22] Heuer H, Schwabe M, Gentine P, Giorgetta M A and Eyring V 2024 *J. Adv. Modeling Earth Syst.* **16** e2024MS004398
[23] Grundner A, Beucler T, Gentine P, Iglesias-Suarez F, Giorgetta M A and Eyring V 2022 *J. Adv. Modeling Earth Syst.* **14** e2021MS002959
[24] Grundner A, Beucler T, Gentine P and Eyring V 2024 *J. Adv. Modeling Earth Syst.* **16** e2023MS003763
[25] Perdomo-Ortiz A, Benedetti M, Realpe-Gómez J and Biswas R 2018 *Quantum Sci. Technol.* **3** 030502
[26] Benedetti M, Lloyd E, Sack S and Fiorentini M 2019 *Quantum Sci. Technol.* **4** 043001
[27] Cerezo M, Verdon G, Huang H-Y, Cincio L and Coles P J 2022 *Nat. Comput. Sci.* **2** 567–76
[28] Farhi E and Neven H 2018 Classification with quantum neural networks on near term processors (arXiv:1802.06002)
[29] Du Y, Hsieh M-H, Liu T and Tao D 2020 *Phys. Rev. Res.* **2** 033125
[30] Yu Z, Chen Q, Jiao Y, Li Y, Lu X, Wang X and Yang J Z 2024 Non-asymptotic approximation error bounds of parameterized quantum circuits (arXiv:2310.07528)
[31] Caro M C, Gil-Fuster E, Meyer J J, Eisert J and Sweke R 2021 *Quantum* **5** 582
[32] Banchi L, Pereira J and Pirandola S 2021 *PRX Quantum* **2** 040321
[33] Caro M C, Huang H-Y, Cerezo M, Sharma K, Sornborger A, Cincio L and Coles P J 2022 *Nat. Commun.* **13** 4919
[34] Haug T and Kim M S 2024 *Phys. Rev. Lett.* **133** 050603
[35] Abbas A, Sutter D, Zoufal C, Lucchi A, Figalli A and Woerner S 2021 *Nat. Comput. Sci.* **1** 403–9
[36] Amin M H, Andriyash E, Rolfe J, Kulchytskyy B and Melko R 2018 *Phys. Rev. X* **8** 021050
[37] Dallaire-Demers P-L and Killoran N 2018 *Phys. Rev. A* **98** 012324
[38] Coyle B, Mills D, Danos V and Kashefi E 2020 *npj Quantum Inf.* **6** 60
[39] Gao X, Anschuetz E R, Wang S T, Cirac J I and Lukin M D 2022 *Phys. Rev. X* **12** 021037
[40] Havlíček V, Córcoles A D, Temme K, Harrow A W, Kandala A, Chow J M and Gambetta J M 2019 *Nature* **567** 209–12

[41] Schuld M and Killoran N 2019 *Phys. Rev. Lett.* **122** 040504
[42] Hur T, Kim L and Park D K 2022 *Quantum Mach. Intell.* **4** 3
[43] Shen K, Jobst B, Shishenina E and Pollmann F 2024 Classification of the fashion-mnist dataset on a quantum computer (arXiv:2403.02405)
[44] Chen S Y C, Wei T-C, Zhang C, Yu H and Yoo S 2022 *Phys. Rev. Res.* **4** 013231
[45] Belis V, Woźniak K A, Puljak E, Barkoutsos P, Dissertori G, Grossi M, Pierini M, Reiter F, Tavernelli I and Vallecorsa S 2024 *Commun. Phys.* **7** 334
[46] Slabbert D, Lourens M and Petruccione F 2024 *Quantum Mach. Intell.* **6** 56
[47] Sünkel L, Martyniuk D, Reichwald J J, Morariu A, Seggoju R H, Altmann P, Roch C and Paschke A 2023 Hybrid quantum machine learning assisted classification of covid-19 from computed tomography scans *2023 IEEE Int. Conf. on Quantum Computing and Engineering (QCE)* (IEEE)) pp 356–66
[48] Sakhnenko A, Sikora J and Lorenz J 2024 Buildung continuous quantum-classical Bayesian neural networks for a classical clinical dataset *Proc. Recent Advances in Quantum Computing and Technology* (*ReAQCT'24*) (ACM) pp 62–72
[49] Corli S, Moro L, Dragoni D, Dispenza M and Prati E 2024 Quantum machine learning algorithms for anomaly detection: a survey (arXiv:2408.11047)
[50] Duneau T, Bruhn S, Matos G, Laakkonen T, Saiti K, Pearson A, Meichanetzidis K and Coecke B 2024 Scalable and interpretable quantum natural language processing: an implementation on trapped ions (arXiv:2409.08777)
[51] Aizpurua B, Jahromi S S, Singh S and Orus R 2024 Quantum large language models via tensor network disentanglers (arXiv:2410.17397)
[52] Tennie F and Palmer T N 2023 *Bull. Am. Meteorol. Soc.* **104** E488–500
[53] Jaderberg B, Gentile A A, Ghosh A, Elfving V E, Jones C, Vodola D, Manobianco J and Weiss H 2024 Potential of quantum scientific machine learning applied to weather modelling (arXiv:2404.08737)
[54] Matsuta T and Furue R 2024 Formulation and evaluation of ocean dynamics problems as optimization problems for quantum annealing machines (arXiv:2405.11782)
[55] Bazgir A and Zhang Y 2024 Qesm: a leap towards quantum-enhanced ml emulation framework for earth and climate modeling (arXiv:2410.01551)
[56] Otgonbaatar S and Kranzlmüller D 2023 Quantum-inspired tensor network for earth science (arXiv:2301.07528)
[57] Nivelkar M, Bhirud S, Singh M, Ranjan R and Kumar B 2023 *IEEE Access* **11** 70679–90
[58] Nammouchi A, Kassler A and Theorachis A 2023 Quantum machine learning in climate change and sustainability: a review (arXiv:2310.09162)
[59] Ho K T M, Chen K C, Lee L, Burt F, Yu S, Po-Heng L 2024 Quantum computing for climate resilience and sustainability challenges (arXiv:2407.16296)
[60] Lachure S S, Lohidasan A, Tiwari A, Dhabu M and Bokde N D 2023 *Quantum Machine Learning Applications to Address Climate Change: A Short Review* (IGI Global Scientific Publishing) ch 4
[61] Rahman S M, Alkhalaf O H, Alam M S, Tiwari S P, Shafiullah M, Al-Judaibi S M and Al-Ismail F S 2024 *Earth Syst. Environ.* **8** 705–22
[62] Schwabe M, Pastori L, de Vega I, Gentine P, Iapichino L, Lahtinen V, Leib M, Lorenz J M and Eyring V 2025 *Environ. Data Sci.* **4** e35
[63] Preskill J 2018 *Quantum* **2** 79
[64] Harrow A W and Montanaro A 2017 *Nature* **549** 203–9
[65] Cerezo M *et al* 2024 Does provable absence of barren plateaus imply classical simulability? Or, why we need to rethink variational quantum computing (arXiv:2312.09121)
[66] Gil-Fuster E, Gyurik C, Pérez-Salinas A and Dunjko V 2024 On the relation between trainability and dequantization of variational quantum learning models (arXiv:2406.07072)
[67] Bermejo P, Braccia P, Rudolph M S, Holmes Z, Cincio L and Cerezo M 2024 Quantum convolutional neural networks are (effectively) classically simulable (arXiv:2408.12739)
[68] Kölle M, Maurer J, Altmann P, Sünkel L, Stein J and Linnhoff-Popien C 2024 Disentangling quantum and classical contributions in hybrid quantum machine learning architectures (arXiv:2311.05559)
[69] Bowles J, Ahmed S and Schuld M 2024 Better than classical? The subtle art of benchmarking quantum machine learning models (arXiv:2403.07059)
[70] Quaas J 2012 *J. Geophys. Res.* **117** D09208
[71] Lohmann U and Roeckner E 1996 *Clim. Dyn.* **12** 557–72
[72] Pincus R and Stevens B 2013 *J. Adv. Modeling Earth Syst.* **5** 225–33
[73] Sundqvist H 1978 *Q. J. R. Meteorol. Soc.* **104** 677–90
[74] Sundqvist H, Berge E and Kristjánsson J E 1989 *Mon. Weather Rev.* **117** 1641–57
[75] Xu K-M and Randall D A 1996 *J. Atmos. Sci.* **53** 3084–102
[76] Teixeira J 2001 *Mon. Weather Rev.* **129** 1750–3
[77] Walcek C J 1994 *Mon. Weather Rev.* **122** 1021–35
[78] Giorgetta M A *et al* 2018 *J. Adv. Modeling Earth Syst.* **10** 1613–37
[79] Stevens B *et al* 2019 *Prog. Earth Planet. Sci.* **6** 61
[80] Duras J, Ziemen F and Klocke D 2021 The DYAMOND winter data collection *EGU General Assembly 2021, (19 April–30 April 2021)*
[81] Nielsen M A and Chuang I L 2010 *Quantum Computation and Quantum Information: 10th Anniversary Edn* (Cambridge University Press)
[82] Stephan C C, Duras J, Harris L, Klocke D, Putman W M, Taylor M, Wedi N P, Žagar N and Ziemen F 2022 *Tellus B* **74** 280–99
[83] Stevens B *et al* 2020 *J. Meteorol. Soc. Japan II* **98** 395–435
[84] Giorgetta M A *et al* 2022 *Geosci. Model Dev.* **15** 6985–7016
[85] Pérez-Salinas A, Cervera-Lierta A, Gil-Fuster E and Latorre J I 2020 *Quantum* **4** 226
[86] Schuld M, Sweke R and Meyer J J 2021 *Phys. Rev. A* **103** 032430
[87] Casas B and Cervera-Lierta A 2023 *Phys. Rev. A* **107** 062612
[88] Bergholm V *et al* 2022 Pennylane: automatic differentiation of hybrid quantum-classical computations (arXiv:1811.04968)
[89] JAX 2024 JAX: high performance array computing (available at: https://jax.readthedocs.io/en/latest/index.html)
[90] McClean J R, Romero J, Babbush R and Aspuru-Guzik A 2016 *New J. Phys.* **18** 023023

[91] Cerezo M et al 2021 Nat. Rev. Phys. 3 625–44
[92] Kingma D P and Ba J 2017 Adam: a method for stochastic optimization (arXiv:1412.6980)
[93] Mitarai K, Negoro M, Kitagawa M and Fujii K 2018 Phys. Rev. A 98 032309
[94] Schuld M, Bergholm V, Gogolin C, Izaac J and Killoran N 2019 Phys. Rev. A 99 032331
[95] Wang Y, Yang S, Chen G, Bao Q and Li J 2023 Atmos. Res. 282 106510
[96] Murray F W 1967 J. Appl. Meteorol. Climatol. 6 203–4
[97] Chakraborty S, Jiang J H, Su H and Fu R 2020 J. Geophys. Res. 125 e2019JD030962
[98] Peters O and Neelin J D 2006 Nat. Phys. 2 393–6
[99] Holloway C E and Neelin J D 2009 J. Atmos. Sci. 66 1665–83
[100] Wu C M, Stevens B and Arakawa A 2009 J. Atmos. Sci. 66 1793–806
[101] Shin S, Teo Y S and Jeong H 2023 Phys. Rev. A 107 012422
[102] Williams C A, Paine A E, Wu H Y, Elfving V E and Kyriienko O 2023 Quantum chebyshev transform: mapping, embedding, learning and sampling distributions (arXiv:2306.17026)
[103] Amari S i 1985 Differential-Geometrical Methods in Statistics (Lecture Notes in Statistics) (Springer)
[104] Amari S-I 1997 Neural Comput. 10 251–76
[105] Pascanu R and Bengio Y 2014 Revisiting natural gradient for deep networks (arXiv:1301.3584)
[106] Pennington J and Worah P 2018 The spectrum of the fisher information matrix of a single-hidden-layer neural network Advances in Neural Information Processing Systems vol 31, ed S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi and R Garnett (Curran Associates, Inc.)
[107] Karakida R, Akaho S and Amari S-I 2020 J. Stat. Mech. 124005
[108] Mingard C, Pointing J, London C, Nam Y and Louis A A 2024 Exploiting the equivalence between quantum neural networks and perceptrons (arXiv:2407.04371)
[109] Pastori L, Eyring V and Schwabe M 2025 Fisher information, training and bias in Fourier regression models (arXiv:2510.06945)
[110] McClean J R, Boixo S, Smelyanskiy V N, Babbush R and Neven H 2018 Nat. Commun. 9 4812
[111] Thanasilp S, Wang S, Nghiem N A, Coles P and Cerezo M 2023 Quantum Mach. Intell. 5 21
[112] Larocca M, Thanasilp S, Wang S, Sharma K, Biamonte J, Coles P J, Cincio L, McClean J R, Holmes Z and Cerezo M 2024 A review of barren plateaus in variational quantum computing (arXiv:2405.00781)
[113] Kreplin D A and Roth M 2024 Quantum 8 1385
[114] Cong I, Choi S and Lukin M D 2019 Nat. Phys. 15 1273–8
[115] Foss-Feig M et al 2023 Experimental demonstration of the advantage of adaptive quantum circuits (arXiv:2302.03029)
[116] Sahay R and Verresen R 2024 Finite-depth preparation of tensor network states from measurement (arXiv:2404.17087)
[117] Chan A, Shi Z, Dellantonio L, Dür W and Muschik C A 2024 Phys. Rev. Lett. 132 240601
[118] Iqbal M et al 2024 Commun. Phys. 7 205
[119] Pesah A, Cerezo M, Wang S, Volkoff T, Sornborger A T and Coles P J 2021 Phys. Rev. X 11 041011
[120] Grant E, Wossnig L, Ostaszewski M and Benedetti M 2019 Quantum 3 214
[121] Friedrich L and Maziero J 2022 Phys. Rev. A 106 042433
[122] Zhang K, Liu L, Hsieh M H and Tao D 2022 Escaping from the barren plateau via Gaussian initializations in deep variational quantum circuits (arXiv:2203.09376)
[123] Gelman M 2024 A survey of methods for mitigating barren plateaus for parameterized quantum circuits (arXiv:2406.14285)
[124] Döbler P and Jattana M S 2025 A survey on integrating quantum computers into high performance computing systems (arXiv:2507.03540)
[125] Schreiber F J, Eisert J and Meyer J J 2023 Phys. Rev. Lett. 131 100803
[126] Landman J, Thabet S, Dalyac C, Mhiri H and Kashefi E 2022 Classically approximating variational quantum machine learning with random Fourier features (arXiv:2210.13200)
[127] Sweke R, Recio E, Jerbi S, Gil-Fuster E, Fuller B, Eisert J and Meyer J J 2023 Potential and limitations of random Fourier features for dequantizing quantum machine learning (arXiv:2309.11647)
[128] Jerbi S, Gyurik C, Marshall S C, Molteni R and Dunjko V 2024 Nat. Commun. 15 5676
[129] Orús R 2019 Nat. Rev. Phys. 1 538–50
[130] Stoudenmire E and Schwab D J 2016 Supervised learning with tensor networks Advances in Neural Information Processing Systems vol 29, ed D Lee, M Sugiyama, U Luxburg, I Guyon and R Garnett (Curran Associates, Inc.)
[131] Efthymiou S, Hidary J and Leichenauer S 2019 Tensornetwork for machine learning (arXiv:1906.06329)
[132] Ran S-J and Su G 2023 Intell. Comput. 2 0061
[133] Rieser H-M, Köster F and Raulf A P 2023 Proc. R. Soc. A 479 20230218
[134] Dilip R, Liu Y-J, Smith A and Pollmann F 2022 Phys. Rev. Res. 4 043007
[135] Jumade R and Sawaya N P 2023 Data is often loadable in short depth: Quantum circuits from tensor networks for finance, images, fluids, and proteins (arXiv:2309.13108)
[136] Jobst B, Shen K, Riofrío C A, Shishenina E and Pollmann F 2023 Efficient mps representations and quantum circuits from the fourier modes of classical image data (arXiv:2311.07666)
[137] Han Z Y, Wang J, Fan H, Wang L and Zhang P 2018 Phys. Rev. X 8 031012
[138] Merbis W, de Mulatier C and Corboz P 2023 Efficient simulations of epidemic models with tensor networks: application to the one-dimensional sis model (arXiv:2305.06815)
[139] Liu J, Li S, Zhang J and Zhang P 2023 Phys. Rev. E 107 L012103