Deep-learning-based quantum algorithms for solving nonlinear partial differential equations

Lukas Mouton,[1,2,*] Florentin Reiter,[2] Ying Chen,[3] and Patrick Rebentrost[1,†]
[1]*Centre for Quantum Technologies, National University of Singapore, Singapore*
[2]*Institute for Quantum Electronics, ETH Zürich, 8093 Zürich, Switzerland*
[3]*Department of Mathematics, Asian Institute of Digital Finance, and Risk Management Institute, National University of Singapore, Singapore*

(Received 25 August 2023; revised 3 March 2024; accepted 15 July 2024; published 14 August 2024)

Partial differential equations frequently appear in the natural sciences and related disciplines. In this work, we explore the potential for enhancing a classical deep-learning-based method for solving high-dimensional nonlinear partial differential equations with suitable quantum subroutines. In a first approach, we construct a deep-learning architecture based on variational quantum circuits without provable guarantees. In a second approach, tailored towards fault-tolerant quantum computers, find that quantum-accelerated Monte Carlo methods offer the potential to speed up the estimation of the loss function. In addition, we identify and analyze the trade-offs when using quantum-accelerated Monte Carlo methods to estimate the gradients with different methods, including a recently developed backpropagation-free forward gradient method. Finally, we discuss the usage of a suitable quantum algorithm for accelerating the training of feed-forward neural networks. Hence, this work provides different avenues with the potential for polynomial speedups for deep-learning-based methods for nonlinear partial differential equations.

DOI: 10.1103/PhysRevA.110.022612

I. INTRODUCTION

Differential equations naturally appear in many disciplines and play a fundamental role in the sciences and engineering [1,2] by mathematically modeling processes in fields such as physics and biology, as well as finance and sociology. In particular, nonlinear parabolic partial differential equations (PDEs) can model, e.g., the pricing of financial derivatives [3], intelligent decision making in game theory [4], and reaction-diffusion processes in physics [5]. Solving high-dimensional PDEs is particularly challenging, as numerical methods typically rely on high-dimensional discretizations of continuous functions. This procedure leads to what is known as the "curse of dimensionality," i.e., the computational cost scaling exponentially with the dimension [6]. If the PDE is nonlinear, approximating nonlinear terms in high-dimensional PDEs with polynomials or other basis functions further contributes to this problem.

Quantum computers have the potential to provide speedups for problems that may otherwise be intractable for classical computers, e.g., in machine learning [7], linear algebra [8], optimization [9], or chemistry [10]. Feynman famously envisioned the possibility of simulating quantum physics by using a quantum mechanical device [11]. Subsequent early breakthrough algorithms were Shor's algorithm [12] for factoring integers and Grover's algorithm [13] for searching an unstructured database. These algorithms motivated the development of increasingly better hardware, as well as a broader set of quantum algorithms and software [14–18]. The seminal algorithm by Harrow, Hassidim, and Lloyd (HHL) shows the potential for exponential quantum speedup for solving linear systems of equations [19]. It has subsequently been applied to solving ordinary differential equations (ODEs) [20]. Many of these algorithms require fault-tolerance, meaning that a certain amount of inaccuracy on the hardware level is tolerable thanks to error-correction codes [21,22]. These quantum algorithms are commonly referred to as fault-tolerant quantum algorithms. In addition to the work on fault-tolerant quantum algorithms, there is also active research on quantum algorithms which aim to make use of the currently available noisy intermediate-scale quantum (NISQ) computers to solve problems of practical relevance [23,24].

Recently, an effective classical algorithm has been proposed for solving high-dimensional semilinear parabolic PDEs (see Sec. I B and Ref. [25]). This algorithm reformulates the nonlinear PDE in terms of a stochastic differential equation (SDE), exploiting a link that has been extensively investigated [26–29]. The authors of Ref. [25] then use deep-learning methods to approximate the spatial gradient of the sought after function. It is well known that neural networks (NNs) can approximate a wide range of functions [30]. By employing NNs instead of polynomial or other basis functions, the authors of Ref. [25] avoid the curse of dimensionality. The gradient is then used in a numerical scheme to solve the nonlinear PDE over a given time interval. While the nature of the algorithm from Ref. [25], namely, being based on deep-learning techniques, does not allow for provable guarantees of finding a solution, it has several benefits in practice. The algorithm from Ref. [25] is able to solve nonlinear parabolic PDEs, which, as outlined above, are relevant in a variety of fields, but are often hard to solve, particularly in high dimensions. Furthermore, the kinds of PDEs that can be solved

*Contact author: lukasjmouton@gmail.com
†Contact author: cqtfpr@nus.edu.sg

are very general, more so than the kinds of PDEs for which quantum algorithms have already been proposed, see Sec. I C. The introduction of NNs improves upon using polynomials or other basis functions to approximate nonlinear unknowns. However, some computational bottlenecks remain in this algorithm. By reformulating the PDE in terms of SDEs, the authors introduce stochasticity into the architecture. Therefore, one requires a certain number of samples to reliable estimate sought after quantities (such as the loss function or the gradient of the NNs) with a certain error tolerance. Furthermore, the runtime of evaluating and training the NNs scales approximately as the input dimension squared, where the input dimension is the spatial dimension. Investigating [25] in the quantum computing context, we thus hope to exhibit advantages when solving a relatively wide range of nonlinear PDEs, as well as introduce a new paradigm for solving differential equations to the field of quantum algorithms, since, to the best of our knowledge, other quantum algorithms for solving differential equations have not attempted to make use of such a deep-learning approach.

In this work, we investigate quantum algorithms for the solving of PDEs by exploring ways in which the deep-learning architecture from Ref. [25] can be sped up with quantum subroutines. We follow three main threads in this paper: (i) a hybridized classical-quantum variational architecture, (ii) quantum estimation of the loss function and gradients for efficiently computable neural networks, and (iii) quantum advantages in evaluation of large-dimensional neural networks.

First (i), as the deep-learning architecture considers a sequence of NNs, we develop a hybridised classical-quantum architecture for solving PDEs by employing variational circuits as feature maps in the NNs. We make use of an existing scheme to avoid the barren plateau issue and carry out simulations to assess the effectiveness of this hybrid approach. While this approach it may exhibit the benefits and issues arising from the use of quantum neural networks, it is by its nature without provable guarantees and relies on further large-scale experimentation.

Stepping away from the variational method, we investigate fault-tolerant approaches. (ii) To overcome the limitation imposed by Chebyshev's inequality on Monte Carlo (MC) sampling, we employ the quantum-accelerated MC (QAMC) method. To this end, we combine QAMC with NNs in order to speed up the estimation of the loss function and of the gradient of the NN. We show how the recently introduced *forward gradient* method can be used in the quantum subroutine. We carry out error and query complexity analyses to quantify the possible speedup. Moreover (iii), to address the bottleneck of training the NNs, we incorporate a quantum algorithm for accelerating the training and evaluation of classical NNs in a separate approach. We outline the advantages and limitations of this approach, in particular its lack of suitability for being combined with the previous QAMC approach.

The paper is structured as follows: In Sec. II we give an introduction to variational quantum algorithms as well as the challenges associated with them, and discuss the combination of variational quantum algorithms and Ref. [25]. Section III outlines the idea of using QAMC (more details on which can be found in Appendix B 4 a) in combination with NNs. We highlight hurdles that arise when combining these two methods and examine different avenues to address these, among which is the recently introduced forward gradient method. We then use this tool in the following section. In Sec. IV we return to our goal of accelerating the algorithm from Ref. [25] and make use of the discussion in Sec. III. Using our results from the preceding section, we incorporate QAMC into the algorithm from Ref. [25]. Finally, we carry out error analyses and calculate query complexities to quantify the performance. In Sec. V we summarize a fault-tolerant quantum algorithm that accelerates the evaluation and training of classical feedforward NNs. We proceed to outline how one can introduce this algorithm into the architecture from Ref. [25], and what advantages and drawbacks this entails. Finally, in Sec. VI we review our work and results, draw conclusions and outline possible future research directions.

In Appendix A, we present the computational model we will work in. In Appendix B we give an introduction to NNs and automatic differentiation (AD) and provide a detailed overview over the algorithm from Ref. [25]. This classical algorithm for solving nonlinear PDEs will be the starting point of our work, i.e., the algorithm which we want to speed up using quantum subroutines. Furthermore, we introduce MC methods and their quantum-accelerated version as well as other quantum algorithms and subroutines.

A. Notation

As in Ref. [31] in Definition 8.1, we define big O notation as follows. Let f and g be two functions $f, g : X \to X$. We say that $f = O(g)$ if there is a constant $C \in \mathbb{R}^+$ such that for all $x \in X$ $|f(x)| \leqslant C|g(x)|$. We also use $\tilde{O}$, which hides polylogarithmic factors.

We define Lipschitz continuity as in Ref. [32] (Definition 9.4.1). Suppose (X, d_X) and (Y, d_Y) are metric spaces (where d_X and d_Y denote metrics on X and Y, respectively) and $f : X \to Y$. If there exists $L \in \mathbb{R}^+$ such that $d_Y(f(a), f(b)) \leqslant L d_X(a, b)$ for all $a, b \in X$, then f is called Lipschitz continuous on X with Lipschitz constant L. In this work we will use the squared l_2 norm, denoted by $\|\cdot\|_2^2$, when dealing with Lipschitz continuity.

In the relevant literature, one often encounters the additive and the mean-squared error. When estimating a quantity a with an estimator $\tilde{a}$, the additive error is given by $|a - \tilde{a}|$ and the mean-squared error by $\mathbb{E}[(a - \tilde{a})^2]$. In Appendix A of Ref. [33] the authors point out that these two definitions are almost equivalent. Concretely, up to a logarithmic overhead, the mean-squared error being ϵ^2 indicates that the additive error is ϵ with probability at least 0.99 and vice versa, using Chebyshev's inequality. Unless stated otherwise we shall, as is common in the literature, use the mean-squared error when analyzing classical algorithms and use the additive error when analyzing quantum algorithms.

In the following section, we provide a detailed description of the algorithm from Ref. [25] in Sec. I B.

B. Solving partial differential equations with deep learning

In this section we outline the deep-learning-based algorithm for solving partial differential equations (PDEs) from Ref. [25], which will serve as a starting point for much of

our work. In Ref. [25], the authors put forward a method to solve semilinear parabolic PDEs. Semilinear refers to a sum of linear terms (in the variable of the PDE) and one nonlinear one. To obtain an approximation of the solution of the PDE, the authors reformulate the PDE as a backward stochastic differential equation (SDE). They then discretize the SDE temporally and approximate the spatial gradient of the unknown solution using NNs at each time step in the approximation. Let $x \in \mathbb{R}^d$ be the spatial variable, $t \in \mathbb{R}$ the temporal variable, and $u(t,x) \in \mathbb{R}$ the unknown solution. Furthermore, let $\sigma(t,x) \in \mathbb{R}^{d\times d}$, $\mu(t,x) \in \mathbb{R}^d$, and $f(t,x,u(t,x)\sigma^\top \nabla u(t,x)) \in \mathbb{R}$ be known functions, where f is the nonlinearity. The main task is to solve PDEs of the following form [25]:

$$\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\mathrm{Tr}[\sigma\sigma^\top(t,x)(\mathrm{Hess}_x u)(t,x)] + \nabla u(t,x)\cdot\mu(t,x) + f(t,x,u(t,x),\sigma^\top(t,x)\nabla u(t,x)) = 0, \quad (1)$$

on some interval $[t_0, T]$. We assume to have a known terminal condition $u(T,x) = g(x) \in \mathbb{R}$. The quantities ∇u and $\mathrm{Hess}_x u$ refer to the gradient and Hessian, respectively, of u with respect to the spatial variable x.

Let W_t be a Brownian motion, meaning that $W_0 = 0$, W_t is continuous in t, and for the increments $W_{t+t_1} - W_t$ it holds that $W_{t+t_1} - W_t$ is distributed according to $\mathcal{N}(0,t_1)$ and independent of past values W_s for $s < t$ [34]. In the case of a multivariate Brownian motion, the above holds for each entry, which are independently and identically distributed (iid) according to $\mathcal{N}(0,t_1)$. For a d-dimensional Brownian motion $\{W_t\}_{t\in[0,T]}$ and a d-dimensional stochastic process $\{X_t\}_{t\in[0,T]}$ satisfying

$$X_t = \xi + \int_0^T \mu(s,X_s)\,ds + \int_0^T \sigma(s,X_s)\,dW_s, \quad (2)$$

the solution of Eq. (1) satisfies the following backward SDE in integral form [26,27],

$$u(t,X_t) = u(t,X_{t_0}) - \int_0^t f(s,X_s,u(s,X_s),\sigma^\top(s,X_s) \times \nabla u(s,X_s))\,ds + \int_0^t [\nabla u(s,X_s)]^\top \sigma(s,X_s)\,dW_s. \quad (3)$$

One can approximate Eqs. (2) and (3) by temporally discretizing them, using, e.g., Euler's method, with $t \in \{t_0 = 0, t_1, \ldots, t_N = T\}$. We then arrive at

$$X_{t_{n+1}} - X_{t_n} \approx \mu(t_n, X_{t_n})\Delta t_n + \sigma(t_n, X_{t_n})\Delta W_{t_n}, \quad (4)$$

and

$$u(t_{n+1}, X_{t_{n+1}}) - u(t_n, X_{t_n}) \approx -f(t_n, X_{t_n}, u(t_n, X_{t_n}), \sigma^\top(t_n, X_{t_n})\nabla u(t_n, X_{t_n}))\Delta t_n + [\nabla u(t_n, X_{t_n})]^\top \sigma(t_n, X_{t_n})\Delta W_{t_n}, \quad (5)$$

where

$$\Delta t_n = t_{n+1} - t_n, \quad \Delta W_{t_n} = W_{t_{n+1}} - W_{t_n}, \quad (6)$$

where the entries of ΔW_{t_n} are iid according to $\mathcal{N}(0,\ \Delta t_n)$. The discretization in Eq. (4) allows us to sample paths $\{\hat{X}_{t_n}\}_{n\in[0,N]}$, where we use the hat (i.e., $\hat{X}_t$) to indicate the discretized estimation of X_t. In a next step, the authors of Ref. [25] approximate the function $x \to \sigma^\top(t,X_t)\nabla u(t,X_t)$ at each time step $t = t_n$ using a feedforward NN,

$$\sigma^\top(t,X_t)\nabla u(t,X_t) = (\sigma^\top \nabla u)(t,X_t) \approx (\sigma^\top \nabla u)(\hat{X}_t|\theta_n), \quad (7)$$

where θ_n denotes the parameters of the nth NN. The reason for assuming this would work is that feedforward NNs may serve as universal function approximators [30]. The authors in Ref. [25] stack together the NNs together with Eqs. (2) and (3) to step by step calculate $\hat{u}$. Figure 1 illustrates this process. As seen in Fig. 1, no discretization of the whole spatial domain takes place in this deep-learning-based algorithm. The discretization of the spatial domain is what typically leads to the curse of dimensionality when solving PDEs, as the cost of solving the problem then grows exponentially in its size (the spatial dimension, in this case). In order to train the NNs, the authors use the expected difference between the terminal condition $g(\hat{X}_{t_N})$, and the computed $\hat{u}_{t_N}$ after the final step, as a loss function,

$$l(\theta) = \mathbb{E}\left[\left|g(\hat{X}_{t_N}) - \hat{u}_{t_N}\left(\{\hat{X}_{t_n}\}_{0\leqslant n\leqslant N}, \{\Delta W_{t_n}\}_{1\leqslant n\leqslant N}\right)\right|^2\right]. \quad (8)$$

We refer to the function inside the expectation value as the payoff function f_p,

$$f_p\left(\{\hat{X}_{t_n}\}_{0\leqslant n\leqslant N}, \{\Delta W_{t_n}\}_{1\leqslant n\leqslant N}\right) = \left|g(\hat{X}_{t_N}) - \hat{u}_{t_N}\left(\{\hat{X}_{t_n}\}_{0\leqslant n\leqslant N}, \{\Delta W_{t_n}\}_{1\leqslant n\leqslant N}\right)\right|^2. \quad (9)$$

The total set of trainable parameters consists of the initial conditions $u(0,X_{t_0})$ and $\nabla u(0,X_{t_0})$ as well as the parameters for each of the NNs, $\{\theta_n\}_{0\leqslant n\leqslant N-1}$. The total number of trainable parameters we call n_θ. The focus of this work is to investigate methods for accelerating the deep-learning method introduced in this section.

C. Related work: Quantum algorithms for differential equations

There exist several proposed quantum algorithms for solving differential equations (ODEs as well as PDEs). In Ref. [20] the author puts forward a fault-tolerant quantum algorithm which solves sparse systems of linear ODEs by discretizing the system of ODEs and subsequently employing the HHL algorithm [19] to solve the resulting system of linear equations. In Ref. [35], the authors also propose a quantum algorithm for solving linear ODEs, which, however, relies on spectral methods. Spectral methods use linear combinations of basis functions (e.g., a Fourier basis) to approximate the solution. This approach also ends with solving a linear system of equations on a quantum computer. A quantum algorithm to solve quadratically nonlinear ODEs under certain conditions is described in Ref. [36]. The authors make use the Carleman linearization [37–39] to approximate the nonlinear part. The Carleman linearization represents a finite-dimensional polynomially nonlinear system by an infinite-dimensional linear system. To make use of the Carleman linearization, the infinite-dimensional linear system is truncated at a certain point. Subsequently, the authors from Ref. [36] again discretize the resulting system and solve the linear system with HHL [19]. The algorithm presented in Ref. [36] may also be applied to solving certain PDEs (with a restricted kind of

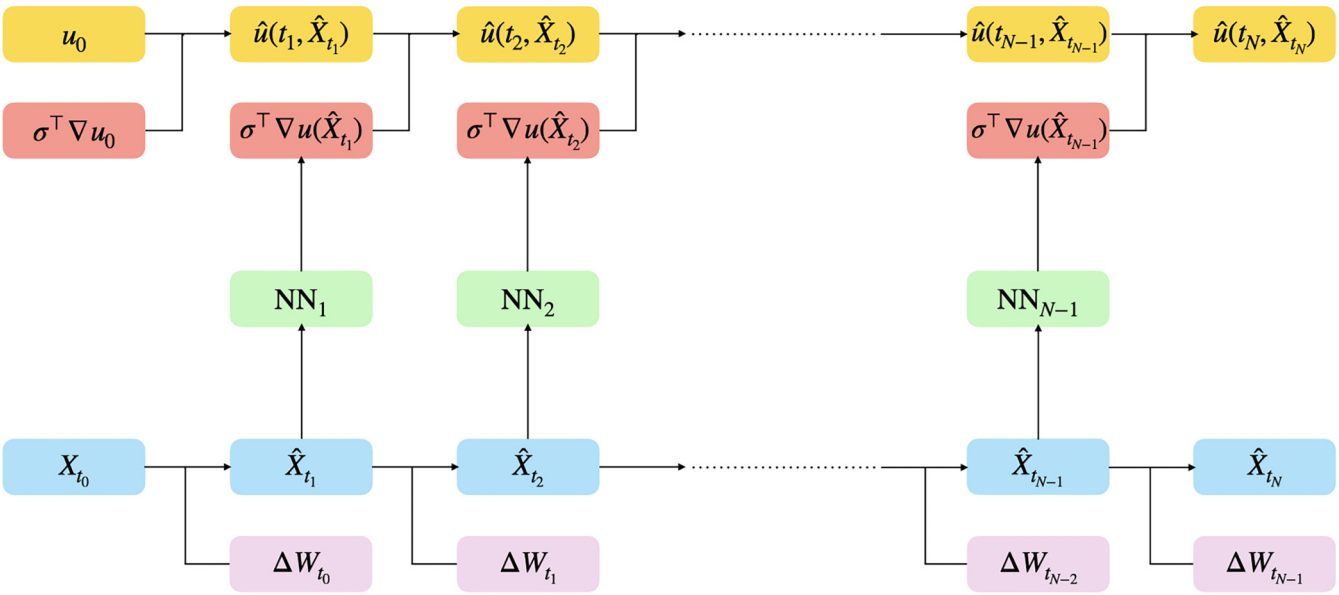

FIG. 1. Architecture used for solving semilinear parabolic partial differential equations via a stochastic differential equation approach and the usage of neural networks for approximating spatial gradient information. The approach is a temporally discretized N-step procedure ($i \in [N]$) which is driven by stochastic Brownian increments ΔW_{t_i} determining the randomness of the state variables X_{t_i}. The neural networks represent the computation of the spatial gradient information from the state variables, which is then used to propagate the solution u. The arrows indicate which quantities are used to approximate which other quantities. Figure adapted from Ref. [25].

nonlinearity), as the discretization of a PDE in all but one dimension generally results in a system of ODEs. In Ref. [40], the author improves on the algorithm from Ref. [36] and in Ref. [41] it is modified to address problems in machine learning. The authors in Ref. [42] present an algorithm for solving (nonlinear) reaction-diffusion equations. Using Euler's method, as well as the Carleman linearization for the nonlinearity, they discretize the PDE and solve the resulting system with HHL. The authors of Ref. [43] present quantum algorithms for solving linear PDEs by making use of the finite difference method (FDM) and spectral methods separately. In the FDM approach, they discretize the PDE on a grid. Both cases result in a linear system of equations which needs to be solved. The authors of Ref. [44] also present a quantum algorithm for solving linear PDEs, which relies on Hamiltonian simulation of a cleverly chosen Hamiltonian, which encodes certain properties of the PDE. In Ref. [45] a quantum algorithm to solve nonlinear ODEs by mapping the ODE to the nonlinear Schrödinger equation, which is then solved using Trotterization, is outlined. Another numerical scheme, the finite element method (FEM, which approximates the solution by using interpolating functions within each discretized element) and HHL are used to solve elliptic PDEs in Ref. [46]. In Refs. [47,48], the authors derive quantum algorithms for solving nonlinear ODEs as well as the nonlinear Hamilton-Jacobi (HJ) equation (which is a special case of the nonlinear Hamilton-Jacobi-Bellman PDE). They do so, by mapping the nonlinear ODEs and the nonlinear HJ equation to linear ODEs and HJ equations using linear representation methods, such as the level set method, and then use HHL to solve the linear system. For a detailed discussion on HHL-based quantum algorithms for solving differential equations, we refer to Ref. [49].

A comment on the form of the solution of HHL is in order. In a classical setting, the vector describing the solution to a linear system of equations can be copied and its entries can be accessed at will. In the quantum setting, however, the solution is encoded in the amplitudes of a quantum superposition (amplitude encoding). The no-cloning theorem [50] states that quantum states cannot be cloned (duplicated), and a measurement would collapse the quantum state to the eigenstate corresponding to the measurement outcome. One therefore resorts to multiple computations and subsequent measurements in order to generate reliable statistics to infer the sought results. The procedure of producing a quantum state proportional to the solution (of a system of linear equations, or differential equations) may in some applications not be enough, as a full quantum state cannot be read out efficiently [36]. In some scenarios it may suffice to extract relevant information from sampling an observable. However, in other cases, more intricate post-processing may be needed to gain the desired insights.

An example of a variational quantum algorithm for solving differential equations can be found in Ref. [51]. The authors define a loss function which measures how well a certain candidate function solves the differential equation. The differential equation is then encoded via its key properties such that it is mapped to a high-dimensional feature space, which is explored by a classical solver in order to find a solution. The variational quantum algorithm put forth in Ref. [52] aims to solve nonlinear differential equations by using a so-called quantum nonlinear processing unit. The latter computes polynomially nonlinear terms appearing in the differential equations.

In addition, there exist multiple proposed quantum algorithms to solve specific differential equations, such as the

Poisson equation [53], the wave equation [54], the Vlasov equation [55], and the heat equation [56].

II. VARIATIONAL APPROACH FOR DEEP-LEARNING METHOD FOR PARTIAL DIFFERENTIAL EQUATIONS

In this section we outline our first approach for enhancing the deep-learning architecture from Ref. [25] using quantum computation. We consider employing variational quantum methods in order to enhance the algorithm from Ref. [25] by introducing parametrized quantum circuits (PQCs) into the neural networks (NNs).

In the context of machine learning, a motivation for making use of variational quantum circuits is the exploitation of the exponentially large Hilbert space via controllable entanglement and interference [57–61]. In this sense, the quantum circuit is employed as a trainable feature map. As the circuit depth may be kept relatively short compared to fault-tolerant quantum algorithms, this approach is more suitable for the noisy intermediate-scale quantum (NISQ) era. If the classical starting point is a classical NN (as in the architecture from Ref. [25]), a straightforward place to introduce PQCs is within the NNs, as described in Ref. [62]. The resulting network is termed a hybrid quantum neural network (hybrid QNN).

In a hybrid QNN, a PQC is inserted into the classical NN such that the outputs (post-activation) of a given layer are fed as arguments (e.g, as the rotation angles of gates in the PQC) into the PQC. The measurement outcomes of the PQC, measured in the computational basis, are then again passed forward as inputs to the succeeding classical layer. In order to update the weights of a hybrid QNN, one generally makes use of backpropagation, see Appendix B 1. In order to evaluate the gradients of the parameters in the PQC, we make use of the parameter shift rule, see Lemma 2.

A commonly used PQC ansatz circuit is the so-called hardware efficient ansatz (HEA), because it makes use of the native gates of a given quantum hardware platform, thus avoiding the transpilation overhead [63]. By doing so, it can keep the circuit depth very low while still allowing for entanglement among all qubits and introducing trainable parameters for each qubit. The HEA is of the form

$$\mathcal{U}(\theta) = \prod_k \mathcal{U}_k(\theta_k)\mathcal{W}_k, \tag{10}$$

where $\mathcal{U}_k(\theta_k)$ consists of single-qubit rotations on each qubit, the angles of which are the variational parameters to be optimized. Furthermore, $\mathcal{W}_k$ are (entangling) two-qubit gates. Note that the form of the HEA is similar to the form outlined in Eq. (B21). The HEA is known to suffer from barren plateaus, especially for larger circuit depths [64]. Reference [65] outlined a technique for preventing the barren plateau issue from occurring in the HEA for short circuit depths. Their finding suggests that using short PQCs of the form of the HEA may still be a viable option. However, we note that recent work has also considered the efficient classical simulatability of barren-plateau-free circuits [66–68]. The authors propose an initial state which satisfies certain entanglement criteria,

$$|\psi_t\rangle = e^{-iHt}|\psi_0\rangle, \tag{11}$$

where $|\psi_0\rangle$ is a product state, i.e., not entangled. For t above a certain threshold, the qubits are suitably entangled. The Hamiltonian H is of the form

$$H = \sum_{i=1}^{n} X_iX_{i+1} + Y_iY_{i+1} + 2Z_iZ_{i+1} + X_i, \tag{12}$$

where n is the number of qubits in the circuit and X_iX_{i+1}, Y_iY_{i+1}, and Z_iZ_{i+1} refer to the two-qubit Pauli gates acting on the ith and $i+1$st qubit. The index $n+1$ refers back to 1. Next, we discuss how the methods introduced above may be applied to the deep-learning architecture.

A. Applying variational algorithms to the deep-learning approach

We proceed to discuss the application of variational quantum methods in the deep-learning architecture from Ref. [25] and its consequences. We replace the NNs in the architecture from Ref. [25] with (generic) PQCs and calculate the gradient of the loss function $l(\theta)$ from Eq. (8) with respect to a parameter θ_k in the nth variational circuit $\mathcal{V}_n$ (i.e., at the nth step of the temporal discretization). This calculation gives

$$\begin{aligned}\partial_{\theta_k} l(\theta) &= \partial_{\theta_k}\mathbb{E}\big[\big|g-\hat{u}_{t_N}\big|^2\big] = 2\mathbb{E}\big[\big(g-\hat{u}_{t_N}\big)\partial_{\theta_k}\hat{u}_{t_N}\big] \\ &= 2\mathbb{E}\Bigg[\big(g-\hat{u}_{t_N}\big)\Bigg(\prod_{i=N-1}^{n+1}\partial_{\hat{u}_i} f(u_i,\,\sigma^\top\nabla u_i)\Delta t_i\Bigg) \\ &\quad\times\Big\{\big[\partial_{\sigma^\top\nabla u_n} f(u_n,\,\sigma^\top\nabla u_n)\Delta t_n + \Delta W_{t_n}\big]\partial_{\theta_k}\mathcal{V}_n\Big\}\Bigg],\end{aligned} \tag{13}$$

where we omitted the arguments of functions that are not of importance for this calculation. The key observation is that the term $\partial_{\theta_k}\mathcal{V}_n$ appears as a factor in the derivative. Therefore, we expect the barren plateau issue to show up in our context as well, provided that the PQCs we employ to replace the classical NNs exhibit barren plateaus in the first place.

B. Simulations

We now simulate the variational architecture with the considerations from Ref. [65] to address the barren plateau issue. We compare the classical case to the case with PQCs. We carry out simulations with small-sized quantum circuits, such that we can simulate them within reasonable time. We investigate whether including a PQC in the NNs in the architecture from Ref. [25] can improve the performance, measured by the loss function. The PQC we employ, for a variable number of qubits, is displayed in Fig. 2. The circuit in Fig. 2 is initialized in the state displayed in Eq. (11) (to avoid the barren plateau from occurring), the inputs are loaded via angle embedding (using R_X rotations), followed by a HEA with R_X rotations and circular entanglement via CNOT gates. In order to compare the classical base case to hybrid scenarios, we solve the Hamilton-Jacobi-Bellman (HJB) equation, which is a special case of the PDE from Eq. (1) with $\sigma(t,X) = 2\mathbb{I}$, $\mu(t,X) = 0$ and $f(t,X,u(t,X),\sigma^\top(t,X)\nabla u(t,X)) = \|\nabla u(t,X)\|_2^2$. To evaluate the loss function, we have $g(X) = \ln[(1+\|X\|_2^2)/2]$ for the HJB partial differential equation (PDE). We choose to simulate the classical and the quantum case for the HJB

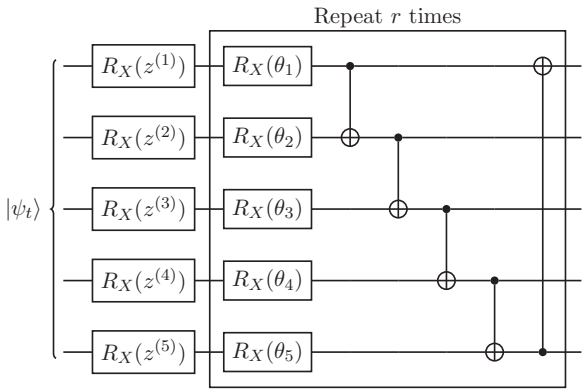

FIG. 2. Example PQC with 5 qubits with $|\psi_t\rangle$ from Eq. (11). The z values are inputs to the circuit (which are given by the outputs of the preceding classical layer, in a hybrid QNN) and the θ values are trainable parameters. The part in the box is repeated r times, with different parameters θ for each repetition.

PDE, because the output of the NNs plays a more important role compared to the other PDEs, and the performance of the NNs (or hybrid QNNs) is under investigation. For our implementations we use PyTorch [69] and PennyLane [70].

In our first experiment we compare the baseline classical architectures with hybrid QNNs with the same total number of trainable parameters in order to investigate whether one can observe improvements of the hybrid case over the classical case by using the PQC as a feature map. The classical NNs have an input layer, two hidden layers, and an output layer (as in Ref. [25]), which are all fully connected. In the hybrid QNN, we replace nodes in the second hidden layer with a PQC of the form from Fig. 2, as seen in Fig. 3(a).

We introduce the PQC in a way such that the total number of trainable parameters does not change, by tuning the number of classical neurons in the hidden layers. In the hybrid case, as opposed to the setting illustrated in Fig. 3(b), some of the trainable parameters are parameters in the PQC (the values of θ in Fig. 2), as opposed to trainable weights in the classical part. We compare three cases, each having a different number of trainable parameters. We compare the performance for the classical base case and the hybrid case, allowing us to estimate if a greater percentage of the trainable parameters in the PQC offers any advantage. Note that while we can easily increase the number of classical parameters, increasing the number of variational quantum parameters is computationally expensive to simulate. Consequently, in the hybrid scenarios of the different cases, a different percentage of the total number of trainable parameters are variational quantum parameters. The cases 1, 2, and 3 we compare in Fig. 4 solve the HJB PDE using the architecture from Ref. [25] in 5, 10, and 20 dimensions with 225, 565, and 1260 total trainable parameters, respectively. In the hybrid cases, we employ an eight-dimensional PQC of the form from Fig. 2 with $r = 2$ in the second hidden layer, introducing 16 trainable variational quantum parameters. Thus, the hybrid model in case 1 has the greatest percentage of variational quantum parameters, and the hybrid model in case 3 the smallest.

As one can see in Fig. 4, the classical models and the hybrid models in each of the three cases perform comparably

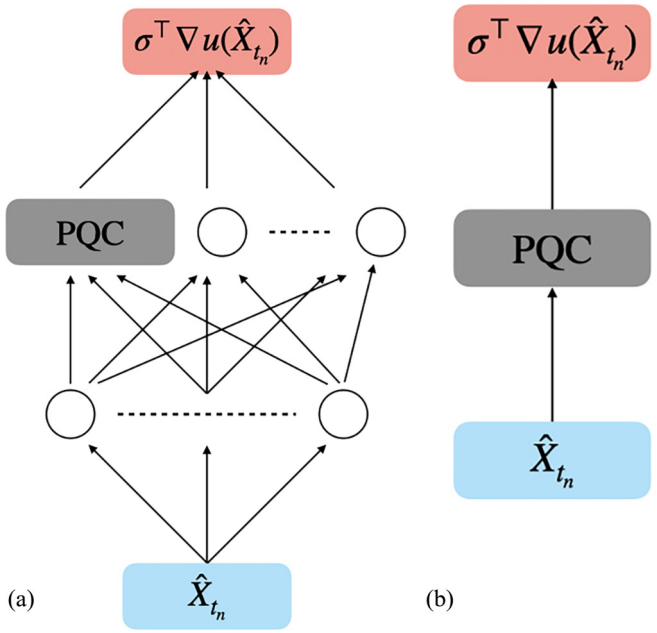

FIG. 3. (a) Hybrid QNN used for the first experiment, where the circles represent classical neurons, the grey box a PQC, and the input and output are labeled according to Fig. 1 (b) PQC employed in the deep-learning architecture as in Fig. 1 without the assistance of classical neurons, as used in the second experiment.

well after a sufficient number of training iterations, with the models with a greater number of trainable parameters taking longer to converge. Figure 4 suggest that introducing the PQC does not provide an improvement over the classical base case. Furthermore, we cannot observe a difference in performance

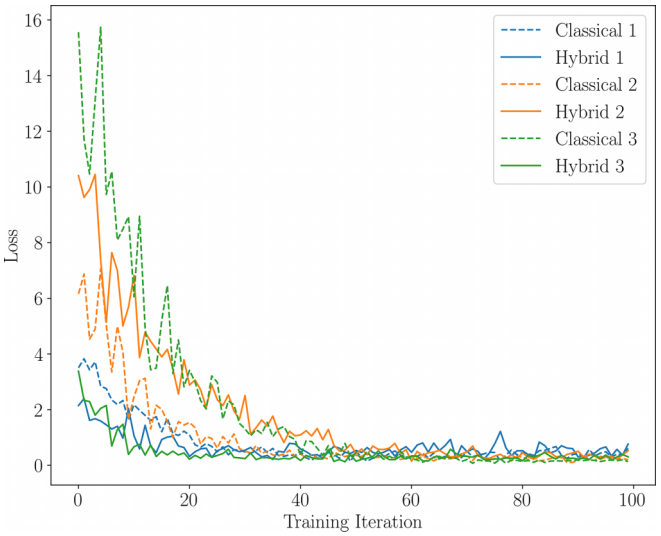

FIG. 4. Loss from Eq. (8) for the HJB PDE against the number of training iterations with 20 samples per iteration for three different scenarios (225, 565, and 1260 total trainable parameters in the classical and hybrid case where the spatial dimension is 5, 10, and 20, respectively). The learning rate in all cases is 0.05. In each of the hybrid cases, 16 of the trainable parameters are variational parameters in a PQC in a hybrid QNN, as shown in Fig. 3(a), where the PQC is of the form of Fig. 2 with $r = 2$.

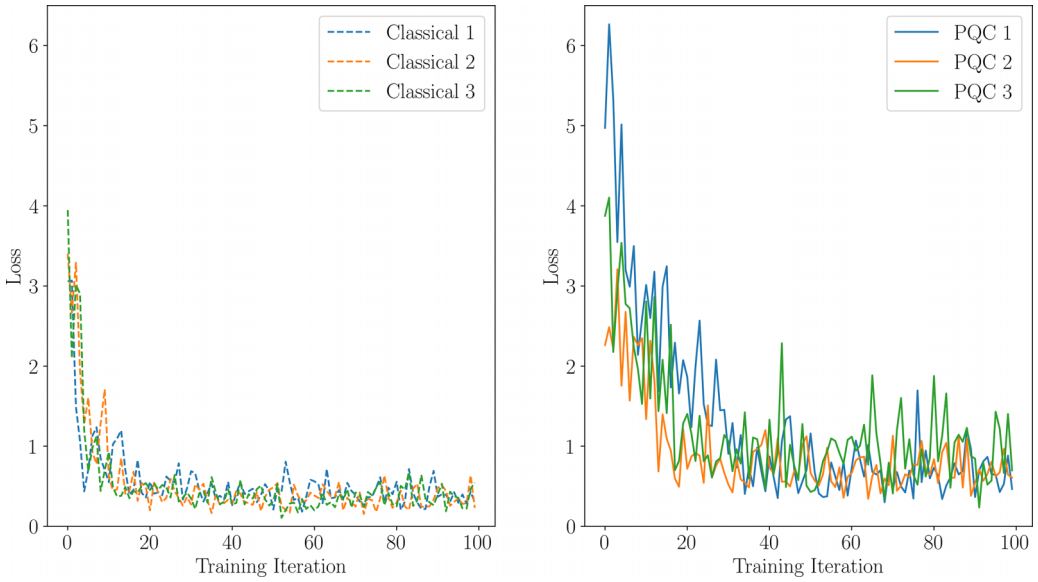

FIG. 5. Loss from Eq. (8) for the HJB PDE against the number of training iterations with 20 samples per iteration for three different scenarios (20, 30, and 42 trainable parameters, respectively) for classical NNs and PQCs with the same number of trainable parameters. The learning rate in all cases is 0.05. In each of the quantum cases, all of the trainable parameters are variational parameters in a PQC, as shown in Fig. 3(b), where the PQC is of the form of Fig. 2.

for a different percentage of variational quantum parameters in the hybrid models. Possible reasons for this include that the percentage of variational quantum parameters is too small to make a difference, or that the classical models are just as suited (if not more so) for the problem at hand. To understand which of these possibilities is more plausible, we carry out a second experiment.

In the next experiment, we directly compare PCQs (as opposed to hybrid QNNs) against simple NNs with the same number of trainable parameters, as shown in Fig. 3(b). Again, an argument for introducing the PQC is the hope that it may find solutions in the Hilbert space serving as a feature space. We want to see whether the hybrid case still performs similarly well as the classical case. Based on the outcome, we want to learn whether in the first experiment, introducing the PQC did not improve the performance because the PQCs and classical NNs perform similarly well, or because the NNs were "carrying" the hybrid models. We again compare three cases where we solve the HJB PDE using the deep-learning architecture for the four-, five-, and six-dimensional case. The classical models are NNs with just one one input layer and one output layer and the PQCs are again of the form from Fig. 2 with the number of qubits corresponding to the dimension d of the input, and the number of repetitions $r = d + 1$ such that the total number of trainable parameters is the same in the classical and the hybrid case. The loss for each of these cases is shown in Fig. 5. We separate the classical and the hybrid models for better visibility. In Fig. 5 it is clearly visible that the classical models outperform the hybrid models, which appears to refute the hypothesis that the classical models and PQCs are similarly suited in this context. This observation suggests that in the first experiment, the classical parts of the hybrid QNNs were to some extent adapting to or providing good input features to the PQC, allowing the hybrid model to perform as well as the classical base model.

To conclude, our evidence suggests that, in the deep-learning architecture from Ref. [25], hybrid models perform similarly well as classical models, but completely replacing the NNs with PQCs worsens the performance. However, we cannot rule out the possibility that, in scenarios beyond the scope of our simulations (in particular, very-high-dimensional cases beyond the reach of classical simulations) hybrid architectures may provide better performance (in the sense of absence of evidence not being evidence of absence). Since the problem of learning the quantity $\sigma^\top(t, X)\nabla u(t, X)$ from samples of X_t is of classical nature, the motivation for including PQCs into the NNs was to use the PQCs as feature map, as done elsewhere. A hypothetical scenario in which the introduction of PQCs into the architecture would be further motivated by the nature of the problem at hand being quantum mechanical, might be if the dependence of the spatial gradient on the stochastic process X_t was given according to some unitary evolution. The investigation of this problem we leave to future work.

III. TRAINING NEURAL NETWORKS WITH QUANTUM-ACCELERATED MONTE CARLO METHODS

The PDE solver combines classical neural networks (NNs) and training on data that are sampled from known distributions, such as the Gaussian distribution. Training and evaluating NNs with data obtained with MC sampling is, however, also used elsewhere, e.g., in quantitative finance [71,72], physics and chemistry simulations [73,74], image and video processing [75,76], drug discovery [77], and robotics [78]. In this section, we explore how the quantum-accelerated Monte Carlo (QAMC, see Appendix B 4 a) method offers a speedup in the number of samples to achieve the same error in evaluating the loss function and the gradients of a neural network. We analyze different methods for training classical

NNs based on quantum circuits and compare them to each other. We will apply the methods discussed in this section to the algorithm from Ref. [25] in Sec. IV.

In Sec. III A we apply QAMC to estimating loss functions of NNs given fixed values for the trainable parameters. Moreover, the NNs may not be pretrained, their parameters thus requiring fine-tuning in order to be useful, hence we investigate the use of QAMC for training as well. In the classical machine learning literature, a novel method for training the weights of NNs, termed the *forward gradient* [79], has been proposed, and we will introduce it in Sec. III B. We then compare different approaches to incorporating the forward gradient with other methods for training NNs in the classical and quantum scenario in Sec. III C.

In the classical scenario, backpropagation (see Appendix B 1) is a widely used algorithm to train the weights in NNs, in spite of its excessive memory usage [80,81], which is difficult to predict in advance [82]. Due to memory in quantum circuits (i.e., qubits) being especially expensive for the foreseeable future, this concern is further amplified in the quantum case. The *forward gradient* method is a generally usable method for evaluating gradients, with the advantage of having lower memory requirements than backpropagation, but with the drawback of introducing additional stochasticity. The forward gradient method relies on forward mode automatic differentiation (AD) as opposed to reverse mode AD, on which backpropagation is based (see Appendix B 2). This difference results in the forward gradient requiring less memory, at the cost of introducing stochasticity into the gradient. Furthermore, being based on AD, it does not suffer from numerical issues as numerical differentiation does. We show that this method also finds application in the deep-learning architecture from Ref. [25]. Since MC methods (classical or quantum) also leverage stochasticity, we investigate how the forward gradient method interacts with MC methods.

The memory advantage of the forward gradient method is particularly beneficial in a quantum setting. We consider the NNs to be classical circuits which are implemented via a unitary (see Definition 4) performing the equivalent quantum circuit. A classical feed-forward NN may be implemented in a quantum circuit as follows: As mentioned in Appendix A, any classical circuit can be implemented on a quantum circuit with only a minimal overhead. Reference [83] discusses further how classical feedforward NNs may be implemented in a quantum circuit. In this reference the authors only outline how to implement a certain activation function which is not Lipschitz continuous. Common activation functions such as the sigmoid function or the rectified linear unit (ReLU) are, however, Lipschitz continuous. When backpropagation is applied to a classical NN implemented in a quantum circuit, where the inputs may be in superposition, the forward and the backward pass occur in the same circuit, and measurement only takes place after both passes have been carried out.

A. Estimating the output of a neural network

We first quantify the potential speedup obtained when using QAMC to estimate the loss function of a NN, where the stochasticity comes from the input of the neural network. Note Definition 5 on preparing probability distributions in superposition. We proceed to outline the setting with a quantum circuit. In this section, a sample or a query refers to a query to the unitary we introduce below (or its modified forms, which we introduce later in Sec. III C). Note that the number of wires is not representative of how many qubits are used to represent a certain state.

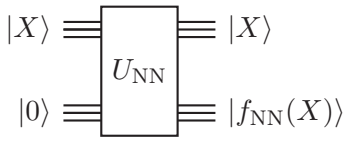

FIG. 6. Unitary evaluating NN.

Consider a classical NN including a scalar loss function represented by the function $f_{\mathrm{NN}} : \mathbb{R}^d \mapsto \mathbb{R}$ and a random variable (RV) X, distributed according to p_X, i.e., X_i occurs with probability p_{X_i}. Let the NN be implemented in a unitary in a quantum circuit, as displayed in Fig. 6. As we outline in Appendix A, any classical circuit can be implemented on a quantum circuit with only a minimal overhead. This is due to the fact that the NAND gate is a universal gate for classical computation, and that the NAND gate can be represented using a Toffoli gate in a quantum circuit. Thus, the classical circuit computing the function f_{NN} can also be implemented in a quantum circuit. We do not make the trainable parameters explicit in this definition.

For an input state as seen in Definition 5, U_{NN} obtains

$$\sum_i \sqrt{p_{X,i}}|X_i\rangle|0\rangle \to \sum_i \sqrt{p_{X,i}}|X_i\rangle|f_{\mathrm{NN}}(X_i)\rangle. \tag{14}$$

Applying QAMC to estimate the mean of $f_{\mathrm{NN}}(X)$, weighted by the distribution p_X, as described in Appendix B 4 a, we need to bound the variance of $f_{\mathrm{NN}}(X)$ in order to determine the required sample complexity and quantify the potential quantum speedup.

At this point, we comment on a property of NNs, namely that NNs are in practice Lipschitz continuous. As shown in Ref. [84], the Lipschitz constant of a (feedforward) NN can be upper bounded by the product of the Lipschitz constants of its constituent layers. The Lipschitz constant of a particular layer can, in turn, be upper bounded by the largest singular value of the matrix describing the weights of that particular layer, $\sigma_{\max}(W_l)$, multiplied by the Lipschitz constant $L_{f_{\mathrm{nl},l}}$ of the activation function of the same layer, $f_{\mathrm{nl},l}$,

$$L_{f_{\mathrm{NN}}} \leqslant \prod_{l=1}^{L} \sigma_{\max}(W_l) L_{f_{\mathrm{nl},l}}, \tag{15}$$

where there are L layers in the NN. The individual layers are Lipschitz continuous if the nonlinear activation functions $f_{\mathrm{nl},l}$ are so, and common activation functions, such as ReLU or the sigmoid function, are Lipschitz continuous. Furthermore, we assume the loss function is Lipschitz continuous, at least on the bounded domain of interest. Thus, in practice, NNs are typically Lipschitz continuous. There exist regularization techniques, termed spectral normalization, that control the Lipschitz constant of a NN by normalizing the singular values of the weight matrices for each layer to a desired value [85,86].

Given the Lipschitz constant for a NN, $L_{f_{\mathrm{NN}}}$, we can also derive the following growth bound: By the Lipschitz continuity of f_{NN} we have

$$|f_{\mathrm{NN}}(X) - f_{\mathrm{NN}}(0)|^2 \leqslant L_{f_{\mathrm{NN}}} \|X - 0\|_2^2 = L_{f_{\mathrm{NN}}} \|X\|_2^2. \quad (16)$$

Therefore, we have that

$$\begin{aligned} |f_{\mathrm{NN}}(X)|^2 &= |f_{\mathrm{NN}}(X) - f_{\mathrm{NN}}(0) + f_{\mathrm{NN}}(0)|^2 \\ &\leqslant |f_{\mathrm{NN}}(X) - f_{\mathrm{NN}}(0)|^2 + |f_{\mathrm{NN}}(0)|^2 \\ &\leqslant L_{f_{\mathrm{NN}}} \|X\|_2^2 + |f_{\mathrm{NN}}(0)|^2 \\ &\leqslant \left[L_{f_{\mathrm{NN}}} + |f_{\mathrm{NN}}(0)|^2\right]\left(1 + \|X\|_2^2\right). \end{aligned} \quad (17)$$

We now proceed to find an upper bound on the variance of $f_{\mathrm{NN}}(X)$. We have

$$\begin{aligned} \mathbb{V}[f_{\mathrm{NN}}(X)] &= \mathbb{E}[f_{\mathrm{NN}}(X)^2] - \mathbb{E}[f_{\mathrm{NN}}(X)]^2 \\ &\leqslant \mathbb{E}[f_{\mathrm{NN}}(X)^2] \\ &\leqslant \mathbb{E}\left[\left(L_{f_{\mathrm{NN}}} + |f_{\mathrm{NN}}(0)|^2\right)\left(1 + \|X\|_2^2\right)\right] \\ &= \left(L_{f_{\mathrm{NN}}} + |f_{\mathrm{NN}}(0)|^2\right)\left(1 + \mathbb{E}\left[\|X\|_2^2\right]\right), \end{aligned} \quad (18)$$

which allows us the formulate the following result, by applying Lemma 6.

Result 1. Quantum-accelerated estimation of neural network loss function. Consider a classical NN including its scalar loss function, $f_{\mathrm{NN}} : \mathbb{R}^d \mapsto \mathbb{R}$, with Lipschitz constant $L_{f_{\mathrm{NN}}}$. Let the NN be implemented in a quantum circuit in the form of a unitary U_{NN}. Let the input to U_{NN} be given by a random variable X, governed by a distribution p_X, for which we have quantum sample access. Then we can estimate the mean output of f_{NN} using QAMC with a sample complexity, with respect to U_{NN}, of

$$\tilde{O}\left[\left(L_{f_{\mathrm{NN}}} + |f_{\mathrm{NN}}(0)|^2\right)\left(1 + \mathbb{E}\left[\|X\|_2^2\right]\right)/\epsilon\right], \quad (19)$$

up to additive error ϵ with probability 0.99, thus offering a quadratic speedup over the classical case.

We proceed to discuss the nature of the speedup, given the upper bound on $\mathbb{V}[f_{\mathrm{NN}}(X)]$. The first factor, $(L_{f_{\mathrm{NN}}} + |f_{\mathrm{NN}}(0)|^2)$, is hard to upper bound in practice, as hinted at in Eq. (15). As mentioned above, there exist regularization techniques to normalize the Lipschitz constant of a NN. The quantity $|f_{\mathrm{NN}}(0)|^2$ is also hard to estimate in a general setting, and we offer no general bound on it. The quantity $\mathbb{E}[\|X\|_2^2]$ depends naturally on the distribution p_X, but we can expect it to grow with the dimension d of X. A key variable in which we get a speedup is the error ϵ, allowing us to achieve the same error tolerance with quadratically fewer samples.

B. Forward gradient and other training methods

Now we discuss the *training* part. We discuss methods for training the NNs when the input is distributed according to a known distribution. We first discuss numerical differentiation, then proceed to methods based on AD. In particular, we introduce the forward gradient method from Ref. [79].

We first recall basic properties of NNs. Let $\{\theta_k\}_k$ denote the set of trainable parameters in the NN, where $k \in \{1, \ldots, n_\theta\}$, n_θ being the number of trainable parameters. The number of trainable parameters in a feedforward NN is given by [87]

$$n_\theta := \sum_{l=1}^{L-1} n_l(n_{l+1} + 1), \quad (20)$$

where L is the number of layers, n_l is the number of neurons in the lth layer and the additional n_l parameters per layer are the biases. We can state the following simple fact:

Fact 1. If in a classical feedforward NN of interest, the number of layers is constant in the input dimension d, and the number of neurons per layer grows at most linearly in d, we have that for the number of trainable parameters n_θ,

$$n_\theta = O(d^2). \quad (21)$$

For one training step updating the weights of the NNs of the architecture from Ref. [25] via gradient descent,

$$\theta^{(j+1)} = \theta^{(j)} - \eta \nabla l(\theta^{(j)}), \quad (22)$$

where η is the learning rate. Due to the nonconvex training landscape of the NNs, provable guarantees on how many training steps are needed to train the NNs to a satisfactory precision are in general not possible. Under various assumptions, provable guarantees can be derived for gradient descent and stochastic gradient descent [88].

When updating the weights in NNs, a straightforward method is to resort to numerical differentiation. Numerical differentiation then approximates the derivative with respect to an individual weight in the NN, θ_k, as

$$\frac{\partial}{\partial \theta_k} f_{\mathrm{NN}}(X; \theta_k) \approx \frac{f_{\mathrm{NN}}(X; \theta_k + h) - f_{\mathrm{NN}}(X; \theta_k - h)}{2h}, \quad (23)$$

for a suitably chosen h. Compared to AD-based methods such as the backpropagation, numerical differentiation may suffer from numerical issues, i.e., the truncation error resulting from the finite difference approximations used to approximate derivatives. For one training step, when using numerical differentiation, we require $O(n_\theta)$ evaluations of the loss function from Eq. (8) to get an estimate of the gradient.

In contrast to numerical differentiation, with backpropagation the cost for a single training step is $O(1)$ times the runtime for the evaluation of the NN [89]. As a consequence, backpropagation is widely used to train NNs. However, backpropagation comes with the downside of memory requirements that are greater and harder to predict compared to methods based on forward mode AD, where the memory requirements are simply twice that of the function evaluation [82].

A possible alternative to backpropagation and numerical differentiation is the forward gradient method presented in Ref. [79], which estimates the so-called forward gradient,

$$(\nabla f_{\mathrm{NN}} \cdot v)v, \quad (24)$$

which, for a suitably chosen vector v, constitutes an unbiased estimate of the gradient ∇f_{NN}. A key ingredient for the forward gradient method in real world applications is, as for backpropagation, AD. However, using the forward gradient method does not necessitate holding on to intermediate activations, unlike in the case of backpropagation, as it relies on forward-mode AD, as opposed to reverse-mode AD. Being based on AD, the forward gradient method does not suffer

from numerical issues in the way numerical differentiation may do. In terms of the runtime, there is only a constant overhead compared to the evaluation of the function (i.e., the NN) [89]. The forward gradient method, by using AD and with a runtime with only a constant overhead over the evaluation of the NN (represented by the function f_{NN}), returns not only the output of the NN but also $\nabla f_{\mathrm{NN}} \cdot v$ in a single forward run, without computing ∇f_{NN} itself. Here the gradient is taken with respect to the trainable parameters of the NN and v is a vector along which the gradient is projected. The computation of ∇f_{NN} with the forward gradient method would, as the authors of Ref. [79] point out, require $O(n_\theta)$ evaluations, by choosing v as each basis vector once. In order to be competitive with backpropagation, they need to work with $O(1)$ evaluations, not $O(n_\theta)$. They have to thus choose v such that the overall sensitivity is attributed back to each individual weight parameter in the NN. This feat is achievable by choosing the entries of v to be independently and identically distributed (iid) according to $\mathcal{N}(0, 1)$. In Ref. [79], the authors show that the forward gradient $(\nabla f_{\mathrm{NN}} \cdot v)v$ is an unbiased estimator of the gradient ∇f_{NN}. Furthermore, they provide numerical results indicating that the forward gradient method is competitive with backpropagation with regard to the runtime as well as to number of update iterations to bring the value of the loss function to a certain threshold, for different network architectures. What is more, the forward gradient method requires significantly less memory than backpropagation, as it does not require the storage of all intermediate steps, as pointed out in Ref. [79]. It is worth highlighting, that the forward gradient is by no means the only noisy estimate of a gradient used in machine learning. In stochastic gradient descent (SGD), one estimates the gradient with only a small subset of the available data [90,91]. While the convergence speed is limited by the noisy approximation of the true gradient, SGD is nevertheless widely used in practice. There exist several techniques for dealing with noisy estimates of gradients in the field of machine learning. Examples of these techniques are gradient clipping [92], whereby each entry of the gradient exceeding a certain threshold in absolute value is set back to that threshold, or gradient norm scaling [93], where the norm of the gradient is scaled down to prevent so-called "gradient explosions," which may occur otherwise.

Note that there also exist quantum algorithms which can compute gradients of functions, where the runtime is linear in the runtime of the function itself [94,95]. However, these methods require phase oracles or probability oracles, which is less practical in our case of interest, see Sec. IV. Going forward, we discuss how we may apply the methods discussed above, in particular the forward gradient method in the quantum case.

C. Training neural networks with quantum-accelerated Monte Carlo

We describe and compare different ways of applying the forward gradient method in the classical and quantum setting and also compare it to other methods for training a NN.

When combining the forward gradient method with QAMC and AD, there are *a priori* two options regarding how to estimate the forward gradient. In the first option, v is loaded

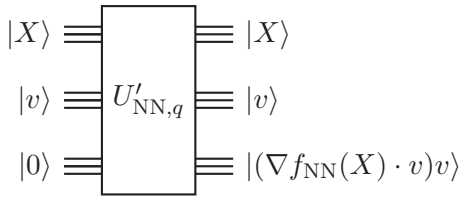

FIG. 7. Unitary evaluating the forward gradient, with v encoded in a quantum state.

in the form of a (quantum mechanical) superposition state, as in Definition 5, according to its distribution p_v. In this option, we hope to get the speedup from using QAMC, but expect to get a slowdown from having to resort to estimating each of the n_θ entries of the forward gradient separately, as QAMC only allows for the estimation of scalar quantities (see the argument in Appendix B 4 b). In the second option we sample v classically, which will prevent us from getting the speedup from QAMC for v (but still for X). This is motivated by only estimating the scalar $\nabla f_{\mathrm{NN}} \cdot v$ (as opposed to a n_θ-dimensional vector), such that we hope to avoid the slowdown in n_θ. Furthermore, it would allow us to use classical Multivariate Monte Carlo (MVMC) (see Appendix B 4 b). It is worth pointing out that since the application of forward mode AD (as well as reverse mode AD) results in a runtime with only a constant overhead compared to just evaluating the NN, it is meaningful to compare the query complexities to the unitaries implementing the evaluation of the NN, with or without some form of AD [89].

We proceed to discuss the first option, where we prepare v (as well as X) as a quantum state according to their respective distribution, as in Definition 5 and estimate each entry of the forward gradient using QAMC. In this case, the gradient we estimate with QAMC is the weighted average of the gradients, weighed by the probabilities of the inputs X of the NNs, as well as those in v. We name the unitary implementing the computation of the forward gradient for the first option $U'_{\mathrm{NN},q}$, as shown in Fig. 7. Recall that, as we outline in Appendix A, any classical circuit can be implemented on a quantum circuit with only a small overhead. Thus, the classical circuit computing the forward gradient can be adapted to also be represented in a quantum circuit.

In this case, the quantum state for a single NN before the unitary is

$$\sum_{i,j} \sqrt{p_{X,i}}\sqrt{p_{v,j}}|X_i\rangle|v_j\rangle|0\rangle, \tag{25}$$

where v is distributed according to p_v. As the output of the unitary $U'_{\mathrm{NN},q}$ we get

$$\sum_{i,j} \sqrt{p_{X,i}}\sqrt{p_{v,j}}|X_i\rangle|v_j\rangle|[\nabla f_{\mathrm{NN}}(X_j) \cdot v_j]v_j\rangle. \tag{26}$$

When estimating the mean of $[\nabla f_{\mathrm{NN}}(X) \cdot v]v$ using QAMC, we now have to consider the randomness from X, quantified by p_X, as well as that of v, quantified by p_v. One issue pops up with this method. In order to estimate the forward gradient, we need to apply QAMC to estimate each entry of the forward gradient, leading to a slowdown of $O(n_\theta)$. Recall

that n_θ is the number of trainable parameters in the NN, which corresponds to the dimension of $[\nabla f_{\mathrm{NN}}(X)\cdot v]v$. Therefore, while we might get a speedup in the error ϵ, we still expect a slowdown in n_θ compared to classical backpropagation. In order to derive the query complexity for estimating the forward gradient in the case where X and v are encoded in quantum states according to their distributions p_X and p_v, respectively, we proceed to upper bound the variance of each entry of the forward gradient with respect to both sources of randomness. Next, we derive an upper bound on the variance with respect to p_X and p_v of the jth component of $[\nabla f_{\mathrm{NN}}(X)\cdot v]v$, i.e., $\mathbb{V}[[(\nabla f_{\mathrm{NN}}(X)\cdot v)v]^{(j)}]$, which we will need to quantify the query complexity. To make our notation more compact, we will refer to $\nabla f_{\mathrm{NN}}(X)$ as ∇f_{NN}. We have that

$$\mathbb{V}[[(\nabla f_{\mathrm{NN}}\cdot v)v]^{(j)}] = \mathbb{V}[(\nabla f_{\mathrm{NN}}\cdot v)v^{(j)}] = \mathbb{V}\left[\sum_{k=1}^{d}(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}\right] = \sum_{k=1}^{d}\mathbb{V}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}] + \sum_{k\neq i}\mathbb{E}[\{(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)} - \mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}]\}\{(\nabla f_{\mathrm{NN}})^{(i)}v^{(i)}v^{(j)} - \mathbb{E}[(\nabla f_{\mathrm{NN}})^{(i)}v^{(i)}v^{(j)}]\}], \quad (27)$$

where we used the formula for the variance of a sum in the last step. The second summand contains the covariance terms, which we have rewritten in the form of the definition in terms of expectation values to allow for further analysis. The covariance terms are

$$\begin{aligned}&\sum_{k\neq i}\mathbb{E}[\{(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)} - \mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}]\}\{(\nabla f_{\mathrm{NN}})^{(i)}v^{(i)}v^{(j)} - \mathbb{E}[(\nabla f_{\mathrm{NN}})^{(i)}v^{(i)}v^{(j)}]\}]\\ &= \sum_{k\neq i}\mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}(\nabla f_{\mathrm{NN}})^{(i)}v^{(i)}v^{(j)}] + \mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}\mathbb{E}[(\nabla f_{\mathrm{NN}})^{(i)}v^{(i)}v^{(j)}]]\\ &\quad + \mathbb{E}[\mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}](\nabla f_{\mathrm{NN}})^{(i)}v^{(i)}v^{(j)}] + \mathbb{E}[\mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}]\mathbb{E}[(\nabla f_{\mathrm{NN}})^{(i)}v^{(i)}v^{(j)}]]. \quad (28)\end{aligned}$$

Note that in each of the four terms in the sum in Eq. (28), there is at least one entry of v that appears only once inside an expectation value because in each term $k\neq i$ (it is possible that $j=k$ or $j=i$). Since the individual components of v are iid with mean 0 each of the four terms in the sum in Eq. (28) will vanish in every summand. We continue with our analysis of the variance of the output of the NN we aim to estimate using QAMC, namely,

$$\mathbb{V}[[(\nabla f_{\mathrm{NN}}\cdot v)v]^{(j)}] = \sum_{k=1}^{d}\mathbb{V}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}]. \quad (29)$$

For the summands from Eq. (29) we make a case distinction to separate the cases where $k\neq j$ from the case where $k=j$. Introducing

$$g_{\max} =: \|\nabla f_{\mathrm{NN}}\|_\infty^2, \quad (30)$$

we have for $k\neq j$

$$\begin{aligned}&\mathbb{V}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}]\\ &= \mathbb{E}[[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}]^2] - \mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}]^2\\ &= \mathbb{E}[[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}]^2]\\ &= \mathbb{E}[[(\nabla f_{\mathrm{NN}})^{(k)}]^2]\mathbb{E}[(v^{(k)})^2]\mathbb{E}[(v^{(j)})^2]\\ &= \mathbb{E}[[(\nabla f_{\mathrm{NN}})^{(k)}]^2]\\ &\leqslant \mathbb{E}\big[\|\nabla f_{\mathrm{NN}}\|_\infty^2\big] \leqslant g_{\max}, \quad (31)\end{aligned}$$

where the second equality is due to the fact that $k\neq j$ and thus the three terms $(\nabla f_{\mathrm{NN}})^{(k)}$, $v^{(k)}$ and $v^{(j)}$ are independent, with $\mathbb{E}[v^{(k)}]=\mathbb{E}[v^{(j)}]=0$. The third equality again follows from the independence of $(\nabla f_{\mathrm{NN}})^{(k)}$, $v^{(k)}$, and $v^{(j)}$. The first equality rests on $\mathbb{V}[v^{(k)}]=\mathbb{V}[v^{(j)}]=1$ being true. In the case where $k=j$ we have

$$\begin{aligned}&\mathbb{V}[(\nabla f_{\mathrm{NN}})^{(k)}(v^{(j)})^2]\\ &= \mathbb{E}[[(\nabla f_{\mathrm{NN}})^{(k)}]^2(v^{(j)})^4] - \mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(j)}]^2\\ &= \mathbb{E}[[(\nabla f_{\mathrm{NN}})^{(k)}]^2]\mathbb{E}[(v^{(j)})^4] - \mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}]^2\mathbb{E}[v^{(j)}]^2\\ &= 3\mathbb{E}[[(\nabla f_{\mathrm{NN}})^{(k)}]^2] - \mathbb{E}[(\nabla f_{\mathrm{NN}})^{(k)}]^2 \leqslant 3g_{\max}, \quad (32)\end{aligned}$$

where we used that $\mathbb{E}[(v^{(j)})^4]=3$. Combining these results, we have that

$$\begin{aligned}&\mathbb{V}[[(\nabla f_{\mathrm{NN}}\cdot v)v]^{(j)}]\\ &= \sum_{k=1}^{d}\mathbb{V}[(\nabla f_{\mathrm{NN}})^{(k)}v^{(k)}v^{(j)}] \leqslant (d+2)g_{\max}. \quad (33)\end{aligned}$$

This result, which also holds in the classical case, we may use for an upper bound of the variance of the forward gradient, which we want to estimate using QAMC.

Result 2. Forward gradient for NN training with QAMC, first option. Consider a classical NN including its scalar loss function, $f_{\mathrm{NN}}:\mathbb{R}^d\mapsto\mathbb{R}$, with n_θ trainable parameters. Let the function computing the quantity $[\nabla f_{\mathrm{NN}}(X)\cdot v]v$ using forward mode AD be implemented in a quantum circuit in the form of a unitary $U'_{\mathrm{NN},q}$ as seen above, where the gradient is taken with respect to the trainable parameters in the NN. Let the input to the NN be given by a d-dimensional random variable X, governed by a distribution p_X, as well as a n_θ-dimensional random variable v, whose components are distributed iid according to $\mathcal{N}(0,1)$. Then we can estimate the mean output of $[\nabla f_{\mathrm{NN}}(X)\cdot v]v$ with respect to the distributions of X and v using QAMC with a sample complexity, with respect to the unitary $U'_{\mathrm{NN},q}$ computing

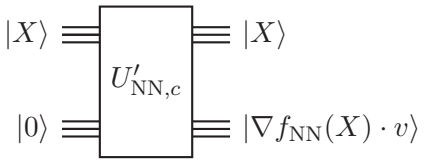

FIG. 8. Unitary evaluating the directional derivative, with v classical.

$[\nabla f_{\mathrm{NN}}(X)\cdot v]v$ of

$$O(n_\theta\sqrt{d g_{\max}}/\epsilon), \tag{34}$$

up to error ϵ in the l_∞ norm with success probability 0.99.

As we expected, we get a slowdown of n_θ in the query complexity, but a speedup in the error tolerance ϵ compared to classically using backpropagation. We proceed to discuss the second option of applying the forward gradient method to the setting at hand.

In the second option, we classically sample the vector v. We use QAMC to estimate the inner product $\nabla f_{\mathrm{NN}}\cdot v$ with respect to the distribution of the input X, p_X, for a fixed classical sample of v. To compute the forward gradient, we classically postprocess $\nabla f_{\mathrm{NN}}\cdot v$ to arrive at $(\nabla f_{\mathrm{NN}}\cdot v)v$. Since $\nabla f_{\mathrm{NN}}\cdot v$ is a scalar, we hope to avoid a slowdown in n_θ. We proceed to provide some equations and circuits to better illustrate this option. Figure 8 displays a unitary also returning the forward gradient. We name this unitary $U'_{\mathrm{NN},c}$.

Similarly to the case in Eq. (14), acting with $U'_{\mathrm{NN},c}$ obtains

$$\sum_i \sqrt{p_{X,i}}|X_i\rangle|0\rangle \to \sum_i \sqrt{p_{X,i}}|X_i\rangle|\nabla f_{\mathrm{NN}}(X_i)\cdot v\rangle. \tag{35}$$

The unitary $U'_{\mathrm{NN},c}$ has access to the fixed sample of v classically and returns the directional derivative using forward mode AD. In this option, the quantity we estimate with QAMC is $\nabla f_{\mathrm{NN}}(X)\cdot v$, which is a scalar. Once we have estimated $\nabla f_{\mathrm{NN}}(X)\cdot v$ we multiply it with the same classical sample of v to get the forward gradient. Carrying out the multiplication of $\nabla f_{\mathrm{NN}}(X)\cdot v$ and v we obtain an overhead in runtime of $O(n_\theta)$. However, we do not need to query the unitary implementing the NN $O(n_\theta)$ times to compute the directional derivative. As outlined in Appendixes B 1 and B 2, the runtime of evaluating the NN (as well as running AD in forward or reverse mode) is $O(n_\theta)$. However, we only query the NN a constant number of times to obtain a sample of the forward gradient. In the case of, e.g., numerical differentiation or the first option for using the forward gradient with QAMC outlined above, we have to query the unitary $O(n_\theta)$ times, in the case of the second option (as in the classical case with backpropagation) only once, to obtain one sample. For now, we proceed to further discuss the second option (where v is classical).

In order to incorporate the second option of evaluating the forward gradient method into our framework of using NNs together with QAMC and quantify a potential speedup, we need to bound the variance of the quantity $\nabla f_{\mathrm{NN}}(X)\cdot v$ with respect to p_X, if we are to estimate it using AD alongside the evaluation of the NN itself. We have

$$\begin{aligned}\mathbb{V}[\nabla f_{\mathrm{NN}}(X)\cdot v] &= \mathbb{E}[|\nabla f_{\mathrm{NN}}(X)\cdot v|^2] - \mathbb{E}[\nabla f_{\mathrm{NN}}(X)\cdot v]^2\\ &\leqslant \mathbb{E}[|\nabla f_{\mathrm{NN}}(X)\cdot v|^2]\\ &\leqslant \mathbb{E}\big[\|\nabla f_{\mathrm{NN}}(X)\|_\infty^2\|v\|_1^2\big]\\ &\leqslant \mathbb{E}\big[\|\nabla f_{\mathrm{NN}}(X)\|_\infty^2 n_\theta\|v\|_2^2\big]\\ &\leqslant g_{\max}n_\theta\|v\|_2^2,\end{aligned} \tag{36}$$

where we used Hölder's inequality in the second inequality and the norm inequalities in the third inequality. We point out that due to the norm inequalities, the term n_θ shows up. We proceed to formulate the following result.

Result 3. Directional derivative for NN training with QAMC, second option. Consider a classical NN including its scalar loss function, $f_{\mathrm{NN}}:\mathbb{R}^d\mapsto\mathbb{R}$, with n_θ trainable parameters. Let the function computing the quantity $\nabla f_{\mathrm{NN}}(X)\cdot v$ using forward mode AD be implemented in a quantum circuit in the form of a unitary $U'_{\mathrm{NN},c}$, where the gradient is taken with respect to the trainable parameters in the NN. Let the input to the NN be given by a d-dimensional random variable X, governed by a distribution p_X. Furthermore, let the components of v be fixed as classical iid samples from the distribution $\mathcal{N}(0,1)$. Then we can estimate the mean output of $\nabla f_{\mathrm{NN}}(X)\cdot v$ with respect to the input distribution p_X using QAMC with a query complexity, with respect to $U'_{\mathrm{NN},c}$, of

$$\tilde{O}(\sqrt{g_{\max}n_\theta}\|v\|_2/\epsilon) \tag{37}$$

up to additive error ϵ with probability 0.99.

As mentioned above, the entries of v are iid according to $\mathcal{N}(0,1)$. Therefore, $\|v\|_2^2$ follows a χ^2 distribution with mean n_θ and variance $2n_\theta$. Using Chebyshev's inequality, we can upper bound the probability that $\|v\|_2^2$ deviates from its mean by more than k standard deviations,

$$P\big[|\|v\|_2^2 - n_\theta| \geqslant k\sqrt{2n_\theta}\big] \leqslant \frac{1}{k^2}. \tag{38}$$

Thus, $\|v\|_2$ grows as $\sqrt{n_\theta}$, since, due to Jensen's inequality,

$$\mathbb{E}\big[\sqrt{\|v\|_2^2}\big] \leqslant \sqrt{\mathbb{E}\big[\|v\|_2^2\big]} = \sqrt{n_\theta}. \tag{39}$$

We see that with the second option n_θ shows up in the query complexity, despite estimating only a scalar.

So far in the second option we have treated v as a fixed sample. However, in practice one would also have to sample v multiple times. To upper bound the error when considering the stochasticity originating from X as well as v, we proceed as follows. Let w denote our estimate of $\nabla f_{\mathrm{NN}}\cdot v$ as in Result 3, taking into account the stochasticity of X for a fixed sample of v. Let $(\nabla f_{\mathrm{NN}}\cdot v)v$ denote the forward gradient where the only source of error stems from the stochasticity of v, and let ∇f_{NN} be the true gradient. Then we have

$$\begin{aligned}&\|wv - \nabla f_{\mathrm{NN}}\|_\infty\\ &\quad= \|w - (\nabla f_{\mathrm{NN}}\cdot v)v + (\nabla f_{\mathrm{NN}}\cdot v)v - \nabla f_{\mathrm{NN}}\|_\infty\\ &\quad\leqslant \|wv - (\nabla f_{\mathrm{NN}}\cdot v)v\|_\infty + \|(\nabla f_{\mathrm{NN}}\cdot v)v - \nabla f_{\mathrm{NN}}\|_\infty\\ &\quad= \|[w - (\nabla f_{\mathrm{NN}}\cdot v)]v\|_\infty + \|(\nabla f_{\mathrm{NN}}\cdot v)v - \nabla f_{\mathrm{NN}}\|_\infty\\ &\quad\leqslant 3|w - (\nabla f_{\mathrm{NN}}\cdot v)| + \|(\nabla f_{\mathrm{NN}}\cdot v)v - \nabla f_{\mathrm{NN}}\|_\infty,\end{aligned} \tag{40}$$

where we made use of the following observation in the last inequality. Since each entry of v is distributed according to $\mathcal{N}(0,1)$, we may truncate each entry of v at ± 3, as the error contributions from the tails of the Gaussian outside of

this range occur with a probability of roughly 0.003. This argument may also be extended to greater constants, e.g, if the Gaussians are truncated at ± 7, which is still a constant, the probability of the Gaussian being outside of this range is around 2.56×10^{-12}. In Result 3 we outlined the query complexity for estimating w such that the first summand from the last line of Eq. (40) is below a certain error ϵ. In order to do the same for the second summand, i.e., the query complexity considering samples of v to estimate the gradient up to a certain error tolerance, we can make use of Lemma 7, as v is sampled classically. To estimate the gradient ∇f_{NN} up to error ϵ in the l_∞ norm using the forward gradient $(\nabla f_{\mathrm{NN}} \cdot v)v$, i.e.,

$$\|(\nabla f_{\mathrm{NN}} \cdot v)v - \nabla f_{\mathrm{NN}}\|_\infty < \epsilon, \tag{41}$$

we upper bound $\|(\nabla f_{\mathrm{NN}} \cdot v)v\|_\infty$ as follows: Making use of the same arguments as in Eq. (36), we have

$$\begin{aligned} \|(\nabla f_{\mathrm{NN}} \cdot v)v\|_\infty &\leqslant 3|\nabla f_{\mathrm{NN}}(X) \cdot v| \\ &\leqslant 3\|\nabla f_{\mathrm{NN}}(X)\|_\infty \sqrt{n_\theta}\|v\|_2 \\ &= O(\sqrt{g_{\max} n_\theta}\|v\|_2) \\ &= O(\sqrt{g_{\max} n_\theta}), \end{aligned} \tag{42}$$

in expectation, where we use the same argument as in Eq. (40) for upper bounding the greatest entry of v. Consequently, with success probability δ, we need

$$O\left(\frac{g_{\max} n_\theta^2}{\epsilon^2} \log \frac{n_\theta}{\delta}\right), \tag{43}$$

many samples in expectation. Since for each sample of v, we require the sample complexity outlined in Result 3, we formulate the following corollary:

Corollary 1 (Forward gradient for NN training with QAMC, second option). Consider a classical NN including its scalar loss function, $f_{\mathrm{NN}} : \mathbb{R}^d \mapsto \mathbb{R}$, with n_θ trainable parameters. Let the function computing the quantity $\nabla f_{\mathrm{NN}}(X) \cdot v$ using forward mode AD be implemented in a quantum circuit in the form of a unitary $U'_{\mathrm{NN},c}$, where the gradient is taken with respect to the trainable parameters in the NN. Let the input to the NN be given by a d-dimensional random variable X, governed by a distribution p_X. Furthermore, let the components of v be sampled as classical iid samples from the distribution $\mathcal{N}(0, 1)$. Then we can estimate the mean of $[\nabla f_{\mathrm{NN}}(X) \cdot v]v$ with respect to the input distribution p_X and the distribution p_v of v using QAMC to estimate the directional derivative $\nabla f_{\mathrm{NN}}(X) \cdot v$ for a fixed sample of v, followed by classical MVMC estimation in v as in Lemma 7 with a query complexity, with respect to $U'_{\mathrm{NN},c}$, of

$$\tilde{O}\left(\frac{n_\theta^{2.5} g_{\max}^{1.5}}{\epsilon^3}\right). \tag{44}$$

up to error ϵ in the l_∞ norm with probability 0.99.

We proceed to summarize the query complexities for estimating the gradient of a NN in different settings and using different methods. We also include the purely classical results. The query complexities are for estimating the gradient ∇f_{NN} up to error ϵ in the l_∞ norm with success probability 0.99. Some further comments on are in order. Compared with the classical case of using the forward gradient method to train the NN, we get a speedup in $g_{\max}$ and ϵ by using QAMC to estimate $(\nabla f_{\mathrm{NN}} \cdot v)v$ in the fully quantum case. However, in the classical case one would presumably resort to backpropagation to compute the gradient of the NN with respect to the trainable parameters instead of using the forward gradient method (provided memory constraints are not a concern), as no additional stochasticity is introduced. When using backpropagation, the randomness with respect to the input X would remain, but there is no need for sampling v. Without the stochasticity introduced by v, the sample complexity is proportional to ϵ^{-2}, as per Lemma 7. It is also worth pointing out that in all the scenarios where X is quantum, we have a slowdown of at least d^2 compared to classical backpropagation. This stems from the fact that QAMC, as seen in Lemma 6, only allows for the estimation of a scalar at a time, unlike in the classical case (see the discussion in Appendix B 4 b). Nevertheless, we have shown that with the purely quantum cases, we can achieve a speedup in ϵ, as we had hoped for. The case where X is quantum and v is classical did not turn out to give us the best of both worlds, as we had envisaged. This is due to the fact that, while we can get a speedup in estimating the directional derivative (in an inner loop), we have an outer loop where we sample v classically, resulting in the slowdown in ϵ. This consideration is shown mathematically in Eq. (40).

To conclude, as seen in Result 1, when using QAMC to estimate the loss function of the NN where the input is sampled from a known distribution, we achieve a quadratic speedup for the query complexity in the error tolerance ϵ compared to the classical case, as was our goal when applying QAMC. When training classical NNs implemented in quantum circuits where the input is given by samples from a known distribution, we compared several classical as well as quantum alternatives. As seen in Table I, by using QAMC to estimate the gradient of the NN, we get a slowdown in d compared to classical backpropagation, but a speedup in ϵ. This holds when applying QAMC to backpropagation, the forward gradient method (in the fully quantum scenario) as well as numerical differentiation. However, compared to when QAMC is applied to backpropagation, an application of QAMC to the forward gradient method and numerical differentiation incurs a slowdown of $\sqrt{d}$. The forward gradient method, however, being based on AD, is less prone to numerical issues as numerical differentiation, and has the potential for being less memory intensive than backpropagation, which is vital in quantum circuits.

IV. QUANTUM-ACCELERATED MONTE CARLO METHODS AND PARTIAL DIFFERENTIAL EQUATIONS

In this section we proceed with the algorithm from Ref. [25] to explore the possible application of quantum-accelerated Monte Carlo (QAMC). Our goal is to show that using QAMC, the deep-learning architecture from Ref. [25] requires fewer samples and thus fewer evaluations of the neural networks (NNs) to estimate the loss function and the gradients for the NNs up to a certain error tolerance. As outlined in Sec. I B, the deep-learning algorithm from Ref. [25] uses NNs to solve nonlinear partial differential equations (PDEs) by reformulating the nonlinear PDE as a stochastic differential equation (SDE). In order to approximate the

TABLE I. Query complexities (to the NN) to estimate the gradient of a NN with respect to its trainable parameters for different methods when the input X is sampled from a distribution p_X and v (for the forward gradient) from p_v, either classically (C) or quantumly (Q). We make use of Fact 1, replacing n_θ by d^2 in the complexities and taking the expectation value for the terms containing norms of v, as in Eqs. (38) and (39).

X	v	Method	Query complexity
C		Backpropagation	$\tilde{O}(g_{\max}/\epsilon^2)$ [Lemma 7 and Eq. (30)]
C	C	Forward gradient	$\tilde{O}(d^4 g_{\max}/\epsilon^2)$ [Lemma 7 and Eq. (42)]
C		Numerical differentiation	$\tilde{O}(d^2 g_{\max}/\epsilon^2)$ [Equations (23), (30), and (B23)]
Q		Backpropagation	$\tilde{O}(d^2\sqrt{g_{\max}}/\epsilon)$ [Lemma 6 and Eq. (30)]
Q	C	Forward gradient	$\tilde{O}(d^5 g_{\max}^{1.5}/\epsilon^3)$ (Corrolary 1)
Q	Q	Forward gradient	$\tilde{O}(d^{2.5}\sqrt{g_{\max}}/\epsilon)$ (Result 2)
Q		Numerical differentiation	$\tilde{O}(d^{2.5}\sqrt{g_{\max}}/\epsilon)$ [Lemma 6 and Eq. (33)]

gradient in each temporal discretization step of the function described by the nonlinear PDE, a NN is employed to learn the spatial gradient based on samples from the SDE governing the spatial variable. In order to achieve a quantum speedup in the algorithm presented in Ref. [25], one possibility consists of using quantum subroutines for estimating the loss function and the solution to the PDE, which depend on the stochastic process governing the spatial variable. A method which may allow us to do so is QAMC, presented in Appendix B 4 a, by addressing the bottleneck imposed by Chebyshev's inequality.

A. Applying quantum-accelerated Monte Carlo to loss function estimation

Setting. We now proceed to apply QAMC to the architecture from Ref. [25] with the hope of achieving a quantum speedup. We briefly outline how we aim to do so, and then proceed to elaborate on the individual steps. Our goal is to estimate the payoff function f_p from Eq. (9), $|g(\hat{X}_{t_N}) - \hat{u}_{t_N}|^2$, as well as the gradients of the parameters of the NNs, with QAMC.

In order to apply QAMC to the setting from Ref. [25], we begin by representing the stochastic process governing the stochastic process X_t [see Eq. (2)] in its differential form as follows:

$$\mathrm{d}X_t = \mu(t, X_t) + \sigma(t, X_t)\mathrm{d}W_t, \tag{45}$$

where μ and σ are real functions and W_t is a standard Brownian motion (as defined in Sec. I B). Furthermore, we also represent u_t [see Eq. (3)] in its differential form as follows:

$$\begin{aligned}\mathrm{d}u_t =& -f(t, X_t, u(t, X_t), \sigma^\top(t, X_t)\nabla u(t, X_t)) \\ &+ \nabla u(t, X_t)^\top \sigma(t, X_t)\mathrm{d}W_t,\end{aligned} \tag{46}$$

where f is a real function and in our setting, as seen in Eq. (7), $\nabla u(t, X_t)^\top \sigma(t, X_t) = (\sigma^\top \nabla u)(t, X_t)$ is a function represented by the NNs. Next, we need to make a set of assumptions, the use of which we will lay out shortly.

Assumption 1. We assume that μ, σ, f, and $\sigma^\top \nabla u$ are Lipschitz continuous, as introduced in Sec. I A, on the domain of interest in the squared l_2 norm.

By making Assumption 1, we derive the following growth bound on the functions mentioned in Assumption 1. For μ we have

$$\begin{aligned}\|\mu(t, X)\|_2^2 &= \|\mu(t, X) - \mu(0, 0) + \mu(0, 0)\|_2^2 \\ &\leqslant \|\mu(t, X) - \mu(0, 0)\|_2^2 + \|\mu(0, 0)\|_2^2 \\ &\leqslant L_\mu\left(t^2 + \|X\|_2^2\right) + \|\mu(0, 0)\|_2^2 \\ &\leqslant \left(L_\mu + \|\mu(0, 0)\|_2^2\right)\left(1 + t^2 + \|X\|_2^2\right).\end{aligned} \tag{47}$$

Similar bounds may be derived analogously for the other functions mentioned in Assumption 1. We continue with our next assumptions.

Assumption 2. We assume that $\mathbb{E}(\|X_{t_0}\|_2^m) < \infty$ for $m \geqslant 0$.

Assumptions 1 and 2 guarantee the existence and uniqueness of a strong solution of the SDEs, meaning that for every Brownian path that W_t may take, the SDEs are guaranteed to have a unique solution [96]. The next assumption we will make use of as this section progresses.

Assumption 3. We assume that the function inside the expectation value in Eq. (8), which we call the payoff function f_p [see Eq. (9)], is Lipschitz continuous on the domain of interest in the sense that

$$\begin{aligned}&\left|f_p\left(\left\{\hat{X}_{t_n}\right\}_{0\leqslant n\leqslant N}, \left\{\Delta W_{t_n}\right\}_{0\leqslant n\leqslant N}\right)\right. \\ &\quad\left. - f_p\left(\left\{\tilde{X}_{t_n}\right\}_{0\leqslant n\leqslant N}, \left\{\Delta \tilde{W}_{t_n}\right\}_{0\leqslant n\leqslant N}\right)\right|^2 \\ &\leqslant L_{f_p} \sup_{0\leqslant n\leqslant N} \left\|\hat{X}_{t_n} - \tilde{X}_{t_n}\right\|_2^2.\end{aligned} \tag{48}$$

Analogously to Eq. (47), we furthermore derive the following growth bound for f_p:

$$\begin{aligned}&f_p\left(\left\{\hat{X}_{t_n}\right\}_{0\leqslant n\leqslant N}, \left\{\Delta W_{t_n}\right\}_{0\leqslant n\leqslant N}\right) \\ &\leqslant \left(L_{f_p} + \left|f_p\left(\left\{\hat{Y}_{t_n}\right\}_{0\leqslant n\leqslant N}\right)\right|^2\right)\left(1 + \sup_{0\leqslant n\leqslant N}\left\|\hat{X}_{t_n}\right\|_2^2\right) \\ &:= K_{f_p}\left(1 + \sup_{0\leqslant n\leqslant N}\left\|\hat{X}_{t_n}\right\|_2^2\right),\end{aligned} \tag{49}$$

where Y_{t_n} signifies the path such that $\sup_{0\leqslant n\leqslant N}\|\hat{X}_{t_n} - Y_{t_n}\|_2^2 = \sup_{0\leqslant n\leqslant N}\|\hat{X}_{t_n}\|_2^2$ for all other paths $\hat{X}_{t_n}$.

Note that the payoff function presented in Eq. (9), $|g(\hat{X}_{t_N}) - \hat{u}_{t_N}|^2$, is not globally Lipschitz continuous in general, and also involves outputs of the nonlinear function f. To avoid violating the Lipschitz condition, we argue that in any practical setting, one would only deal with a bounded interval

of interest. Assuming that the loss function does not contain any singularities on the interval of interest, it is Lipschitz continuous on the same interval, with the Lipschitz constant being the maximum of its derivative on the interval. The same argument can be applied to justify the growth bound assumption.

As mentioned above, we make Assumptions 1 and 2 so that Eqs. (45) and (46) are guaranteed to have a unique strong solution [96]. For the solution of the SDE it then holds that

$$\sup_{t_0\leqslant t\leqslant T}\mathbb{E}\big[\|X_t\|_2^2\big]<\infty, \tag{50}$$

as well as

$$\mathbb{E}\Big[\sup_{0\leqslant n\leqslant N}\|X_{t_n}\|_2^2\Big]\leqslant\mathbb{E}\big[C\big(1+\|X_{t_0}\|_2^2\big)\big], \tag{51}$$

where C may depend on the time interval $T-t_0$ and on the Lipschitz constants of μ and σ [96]. If there is a solution to an SDE of the form in Eqs. (45) and (46) for each given path the Brownian process W_t takes, we say that the SDE has a strong solution [96]. Assumptions 1 and 2 are satisfied (at least on a bounded domain) for several important special cases surveyed in Ref. [25], such as the nonlinear Black-Scholes equation, the Hamilton-Jacobi-Bellman equation as well as the Allen-Cahn equation. In the same three example cases, Assumption 3 is satisfied for bounded domains.

In order to present the cost of using QAMC to solve PDEs with the architecture from Ref. [25] we assume that access to $t_0, u_0, \sigma^\top\nabla u_0, X_{t_0}, \mu, \sigma$, and f is given via unitaries, which we denote by $U_{t_0}, U_{u_0}, U_{\sigma^\top\nabla u_0}, U_{X_{t_0}}, U_\mu, U_\sigma$, and U_f, respectively. In addition, we assume that we have access to a unitary U_{Gauss} which can prepare a state of the form presented in Definition 5 with a Gaussian distribution. We also assume that evaluating the NNs (i.e., carrying out a single feedforward pass) is done by querying a unitary U_{NN}. Since the architecture of all the NNs in Ref. [25] is the same, we also treat the unitaries for the different NNs in the architecture from Ref. [25] as having the same cost from a query complexity point of view.

Variance bound. In order to apply Lemma 6 to the architecture outlined in Ref. [25] and quantify a potential quantum speedup, we proceed to find an upper bound on the variance λ^2 of the quantity whose expectation value we aim to estimate, namely the payoff function from Eq. (9). We have for λ^2:

$$\begin{aligned}
\lambda^2&=\mathbb{V}\big[f_p\big(\{\hat{X}_{t_n}\}_{0\leqslant n\leqslant N},\{\Delta W_{t_n}\}_{0\leqslant n\leqslant N}\big)\big]\\
&\leqslant\mathbb{E}\big[\big|f_p\big(\{\hat{X}_{t_n}\}_{0\leqslant n\leqslant N},\{\Delta W_{t_n}\}_{0\leqslant n\leqslant N}\big)\big|^2\big]\\
&\leqslant\mathbb{E}\Big[K_{f_p}\Big(1+\sup_{0\leqslant n\leqslant N}\big\|\hat{X}_{t_n}\big\|_2^2\Big)\Big]\\
&=\mathbb{E}\Big[K_{f_p}\Big(1+\sup_{0\leqslant n\leqslant N}\big\|\hat{X}_{t_n}-X_{t_n}+X_{t_n}\big\|_2^2\Big)\Big]\\
&\leqslant\mathbb{E}\Big[K_{f_p}\Big(1+\sup_{0\leqslant n\leqslant N}\big\|\hat{X}_{t_n}-X_{t_n}\big\|_2^2+\sup_{0\leqslant n\leqslant N}\big\|X_{t_n}\big\|_2^2\Big)\Big]\\
&\leqslant K_{f_p}\Big(1+\mathbb{E}\Big[\sup_{0\leqslant n\leqslant N}\big\|\hat{X}_{t_n}-X_{t_n}\big\|_2^2\Big]+\mathbb{E}\Big[\sup_{0\leqslant n\leqslant N}\big\|X_{t_n}\big\|_2^2\Big]\Big)\\
&\leqslant K_{f_p}\big[1+K_2(\Delta t)^{2r}+C\big(1+\|X_{t_0}\|_2^2\big)\big]=:\lambda_{\max}^2,
\end{aligned} \tag{52}$$

where the first inequality follows from the definition of the variance, the second inequality stems from the growth bound of the payoff function (see Assumption 3), and the last inequality stems from the definition the strong order r in Eq. (B42) and from Eq. (51), thanks to Assumptions 1 to 3. Note that the bound in Eq. (52) also holds in the classical case, as we have not yet employed any quantum subroutines.

Error analysis. Before going through the application of QAMC to the architecture from Ref. [25], we comment on the error sources when estimating the payoff function f_p, in order to understand what effects different errors have and how they differ in the classical and quantum scenario. Note that the inaccuracy of the NNs when representing the spatial gradient as well as the error stemming from the temporal discretization of the SDEs are not error sources hindering us at accurately estimating the payoff function, rather they have an effect on the (true) value of the solution. When estimating the payoff function, we consider two kinds of errors:

(1) The discretization error of the Gaussian increments ΔW_t. The estimate of the payoff function considering only the discretization error of the Gaussian increments ΔW_t we denote as I_2.

(2) The estimation error, when using QAMC. The estimate which contains both kinds of errors we denote by I_3.

In order to bound the error between the ideal estimation of the payoff function (with no error whatsoever), which we denote by I_1, and our actual estimate containing both errors, I_3, we proceed as follows:

$$|I_1-I_3|=|I_1-I_2+I_2-I_3|\leqslant|I_1-I_2|+|I_2-I_3|. \tag{53}$$

In order to upper bound the total error $|I_1-I_3|$ by ϵ we proceed to bound the terms on the right hand side of Eq. (53).

We begin by bounding the first error term, $|I_1-I_2|$, which results from discretizing the Gaussian increments ΔW_{t_n}. The number $N_{\text{Gauss}}=2^{n_{\text{Gauss}}}$ represent how finely we discretize the Gaussian distribution, where n_{Gauss} is the number of qubits we use to represent a single univariate Gaussian distribution. The error resulting from this discretization of the Gaussian increments may be bounded by a Riemann sum. In general, the following holds for a left- or right-rule Riemann sum approximating an integral of a function f_r [97],

$$\left|\int_a^b f_r(x)\,dx-\sum_{k=0}^{n-1}\Delta x f_r(a+k\Delta x)\right|\leqslant\frac{L_{f_r}(b-a)^2}{2n}, \tag{54}$$

where $\Delta x=|a-b|/N$, L_{f_r} is the Lipschitz constant of f_r (on the relevant interval) and n represents how finely we discretize the integral. In the case of two integrals being approximated

by a left- or right-rule Riemann sum, we have

$$\begin{aligned}
&\left|\int_a^b\int_c^d f_r(x,y)\,dydx - \sum_{k=0}^{n-1}\sum_{l=0}^{m-1}\Delta x\Delta y f_r(a+k\Delta x, c+l\Delta y)\right| \\
&\leqslant \left|\int_a^b\int_c^d f_r(x,y)\,dydx - \int_a^b\sum_{l=0}^{m-1}\Delta y f_r(x,c+l\Delta y)\,dx\right| \\
&\quad + \left|\int_a^b\sum_{l=0}^{m-1}\Delta y f_r(x,c+l\Delta y)\,dx - \sum_{k=0}^{n-1}\sum_{l=0}^{m-1}\Delta x\Delta y f_r(a+k\Delta x, c+l\Delta y)\right| \\
&\leqslant \int_a^b\left|\int_c^d f_r(x,y)\,dy - \sum_{l=0}^{m-1}\Delta y f_r(x,c+l\Delta y)\right|dx \\
&\quad + \sum_{l=0}^{m-1}\Delta y\left|\int_a^b f_r(x,c+l\Delta y)\,dx - \sum_{k=0}^{n-1}\Delta x f_r(a+k\Delta x, c+l\Delta y)\right| \\
&\leqslant \frac{L_{f_r}(d-c)^2}{2m}|b-a| + \frac{L_{f_r}(b-a)^2}{2n}|d-c|. \qquad (55)
\end{aligned}$$

If, as in our case of interest, $a=c$, $b=d$, and $n=m$ we end up with an error bound of

$$\frac{2L_{f_r}|b-a|^3}{2n}. \qquad (56)$$

We generalize the above error to the case of M integrals and we get an error bound of

$$\frac{ML_{f_r}|b-a|^{M+1}}{2n}. \qquad (57)$$

We return to our setting where we are analyzing the discretization error for the Gaussian increments. If we set the bounds to be at $\pm 3\sqrt{\Delta t}$ (recall that Δt is the variance of our Gaussian increments) away from the mean, respectively, we can consider the error contributions from the tails of the Gaussian which lie outside of our discretized range to be negligible, at roughly 0.003. We then upper bound the error resulting from the discretization of the Gaussian increments by using Eq. (57) as follows:

$$|I_1 - I_2| \leqslant \frac{NdL_{f_p}|6\Delta t|^{Nd+1}}{2N_{\text{Gauss}}}, \qquad (58)$$

where N, t_0, and T are as seen in Fig. 1, and L_{f_p} is the Lipschitz constant of the payoff function, see Assumption 3. Note that this result also holds for the classical case (i.e., using classical MC methods) and is in line with a similar result in Ref. [98]. In order to upper bound the error from discretizing the Gaussian increments by $\epsilon/3$, we choose,

$$N_{\text{Gauss}} \geqslant \frac{3NdL_{f_p}|6\Delta t|^{Nd+1}}{2\epsilon} = O\big(\epsilon^{-1-1/r}dL_{f_p}|6\Delta t|^{\epsilon^{-1/r}d+1}\big), \qquad (59)$$

where we inserted the relation between N and ϵ from Eq. (71). Since $N_{\text{Gauss}} = 2^{n_{\text{Gauss}}}$, where n_{Gauss} is the number of qubits we require for a discretization of a one-dimensional univariate Gaussian distribution, we thus have to deploy,

$$n_{\text{Gauss}} = \tilde{O}(\epsilon^{-1/r}d), \qquad (60)$$

qubits for the discretization of the Gaussian increments in the whole architecture from Ref. [25] to achieve the desired precision. In the above calculation we did not make use of any quantum subroutines, and thus the analysis of the error stemming from the discretization of the Gaussian increments holds in the classical case (for bits) as well as in the quantum case (for qubits).

The second error source when estimating the payoff function stems from estimating I_2 with I_3. According to Lemmas 4 and 6 there exists a quantum algorithm that estimates I_2 up to error

$$|I_2 - I_3| \leqslant \epsilon/3, \qquad (61)$$

with probability at least 0.99 with $\tilde{O}(\epsilon^{-1})$ estimations of I_2. Using classical Monte Carlo (MC) methods, we would require $\tilde{O}(\epsilon^{-2})$ estimations. This is where the quantum speedup comes into play.

Quantum circuit. We now proceed to describe the quantum circuit for the proposed algorithm step by step, by writing out the state after each operation, and simultaneously keeping count of the number of unitary calls and arithmetic operations we make. We assume that we have a sufficient number of qubits available to us. We indicate the spare qubits, which we may use in the future, by $|0\cdots 0\rangle$. We start off by calling the unitaries $U_{X_{t_0}}$, U_{t_0}, $U_{\sigma^\top\nabla u_0}$, and U_{u_0} to load the initial values t_0, u_0, $\sigma^\top\nabla u_0$, and X_{t_0}, respectively. Each of these unitaries we call only once for each estimation of the mean in Eq. (8). After calling each of these unitaries, our quantum state is

$$|X_{t_0}\rangle|t_0\rangle|\sigma^\top\nabla u_0\rangle|u_0\rangle|0\cdots 0\rangle. \qquad (62)$$

Note that since the values are classical parameters and will not be in superposition or entangled later on, storing the initial values in qubits is not necessary, they could be introduced into the circuit classically. However, in view of later operations on the quantum circuit, we think it nevertheless makes sense for illustrative reasons. Next, we use N additions to increase t,

$$|X_{t_0}\rangle|t_0\rangle|\sigma^\top\nabla u_0\rangle|u_0\rangle|t_1\rangle\cdots|t_N\rangle|0\cdots 0\rangle. \qquad (63)$$

For the Gaussian increments we need Nd univariate Gaussian random variables (RVs) to represent the N d-dimensional Gaussian RVs, whose entries are identically and independently distributed (iid), loaded in the sense of Definition 5 by a unitary U_{Gauss}. Therefore, we need to call U_{Gauss} Nd times. After calling U_{Gauss} Nd times, we have

$$|X_{t_0}\rangle|t_0\rangle|\sigma^\top\nabla u_0\rangle|u_0\rangle \sum_{k_{t_0,1}=1}^{N_{\text{Gauss}}}\cdots\sum_{k_{t_{N-1},d}=1}^{N_{\text{Gauss}}}\sqrt{p_{k_{t_0,1}}}\cdots\sqrt{p_{k_{t_{N-1},d}}}|k_{t_0,1}\rangle|\Delta W_{t_0,1}(k_{t_0,1})\rangle$$
$$\cdots|k_{t_{N-1},d}\rangle|\Delta W_{t_{N-1},1}(k_{t_{N-1},d})\rangle|t_1\rangle\cdots|t_N\rangle|0\cdots0\rangle. \tag{64}$$

After that, to prepare $\hat{X}_t$ with $t\in[1,\ldots,N]$, we make N calls to U_μ and U_σ each, which act as $U_\mu|t,X\rangle|0\rangle=|t,X\rangle|\mu(t,X)\rangle$ and $U_\sigma|t,X\rangle|0\rangle=|t,X\rangle|\sigma(t,X)\rangle$, respectively. We also require dN multiplications (multiplying the μ vectors with the scalars Δt at each time step) and d^2N multiplications as well as $(d-1)dN$ additions for the multiplications of the σ matrices with the respective ΔW_{t_n} vectors. Finally, we require $2N$ additions to carry out the update steps, as seen in Eq. (4). We then arrive at the following state:

$$|X_{t_0}\rangle|t_0\rangle|\sigma^\top\nabla u_0\rangle|u_0\rangle \sum_{k_{t_0,1}=1}^{N_{\text{Gauss}}}\cdots\sum_{k_{t_{N-1},d}=1}^{N_{\text{Gauss}}}\sqrt{p_{k_{t_0,1}}}\cdots\sqrt{p_{k_{t_{N-1},d}}}|k_{t_0,1}\rangle|\Delta W_{t_0,1}(k_{t_0,1})\rangle\times\cdots\times|k_{t_{N-1},d}\rangle|\Delta W_{t_{N-1},1}(k_{t_{N-1},d})\rangle$$
$$\times|\hat{X}_{t_1}(k_{t_0,1},\ldots,k_{t_0,d})\rangle\cdots|\hat{X}_{t_N}(k_{t_0,1},\ldots,k_{t_{N-1},d})\rangle|t_1\rangle\cdots|t_N\rangle|0\cdots0\rangle. \tag{65}$$

Next, we make $N-1$ calls to the neural network unitaries U_{NN} to compute the quantities $\sigma^\top\nabla u_t$, in the sense that $U_{\text{NN}}|\hat{X}_{t_n}\rangle|0\rangle=|\hat{X}_{t_n}\rangle|\sigma^\top\nabla u_t(\hat{X}_{t_n})\rangle$. The state thus becomes the following, where $|\sigma^\top\nabla u_t\rangle$ is only present for $t\in[1,N-1]$,

$$|u_0\rangle|\sigma^\top\nabla u_0\rangle|t_0\rangle|X_{t_0}\rangle \sum_{k_{t_0,1}=1}^{N_{\text{Gauss}}}\cdots\sum_{k_{t_{N-1},d}=1}^{N_{\text{Gauss}}}\sqrt{p_{k_{t_0,1}}}\cdots\sqrt{p_{k_{t_{N-1},d}}}|k_{t_0,1}\rangle|\Delta W_{t_0,1}(k_{t_0,1})\rangle\times\cdots\times|k_{t_{N-1},d}\rangle|\Delta W_{t_{N-1},1}(k_{t_{N-1},d})\rangle$$
$$\times|\hat{X}_{t_1}(k_{t_0,1},\ldots,k_{1,d})\rangle\cdots|\hat{X}_{t_N}(k_{t_0,1},\ldots,k_{t_{N-1},d})\rangle|t_1\rangle\cdots|t_N\rangle|\sigma^\top\nabla u_t(\hat{X}_{t_1})\rangle\cdots|\sigma^\top\nabla u_t(\hat{X}_{t_{N-1}})\rangle|0\cdots0\rangle. \tag{66}$$

It is worth pointing out that, in the above equation, we omitted the pointer arguments k for the states of the form $|\sigma^\top\nabla u_t(\hat{X}_{t_n})\rangle$ to allow for a more compact notation.

Next, we proceed to compute $\hat{u}_t$ for $t\in[1,\ldots,N]$. This computation requires N calls to U_f, N additions and multiplications each for multiplying the outcome of U_f with Δt and adding it to the previous value of $\hat{u}_t$, and dN multiplications as well as dN additions for multiplying the vectors $\sigma^\top\nabla u_t(\hat{X}_{t_n})$ and adding the result to the previous value of $\hat{u}_t$. We then have the following quantum state:

$$|u_0\rangle|\sigma^\top\nabla u_0\rangle|t_0\rangle|X_{t_0}\rangle \sum_{k_{t_0,1}=1}^{N_{\text{Gauss}}}\cdots\sum_{k_{t_{N-1},d}=1}^{N_{\text{Gauss}}}\sqrt{p_{k_{t_0,1}}}\cdots\sqrt{p_{k_{t_{N-1},d}}}|k_{t_0,1}\rangle|\Delta W_{t_0,1}(k_{t_0,1})\rangle\times\cdots\times|k_{t_{N-1},d}\rangle|\Delta W_{t_{N-1},1}(k_{t_{N-1},d})\rangle$$
$$\times|\hat{X}_{t_1}(k_{t_0,1},\ldots,k_{1,d})\rangle\cdots|\hat{X}_{t_N}(k_{t_0,1},\ldots,k_{t_{N-1},d})\rangle|t_1\rangle\cdots|t_N\rangle|\sigma^\top\nabla u_t(\hat{X}_{t_1})\rangle\cdots|\sigma^\top\nabla u_t(\hat{X}_{t_{N-1}})\rangle|\hat{u}_{t_1}\rangle\cdots|\hat{u}_{t_N}\rangle|0\cdots0\rangle. \tag{67}$$

Again, we omitted the pointer arguments k for the states $|\hat{u}_{t_n}\rangle$ to simplify the notation. The pointer arguments for $|\hat{u}_{t_n}\rangle$ would be the same as for $|\hat{X}_{t_n}\rangle$, i.e., $(k_{t_0,1},\ldots,k_{t_{n-1},d})$.

Finally, we require one call to the unitary U_{loss} to compute the loss function. The quantum state then becomes

$$|u_0\rangle|\sigma^\top\nabla u_0\rangle|t_0\rangle|X_{t_0}\rangle \sum_{k_{t_0,1}=1}^{N_{\text{Gauss}}}\cdots\sum_{k_{t_{N-1},d}=1}^{N_{\text{Gauss}}}\sqrt{p_{k_{t_0,1}}}\cdots\sqrt{p_{k_{t_{N-1},d}}}|k_{t_0,1}\rangle|\Delta W_{t_0,1}(k_{t_0,1})\rangle\times\cdots\times|k_{t_{N-1},d}\rangle|\Delta W_{t_{N-1},1}(k_{t_{N-1},d})\rangle$$
$$\times|\hat{X}_{t_1}(k_{t_0,1},\ldots,k_{1,d})\rangle\cdots|\hat{X}_{t_N}(k_{t_0,1},\ldots,k_{t_{N-1},d})\rangle|t_1\rangle\cdots|t_N\rangle$$
$$\times|\sigma^\top\nabla u_t(\hat{X}_{t_1})\rangle\cdots|\sigma^\top\nabla u_t(\hat{X}_{t_{N-1}})\rangle|\hat{u}_{t_1}\rangle\cdots|\hat{u}_{t_N}\rangle|f_p(\{\hat{X}_{t_n}\}_{0\leqslant n\leqslant N},\{\Delta W_{t_n}\}_{0\leqslant n\leqslant N})\rangle|0\cdots0\rangle. \tag{68}$$

Now that we have $|f_{\text{p}}(\hat{X}_{t_N},\hat{u}_{t_N})\rangle$ in the superposition, weighted by the distribution of the Gaussian increments, we can apply QAMC from Appendix B 4 a to estimate the mean of $f_{\text{p}}(\hat{X}_{t_N},\hat{u}_{t_N})$ with respect to the distributions given by the Gaussian increments ΔW_{t_n}. We proceed to summarize the query complexities for the individual unitaries.

Query complexities. Based on the above, estimating the payoff function from Eq. (9) using QAMC, thus requires the asymptotic number of queries to the following unitaries and arithmetic operations:

(1) $\tilde{O}(\lambda_{\max}/\epsilon)$ queries to $U_{X_{t_0}}$, U_{u_0}, $U_{\sigma^\top\nabla u_0}$, U_{t_0}, and U_{loss} each;
(2) $\tilde{O}(N\lambda_{\max}/\epsilon)$ queries to U_μ, U_σ, U_f, and U_{NN} each;
(3) $\tilde{O}(dN\lambda_{\max}/\epsilon)$ queries to U_{Gauss};

(4) $\tilde{O}(d^2 N \lambda_{\max}/\epsilon)$ arithmetic operations.

How does this compare to the classical case? We do not get the speedup from QAMC in the classical case. Therefore, we have

(1) $O(\lambda_{\max}^2/\epsilon^2)$ queries to $U_{X_{t_0}}$, U_{u_0}, $U_{\sigma^\top \nabla u_0}$, U_{t_0}, and U_{loss} each,

(2) $O(N\lambda_{\max}^2/\epsilon^2)$ queries to U_μ, U_σ, U_f, and U_{NN} each,

(3) $O(dN\lambda_{\max}^2/\epsilon^2)$ queries to U_{Gauss},

(4) $O(d^2 N\lambda_{\max}^2/\epsilon^2)$ arithmetic operations,

from which we can see the slowdown in $\lambda_{\max}$ and ϵ compared to the quantum enhanced version. We conclude the analysis with the following result:

Result 4. Quantum-accelerated Monte Carlo evaluation. Using the QAMC method from Ref. [99] and described in Appendix B 4 a, there is the potential for achieving a quantum speedup from a query complexity point of view for estimating the loss function in the algorithm for solving nonlinear PDEs described in Ref. [25] with the number of calls to unitaries as well as arithmetic operations as outlined above.

Subsequently, we discuss the problem of training the NNs, referencing the findings from Sec. III.

B. Training the neural networks

As described in Sec. III, using the forward gradient method allows us to estimate the gradients of the parameters in the NNs using QAMC. Since the payoff function f_p [see Eq. (9)] is assumed to be Lipschitz continuous (see Assumption 3) and has a scalar output we may apply the methods surveyed in Table I, where the gradient is taken with respect to the trainable parameters in the architecture from Ref. [25], i.e., the weights in the NNs and the initial values of u and $\sigma^\top \nabla u$. To numerically verify that the forward gradient method as well as numerical differentiation work in the deep-learning architecture from Ref. [25], we implement these methods in the purely classical setting to update the weights when solving the Hamilton-Jacobi-Bellman (HJB) PDE. The HJB PDE is a special case of Eq. (1) where $\sigma(t,X) = 2\mathbb{I}$, $\mu(t,X) = 0$, and $f(t,X,u(t,X),\sigma^\top(t,X)\nabla u(t,X)) = \|\nabla u(t,X)\|_2^2$, and, to evaluate the loss function, $g(X) = \ln[(1+\|X\|_2^2)/2]$. We take $N = 20$ temporal discretization steps, where we approximate the spatial gradient at each step with a NN with 225 trainable parameters, and the terminal time is $T = 1$. We present the values of the loss function at each training iteration in Fig. 9. As we can see in Fig. 9, and as experimentally shown for other problems in Ref. [79], the forward gradient method is indeed competitive with backpropagation, although the loss function in the latter case appears to converge slightly below the value in the forward gradient case, possibly due to the additional stochasticity introduced by the forward gradient, as mentioned in Ref. [79]. The same observation holds for numerical differentiation, where the slightly higher loss at convergence may stem from the truncation error.

C. Multilevel Monte Carlo methods and partial differential equations

We have so far discussed estimating the loss function in the deep-learning architecture from Ref. [25]. We now discuss the estimation of the solution to the PDE itself, once the loss function has been reduced to a satisfactory level. Once the loss function is below a certain threshold or has ceased to improve, the circuits outlined above may simply be truncated after the nth time step and used to estimate u_{t_n}. Since the NNs as well as μ and σ are Lipschitz continuous, one may estimate u_{t_n} with the given assumptions (Assumptions 1 to 3). As u_{t_n} are scalars, one may again make use of QAMC to estimate u_{t_n} with a speedup in ϵ compared to the classical case.

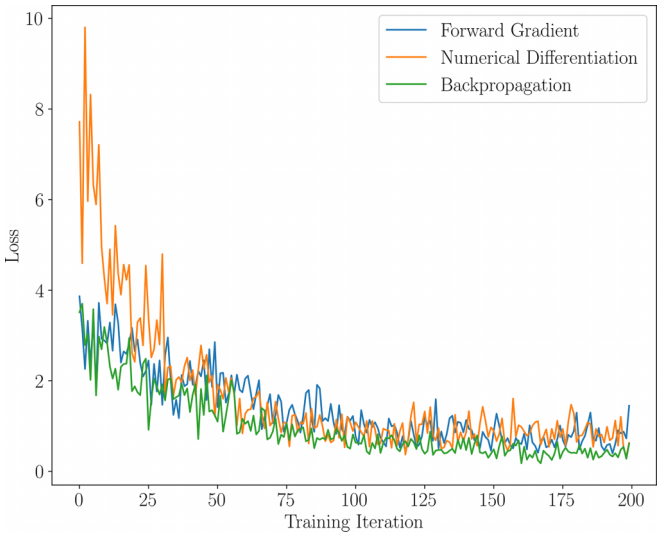

FIG. 9. Loss from Eq. (8) for the HJB PDE against the number of training iterations when the forward gradient method, numerical differentiation and backpropagation are employed to update the weights. In each case, we have $N = 20$ temporal discretization steps, each of the 20 neural networks has 225 trainable parameters, and the terminal time is $T = 1$. We take 20 training samples per iteration and the learning rate is 0.01. In the case when the forward gradient method is employed, we take 100 samples of v per iteration. For numerical differentiation, h from Eq. (23) is 0.001.

When considering the precision with which we estimate the solution at final time, u_{t_N}, we now also have to consider the error resulting from the discretization of the SDEs. It is worth mentioning that the error stemming from the inaccuracies of the NNs when representing the spatial gradients are also an error source contributing to the total error when estimating the solution. However, the highly nonconvex training landscapes of the NNs prevent us from bounding this contribution to the error. We proceed to analyze the contribution from the discretization of the SDEs, neglecting the contribution from the inaccuracies of the NNs.

As in the case where we estimate the loss function, the error originating from the discretization of the Gaussian increments, as well as the estimation error when using MC methods, are relevant. Let the ideal estimation of the solution be I_0 and the estimate of the solution considering only the discretization error of the SDE be I_1. Furthermore, analogously to the discussion in Sec. IV A, let I_2 be the estimate of the solution u_{t_N} when also considering the error from the discretization of the Gaussian increments and let I_3 be the estimate where all three kinds of errors are taken into account.

Then we have

$$|I_0 - I_3| = |I_0 - I_1 + I_1 - I_2 + I_2 - I_3| \leqslant |I_0 - I_1| + |I_1 - I_2| + |I_2 - I_3|. \quad (69)$$

In order to estimate the solution u_{t_N} up to error ϵ we need to upper bound the term on the left hand side in Eq. (69), which we can achieve by upper bounding the right hand side of the same equation. In Sec. IV A we showed that upper bounding the term $|I_1 - I_2|$ gives us a requirement on how many bits or qubits are needed to represent the Gaussian increments. Furthermore, upper bounding $|I_2 - I_3|$ gave us a requirement on how many samples we need to take with classical MC or QAMC. We thus still need to upper bound the first term, $|I_0 - I_1|$:

$$\begin{aligned} |I_0 - I_1| &= \left|\mathbb{E}\left[u_{t_N}\left(\{X_{t_n}\}_{0\leqslant n\leqslant N}\right)\right] - \mathbb{E}\left[u_{t_N}\left(\{\hat{X}_{t_n}\}_{0\leqslant n\leqslant N}\right)\right]\right| \\ &\leqslant \mathbb{E}\left[\left|u_{t_N}\left(\{X_{t_n}\}_{0\leqslant n\leqslant N}\right) - u_{t_N}\left(\{\hat{X}_{t_n}\}_{0\leqslant n\leqslant N}\right)\right|\right] \\ &\leqslant L_{u_{t_N}}\mathbb{E}\left[\sup_{0\leqslant n\leqslant N}\left\|X_{t_n} - \hat{X}_{t_n}\right\|_2\right] = O((\Delta t)^r), \end{aligned} \quad (70)$$

where the first equality is the definition of I_0 and I_1 (we dropped the Gaussian increments in the arguments to simplify the notation), the first inequality follows from applying Jensen's inequality (recall that the absolute value is a convex function). The second inequality follows from Assumption 3, since the local Lipschitz continuity of f_p implies local Lipschitz continuity of u_{t_N} (we name its Lipschitz constant $L_{u_{t_N}}$). The last equation follows from the definition of the strong order r, see Eq. (B42). In order to bound $|I_0 - I_1|$ by $\epsilon/3$, we need to choose $\Delta t = O(\epsilon^{1/r})$ and, consequently,

$$N = O(\epsilon^{-1/r}). \quad (71)$$

It is worth pointing out that this relation between N and ϵ holds as much for the quantum case as for the classical case.

By inserting the dependency of N on ϵ from Eq. (71) into the query complexities from Sec. IV A, we arrive at the following complexities:

(1) $\tilde{O}(\lambda_{\max}/\epsilon)$ queries to $U_{X_{t_0}}$, U_{u_0}, $U_{\sigma^\top \nabla u_0}$, U_{t_0}, and U_{loss} each;
(2) $\tilde{O}(\lambda_{\max}/\epsilon^{1+1/r})$ queries to U_μ, U_σ, U_f, and U_{NN} each;
(3) $\tilde{O}(d\lambda_{\max}/\epsilon^{1+1/r})$ queries to U_{Gauss};
(4) $\tilde{O}(d^2\lambda_{\max}/\epsilon^{1+1/r})$ arithmetic operations,

and by inserting $r = 1/2$ for the Euler-Maruyama scheme which the architecture from Ref. [25] uses, we arrive at

(1) $\tilde{O}(\lambda_{\max}/\epsilon)$ queries to $U_{X_{t_0}}$, U_{u_0}, $U_{\sigma^\top \nabla u_0}$, U_{t_0}, and U_{loss} each;
(2) $\tilde{O}(\lambda_{\max}/\epsilon^3)$ queries to U_μ, U_σ, U_f, and U_{NN} each;
(3) $\tilde{O}(d\lambda_{\max}/\epsilon^3)$ queries to U_{Gauss};
(4) $\tilde{O}(d^2\lambda_{\max}/\epsilon^3)$ arithmetic operations.

How does this compare to the classical case? The relation between N and ϵ from Eq. (71) holds in the classical case as well. However, we do not get the speedup from QAMC in the classical case. Therefore, we have

(1) $O(\lambda_{\max}^2/\epsilon^2)$ queries to $U_{X_{t_0}}$, U_{u_0}, $U_{\sigma^\top \nabla u_0}$, U_{t_0}, and U_{loss} each;
(2) $O(N\lambda_{\max}^2/\epsilon^2)$ queries to U_μ, U_σ, U_f, and U_{NN} each;
(3) $O(dN\lambda_{\max}^2/\epsilon^2)$ queries to U_{Gauss};
(4) $O(d^2N\lambda_{\max}^2/\epsilon^2)$ arithmetic operations,

which, after inserting the expressions for N and r leaves us with

(1) $O(\lambda_{\max}^2/\epsilon^2)$ queries to $U_{X_{t_0}}$, U_{u_0}, $U_{\sigma^\top \nabla u_0}$, U_{t_0}, and U_{loss} each;
(2) $O(\lambda_{\max}^2/\epsilon^4)$ queries to U_μ, U_σ, U_f, and U_{NN} each;
(3) $O(d\lambda_{\max}^2/\epsilon^4)$ queries to U_{Gauss};
(4) $O(d^2\lambda_{\max}^2/\epsilon^4)$ arithmetic operations.

As in the case of estimating the loss function (see Sec. IV A), QAMC thus offers the potential for a quantum speedup in ϵ when estimating the solution u_{t_N} of the PDE.

As described in Appendix B 4 c, multilevel MC (MLMC) methods offer the potential to improve the sample complexity when estimating the mean value of functions depending on the discretized solutions of SDEs. We thus discuss the application of MLMC as well as quantum-accelerated MLMC (QAMLMC) to the problem of estimating the solution u_{t_N} in the deep-learning architecture from Ref. [25].

We begin by checking that Assumptions 4 to 6 are satisfied in the setting of this section so far. Assumption 4 is satisfied since we make Assumption 1. The numerical scheme employed in Ref. [25] is the Euler-Maruyama scheme which is of the form Eq. (B47) and satisfies Assumption 5 [96]. Finally, Assumption 6 is satisfied since we make Assumption 3, the latter being a stronger assumption. Thus, the assumptions made in Ref. [33] are satisfied and we can in principle apply Lemmas 8 and 9.

We continue with considerations on how one might use QAMLMC (or classical MLMC) in combination with the architecture from Ref. [25]. It is worth pointing out that estimating each of the terms Y_k from Eq. (B44) which approximate the solution at final time u_{t_N} with increasing precision, having N_k discretization steps in the architecture from Ref. [25], will also have N_k NNs to train. In total, with K estimators Y_k with 2^k NNs each, where $K = O(\log(2\epsilon^{-1}))$, we would have $O(2^{\log(2\epsilon^{-1})}) = O(\epsilon^{-1})$ NNs to train. Since in each estimator there are different NNs (i.e., we do not assume that any weight sharing across different estimators takes place, although that might be possible), MLMC or QAMLMC may only offer a speedup to the estimation of the solution u_{t_N} and would not be of help in estimating the gradients for each estimator. The gradients would need to be computed as discussed in Sec. III C.

When using classical or quantum-accelerated MLMC methods, observe that the cutoff strong order for which the $1/r$ term vanishes in the exponent of ϵ is not the same for the classical and the quantum case, as seen in Lemmas 8 and 9. In the classical case, it is $r = 1/2$ and in the quantum case it is $r = 1$. This difference stems from the different base case MC sample complexities, $O(\lambda^2/\epsilon^2)$ in the classical case and $\tilde{O}(\lambda/\epsilon)$ in the quantum case, where λ^2 is the variance of the term in question [33]. Recall that in the architecture from Ref. [25] we have $r = 1/2$ as the Euler-Maruyama scheme is used. We continue to analyze potential speedups when using classical and quantum-accelerated MLMC methods to estimate the solution u_{t_N} from the deep-learning architecture.

Employing MLMC methods in the deep-learning architecture in the classical case to estimate the solution u_{t_N}, with $r = 1/2$ we have as per Lemma 8,

(1) $\tilde{O}(\epsilon^{-2})$ queries to $U_{X_{t_0}}, U_{u_0}, U_{\sigma^\top \nabla u_0}, U_{t_0}, U_{\text{loss}}, U_\mu, U_\sigma, U_f$, and U_{NN} each;
(2) $\tilde{O}(d\epsilon^{-2})$ queries to U_{Gauss};
(3) $\tilde{O}(d^2\epsilon^{-2})$ arithmetic operations,

where d is the dimension of the stochastic process X_t and where we treated the individual NNs as equivalent from a query complexity point of view. In the case of QAMLMC we have in the same setting:

(1) $\tilde{O}(\epsilon^{-2})$ queries to $U_{X_{t_0}}, U_{u_0}, U_{\sigma^\top \nabla u_0}, U_{t_0}, U_{\text{loss}}, U_\mu, U_\sigma, U_f$, and U_{NN} each;
(2) $\tilde{O}(d\epsilon^{-2})$ queries to U_{Gauss};
(3) $\tilde{O}(d^2\epsilon^{-2})$ arithmetic operations,

which is the same as for the classical case (the lower threshold in the strong order r for the classical case compensates for the speedup in ϵ in the quantum case). However, both methods improve on the complexities outlined above in this section. It is worth highlighting again, that MLMC methods would only speed up the estimation of the solution but not the estimation of the gradients with respect to the parameters of the NNs. Thus, the results of Sec. III in Table I are still relevant for training the NNs in the sense that they offer the potential for speedups in ϵ when estimating the gradient, even in the scenario where MLMC methods are applied to estimating the solution.

Improvements in the query complexities for QAMLMC (which would allow for QAMLMC to outperform classical MLMC) may be brought about by implementing a numerical scheme with $r > 1/2$, e.g., the Milstein scheme [100] with $r = 1$. However, this step comes with its own difficulties, particularly for the architecture from Ref. [25]. For implementing higher order schemes such as the Milstein scheme, derivatives of σ and $\sigma^\top \nabla u$ come into play. A possibility to compute what amounts to the Hessian of u might be to add a second NN per discretization step. However, we leave this investigation to future research.

Summarizing our findings from this section, we saw that by applying QAMC to the deep-learning architecture for solving nonlinear PDEs from Ref. [25] there exists the potential for a speedup in the error tolerance ϵ when estimating the loss function. In Sec. IV C we found that applying classical MLMC methods offers the potential for an even greater speedup in ϵ compared to applying QAMC when estimating the solution of the PDE at final time. Interestingly, with the numerical scheme employed in the deep-learning architecture, QAMLMC does not offer a speedup (nor suffers from a slowdown) compared to (classical) MLMC. Future research may, however, be able to find the possibility for a quantum speedup by increasing the strong order r of the numerical scheme for approximating the SDE. This might allow QAMLMC to provide a speedup over classical MLMC again. As mentioned in Sec. IV C, increasing the strong order r in the deep-learning architecture comes with its own set of challenges, which we leave for future research to address. It is worth pointing out that applying MLMC methods only offers the potential of speeding up the estimation of the loss function, but not the estimation of the gradient, thus our findings from Sec. III on estimating the gradient with QAMC methods is still relevant.

V. QUANTUM ALGORITHM FOR ACCELERATING NEURAL NETWORK TRAINING

In this section we discuss the application of a quantum algorithm from the literature (see [101]) which offers the potential for accelerating the training and evaluating of classical NNs. This algorithm constitutes a fault-tolerant quantum algorithm that we aim to incorporate into the architecture from Ref. [25] for solving nonlinear partial differential equations (PDEs). After having reviewed the algorithm, we discuss how it can be applied in the setting from Ref. [25], and what advantages and disadvantages are associated with doing so.

It is worth pointing out that while in previous sections we mainly discussed the query complexity to the unitaries implementing the NNs, this section (and the algorithm we introduce therein) are not about the query complexity to the unitaries, but rather the runtime of the NN, i.e., the runtime of the unitaries. A key component of this algorithm is the robust inner product estimation (RIPE) quantum subroutine, introduced in Appendix B 5. At the heart of many machine learning architectures lie NNs that are trained in a supervised setting, meaning that the NN is trained with labeled examples. Applications of supervised NNs are found in manufacturing [102], drug discovery [103], and solving differential equations, as in the algorithm from Ref. [25], which we outlined in Sec. I B.

When classically training feedforward NNs, the total runtime of the training algorithm is $O(T_{\text{iter}}ME)$, where T_{iter} is the number of training steps, M is the number of data points trained on per training step, and E is the number of edges in the network. This linear dependence on the number of edges renders the training of large fully connected feedforward NNs expensive. To alleviate this issue, the authors in Ref. [101] present a quantum-enhanced algorithm for training and evaluating feedforward NNs, where the linear dependence on the number of edges E may be exchanged for a linear dependence in the number of neurons n_{nodes}, at the cost of the dependence on $T_{\text{iter}}M$ becoming $(T_{\text{iter}}M)^{3/2}$, i.e., $O((T_{\text{iter}}M)^{3/2}n_{\text{nodes}})$. Recall the NN training via backpropagation outlined in Appendix B 1. In particular notice that we can reconsider Eqs. (B1) and (B2) by emphasizing the inner product in each of the two equations,

$$z_l^{(j)} = \left(W_l^{(j)} a_{l-1}\right) + b_l^{(j)}, \tag{72}$$

and

$$\delta_l^{(j)} = f'_{\text{nl}}\left(z_l^{(j)}\right)\left[\left(W_{l+1}^\top\right)^{(j)}\delta_{l+1}\right]. \tag{73}$$

The authors in Ref. [101] point out that when training and evaluating NNs using Eqs. (B1) and (B2) in the classical setting, the runtime is $O(T_{\text{iter}}ME)$. This complexity with its linear dependence on E arises for each training iteration and for each data point for the following reason: One has to, in each layer, evaluate one activation function per neuron, as well as compute the inner product from Eq. (72), which depends on the previous layer. Thus, we have a time complexity of $O(\sum_{l=2}^{L} n_l n_{l-1} = E)$ per data point per training iteration.

The authors then propose to use the RIPE algorithm (see Lemma 11) to estimate the inner products in Eqs. (72) and (73). In addition to achieving potential speedups, the authors

argue that introducing noise into the inner product estimation may help regularise NNs, i.e., prevent them from overfitting. Overfitting refers to the phenomenon whereby NNs fail to identify patterns underlying the data and instead memorise the data on which they are trained and consequently fail to generalize well to new data. Next, the authors point out that merely initializing and storing the weight matrices W_l takes time at least $O(E)$. To circumvent this issue, the authors in Ref. [101] come up with two solutions. Firstly, they use low-rank initialization for the weight matrices. Observe that, due to Eq. (B3) and setting $\eta_{0,1} = -1$, we can write the weights as

$$W_{t,l}^{(j,k)} = \sum_{\tau=0}^{t-1} \sum_{m=1}^{M} \frac{-\eta_{\tau,l}}{M} \delta_{\tau,m,l}^{(j)} a_{\tau,m,l-1}^{(k)}. \tag{74}$$

With low-rank initialization, only a fraction of the M summands $a_{0,m,l}^{(k)} \delta_{0,m,l}^{(j)}$ are set to be initially nonzero. They justify this by pointing out that other (classical) algorithms for training NNs have made use of this approach as well to speed up the training of NNs [104–106]. Thus, the authors avoid writing out $O(E)$ weight values for the initial weights. Secondly, they make use of an implicit weight storage scheme using quantum random-access memory (QRAM), see Definition 6. Recall that QRAM is a quantum mechanical analog of classical RAM, allowing for classical data to be queried in superposition. By storing matrices $X_{t,l,j} \in \mathbb{R}^{t \times M}$ with elements $X_{t,l,j}^{(\tau,m)} = \frac{-\eta_{\tau,l}}{M} \delta_{\tau,m,l}^{(j)} \|a_{\tau,m,l-1}\|_2$ in an l_2 binary search tree in QRAM, the states $|W_{t,l}^{(j)}\rangle$ can be computed efficiently on the fly. An l_2 binary search tree is a tree (data structure) where the nodes have two children, unless they are leaves, and where the root stores $\|x\|_2^2$ and the leaves store the components $(x^{(k)})^2$ along with the sign of $x^{(k)}$. The authors of Ref. [107] showed that storing a classical vector $x \in \mathbb{R}^K$ in an l_2 binary search tree in QRAM takes time $\tilde{O}(K)$ and retrieving it polylogarithmic time in K. For a given training iteration t, there are $n_{\text{nodes}} - n_1$ such matrices stored in QRAM. The key takeaway is that in order to avoid the cost of $O(E)$, the authors in Ref. [101] store the matrices $X_{t,l,j}$ implicitly in QRAM, from which they can compute the weights $W_l^{(j)}$ efficiently on the fly. For more details on the implicit storage of the weight matrices, we refer to Ref. [101]. The cost of performing the forward propagation in an NN then becomes $O(n_{\text{nodes}} T_{\text{RIPE}})$, where T_{RIPE} is the mean time to carry out the RIPE algorithm from Lemma 11, which is applied to estimate the inner products in Eqs. (72) and (73). Using the expression from Eq. (B59), the overall running time for the forward propagation is

$$O\left(\sqrt{T_{\text{iter}} M} n_{\text{nodes}} \frac{\log 1/\gamma}{\epsilon} R_{t,m,a}\right), \tag{75}$$

where γ and ϵ are as in Lemma 11, $R_{t,m,a}$ depends on the matrices $X_{t,l,j}$ and $W_{t,l}^{(j)}$ as well as the terms $a_{t,m,l-1}$. The authors provide experimental evidence, that $R^{t,m,a}$ does, however, in practice not impact the running time significantly. Similarly, by again using low-rank initialization and the implicit weight storage scheme, the authors show that one backpropagation pass can be carried out in time:

$$O\left(\sqrt{T_{\text{iter}} M} N \frac{\log 1/\gamma}{\epsilon} R_{t,m,\delta}\right), \tag{76}$$

where $R_{t,m,\delta}$ depends on the matrices $X_{t,l,j}$ and $W_{t,l}^{(j)}$ and on the terms $\delta_{t,m,l-1}$. Again, the authors provide evidence indicating that in practical settings $R_{t,m,\delta}$ does not significantly impact the runtime. When training the NN for T_{iter} iterations with M data points each, the runtime gets multiplied by a factor of $T_{\text{iter}} M$.

It is worth pointing out again, that the authors of Ref. [101] managed to overcome the dependence on E in the runtime of training and evaluating NNs, which comes at the cost of $\sqrt{M} n_{\text{nodes}}$ showing up in the runtime. Replacing E by n_{nodes} in the runtime approximately amounts to a quadratic speedup (in the case of a fully connected feedforward NN), since each neuron, in that scenario, is connected with every neuron in the preceding and succeeding layer, see Fact 1. Furthermore, in practice, for large NNs, $n_{\text{nodes}} \gg \sqrt{T_{\text{iter}} M}$ [101]. We summarize the key findings from Ref. [101] (Algorithm 3 therein) in Lemma 1.

Lemma 1. RIPE-accelerated NN training and evaluation [101]. There exist quantum algorithms for training and evaluating a feedforward NN using the RIPE algorithm (see Lemma 11). The time complexity for the training procedure is $O((T_{\text{iter}} M)^{3/2} n_{\text{nodes}} \frac{\log 1/\gamma}{\epsilon} R_b)$ and $O(\sqrt{T_{\text{iter}} M} n_{\text{nodes}} \frac{\log 1/\gamma}{\epsilon} R_f)$ for the evaluation of the NN. Here, T_{iter} is the number of training iterations, M is the number of data points trained on per iteration, n_{nodes} is the number of neurons in the NN, γ and ϵ are from the RIPE algorithm (see Lemma 11), and R_f and R_b are factors depending on the NN and the training samples. In practice, the last two parameters are expected to not significantly impact the runtime.

Application to deep-learning approach

We proceed to discuss how the results from Ref. [101] that we introduced above may be applied to the architecture from Ref. [25], and discuss the implications. To begin with, it is worth pointing out that the NNs in the deep-learning architecture are in fact trained in a supervised manner. For the sampled Brownian paths $\{X_{t_n}\}_n$, which constitute the training data, the labels are given by $g(X_{t_N})$, as outlined in Sec. I B. Thus, when starting from the classical algorithm outlined in Ref. [25], one may train and evaluate the NNs using the results from Ref. [101], with the caveat that QRAM is required, unlike in the algorithms discussed so far. As outlined above, this approximately results in a quadratic speedup in d for the training and evaluation of the NNs, where d is the dimension of the stochastic process X_t from Ref. [25], (see Fact 1), whereas the number of neurons per NN is of the order of d.

A question that naturally emerges at this point, is whether the RIPE-accelerated training and evaluation may be combined with other methods, with which we have aimed to enhance the deep-learning architecture. In Sec. IV C we outlined how the estimation of the loss function in the deep-learning may be accelerated using classical multilevel Monte Carlo (MLMC) methods. Since employing the classical MLMC method does not change the nature of any of the NNs involved in the deep-learning architecture from Ref. [25], the application of the RIPE-accelerated methods appears straightforward. However, the combination of the RIPE-accelerated method and quantum-accelerated Monte Carlo (QAMC) methods does not. In the RIPE-accelerated

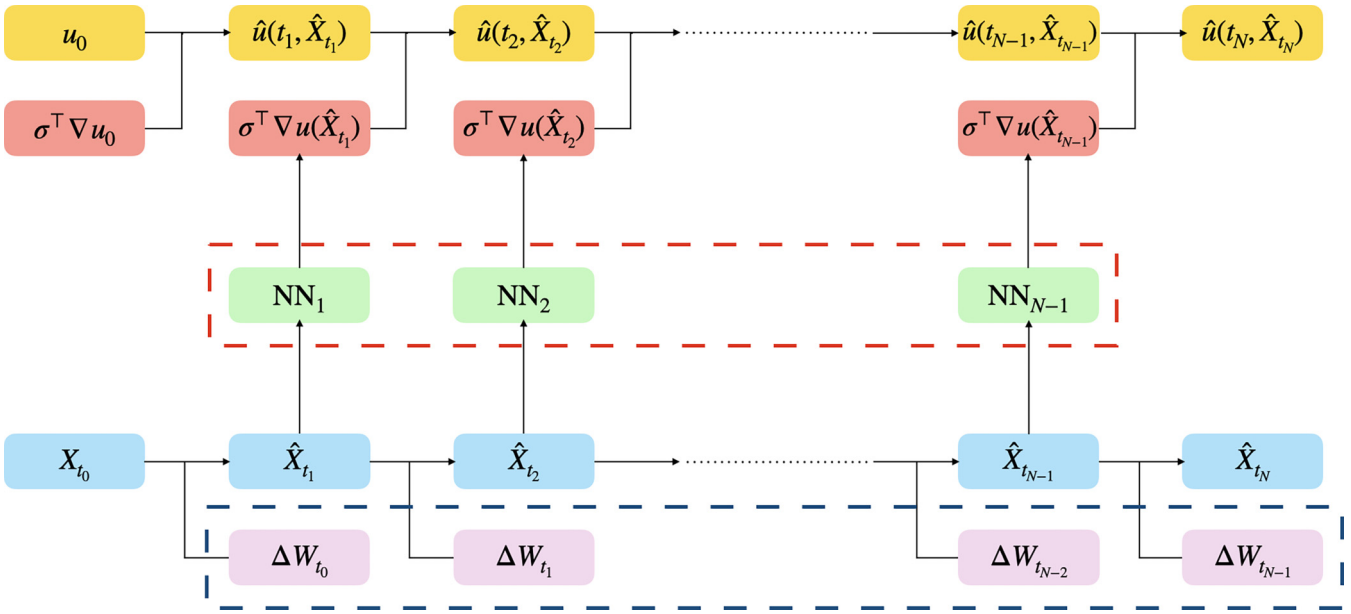

FIG. 10. Flowchart representing the classical algorithm from Ref. [25] as shown in Fig. 1. The parts of the algorithm that we targeted for quantum speedups are inside dashed boxes. We target the neural networks (red dashed box) by making use of variational quantum circuits (with a near-term setting in mind) as well as the robust inner product estimation algorithm (with a far-term setting in mind). We target the sampling process (dark blue dashed box) by making use of quantum-accelerated Monte Carlo for evaluating the loss function and its gradient.

methods, the quantum subroutine that is employed is the RIPE subroutine (Lemma 11). The other operations, such as applying the activation functions, are not carried out on a quantum processor, as described in Ref. [25]. If we were to combine the RIPE-accelerated methods with QAMC, the whole deep-learning architecture, from preparing the stochastic process X_t to computing the payoff function f_p that we aim to estimate, has to be carried out in a quantum circuit, lest the superpositions get collapsed, on which QAMC and the underlying amplitude estimation subroutine crucially rely. Therefore, the implicit weight storage scheme, which is an essential part of the RIPE-accelerated methods, would have to be adapted. Recall that QRAM merely allows for the loading of classical data in superposition, but does not allow for the storage of states in superposition. Thus, the terms that are stored in QRAM in Ref. [101] would instead be stored in qubits in the quantum circuit, in superposition. This procedure would require some kind of random access memory in a quantum circuit, as outlined in, e.g., [108]. We point out that this procedure would most likely be very expensive regarding the number of qubits required. We leave the further exploration of combining RIPE-accelerated NN methods with QAMC to future research.

VI. DISCUSSION

In Fig. 10, the sections of the classical algorithm from Ref. [25] for which we investigated the potential for quantum speedups are highlighted in dashed boxes. The training and evaluation of the neural networks (NNs) inside the red dashed box we aimed to speed up by using parametrized quantum circuits (PQCs) and training them as in variational quantum algorithms, as described in Sec. II. While the variational method could allow for quantum advantages in special scenarios, in our simulations we found no evidence supporting the potential for a quantum speedup using this method. We also investigated the potential for a quantum speedup in the training and evaluation of the NNs by considering an algorithm for fault-tolerant quantum computers, see Sec. V. This algorithm (introduced in Ref. [101]) offers the potential for a quadratic speedup in the dimension d of the spatial variable of the partial differential equation (PDE) by making use of a robust inner product estimation (RIPE) subroutine. We targeted the sampling process (the section inside the dark blue dashed box in Fig. 10) by considering quantum-accelerated Monte Carlo (QAMC) [99] in Secs. III and IV. We showed that for the number of samples required to achieve a certain error tolerance in the loss function and its gradient (with respect to their trainable parameters), there is the potential for a quadratic speedup in the error tolerance, when using this subroutine on a fault-tolerant quantum computer, compared to the classical case. Furthermore, the forward gradient method offers the potential for saving memory, i.e., qubits. When targeting the sampling process, we also identified a classical algorithm, multilevel Monte Carlo (MLMC) for which we could show the potential for an even greater speedup in the error tolerance. The combination of MLMC with QAMC in the context of the deep-learning architecture may be investigated in the future.

For now, since QAMC and MLMC methods offer the potential for a speedup in the error tolerance ϵ when estimating the loss function and the gradient, and RIPE-accelerated methods for evaluating and training classical NNs offers a possible speedup in the input dimension d, we can outline two regimes. In the case where $d \gg 1/\epsilon$, the speedup in d when training the NNs with the RIPE-accelerated method would be more valuable, as may be the case in high-dimensional instances of the Black-Scholes or Hamilton-Jacobi-Bellman PDE. On the contrary, when $d \ll 1/\epsilon$, QAMC and MLMC offer a more

significant advantage, as in the Allen-Cahn equation. The task of combining RIPE-accelerated methods for NNs with QAMC we leave to future research.

ACKNOWLEDGMENTS

We acknowledge valuable discussions with Jia-Yang Gao. This research is supported by the National Research Foundation, Singapore, and A*STAR under its CQT Bridging Grant and its Quantum Engineering Programme under Grant No. NRF2021-QEP2-02-P05. L.M. also acknowledges funding from the Swiss-European Mobility Programme (SEMP). F.R. acknowledges financial support by the Swiss National Science Foundation (Ambizione Grant No. PZ00P2_186040).

APPENDIX A: QUANTUM COMPUTATIONAL MODEL

Before we look at specific quantum algorithms, we review our computational model and other assumptions and definitions. The computational model of quantum computing we work in is the standard quantum circuit model [109]. In a quantum circuit with n qubits, operations on quantum states are represented as unitary matrices (termed quantum gates) acting on at least one qubit. Since the operations are unitary, they are reversible. We illustrate a quantum circuit by a set of horizontal lines (wires) [109]. The unitary operations are represented as boxes on these wires, being executed from left to right. The quantum circuit model is useful for showing how a complicated operation can be constructed from relatively simple operations. At the end of the circuit a measurement, typically in the computational basis, takes place. As mentioned in Ref. [109], an arbitrary classical circuit can be simulated by an equivalent (reversible) quantum circuit. Indeed, the class of problems that can be solved in polynomial time with a probabilistic classical computer is a subset of the class of problems that can be solved in polynomial time with a quantum computer [110]. For a probabilistic classical circuit with runtime T, there exists a corresponding quantum circuit with runtime $O(T^{\log_2(3)})$ [111] (formulation from Ref. [112]).

In our work, we carry out arithmetic computations by employing a fixed point representation of real numbers. We make the assumption that there are enough qubits available to us, such that we can store numbers with enough precision, such that numerical errors become negligible. We use the fixed point encoding for real numbers as in Ref. [113] with the formulation from Ref. [114].

Definition 1. Fixed-point encoding of real numbers. Let c_1 and c_2 be positive integers and $a \in \{0,1\}^{c_1}$, $b \in \{0,1\}^{c_2}$ as well as $s \in \{0,1\}$, then we define a rational number as

$$Q(a,b,s) = (-1)^s (2^{c_1-1}a^{(c_1-1)} + \cdots + 2a^{(1)} + a^{(0)} + 2^{-1}b^{(0)} + \cdots + 2^{-c_2}b^{(c_2-1)}) \in [-R, R], \tag{A1}$$

where $R = 2^{c_1} - 2^{-c_2}$.

Using the fixed-point encoding of real numbers as in Definition 1, we define our arithmetic model.

Definition 2. Quantum arithmetic model. Given c_1, $c_2 \in \mathbb{N}$, we say that we use a quantum arithmetic computing model if the four arithmetic operations can be performed in constant time on a quantum computer.

More elaborate discussions on how one can perform arithmetic operations on a quantum computer using the fixed-point representation of real numbers can be found in Appendix C of Ref. [98], as well as in Ref. [113].

Controlled rotation operations are central to quantum computation and the cost of carrying out controlled rotations is dependent the number of bits needed to specify the rotation angle [115]. In our computational model we associate a controlled rotation with constant cost, with the formulation from Ref. [116].

Definition 3. Controlled rotation. We say we carry out a controlled rotation with an operation $\mathcal{R}$ if, with constant time and for all rational numbers $x \in [0,1]$ defined by a $(1+c_2)$-bit string as defined above in the fixed point arithmetic,

$$\mathcal{R}|x\rangle|0\rangle = |x\rangle(\sqrt{1-x}|0\rangle + \sqrt{x}|1\rangle). \tag{A2}$$

In order to access (classical) functions in a quantum circuit we make use of oracles and unitaries. If we know how to implement the function f under consideration, we speak of a unitary U_f. Typically, the term oracle is used for functions where we do not know how to implement the circuit, e.g., in Grover's search algorithm [13]. The unitary U_f for a classical function f is defined as follows [109]:

Definition 4. Access to function. We say that we have quantum access to a classical function $f : \{0,1\}^n \to \{0,1\}^m$, which can be represented via a classical circuit, via a unitary U_f, if we can perform the quantum operation,

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle, \quad \text{for } x \in \{0,1\}^n, \quad y \in \{0,1\}^m. \tag{A3}$$

in constant time, where $\oplus$ represents the bitwise exclusive OR (XOR) operation.

The oracle O_f acts in the same way, for the case when we do not know how to implement the function f. Importantly, U_f allows for access to function evaluations in superposition, i.e., in a linear combination of states. For complex normalized coefficients $\{c_i\}_{i=1}^N$ and $\{x_i\}_{i=1}^N$ a set of points, we have

$$U_f \sum_{i=1}^{N} c_i|x_i\rangle|0\rangle = \sum_{i=1}^{N} c_i|x_i\rangle|f(x_i)\rangle. \tag{A4}$$

Quantum distribution loading refers to the preparation of quantum states corresponding to probability distributions, i.e.,

$$|\psi\rangle = \sum_i \sqrt{p_i}|i\rangle, \tag{A5}$$

where $\{p_i\}_i$ constitutes a probability distribution. Grover and Rudolph have presented a method to load a discrete approximation of a log-concave probability distribution (e.g., a univariate Gaussian distribution) [117]. The inductive Grover-Rudolph method starts from a state of the form

$$|\psi_m\rangle = \sum_{i=0}^{2^m-1} \sqrt{p_{i,m}}|i\rangle \tag{A6}$$

and then proceeds to further divide the 2^m regions into 2^{m+1}. To do so, the following function is computed:

$$f_{\mathrm{GR}}(i) = \frac{\int_{x_{L,i}}^{(x_{R,i}-x_{L,i})/2} p(x)\,dx}{\int_{x_{L,i}}^{x_{R,i}} p(x)\,dx}, \tag{A7}$$

where $x_{L,i}$ and $x_{R,i}$ are the left and right boundaries of the region i. Next, $\theta_i = \arccos[\sqrt{f_{\mathrm{GR}}(i)}]$ is loaded into a quantum circuit such that the following controlled rotation can be carried out:

$$\sqrt{p_{i,m}}|i\rangle|\theta_i\rangle|0\rangle \mapsto \sqrt{p_{i,m}}|i\rangle|\theta_i\rangle[\cos(\theta_i)|0\rangle + \sin(\theta_i)|1\rangle], \tag{A8}$$

which, after uncomputing $|\theta_i\rangle$, leaves us in $|\psi_{m+1}\rangle$.

There are, however, issues with the Grover-Rudolph method, as pointed out in Refs. [98,118]. Having to compute the integrals in Eq. (A7) may undo any quantum speedup brought about by other quantum subroutines. Nevertheless, in Ref. [116], the authors point out that this argument only applies when one needs to sample from the distribution $p(x)$ in order to compute f_{GR}, which is not always the case. Another way to avoid the slowdown, in the case where the same distribution is to be loaded multiple times, is to compute the quantities θ_i up to the desired m once and store them in a quantum random access memory (see Definition 6), from which they can be loaded multiple times. In Ref. [119], the authors present a different method for distribution loading, namely the quantum generative adversarial networks (QGANs). The QGAN can first learn a distribution (from samples) and later, once trained, repeatedly load the distribution into the quantum circuit efficiently. We conclude this discussion with the following definition for quantum distribution loading.

Definition 5. Quantum distribution loading. We say we have access to distribution loading for the distribution $\{p_i\}_i$ if we have access to a unitary U_p, such that

$$U_p|0\cdots0\rangle = \sum_{i=0}^{N_p} \sqrt{p_i}|i\rangle. \tag{A9}$$

Apart from potentially alleviating the issue around quantum distribution loading, quantum random access memory (QRAM) is a key component of quantum algorithms for, e.g., machine learning [120], evaluating general NAND trees [121], and enforcing privacy in database searches [122]. We conclude this section with the introduction of QRAM.

Definition 6. Quantum random access memory [123]. We refer to a data structure that allows for the loading of classical data in superposition into a quantum circuit, as in

$$\sum_i c_i|i\rangle|0\cdots0\rangle \mapsto \sum_i c_i|i\rangle|D_i\rangle, \tag{A10}$$

where $|i\rangle$ represent the pointer states (in superposition) and $|D_i\rangle$ the data stored at index i, and c_i are normalized complex coefficients, as quantum random access memory.

APPENDIX B: CLASSICAL AND QUANTUM METHODS

In this Appendix we introduce a range of algorithms and subroutines, classical and quantum, which will be relevant later on in this work. We give an introduction to neural networks (NNs) in Appendix B 1 and automatic differentiation (AD) in Appendix B 2. We introduce variational quantum methods in Appendix B 3, which are proposed to be well suited for the noisy intermediate-scale quantum (NISQ) era, where circuit depths are kept short to mitigate the effects of noise. Thereafter, we introduce classical Monte Carlo (MC) methods and their quantum-accelerated counterparts in Appendix B 4. Among the fault-tolerant quantum algorithms we introduce, are quantum-accelerated MC (QAMC) methods in Appendix B 4 a and the robust inner product estimation (RIPE) algorithm in Appendix B 5.

1. Neural networks

Deep learning is a subfield of machine learning which involves leveraging large NNs and has a wide range of applications, such as web search [124,125], computer vision [126], and natural language processing [127]. The algorithm from Ref. [25], which serves as the starting point of our work, applies deep learning to the problem of solving nonlinear partial differential equations (PDEs). We here give a brief overview of feedforward NNs and their training and evaluation algorithms, following [101]. References for further reading include Refs. [87,128].

A feedforward NN consists of a collection of units, organized into L layers, each layer l having n_l neurons. The connections between the neurons of two adjacent layers, e.g., between layers $l-1$ and l, can be described by a weight matrix $W_l \in \mathbb{R}^{n_l \times n_{l-1}}$. Furthermore, each layer has its own bias vector $b_l \in \mathbb{R}^{n_l}$. Given a nonlinear function f_{nl} (termed the activation function), data are propagated through the NN, in what is called forward propagation, by computing $a_l^{(j)} = f_{\mathrm{nl},l}(z_l^{(j)})$ and

$$z_l^{(j)} = \sum_{k=1}^{n_l} W_l^{(j,k)} a_{l-1}^{(k)} + b_l^{(j)}. \tag{B1}$$

The goal of a feedforward NN is generally to make correct predictions from data that it has not been trained on. Note that the nonlinearity of f_{nl} is crucial, otherwise the whole NN would just amount to one linear transformation. Consider a data set of the form $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ where $x_i \in \mathbb{R}^{n_1}$ are data vectors and $y_i \in \mathbb{R}^{n_L}$ are the corresponding labels. We make use of such a data set to train the NN, meaning that its weights W_l and biases b_l are tuned such that the NN minimizes a cost function $C : \mathbb{R}^{n_L} \mapsto \mathbb{R}$. Ideally, the NN is trained long enough such that it makes correct predictions on unseen data points with high accuracy. The weights and biases of the NN are typically trained using a method called backpropagation, which works as follows. Given z_L and a_L at the end of the network, after having applied forward propagation to a data point from the training set, we introduce a vector δ_L with entries $\delta_L^{(j)} = f'_{\mathrm{nl}}(z_L^{(j)})\partial C/\partial a_L^{(j)}$ and proceed to compute the derivatives of the weights and biases backward through the network using the chain rule, at each step calculating and storing the vectors δ_l with entries

$$\delta_l^{(j)} = f'_{\mathrm{nl}}\big(z_l^{(j)}\big) \sum_{k=1}^{n_{l+1}} \big(W_{l+1}^{\top}\big)^{(j,k)} \delta_{l+1}^{(k)}. \tag{B2}$$

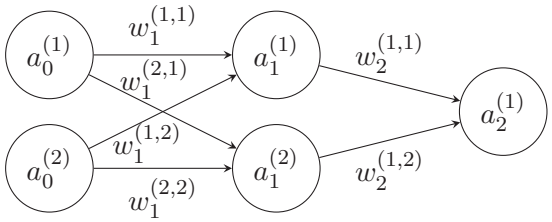

FIG. 11. Toy example of an NN, where the values in the neurons represent the pre-activation values, in the cases where an activation is applied, and the w values represent the weights associated with the respective edges. For simplicity, we omit biases.

The weights and biases are then typically updated via gradient descent,

$$W_{t+1,l}^{(j,k)} = W_{t,l}^{(j,k)} - \eta_{t,l}\frac{1}{M}\sum_{m=1}^{M} a_{t,m,l-1}^{(k)}\,\delta_{t,m,l}^{(j)}, \tag{B3}$$

$$b_{t+1,l}^{(j)} = b_{t,l}^{(j)} - \eta_{t,l}\frac{1}{M}\sum_{m=1}^{M} \delta_{t,m,l}^{(j)}, \tag{B4}$$

where the index t indicates the update step, and η is the learning rate and M is the size of the training data (or a batch thereof). We refer to one step of the form of Eqs. (B3) and (B4) as an iteration step. In the next section, we describe the methods used to efficiently evaluate the derivatives that appear in backpropagation, which may also be applied in other scenarios.

2. Automatic differentiation

We next introduce automatic differentiation (AD), which was first introduced in Ref. [129] and is widely used to numerically evaluate derivatives. AD computes numerical values of derivatives of functions, without developing algebraic expressions for the derivatives, by decomposing the function into a sequence of elementary functional steps. Using predeveloped subroutines (e.g., for the derivatives of certain known functions, as well as for the product rule and the chain rule), the derivative is computed alongside the actual function. The number of arithmetic operations when using AD is increased only by a constant factor compared to the number of arithmetic operations needed to evaluate a given function [89]. There exist two modes of AD, forward and reverse mode [130]. For a function with n inputs and m outputs, forward mode AD computes a column of the Jacobian, and reverse mode computes a row of the Jacobian. We illustrate each mode using the toy example of a NN in Fig. 11, where we omit the biases b_l for simplicity. In the context of NNs, reverse mode AD corresponds to the backpropagation algorithm.

We begin by applying reverse mode AD to the toy example from Fig. 11. Reverse mode AD in the context of NNs, i.e., backpropagation, consists of a forward and backward pass. In the forward pass, the input values are propagated forwards through the NN and the intermediate values are stored. In the backward pass, the derivatives are computed. For our toy NN from Fig. 11, the forward pass looks as follows, where we represent the values we compute at each step in the forward pass in a vector. We begin by loading the initial values, i.e., the input data,

$$\begin{pmatrix} a_0^{(1)} \\ a_0^{(2)} \end{pmatrix} = \begin{pmatrix} x^{(1)} \\ x^{(2)} \end{pmatrix}, \tag{B5}$$

propagating the data through the first linear transformation,

$$\begin{pmatrix} a_1^{(1)} \\ a_1^{(2)} \end{pmatrix} = \begin{pmatrix} x^{(1)}w_1^{(1,1)} + x^{(2)}w_1^{(1,2)} \\ x^{(1)}w_1^{(2,1)} + x^{(2)}w_1^{(2,2)} \end{pmatrix}, \tag{B6}$$

applying the nonlinear activation function,

$$\begin{pmatrix} z_1^{(1)} \\ z_1^{(2)} \end{pmatrix} = \begin{pmatrix} f_{\mathrm{nl}}\big(a_1^{(1)}\big) \\ f_{\mathrm{nl}}\big(a_1^{(2)}\big) \end{pmatrix}, \tag{B7}$$

followed by the second linear transformation,

$$\big(a_2^{(1)}\big) = \big(z_1^{(1)}w_2^{(1,1)} + z_1^{(2)}w_2^{(1,2)}\big), \tag{B8}$$

which leaves us with the final output of the toy NN, to which one typically applies a loss function, f_{loss},

$$C = f_{\mathrm{loss}}\big(a_2^{(1)}\big). \tag{B9}$$

Next, we go through the backward pass of reverse mode AD. At each step in the backward pass, we display the newly computed values, which correspond to the derivatives of the final output of the NN with respect to the weights of a given layer. Note that in practice one only computes the vectors on the right hand side, and not the symbolic derivatives, as doing so may be computationally expensive. We start with the cost of the NN, C, and compute the derivatives with respect to the weights in the final linear transformation,

$$\begin{pmatrix} \frac{\partial C}{\partial w_2^{(1,1)}} \\ \frac{\partial C}{\partial w_2^{(1,2)}} \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial a_2^{(1)}}\frac{\partial a_2^{(1)}}{\partial w_2^{(1,1)}} \\ \frac{\partial C}{\partial a_2^{(1)}}\frac{\partial a_2^{(1)}}{\partial w_2^{(1,2)}} \end{pmatrix} = \begin{pmatrix} z_1^{(1)} \\ z_1^{(2)} \end{pmatrix}, \tag{B10}$$

and proceed backwards, where the nonlinear activation function comes into play,

$$\begin{pmatrix} \frac{\partial C}{\partial w_1^{(1,1)}} \\ \frac{\partial C}{\partial w_1^{(2,1)}} \\ \frac{\partial C}{\partial w_1^{(1,2)}} \\ \frac{\partial C}{\partial w_1^{(2,2)}} \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial a_2^{(1)}}\frac{\partial a_2^{(1)}}{\partial z_1^{(1)}}\frac{\partial z_1^{(1)}}{\partial a_1^{(1)}}\frac{\partial a_1^{(1)}}{\partial w_1^{(1,1)}} \\ \frac{\partial C}{\partial a_2^{(1)}}\frac{\partial a_2^{(1)}}{\partial z_1^{(1)}}\frac{\partial z_1^{(1)}}{\partial a_1^{(1)}}\frac{\partial a_1^{(1)}}{\partial w_1^{(2,1)}} \\ \frac{\partial C}{\partial a_2^{(1)}}\frac{\partial a_2^{(1)}}{\partial z_1^{(2)}}\frac{\partial z_1^{(2)}}{\partial a_1^{(2)}}\frac{\partial a_1^{(2)}}{\partial w_1^{(1,2)}} \\ \frac{\partial C}{\partial a_2^{(1)}}\frac{\partial a_2^{(1)}}{\partial z_1^{(2)}}\frac{\partial z_1^{(2)}}{\partial a_1^{(2)}}\frac{\partial a_1^{(2)}}{\partial w_1^{(2,2)}} \end{pmatrix} = \begin{pmatrix} w_2^{(1,1)} f'_{\mathrm{nl}}\big(a_1^{(1)}\big)a_0^{(1)} \\ w_2^{(1,1)} f'_{\mathrm{nl}}\big(a_1^{(2)}\big)a_0^{(1)} \\ w_2^{(1,2)} f'_{\mathrm{nl}}\big(a_1^{(1)}\big)a_0^{(2)} \\ w_2^{(1,2)} f'_{\mathrm{nl}}\big(a_1^{(2)}\big)a_0^{(2)} \end{pmatrix}. \tag{B11}$$

It is worth pointing out, that we have now computed the entries of the gradient of the output of the NN with respect to the trainable parameters. Since we know the structure of the NN, we do not need to symbolically compute the derivatives (in the sense that we would obtain an algebraic representation of the derivative), but we can merely use the right hand side of the equations in the last two vectors to calculate the numerical values of the derivatives. This allows us to compute the gradient of the NN with respect to the weights with only a constant overhead compared to the evaluation of the NN (i.e., the forward pass) [89], as for the forward as well as the backward pass we need $O(E)$ arithmetic operations and

$O(n_{\text{nodes}})$ evaluations of the activation function and its derivative, where E is the number of trainable parameters and n_{nodes} is the number of nodes in the NN. A key point allowing us to do so is the reusability of terms that we computed and stored in the forward pass for the backward pass.

Next, we outline the process of applying forward mode AD to the toy NN. In forward mode, we evaluate the NN alongside computing the derivatives. Again, with AD we do not compute the symbolic expressions for the derivatives (these are only shown for illustration purposes), but only numerical values. In contrast to the forward pass in reverse mode AD, no intermediate values get stored for later retrieval during forward mode AD. We will represent the evaluations and the derivatives as a pair of vectors for each stage. The entries of the derivative vector (termed the tangents, with ∂) contain the derivatives of the corresponding entries in the vector containing the function evaluation values (the primals). The reader may ask "the derivative with respect to what?" The answer to this question lies in the initialization of the derivative values, as will become clearer by the end of our computation. We initialize the derivatives of the weight parameters w to be given by the parameters v with the labels and indices. We again begin by loading the input values, and in the case of forward mode AD we have

$$\begin{pmatrix} a_0^{(1)} \\ a_0^{(2)} \end{pmatrix} = \begin{pmatrix} x^{(1)} \\ x^{(2)} \end{pmatrix}, \quad \begin{pmatrix} \partial a_0^{(1)} \\ \partial a_0^{(2)} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \tag{B12}$$

and after the linear transformation we have

$$\begin{aligned} \begin{pmatrix} a_1^{(1)} \\ a_1^{(2)} \end{pmatrix} &= \begin{pmatrix} x^{(1)} w_1^{(1,1)} + x^{(2)} w_1^{(1,2)} \\ x^{(1)} w_1^{(2,1)} + x^{(2)} w_1^{(2,2)} \end{pmatrix}, \\ \begin{pmatrix} \partial a_1^{(1)} \\ \partial a_1^{(2)} \end{pmatrix} &= \begin{pmatrix} x^{(1)} v_1^{(1,1)} + x^{(2)} v_1^{(1,2)} \\ x^{(1)} v_1^{(2,1)} + x^{(2)} v_1^{(2,2)} \end{pmatrix}, \end{aligned} \tag{B13}$$

and following the nonlinear activation function we have

$$\begin{pmatrix} z_1^{(1)} \\ z_1^{(2)} \end{pmatrix} = \begin{pmatrix} f_{\text{nl}}(a_1^{(1)}) \\ f_{\text{nl}}(a_1^{(2)}) \end{pmatrix}, \quad \begin{pmatrix} \partial z_1^{(1)} \\ \partial z_1^{(2)} \end{pmatrix} = \begin{pmatrix} f'_{\text{nl}}(a_1^{(1)}) \partial a_1^{(1)} \\ f'_{\text{nl}}(a_1^{(2)}) \partial a_1^{(2)} \end{pmatrix}, \tag{B14}$$

and after the final linear transformation we get

$$\begin{aligned} (a_2^{(1)}) &= \big(f_{\text{nl}}(a_1^{(1)}) w_2^{(1,1)} + f_{\text{nl}}(a_1^{(2)}) w_2^{(1,2)}\big), \\ (\partial a_2^{(1)}) &= \big(f'_{\text{nl}}(a_1^{(1)}) \partial a_1^{(1)} w_2^{(1,1)} + f_{\text{nl}}(a_1^{(1)}) v_2^{(1,1)} \\ &\quad + f'_{\text{nl}}(a_1^{(2)}) \partial a_1^{(2)} w_2^{(1,2)} + f_{\text{nl}}(a_1^{(2)}) v_2^{(1,2)}\big), \end{aligned} \tag{B15}$$

where we applied the product rule. Finally, we compute the loss,

$$C = f_{\text{loss}}(a_2^{(1)}), \quad \partial C = f'_{\text{loss}}(a_2^{(1)}) \partial a_2^{(1)}. \tag{B16}$$

The quantity ∂C, which forward mode AD gives us, corresponds to $\nabla C \cdot v$, whereby ∇C is the gradient of C with respect to the trainable parameters, and v is the vector containing the initializations of the tangents in the same order as they appear in the gradient [130]. To see this in our example, let us expand ∂C,

$$\begin{aligned} \partial C = f'_{\text{loss}}(a_2^{(1)}) \big(f'_{\text{nl}}(a_1^{(1)}) \big(x^{(1)} v_1^{(1,1)} + x^{(2)} v_1^{(1,2)}\big) w_2^{(1,1)} \\ + f_{\text{nl}}(a_1^{(1)}) v_2^{(1,1)} + f'_{\text{nl}}(a_1^{(2)}) \big(x^{(1)} v_1^{(2,1)} \\ + x^{(2)} v_1^{(2,2)}\big) w_2^{(1,2)} + f_{\text{nl}}(a_1^{(2)}) v_2^{(1,2)}\big), \end{aligned} \tag{B17}$$

where this claim is more clearly visible. By choosing the initialization values for the derivatives, we can thus choose with respect to which weights we want to differentiate, by computing the directional derivative with respect to the initial values of the tangents. If we, e.g., set v to only have one nonzero entry, forward mode AD computes the derivative of C with respect to the corresponding parameter. Similarly to reverse mode AD, forward mode AD allows us to compute the directional derivative of the NN with only a constant overhead in runtime compared to the evaluation of the NN [89], with $O(E)$ arithmetic operations and $O(n_{\text{nodes}})$ evaluations of the activation function and its derivative.

To conclude, we can numerically compute the gradient of the NN with respect to the trainable parameters with reverse mode AD (backpropagation, in the context of NNs) and we can compute the inner product of the gradient with respect to the initial tangent values with forward mode AD. Both modes allow for the computation of the respective quantity (the gradient and the directional derivative) with a runtime with only a constant overhead (in practice, a factor of two to three [130]), compared to just the evaluation of the NN.

After having reviewed a range of relevant classical algorithms and subroutines, we proceed in the following sections to give an overview over a range of quantum algorithms and subroutines that we will reference later on in this work.

3. Variational quantum algorithms

We begin by introducing variational quantum algorithms (VQAs) together with their strengths and weaknesses. VQAs have been designed with the aim of being particularly well suited for the noisy intermediate-scale quantum (NISQ) era, by keeping the circuit depths short. Nevertheless, VQAs form a class of quantum algorithms that have been proposed for a variety of problems, such as machine learning [57], optimization [131] and chemistry [132]. In the context of VQAs, one defines a cost function $\mathcal{C}$, which encodes the solution to the problem, as well as a quantum circuit (or ansatz) $|\psi(\theta)\rangle$ parametrized by a set of parameters θ [132]. The quantum processor then evaluates the cost function (or its gradients), and a classical optimizer tries to find the optimal parameters $\theta^\star$ for the ansatz,

$$\theta^\star = \arg\min_\theta \mathcal{C}(|\psi(\theta)\rangle). \tag{B18}$$

The parameter shift rule allows for the evaluation of the gradient of a parametrized quantum circuit (PQC) with respect to a single parameter [133,134].

Lemma 2. Parameter shift rule [133]. Let $\mathcal{B}$ be an observable and let

$$\langle \mathcal{B} \rangle = \langle 0|^{\otimes n} \mathcal{U}^\dagger(\theta)\, \mathcal{B}\, \mathcal{U}(\theta)\, |0\rangle^{\otimes n} \tag{B19}$$

be the expectation value of $\mathcal{B}$ with respect to a parametrized n-qubit quantum circuit $\mathcal{U}(\theta)|0\rangle^{\otimes n}$ where θ denotes the set of parameters. If a given parameter θ_j appears in $\mathcal{U}(\theta)$ only once in the form of a gate $\mathcal{G}(\theta_j) = e^{-i\theta_j G}$ we can write $\mathcal{U}(\theta)|0\rangle^{\otimes n} = \mathcal{U}_1\mathcal{G}(\theta_j)\mathcal{U}_2|0\rangle^{\otimes n}$. Furthermore, if G is a Hermitian operator with two distinct eigenvalues (such as for any single qubit gate), which we can shift without loss of generality to $\pm r$, it holds that

$$\partial_{\theta_j}\langle\mathcal{B}\rangle/r = \langle 0|^{\otimes n}\mathcal{U}_2^\dagger\mathcal{G}^\dagger(\theta_j+s)\mathcal{U}_1^\dagger\mathcal{B}\mathcal{U}_1\mathcal{G}(\theta_j+s)\mathcal{U}_2|0\rangle^{\otimes n} - \langle 0|^{\otimes n}\mathcal{U}_2^\dagger\mathcal{G}^\dagger(\theta_j-s)\mathcal{U}_1^\dagger\mathcal{B}\mathcal{U}_1\mathcal{G}(\theta_j-s)\mathcal{U}_2|0\rangle^{\otimes n}, \tag{B20}$$

where $s = \pi/4r$.

In Ref. [135] the author extends the parameter shift rule to general two-qubit gates, with a constant overhead in runtime.

What makes VQAs uniquely suitable for NISQ devices is their typically short circuit depth, which prevents too many errors from accumulating throughout the computation on the quantum processor. Furthermore, the optimization is outsourced to a classical processor. At the same time, VQAs make use of the Hilbert space whose size increases exponentially in the number of qubits to encode features and find solutions.

A downside of VQAs is, however, the widespread presence of barren plateaus [64]. This phenomenon refers to the vanishing of the gradient, which happens exponentially fast in the number of qubits. The gradient is needed to minimize the cost function. When computing an expectation of an observable $\mathcal{C}$ of a parametrized n-qubit quantum circuit we have

$$\mathcal{U}(\theta) = \prod_k \mathcal{U}_k(\theta_k)\mathcal{W}_k, \tag{B21}$$

where $\mathcal{U}_k(\theta_k) = e^{-i\theta_k\mathcal{V}_k}$ and where $\mathcal{V}_k$ are Hermitian and $\mathcal{W}_k$ are not parametrized unitaries. When tuning the parameters θ, we do so by using a classical optimizer which typically relies on calculating the derivative with respect to a parameter θ_j of an expectation value we want to minimize, i.e.,

$$\partial_{\theta_j}\langle\mathcal{C}\rangle = i\langle 0|^{\otimes n}\mathcal{U}_-^\dagger[\mathcal{V}_j, \mathcal{U}_+^\dagger\mathcal{C}\mathcal{U}_+]\mathcal{U}_-|0\rangle^{\otimes n}, \tag{B22}$$

where $\mathcal{U}_-$ and $\mathcal{U}_+$ refer to the products of the factors from Eq. (B21) with $k < j$ and $k > j$, respectively, and the square brackets denote the commutator. Whenever the training ansatz $\mathcal{U}(\theta)$ is sufficiently random, $\mathcal{U}_-$ or $\mathcal{U}_+$ or both match the Haar random distribution for unitary matrices up to the second moment [136]. In Ref. [64], the authors show that if a circuit is sufficiently deep such that $\mathcal{U}_-$ or $\mathcal{U}_+$ or both form a two-design (matching the Haar random distribution up to the second moment), then with high probability the ansatz state will be on a barren plateau, i.e., the size of the gradient vanishes exponentially fast in the number of qubits and the optimizer will not be able to find a direction along which the parameters can be optimized. Randomly parametrized quantum circuits (PQC) are often used as initial guesses in variational quantum algorithms as a starting point for exploring the space of quantum states [64].

Several potential remedies to combat the emergence of barren plateaus have been put forward, often proposing to reduce the entanglement between qubits, or groups of qubits [137–140]. In Refs. [141,142], the authors show that over-parametrizing variational quantum circuits also improves the trainability. Furthermore, quantum convolutional neural networks [143] have been shown to not exhibit barren plateaus [144].

As touched upon in Sec. I, we refer to quantum algorithms as fault-tolerant, when their design is not concerned about limitations of the underlying hardware. The next sections will feature fault-tolerant quantum algorithms (as well as some classical methods they improve upon) that will be relevant later on in this work.

4. Classical and quantum-accelerated Monte Carlo methods

Monte Carlo (MC) methods use randomness to estimate numerical properties of systems which would prove intractable for an analytical analysis and are often employed in, e.g., physics [145], finance [146], and machine learning [147]. Next, we outline classical and quantum MC methods, starting with the univariate case in Appendix B 4 a and proceeding to the multivariate case in Appendix B 4 b and multilevel MC methods in Appendix B 4 c. We will apply these methods to the deep-learning architecture from Ref. [25] later in this work.

a. Univariate Monte Carlo methods

MC methods are often used to estimate the expected output of a randomised algorithm, we begin by focusing on the case where the quantity we want to estimate is a scalar, with the formulation from Ref. [99] for the general setting. Let $v(\mathcal{A})$ denote the scalar (hence univariate) random variable which returns the outcome of the randomised algorithm $\mathcal{A}$ processed by the function v. MC methods then aim to estimate the expectation value of w of $v(\mathcal{A})$ in the following way: They produce k samples by independently running $\mathcal{A}$ k times and taking the average of the samples to produce $\tilde{w}$ which is used as an estimator of the true value w. Assuming the variance of $v(\mathcal{A})$ is bounded by σ^2, it holds, by Chebyshev's inequality, that

$$P[|w-\tilde{w}| \geqslant \epsilon] \leqslant \frac{\sigma^2}{k\epsilon^2}, \tag{B23}$$

for $\epsilon > 0$. We can thus conclude, that by taking k samples where

$$k = O(\sigma^2/\epsilon^2), \tag{B24}$$

we can estimate w up to error ϵ with a success probability of, e.g., 0.99.

By employing a quantum algorithm, it is possible to achieve a quadratic speedup in σ/ϵ for the MC method, as first shown in Ref. [99]. We start by formulating the setting in terms of a quantum circuit, as done in Ref. [148]. Assume we have an algorithm $\mathcal{A}$ on n qubits, which, upon measurement, produces the n-bit result x with probability $|a_x|^2$. Furthermore, let $v(x) : \{0,1\}^n \mapsto [0,1]$. The goal is then to estimate the expectation value,

$$\mathbb{E}[v(\mathcal{A})] = \sum_{x=0}^{2^n-1}|a_x|^2 v(x). \tag{B25}$$

We may obtain this expectation value by combining $\mathcal{A}$,

$$\mathcal{A}|0\rangle^{\otimes n} = \sum_{x=0}^{2^n-1} a_x|x\rangle, \tag{B26}$$

and the rotation operation seen in Eq. (A2) to arrive at

$$\begin{aligned}\mathcal{R}(\mathcal{A}\otimes\mathcal{I}_2)|0\rangle^{\otimes n+1} &= \sum_{x=0}^{2^n-1} a_x|x\rangle(\sqrt{1-v(x)}|0\rangle + \sqrt{v(x)}|1\rangle)\\ &=: |\chi\rangle,\end{aligned} \tag{B27}$$

where $\mathcal{I}_d$ denotes the d-dimensional identity operation. Measuring the ancilla qubit in the state $|1\rangle$ has as its success probability the sought expectation value,

$$\bar{v} := \langle\chi|(\mathcal{I}_{2^n}\otimes|1\rangle\langle 1|)|\chi\rangle = \mathbb{E}[v(\mathcal{A})]. \tag{B28}$$

However, since the variance of this (Bernoulli) distribution is $\mathbb{V}[v(\mathcal{A})] = \bar{v}(1-\bar{v})/k$, where k is the number of samples, we still have to sample k times, where

$$k = O\left(\frac{\bar{v}(1-\bar{v})}{\epsilon^2}\right), \tag{B29}$$

to obtain a given accuracy ϵ, as in the classical case. The quadratic speedup in Ref. [99] comes from employing amplitude estimation [149]. The speedup with amplitude estimation is attained by encoding the desired expectation value to an eigenfrequency of an oscillating quantum system, and using additional qubits to extract the eigenfrequency. Consider the following unitary:

$$\mathcal{V} = \mathcal{I}_{2^{n+1}} - 2\mathcal{I}_{2^n}\otimes|1\rangle\langle 1|, \tag{B30}$$

such that $\langle\chi|\mathcal{V}|\chi\rangle = 1-2\bar{v}$. Because we can write any quantum state in the $(n+1)$-qubit Hilbert space as

$$\mathcal{V}|\chi\rangle = \cos(\theta/2)|\chi\rangle + e^{i\phi}\sin(\theta/2)|\chi^\perp\rangle, \tag{B31}$$

where $|\chi^\perp\rangle$ is a specific orthogonal complement of $|\chi\rangle$. Now,

$$1-2\bar{v} = \cos(\theta/2), \tag{B32}$$

so the goal is to estimate θ. Next, we define the unitaries,

$$\mathcal{U} = \mathcal{I}_{2^{n+1}} - 2|\chi\rangle\langle\chi|, \tag{B33}$$

and

$$\mathcal{Q} = \mathcal{U}\mathcal{V}\mathcal{U}\mathcal{V}, \tag{B34}$$

where $\mathcal{Q}$ performs a rotation by an angle of 2θ in the two-dimensional Hilbert space spanned by $|\chi\rangle$ and $\mathcal{V}|\chi\rangle$. The unitary $\mathcal{U}$ can be implemented via

$$\mathcal{U} = \mathcal{R}(\mathcal{A}\otimes\mathcal{I}_2)[\mathcal{I}_{2^{n+1}} - 2(|0\rangle\langle 0|)^{\otimes n+1}][\mathcal{R}(\mathcal{A}\otimes\mathcal{I}_2)]^\dagger. \tag{B35}$$

The task now becomes to estimate θ by estimating the eigenvalues $e^{\pm i\theta}$ of $\mathcal{Q}$ (recall that $\mathcal{Q}$ is unitary). This computation can be achieved by using the quantum phase estimation algorithm [109]. We present the amplitude estimation and the quantum-accelerated MC (QAMC) method in the following lemmas.

Lemma 3. Amplitude estimation [149]. The quantum algorithm termed amplitude estimation takes as input a single copy of a quantum state $|\chi\rangle$, unitary transformations $\mathcal{U} = \mathcal{I} - 2|\psi\rangle\langle\psi|$ and $\mathcal{V} = \mathcal{I} - 2\mathcal{P}$ where $\mathcal{P}$ is a projector, and an integer k. Amplitude estimation outputs $\tilde{a}$, an estimate of $a = \langle\chi|\mathcal{P}|\chi\rangle$ such that

$$|a-\tilde{a}| \leqslant 2\pi\frac{\sqrt{a(1-a)}}{k} + \frac{\pi^2}{k^2}, \tag{B36}$$

with probability at least $8/\pi^2$, using $\mathcal{U}$ and $\mathcal{V}$ t times each.

Lemma 4. Powering Lemma [150]. Let $\mathcal{A}$ be a classical or quantum algorithm which aims to estimate some quantity μ and whose output $\tilde{\mu}$ satisfies $|\mu-\tilde{\mu}| \leqslant \epsilon$ except with fixed probability $\gamma < 1/2$. Then, for any $\delta > 0$, it suffices to repeat $\mathcal{A}$ $O(\log 1/\delta)$ times and take the median to obtain an estimate which is accurate within ϵ with probability at least $1-\delta$.

Amplitude estimation, together with the powering lemma, are used in Ref. [99] to achieve a quantum speedup for Monte Carlo mean estimation. We start with the result for $0 \leqslant v(\mathcal{A}) \leqslant 1$, which is a direct application of Lemmas 3 and 4 to the setting outlined so far.

Lemma 5. Mean estimation for [0, 1] bounded functions [99]. Let $\mathcal{A}$ be a quantum circuit on n qubits and let $v(\mathcal{A})$ be the random variable that takes on the value $v(x) \in [0,\ 1]$ when $\mathcal{A}$ outputs x. Let $\mathcal{R}$ be defined such that

$$\mathcal{R}|x\rangle|0\rangle = |x\rangle[\sqrt{1-v(x)}|0\rangle + \sqrt{v(x)}|1\rangle]. \tag{B37}$$

Furthermore, let $|\chi\rangle = \mathcal{R}(\mathcal{A}\otimes\mathcal{I}_2)|0\rangle^{\otimes n+1}$ and let $\mathcal{U} = \mathcal{I}_{2^{n+1}} - 2|\chi\rangle\langle\chi|$. Then there exists a quantum algorithm that uses $O(\log 1/\delta)$ copies of $|\chi\rangle$ and uses $\mathcal{U}$ $O(t\ \log 1/\delta)$ times to output an estimate $\hat{\bar{v}}$ such that

$$|\hat{\bar{v}} - \mathbb{E}[v(\mathcal{A})]| \leqslant C\left(\frac{\sqrt{\mathbb{E}[v(\mathcal{A})]}}{t} + \frac{1}{t^2}\right), \tag{B38}$$

with probability at least $1-\delta$, where C is a universal constant. In particular, for any fixed $\delta > 0$ and and ϵ such that $0 < \epsilon \leqslant 1$, to produce an estimate $\hat{\bar{v}}$ such that $|\hat{\bar{v}} - \mathbb{E}[v(\mathcal{A})]| \leqslant \epsilon\mathbb{E}[v(\mathcal{A})]$ it suffices to take $t = O(1/(\epsilon\sqrt{\mathbb{E}[v(\mathcal{A})]}))$. To achieve $|\hat{\bar{v}} - \mathbb{E}[v(\mathcal{A})]| \leqslant \epsilon$ with probability at least $1-\delta$ it suffices to take $t = O(1/\epsilon)$.

The result in Lemma 5 is improved from $v(\mathcal{A}) \in [0,\ 1]$ to $v(\mathcal{A})$ having a bounded variance.

Lemma 6. Mean estimation for functions with bounded variance [99]. Let $\mathcal{A}$ be a quantum circuit on n qubits and let $v(\mathcal{A})$ be the random variable that takes on the value $v(x)$ when $\mathcal{A}$ outputs x such that $\mathbb{V}[v(\mathcal{A})] < \lambda^2$. Let the accuracy be $\epsilon < 4\lambda$. Let $\mathcal{U}$ and $|\chi\rangle$ be as in Lemma 5. Then there exists a quantum algorithm that uses $O(\log\lambda/\epsilon\ (\log\log\lambda/\epsilon))$ copies of $|\chi\rangle$ and uses $\mathcal{U}$ for a number of times $O[\lambda/\epsilon\ (\log\lambda/\epsilon)^{3/2}(\log\log\lambda/\epsilon)]$ and estimates $\mathbb{E}[v(\mathcal{A})]$ up to additive error ϵ with success probability at least $2/3$.

In Ref. [99], Lemma 6 is derived from Lemma 5 by decomposing $v(\mathcal{A})$ into a set of random variables whose outputs lie in [0, 1] and by estimating the mean of each of these random variables. The success probability can be improved to $1-\delta$ for $\delta > 0$ at a multiplicative cost of $O(\log 1/\delta)$ using Lemma 4. Therefore, we can, e.g., improve the success probability to 0.99, without an extra cost being reflected if we formulate it using the $\tilde{O}$ notation, as $\tilde{O}$ hides polylogarithmic factors. It is worth pointing out that QAMC (see Lemma 6)

offers a (nearly) quadratic speedup in terms of ϵ compared to the classical method [see Eq. (B24)].

We next outline the generalization of MC methods from the univariate case to the multivariate case.

b. Multivariate Monte Carlo methods

In multivariate MC estimation, the goal is to estimate the mean in the case where $v(\mathcal{A}) \in \mathbb{R}^d$. In the classical scenario, estimating all entries $\mathbb{E}[v_1(\mathcal{A})], \ldots, \mathbb{E}[v_d(\mathcal{A})]$ can be done simultaneously with the same executions of $\mathcal{A}$ with an overhead of $\log(d)$ in the sample complexity due to Hoeffding's inequality, as shown in Appendix A of Ref. [151]. We restate the result in Lemma 7 below.

Lemma 7. Classical multivariate Monte Carlo estimation (Appendix A in Ref. [151]). Let an algorithm $\mathcal{A}$ generate a d-dimensional random variable $v(\mathcal{A})$, bounded in l_∞ norm $\|v(\mathcal{A})\|_\infty \leqslant B$, then we can estimate $\mathbb{E}[v(\mathcal{A})]$ up to error ϵ in the l_∞ norm with success probability $1-\delta$ with a sample complexity of,

$$O\left(\frac{B^2}{\epsilon^2}\log\frac{d}{\delta}\right). \tag{B39}$$

In the quantum case, however, this simultaneous estimation is impeded as one relies on amplitude estimation to estimate the mean, which is encoded in the relative phase. Due to the periodicity and boundedness of the relative phase, one cannot use the same executions of $\mathcal{A}$ to simultaneously estimate all entries of $v(\mathcal{A})$. This inconvenience generally results in a linear overhead of d in the sample complexity in the quantum case over the univariate (quantum) scenario. As shown in Ref. [151], there are special cases where this linear dependence can be slightly improved, however, not down to the logarithmic dependence of the classical case.

c. Multilevel Monte Carlo methods

MC methods are often used to estimate an expectation value of a random variable determined by the solution of an SDE [33], as is the case in Ref. [25], see Eq. (8). Here, we outline multilevel MC (MLMC) methods, which have the potential to improve the sample complexity of MC methods for estimating the mean value of a function depending on the (discretized) solution of an SDE, in the classical and quantum case. We follow [33] and begin with the setting and then outline the classical and quantum methods.

Let $X_t \in \mathbb{R}^d$ be a stochastic process defined by the SDE,

$$\mathrm{d}X_t = \mu(t, X_t) + \sigma(t, X_t)\mathrm{d}W_t, \tag{B40}$$

where $\sigma(t, X) \in \mathbb{R}^{d\times d}$, $\mu(t, X) \in \mathbb{R}^d$ and W_t is a standard Brownian motion (as introduced in Sec. I B). Consider the problem of estimating an expectation value at time T of the form (Problem 1 in Ref. [33])

$$\mathbb{E}\big(\mathcal{P}(X_T)|X_{t_0}\big) \in \mathbb{R}, \tag{B41}$$

where $\mathcal{P} : \mathbb{R}^d \mapsto \mathbb{R}$ is some payoff function and X_{t_0} is the value of X_t at the initial time. In the case of a general SDE without an explicit solution, one first has to discretize the SDE on an interval $[t_0, T]$ with step size Δt to produce an approximate solution using a numerical scheme.

A numerical scheme for approximating the solution of an SDE $\hat{X}_{t_n}$ with time step size $\Delta t = T/N$ is of strong order r if there exists a constant $K_m > 0$, for any $m \in \mathbb{N}$, such that

$$\mathbb{E}\left[\sup_{0\leqslant n\leqslant N}\left\|\hat{X}_{t_n} - X_{t_n}\right\|_2^m\right] \leqslant K_m(\Delta t)^{rm}. \tag{B42}$$

When using a numerical scheme of strong order r, the error for estimating Eq. (B41) scales as $\epsilon = O((\Delta t)^r)$ [96], resulting in an ϵ dependence of $O(1/\epsilon^{2+1/r})$ in the sample complexity for the classical case [33]. A way to improve the error dependence in the sample complexity is to make use of the MLMC method [152–154]. We follow the introduction of the MLMC method from Ref. [33].

The MLMC method aims to estimate the expectation value of some random variable P, $\mathbb{E}[P]$, by means of a sequence of random variables $P_0, P_1, \ldots, P_K$ where each element of the sequence approximates P with greater accuracy and $K = O(\log(2\epsilon^{-1}))$. In the setting of a discretized SDE, we can think of P as the payoff function $\mathcal{P}$ evaluated at the terminal time $\mathcal{P}(X_T)$ which we want to estimate, and the index of the P_k relating to how many approximation steps N we take via $N_k = T/(\Delta t_k) = 2^k T$. MLMC methods then estimate $\mathbb{E}[P]$ by observing that the following telescoping sum holds, due to the linearity of the expectation value,

$$\mathbb{E}[P_K] = \mathbb{E}[P_0] + \sum_{k=0}^{K}\mathbb{E}[P_k - P_{k-1}], \tag{B43}$$

where $P_{-1} = 0$. The MLMC method estimates each of the summands in a way that minimizes the cost. To estimate $\mathbb{E}[P]$ we introduce the estimator Y,

$$Y = \sum_{k=0}^{K} Y_k, \tag{B44}$$

where we have for Y_k the following expression:

$$Y_k = \frac{1}{N_k}\sum_{i=0}^{N_k}\left(P_k^{(i)} - P_{k-1}^{(i)}\right), \tag{B45}$$

where i indicates the sample. Each Y_k is then approximated via MC methods, where $P_k^{(i)} - P_{k-1}^{(i)}$ comes from one Brownian path, but with a different number of discretization steps for P_k and P_{k-1}. Further following [33], we bound the error,

$$\mathbb{E}[Y - \mathbb{E}[P]]^2 \leqslant \mathbb{V}[Y] + \mathbb{E}[P_K - P]^2 \leqslant \epsilon^2, \tag{B46}$$

where we need to consider the cost C_k and the variance V_k of $P_k - P_{k-1}$. The total cost and variance of Y are given by $\sum_{k=0}^{K} N_k C_k$ and $\sum_{k=0}^{K} V_k/N_k$, respectively. Assuming that $C_k = O(2^{\gamma k})$ and $V_k = O(2^{-\beta k})$ where $\beta \geqslant \gamma$, it can be shown that the number of samples needed in the classical case to estimate $\mathbb{E}[P]$ within error ϵ is $\tilde{O}(\epsilon^{-2})$, removing the $1/r$ dependence.

When applying MLMC to the problem of estimating $\mathbb{E}(\mathcal{P}(X_T)|X_{t_0})$ from Eq. (B41) with X_t governed by the SDE from Eq. (B40), the authors in Ref. [33] make a set of assumptions, beginning with assumptions on quantities appearing in the SDE.

Assumption 4. (Assumption 11 in Ref. [33]). The functions μ and σ are globally Lipschitz continuous. Furthermore,

they assume that for the initial value X_{t_0} it holds that $\mathbb{E}[X_{t_0}^m] \leqslant C_m$ for constants $C_m \geqslant 0$.

Next, the authors make an assumption on the numerical scheme employed [recall the definition o the strong order r in Eq. (B42)]. The so-called Tayler-Itô schemes constitute a general class of high order schemes (for solving SDEs) of the following form [96]

$$X_{k+1} = \sum_{\alpha} f_\alpha(kh, X_k) I_\alpha, \tag{B47}$$

where the f_α are coefficient functions and the I_α are integrals over the time interval $[kh, (k+1)h]$.

Assumption 5. (Assumption 2 in Ref. [33]). The coefficient functions f_α as in Eq. (B47) are globally Lipschitz continuous with respect to X.

Finally, the authors in Ref. [33] make an assumption on the payoff function.

Assumption 6. (Assumption 3 in Ref. [33]. The payoff function $\mathcal{P}$ from Eq. (B41) is globally Lipschitz continuous.

We summarize the classical result for the application of MLMC to SDEs as follows:

Lemma 8. (Classical MLMC for SDE Payoff Estimation; Proposition 3 in Ref. [33]). Consider the problem of estimating a payoff function as in Eq. (B41) of a discretized SDE of the form of Eq. (B40) under Assumptions 4 to 6. Then MLMC with a scheme of strong order r estimates $\mathbb{E}[\mathcal{P}(X_T)|X_{t_0}]$ up to mean-squared error ϵ^2 with probability at least 0.99 with a sample complexity of

$$\begin{aligned} &O(\epsilon^{-2}) && r > 1/2 \\ &O(\epsilon^{-2}(\log 1/\epsilon)^2) && r = 1/2 \\ &O(\epsilon^{-1/r}) && r < 1/2. \end{aligned} \tag{B48}$$

In Ref. [33], the authors present a quantum-accelerated version of MLMC and QAMLMC. The speedup is derived from making use of QAMC from Ref. [99] (see Appendix B 4 a). In QAMLMC, QAMC is used to estimate the expectation values $\mathbb{E}[P_k - P_{k-1}]$. The result for applying QAMLMC to the problem of estimating quantities of the form Eq. (B41) is summarized below.

Lemma 9. (QAMLMC; Theorem 3 in Ref. [33]). Consider the problem of estimating a payoff function as in Eq. (B41) of a discretized SDE of the form of Eq. (B40) under Assumptions 4 to 6. Then QAMLMC with a scheme of strong order r estimates $\mathbb{E}[\mathcal{P}(X_T)|X_{t_0}]$ up to additive error ϵ with probability at least 0.99 with a sample complexity of

$$\begin{aligned} &O(\epsilon^{-1}(\log 1/\epsilon)^{3/2}(\log\log 1/\epsilon)^2) && r > 1 \\ &O(\epsilon^{-1}(\log 1/\epsilon)^{7/2}(\log\log 1/\epsilon)^2) && r = 1 \\ &O(\epsilon^{-1/r}(\log 1/\epsilon)^{3/2}(\log\log 1/\epsilon)^2) && r < 1. \end{aligned} \tag{B49}$$

5. Robust inner product estimation

Next, we outline a fault-tolerant quantum algorithm for estimating inner products. It is a vital component of the algorithm (presented in the same paper, [101]) for addressing the bottleneck of training the NNs in the deep-learning architecture which we will discuss in Sec. V. The authors of Ref. [101] introduce a quantum algorithm termed robust inner product estimation (RIPE) which is a generalization of the inner product estimation algorithm from Ref. [155]. It allows for estimating the inner product between two states v and c by using their (amplitude encoded) quantum states, i.e.,

$$|v\rangle = \frac{1}{\|v\|_2} \sum_j v^{(j)} |j\rangle, \tag{B50}$$

and analogously for $|c\rangle$. The inner product estimation algorithm (Lemma 4.2 from Ref. [155]) allows for the estimation of the inner product between two vectors v and c, with known norms, up to error ϵ with probability at least $1-\gamma$ and in time $\tilde{O}(\|v\|_2\|c\|_2 T \log(2/\gamma)/\epsilon)$ where T is the time needed to prepare $|v\rangle$ and $|c\rangle$. We here sketch the proof.

We begin in the following state:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle. \tag{B51}$$

Next, we load the vectors $|v\rangle$ and $|c\rangle$ with the operations $|0\rangle|0\rangle \mapsto |0\rangle|v\rangle$ and similarly $|1\rangle|0\rangle \mapsto |1\rangle|c\rangle$. This takes us in the state

$$\frac{1}{\sqrt{2}}(|0\rangle|v\rangle + |1\rangle|c\rangle). \tag{B52}$$

After applying a Hadamard gate on the first qubit, we obtain

$$\tfrac{1}{2}(|0\rangle(|v\rangle + |c\rangle) + |1\rangle(|v\rangle - |c\rangle)). \tag{B53}$$

Given the state in Eq. (B53), the probability of measuring one in the first qubit p_1 is

$$p_1 = \frac{1}{4}[2 - 2\langle v|c\rangle] = \frac{1 - \langle v|c\rangle}{2}, \tag{B54}$$

from which one can calculate $\langle v|c\rangle$, given $\|v\|_2$ and $\|c\|_2$. By rewriting $|1\rangle(|v\rangle - |c\rangle)$ as $|y,\ 1\rangle$ (swapping the qubits), we now have

$$\sqrt{p_1}|y,\ 1\rangle + \sqrt{1-p_1}|G,\ 0\rangle, \tag{B55}$$

where G signifies a garbage state. Next, using amplitude estimation (see Lemma 3), we arrive at the state

$$\sqrt{\alpha}|\hat{p}_1,\ G',\ 1\rangle + \sqrt{1-\alpha}|G'',\ 1\rangle, \tag{B56}$$

where $\alpha > 8/\pi^2$, $|\hat{p}_1 - p_1| \leqslant \epsilon$ and G' as well as G'' are further garbage states. To extract $\hat{p}_1$ the authors from Ref. [155] make use of a result from Ref. [156].

Lemma 10. (Quantum median estimation; Lemma 8 in Ref. [156]). Let $\mathcal{Y}$ be a unitary that maps

$$\mathcal{Y} : |0\rangle^{\otimes n} \mapsto \sqrt{a}|x,\ 1\rangle + \sqrt{1-a}|G,\ 0\rangle, \tag{B57}$$

for some $1/2 < a \leqslant 1$ in time T. Then there exists a quantum algorithm which, for any $\gamma > 0$ and $1/2 < a_0 \leqslant a$, prepares a state $|\psi\rangle$ such that $\||\psi\rangle - |0\rangle^{\otimes nL}|x\rangle\|_2 \leqslant \sqrt{\gamma}$ for some integer L in time

$$2T\left\lceil \frac{\ln 2/\gamma}{2(|a_0| - 1/2)^2} \right\rceil. \tag{B58}$$

Lemma 10 allows us to extract a quantum state $|\psi\rangle$ such that $\||\psi\rangle - |0\rangle^{\otimes L}|\hat{p}_1,\ G'\rangle\|_2 \leqslant \sqrt{\gamma}$. From $\hat{p}_1$ we can then compute $\langle v|c\rangle$ via the relation from Eq. (B54). In Ref. [155], the authors generalize the inner product estimation algorithm. We present their result in form of Lemma 11.

Lemma 11. (Robust inner product estimation (RIPE) [101]). If quantum states $|v\rangle$ and $|c\rangle$ can each be prepared in time T, and if the norms $\|v\|_2$ and $\|c\|_2$ are known within multiplicative error $\epsilon/3$, then the mapping $|v\rangle|c\rangle|0\rangle \mapsto |v\rangle|c\rangle|s\rangle$ where, with probability at least $1-\gamma$,

$$|s - v\cdot c| \leqslant \begin{cases} \epsilon|v\cdot c| & \text{in time } \tilde{O}\left(\frac{T(\log 1/\gamma)\|v\|_2\|c\|_2}{\epsilon|v\cdot c|}\right) \\ \epsilon & \text{in time } \tilde{O}\left(\frac{T(\log 1/\gamma)\|v\|_2\|c\|_2}{\epsilon}\right). \end{cases} \quad \text{(B59)}$$

[1] M. Braun and M. Golubitsky, *Differential Equations and their Applications* (Springer, New York, USA, 1983), Vol. 1.
[2] W. A. Strauss, *Partial Differential Equations: An Introduction* (John Wiley & Sons, Hoboken, New Jersey, USA, 2007).
[3] F. Black and M. Scholes, J. Political Econ. **81**, 637 (1973).
[4] R. Bellman, *Dynamic Programming* (Princeton University Press, Princeton, New Jersey, USA, 1957).
[5] H. Emmerich, *The Diffuse Interface Approach in Materials Science: Thermodynamic Concepts and Applications of Phase-Field Models* (Springer, Berlin, Germany, 2003).
[6] M. Hutzenthaler, A. Jentzen, and T. Kruse, Found. Comput. Math. **22**, 905 (2022).
[7] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, Nature (London) **549**, 195 (2017).
[8] L. Zhao, Z. Zhao, P. Rebentrost, and J. Fitzsimons, Quantum Mach. Intell. **3**, 21 (2021).
[9] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda, Science **292**, 472 (2001).
[10] B. P. Lanyon, J. D. Whitfield, G. G. Gillett, M. E. Goggin, M. P. Almeida, I. Kassal, J. D. Biamonte, M. Mohseni, B. J. Powell, M. Barbieri *et al.*, Nat. Chem. **2**, 106 (2010).
[11] R. P. Feynman, *Feynman and Computation* (CRC Press, 2018), pp. 133–153.
[12] P. W. Shor, in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (IEEE, 1994), pp. 124–134.
[13] L. K. Grover, Phys. Rev. Lett. **80**, 4329 (1998).
[14] J. Preskill, Quantum **2**, 79 (2018).
[15] A. Montanaro, npj Quantum Inf. **2**, 15023 (2016).
[16] S. Krinner, N. Lacroix, A. Remm, A. D. Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann *et al.*, Nature (London) **605**, 669 (2022).
[17] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell *et al.*, Nature (London) **574**, 505 (2019).
[18] G. Q. AI, Nature (London) **614**, 676 (2023).
[19] A. W. Harrow, A. Hassidim, and S. Lloyd, Phys. Rev. Lett. **103**, 150502 (2009).
[20] D. W. Berry, J. Phys. A: Math. Theor. **47**, 105301 (2014).
[21] P. W. Shor, in *Proceedings of 37th Conference on Foundations of Computer Science* (IEEE Computer Society, Washington DC, USA, 1996), pp. 56–65.
[22] S. J. Devitt, W. J. Munro, and K. Nemoto, Rep. Prog. Phys. **76**, 076001 (2013).
[23] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke *et al.*, Rev. Mod. Phys. **94**, 015004 (2022).
[24] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, New J. Phys. **18**, 023023 (2016).
[25] J. Han, A. Jentzen, and W. E, Proc. Natl. Acad. Sci. USA **115**, 8505 (2018).
[26] E. Pardoux and S. Peng, *Stochastic Partial Differential Equations and their Applications* (Springer, 1992), pp. 200–217.
[27] E. Pardoux and S. Tang, Probab. Theory Relat. Fields **114**, 123 (1999).
[28] N. El Karoui, S. Peng, and M. C. Quenez, Math. Finance **7**, 1 (1997).
[29] E. Gobet, *Monte-Carlo Methods and Stochastic Processes: From Linear to Non-Linear* (CRC Press, Boca Raton, Florida, USA, 2016).
[30] K. Hornik, M. Stinchcombe, and H. White, Neural Netw. **2**, 359 (1989).
[31] S. Rubinstein-Salzedo, *Cryptography* (Springer, 2018), pp. 75–83.
[32] M. O'Searcoid, *Metric Spaces* (Springer Science & Business Media, Berlin, Germany, 2006).
[33] D. An, N. Linden, J.-P. Liu, A. Montanaro, C. Shao, and J. Wang, Quantum **5**, 481 (2021).
[34] R. Durrett, *Probability: Theory and Examples* (Cambridge University Press, Cambridge, England, UK, 2019), Vol. 49.
[35] A. M. Childs and J.-P. Liu, Commun. Math. Phys. **375**, 1427 (2020).
[36] J.-P. Liu, H. Ø. Kolden, H. K. Krovi, N. F. Loureiro, K. Trivisa, and A. M. Childs, Proc. Natl. Acad. Sci. USA **118**, e2026805118 (2021).
[37] T. Carleman, Acta Math. **59**, 63 (1932).
[38] M. Forets and A. Pouly, arXiv:1711.02552.
[39] K. Kowalski and W.-H. Steeb, *Nonlinear Dynamical Systems and Carleman Linearization* (World Scientific, Singapore, 1991).
[40] H. Krovi, Quantum **7**, 913 (2023).
[41] J. Liu, M. Liu, J.-P. Liu, Z. Ye, Y. Alexeev, J. Eisert, and L. Jiang, Nat. Commun. **15**, 434 (2024).
[42] J.-P. Liu, D. An, D. Fang, J. Wang, G. H. Low, and S. Jordan, Commun. Math. Phys. **404**, 963 (2023).
[43] A. M. Childs, J.-P. Liu, and A. Ostrander, Quantum **5**, 574 (2021).
[44] J. M. Arrazola, T. Kalajdzievski, C. Weedbrook, and S. Lloyd, Phys. Rev. A **100**, 032306 (2019).
[45] S. Lloyd, G. De Palma, C. Gokler, B. Kiani, Z.-W. Liu, M. Marvian, F. Tennie, and T. Palmer, arXiv:2011.06571.
[46] A. Montanaro and S. Pallister, Phys. Rev. A **93**, 032324 (2016).
[47] S. Jin, N. Liu, and Y. Yu, J. Comput. Phys. **487**, 112149 (2023).
[48] S. Jin and N. Liu, arXiv:2202.07834.
[49] D. An, J.-P. Liu, D. Wang, and Q. Zhao, arXiv:2211.05246.
[50] W. K. Wootters and W. H. Zurek, Nature (London) **299**, 802 (1982).
[51] O. Kyriienko, A. E. Paine, and V. E. Elfving, Phys. Rev. A **103**, 052416 (2021).

[52] M. Lubasch, J. Joo, P. Moinier, M. Kiffner, and D. Jaksch, Phys. Rev. A **101**, 010301(R) (2020).

[53] Y. Cao, A. Papageorgiou, I. Petras, J. Traub, and S. Kais, New J. Phys. **15**, 013021 (2013).

[54] P. C. S. Costa, S. Jordan, and A. Ostrander, Phys. Rev. A **99**, 012323 (2019).

[55] A. Engel, G. Smith, and S. E. Parker, Phys. Rev. A **100**, 062315 (2019).

[56] N. Linden, A. Montanaro, and C. Shao, Commun. Math. Phys. **395**, 601 (2022).

[57] M. Schuld and N. Killoran, Phys. Rev. Lett. **122**, 040504 (2019).

[58] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, Nature (London) **567**, 209 (2019).

[59] E. Farhi and H. Neven, arXiv:1802.06002.

[60] T. Goto, Q. H. Tran, and K. Nakajima, Phys. Rev. Lett. **127**, 090506 (2021).

[61] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac, and N. Killoran, arXiv:2001.03622.

[62] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, and S. Woerner, Nat. Comput. Sci. **1**, 403 (2021).

[63] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, Nature (London) **549**, 242 (2017).

[64] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, Nat. Commun. **9**, 4812 (2018).

[65] L. Leone, S. F. Oliviero, L. Cincio, and M. Cerezo, Quantum **8**, 1395 (2024).

[66] M. Larocca, S. Thanasilp, S. Wang, K. Sharma, J. Biamonte, P. J. Coles, L. Cincio, J. R. McClean, Z. Holmes, and M. Cerezo, arXiv:2405.00781.

[67] M. Ragone, B. N. Bakalov, F. Sauvage, A. F. Kemper, C. O. Marrero, M. Larocca, and M. Cerezo, arXiv:2309.09342.

[68] M. Cerezo, M. Larocca, D. García-Martín, N. Diaz, P. Braccia, E. Fontana, M. S. Rudolph, P. Bermejo, A. Ijaz, S. Thanasilp *et al.*, arXiv:2312.09121.

[69] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, Adv. Neural Inf. Process. Syst. **32**, 8024 (2019).

[70] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi *et al.*, arXiv:1811.04968.

[71] J. F. de Freitas, M. Niranjan, A. H. Gee, and A. Doucet, Neural Comput. **12**, 955 (2000).

[72] P. Pagnottoni, Front. Artif. Intell. **2**, 5 (2019).

[73] R. J. DiRisio, F. Lu, and A. B. McCoy, J. Phys. Chem. A **125**, 5849 (2021).

[74] J. Hermann, J. Spencer, K. Choo, A. Mezzacapo, W. Foulkes, D. Pfau, G. Carleo, and F. Noé, Nat. Rev. Chem. **7**, 692 (2023).

[75] C. Liu, F. Huang, A. Qiu, A. D. N. Initiative *et al.*, Neural Netw. **159**, 14 (2023).

[76] S.-Y. Tsui, C.-Y. Wang, T.-H. Huang, and K.-B. Sung, Biomed. Opt. Express **9**, 1531 (2018).

[77] Y. Xie, C. Shi, H. Zhou, Y. Yang, W. Zhang, Y. Yu, and L. Li, arXiv:2103.10432.

[78] F. Riccio, R. Capobianco, and D. Nardi, *RoboCup 2016: Robot World Cup XX 20* (Springer, 2017), pp. 256–267.

[79] A. G. Baydin, B. A. Pearlmutter, D. Syme, F. Wood, and P. Torr, arXiv:2202.08587.

[80] A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves, Adv. Neural Inf. Process. Syst. **29**, 4125 (2016).

[81] A. Griewank and A. Walther, ACM Trans. Math. Software **26**, 19 (2000).

[82] C. C. Margossian, Wiley Interdiscip. Rev.: Data Min. Knowl. Discovery **9**, e1305 (2019).

[83] K. H. Wan, O. Dahlsten, H. Kristjánsson, R. Gardner, and M. Kim, npj Quantum Inf. **3**, 36 (2017).

[84] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, arXiv:1312.6199.

[85] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, arXiv:1802.05957.

[86] C. Anil, J. Lucas, and R. Grosse, *International Conference on Machine Learning* (PMLR, 2019), pp. 291–301.

[87] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, Massachusetts, USA, 2016).

[88] E. Hazan *et al.*, Found. Trends Optim. **2**, 157 (2016).

[89] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation* (SIAM, Philadelphia, Pennsylvania, USA, 2008).

[90] H. Robbins and S. Monro, Ann. Math. Stat. **22**, 400 (1951).

[91] L. Bottou, *Neural Networks: Tricks of the Trade*, 2nd ed. (Springer, Berlin, Germany, 2012), pp. 421–436.

[92] J. Zhang, T. He, S. Sra, and A. Jadbabaie, arXiv:1905.11881.

[93] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, in *International Conference on Machine Learning* (PMLR, 2018), pp. 794–803.

[94] A. Gilyén, S. Arunachalam, and N. Wiebe, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms* (SIAM, 2019), pp. 1425–1444.

[95] S. P. Jordan, Phys. Rev. Lett. **95**, 050501 (2005).

[96] P. E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations* (Springer, 1992).

[97] D. Hughes-Hallett, A. M. Gleason, and W. G. McCallum, *Calculus: Single and Multivariable* (John Wiley & Sons, Hoboken, New Jersey, USA, 2020).

[98] S. Chakrabarti, R. Krishnakumar, G. Mazzola, N. Stamatopoulos, S. Woerner, and W. J. Zeng, Quantum **5**, 463 (2021).

[99] A. Montanaro, Proc. R. Soc. A **471**, 20150301 (2015).

[100] G. N. Mil'shtein, Teor. Prob. Appl. **19**, 557 (1975).

[101] J. Allcock, C.-Y. Hsieh, I. Kerenidis, and S. Zhang, ACM Trans. Quantum Comput. **1**, 1 (2020).

[102] D. Kim, M. Kim, and W. Kim, IEEE Access **8**, 215125 (2020).

[103] I. I. Baskin, D. Winkler, and I. V. Tetko, Expert Opin. Drug Discovery **11**, 785 (2016).

[104] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, Adv. Neural Inf. Process. Syst. **27**, 1269 (2014).

[105] X. Yu, T. Liu, X. Wang, and D. Tao, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, New York, USA, 2017), pp. 7370–7379.

[106] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (IEEE, 2013), pp. 6655–6659.

[107] I. Kerenidis and A. Prakash, arXiv:1603.08675.

[108] A. Ambainis, SIAM J. Comput. **37**, 210 (2007).

[109] M. A. Nielsen and I. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, England, UK, 2002).

[110] E. Bernstein and U. Vazirani, in *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, USA, 1993), pp. 11–20.

[111] H. Buhrman, J. Tromp, and P. Vitányi, in *Automata, Languages and Programming: 28th International Colloquium, ICALP 2001 Crete, Greece, July 8–12, 2001 Proceedings 28* (Springer, 2001), pp. 1017–1027.

[112] J. Bausch, S. Subramanian, and S. Piddock, Quantum Mach. Intell. **3**, 16 (2021).

[113] T. Häner, M. Roetteler, and K. M. Svore, arXiv:1805.12445.

[114] P. Rebentrost, M. Santha, and S. Yang, Quantum **7**, 1174 (2023).

[115] P. Wocjan, C.-F. Chiang, D. Nagaj, and A. Abeyesinghe, Phys. Rev. A **80**, 022340 (2009).

[116] J. F. Doriguello, A. Luongo, J. Bao, P. Rebentrost, and M. Santha, *17th Conference on the Theory of Quantum Computation, Communication and Cryptography*, Leibniz International Proceedings in Informatics (LIPIcs) (Schloss Dagstuhl Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2022), Vol. 232, pp. 2:1–2:24.

[117] L. Grover and T. Rudolph, arXiv:quant-ph/0208112.

[118] S. Herbert, Phys. Rev. E **103**, 063302 (2021).

[119] C. Zoufal, A. Lucchi, and S. Woerner, npj Quantum Inf. **5**, 103 (2019).

[120] X. Gao, Z. Zhang, and L. Duan, arXiv:1711.02038.

[121] A. M. Childs, B. W. Reichardt, R. Spalek, and S. Zhang, arXiv:quant-ph/0703015.

[122] V. Giovannetti, S. Lloyd, and L. Maccone, Phys. Rev. Lett. **100**, 230502 (2008).

[123] V. Giovannetti, S. Lloyd, and L. Maccone, Phys. Rev. Lett. **100**, 160501 (2008).

[124] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, in *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management* (Association for Computing Machinery, New York, USA, 2013), pp. 2333–2338.

[125] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, J. Am. Soc. Inf. Sci. **41**, 391 (1990).

[126] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis *et al.*, Comput. Intell. Neurosci. **2018**, 1 (2018).

[127] T. Young, D. Hazarika, S. Poria, and E. Cambria, IEEE Comput. Intell. Mag. **13**, 55 (2018).

[128] M. A. Nielsen, *Neural Networks and Deep Learning* (Determination press San Francisco, 2015), Vol. 25.

[129] R. E. Wengert, Commun. ACM **7**, 463 (1964).

[130] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, J. Mach. Learn. Res. **18**, 1 (2018).

[131] E. Farhi, J. Goldstone, and S. Gutmann, arXiv:1411.4028.

[132] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio *et al.*, Nat. Rev. Phys. **3**, 625 (2021).

[133] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, Phys. Rev. A **98**, 032309 (2018).

[134] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, Phys. Rev. A **99**, 032331 (2019).

[135] G. E. Crooks, arXiv:1905.13311.

[136] F. G. Brandao, A. W. Harrow, and M. Horodecki, Commun. Math. Phys. **346**, 397 (2016).

[137] E. Grant, L. Wossnig, M. Ostaszewski, and M. Benedetti, Quantum **3**, 214 (2019).

[138] T. L. Patti, K. Najafi, X. Gao, and S. F. Yelin, Phys. Rev. Res. **3**, 033090 (2021).

[139] A. Skolik, J. R. McClean, M. Mohseni, P. van der Smagt, and M. Leib, Quantum Mach. Intell. **3**, 5 (2021).

[140] T. Volkoff and P. J. Coles, Quantum Sci. Technol. **6**, 025008 (2021).

[141] J. Kim, J. Kim, and D. Rosa, Phys. Rev. Res. **3**, 023203 (2021).

[142] M. Larocca, N. Ju, D. García-Martín, P. J. Coles, and M. Cerezo, Nat. Comput. Sci. **3**, 542 (2023).

[143] I. Cong, S. Choi, and M. D. Lukin, Nat. Phys. **15**, 1273 (2019).

[144] A. Pesah, M. Cerezo, S. Wang, T. Volkoff, A. T. Sornborger, and P. J. Coles, Phys. Rev. X **11**, 041011 (2021).

[145] M. E. Newman and G. T. Barkema, *Monte Carlo Methods in Statistical Physics* (Clarendon Press, Oxford, UK, 1999).

[146] P. Glasserman, *Monte Carlo Methods in Financial Engineering* (Springer, New York, USA, 2004), Vol. 53.

[147] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, Mach. Learn. **50**, 5 (2003).

[148] P. Rebentrost, B. Gupt, and T. R. Bromley, Phys. Rev. A **98**, 022321 (2018).

[149] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, Contemp. Math. **305**, 53 (2002).

[150] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, Theor. Comput. Sci. **43**, 169 (1986).

[151] A. Cornelissen and S. Jerbi, arXiv:2107.03410.

[152] S. Heinrich, in *Large-Scale Scientific Computing: Third International Conference, LSSC 2001 Sozopol, Bulgaria, June 6–10, 2001 Revised Papers 3* (Springer, 2001), pp. 58–67.

[153] M. B. Giles, Oper. Res. **56**, 607 (2008).

[154] M. B. Giles, Acta Numer. **24**, 259 (2015).

[155] I. Kerenidis, J. Landman, A. Luongo, and A. Prakash, Adv. Neural Inf. Process. Syst. **32**, 4134 (2019).

[156] N. Wiebe, A. Kapoor, and K. Svore, Quantum Info. Comput. **15**, 316 (2015).