

Received 19 August 2024; revised 10 January 2025; accepted 14 January 2025; date of publication 20 January 2025; date of current version 20 February 2025.

Digital Object Identifier 10.1109/TQE.2025.3532017

Variational Quantum Algorithms for Differential Equations on a Noisy Quantum Computer

NICLAS SCHILLO^{ID} AND ANDREAS STURM^{ID}

Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO, 70569 Stuttgart, Germany

Corresponding author: Niclas Schillo (e-mail: niclas.schillo@iao.fraunhofer.de).

This work was supported by the Ministry of Economic Affairs, Labour and Tourism Baden-Württemberg in the frame of the Competence Center Quantum Computing Baden-Württemberg through the project SEQUOIA End-to-End.

ABSTRACT The role of differential equations (DEs) in science and engineering is of paramount importance, as they provide the mathematical framework for a multitude of natural phenomena. Since quantum computers promise significant advantages over classical computers, quantum algorithms for the solution of DEs have received a lot of attention. Particularly interesting are algorithms that offer advantages in the current noisy intermediate-scale quantum (NISQ) era, characterized by small and error-prone systems. We consider a framework of variational quantum algorithms, quantum circuit learning (QCL), in conjunction with derivation methods, in particular the parameter shift rule, to solve DEs. As these algorithms were specifically designed for NISQ computers, we analyze their applicability on NISQ devices by implementing QCL on an IBM quantum computer. Our analysis of QCL without the parameter shift rule shows that we can successfully learn different functions with three-qubit circuits. However, the hardware errors accumulate with increasing number of qubits, and thus, only a fraction of the qubits available on the current quantum systems can be effectively used. We further show that it is possible to determine derivatives of the learned functions using the parameter shift rule on the IBM hardware. The parameter shift rule results in higher errors, which limits its execution to low-order derivatives. Despite these limitations, we solve a first-order DE on the IBM quantum computer. We further explore the advantages of using multiple qubits in QCL by learning different functions simultaneously and demonstrate the solution of a coupled DE on a simulator.

INDEX TERMS Differential equations, quantum algorithm, quantum computing.

I. INTRODUCTION

The efficient and accurate solution of differential equations is of great importance in numerous scientific fields and in various branches of industry. However, many differential equations are very difficult to simulate, e.g., due to their high degree of nonlinearities or numerical instability. It is well known that quantum algorithms can significantly speed up the computation of certain problems. Because of this potential advantage, much research is devoted to solving differential equations using quantum algorithms. For example, several quantum algorithms for solving differential equations have been proposed that utilize quantum algorithms only as a subroutine [1], [2], [3], [4]. A common feature of these algorithms is the use of quantum phase estimation [5] as the fundamental component in their quantum subroutines. Furthermore, they employ oracle-based approaches for data access and rely on amplitude-encoded states. This approach

comes with several challenges, including the input and output problem [6], as well as substantial computational overhead in constructing quantum oracles. In particular, for nonlinear differential equations, quantum algorithms are scarce, with only a few notable examples [7].

In addition, these algorithms require a large-scale fault-tolerant quantum computer. However, the current noisy intermediate-scale quantum (NISQ) [8] era is characterized by quantum computers that are error-prone and limited in size. Hence, quantum algorithms that work on NISQ computers are of central interest. The most prominent candidates in this context are variational quantum algorithms. Due to their hybrid quantum–classical architecture, they require fewer qubits and quantum gates so that they can cope with the limitations of NISQ systems.

In this work, we consider variational quantum algorithms based on the quantum circuit learning (QCL) framework [9].

QCL is a loosely defined term that is used differently in the literature. In our consideration, QCL circuits are multiqubit circuits that have one data encoding layer at the beginning where a variable is encoded into the quantum state using quantum feature map encoding [10]. This is followed by a variational layer consisting of parameterized quantum gates. Finally, an expectation value is measured. In this way, we obtain a parameterized function of the encoded variable. The types of functions that can be represented with this method depend on the data encoding layer and the number of qubits.

One application of QCL circuits is to learn 1-D functions by training the parameters with a classical optimizer. Building upon this idea, it is possible to use QCL in combination with the parameter shift rule (PSR) to solve differential equations. The PSR is an approach to obtain gradients of a parameterized quantum circuit [9], [11].

The previously described circuits have already been extensively studied in the literature, and their effectiveness has been demonstrated by classical simulations [9], [12]. Using these circuits to solve differential equations has also been analyzed and classically simulated [13], [14], [15], [16]. Since these algorithms were proposed specifically for NISQ systems, it is of great interest whether these algorithms can be executed on such systems in practice. So far, however, the full algorithm has never been implemented on NISQ hardware.

In this work, we focus on the executability of QCL circuits on current quantum computers and investigate possible error sources. It is shown that different functions can be learned with a three-qubit QCL circuit on a superconducting IBM quantum computer with under 100 optimization steps. However, if we increase the number of qubits, the hardware errors increase significantly and an execution becomes infeasible. Furthermore, we show that the PSR, which is necessary for the solution of differential equations, can be executed on the NISQ hardware but leads to considerable errors because it is very prone to noise. Despite these difficulties, we successfully solve a simple differential equation with QCL circuits and the PSR on the IBM quantum computer.

Another open question regarding QCL algorithms is how to take advantage of the multiqubit nature of these circuits. We present methods to use the multiqubit character of QCL circuits to efficiently learn multiple functions simultaneously or to solve coupled differential equations.

The rest of this article is organized as follows. Section II gives an introduction to QCL and the circuits used in this work. Following this, Section III introduces the IBM hardware on which the QCL algorithm is to be tested and provides additional implementation details. Subsequently, in Section IV, we perform prior simulations with different noise models and experiments on the IBM hardware to determine suitable settings for our subsequent execution of QCL. In Section V-A, we learn exemplary functions with a statevector simulator and, subsequently, in Section V-B, on the IBM quantum computer. Following this, the possibility of solving differential equations in combination with the PSR is explored. In order to investigate how NISQ-friendly it is

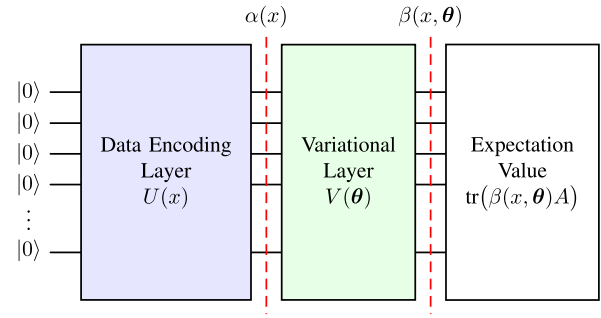


FIGURE 1. Structure of QCL circuits starting with the data encoding layer $U(x)$ followed by the parameterized variational layer $V(\theta)$ and the calculation of the expectation value of an observable A .

to solve differential equations with QCL circuits, the PSR is tested on the IBM quantum computer in Section VI-A. In addition, in Section VI-B, a simple differential equation is solved on the quantum computer. Section VII examines whether it is possible to learn several functions with a single QCL circuit by measuring multiple qubits. With this concept, a coupled differential equation is solved with a single QCL circuit on a simulator in Section VII-A. Finally, Section VIII concludes this article.

II. QUANTUM CIRCUIT LEARNING

The general structure of QCL circuits is shown in Fig. 1. In the data encoding layer $U(x)$, the variable x is encoded into the quantum state with quantum feature map encoding [10]. After the data encoding layer, the variational layer $V(\theta)$ follows, which consists of parameterized quantum gates with the parameters $\theta = (\theta_0, \theta_1, \dots)$. At the end, an expectation value of an observable A is calculated.

The data encoding layer in this work consists of

$$U(x) = \bigotimes_{n=1}^N R_Y(\varphi(x)) \quad (1)$$

where N is the number of qubits and $\varphi(x)$ is an inner function.

For example, a data encoding layer with $\varphi(x) = x$ results in the density matrix

$$\alpha(x) = \frac{1}{2^N} \bigotimes_{n=1}^N (I + \sin(x)X + \cos(x)Z). \quad (2)$$

By multiplying out and using addition theorems, we can see that the expression contains trigonometric functions of the forms $\sin(nx)$ and $\cos(nx)$ with $n = 1, 2, \dots, N$. In this case, the expectation value without the variational layer, $\langle A \rangle_\alpha(x) = \text{tr}(\alpha(x)A)$, has the form

$$\langle A \rangle_\alpha(x) = c_0 + \sum_{n=1}^N (c_n \sin(nx) + c_{n+N} \cos(nx)) \quad (3)$$

where c_n are scalar coefficients that depend on the observable A . To adjust the coefficients c_n , the variational layer is introduced. The new coefficients, which we again denote with c_n , now depend on the variational parameters θ , which allows us

to control their value. Hence, the expectation value after the variational layer, $\langle A \rangle_\beta(x, \theta) = \text{tr}(\beta(x, \theta)A)$, has the form

$$\langle A \rangle_\beta(x, \theta) = c_0(\theta) + \sum_{n=1}^N \left(c_n(\theta) \sin(nx) + c_{n+N}(\theta) \cos(nx) \right). \quad (4)$$

In [9], a different data encoding layer

$$U(x) = \bigotimes_{n=1}^N R_Y(\arcsin(x)) \quad (5)$$

was introduced. This data encoding scheme results in the density matrix

$$\alpha(x) = \frac{1}{2^N} \bigotimes_{n=1}^N \left(I + xX + \sqrt{1-x^2}Z \right) \quad (6)$$

and gives a set of polynomials up to the order of x^N with additional $\sqrt{1-x^2}$ terms.

One application of QCL is to learn arbitrary functions $f(x)$. Here, the parameters are chosen such that the expectation value $\langle A \rangle_\beta$ matches the function $f(x)$. For this purpose, a cost function on several training points x_i is minimized with a classical optimizer.

A. COST FUNCTION

An example of a simple cost function is

$$L(\theta) = \sum_i |f(x_i) - f_{\text{QC}}(x_i, \theta)|^2 \quad (7)$$

where $f_{\text{QC}}(x, \theta) = \text{tr}(\beta(x, \theta)A)$ is the expectation value of the observable A and will be referred to as the quantum model function, \sum_i sums over a number of training points x_i , and $f(x)$ is the function to be learned. While this approach can learn the qualitative behavior of functions, it may show strong deviations in the final result, as discussed in Appendix A. To increase the accuracy of function learning, an improved cost function is introduced that includes a post-processing parameter θ_{post}

$$L(\theta) = \sum_i |f(x_i) - f_{\text{QC}}^{\text{post}}(x_i, \theta)|^2 \quad (8)$$

where $f_{\text{QC}}^{\text{post}}(x, \theta) = f_{\text{QC}}(x, \theta) \cdot \theta_{\text{post}}$. The postprocessing parameter θ_{post} is optimized along with the other parameters. This additional degree of freedom significantly increases the expressivity of the model and leads to faster and more accurate learning. It also extends the value range of the quantum model functions to $f_{\text{QC}}^{\text{post}}(x, \theta) \in \mathbb{R}$ for $\theta_{\text{post}} \in \mathbb{R}$, overcoming the limitation of the Z expectation value range $[-1, 1]$.

B. QUANTUM CIRCUIT

In this work, we analyze QCL circuits with the following structure. First, the input data are encoded using $R_Y(x)$ or $R_Y(\arcsin(x))$ gates applied to each qubit. Next, an entanglement layer is created using CNOT gates in a linear chain

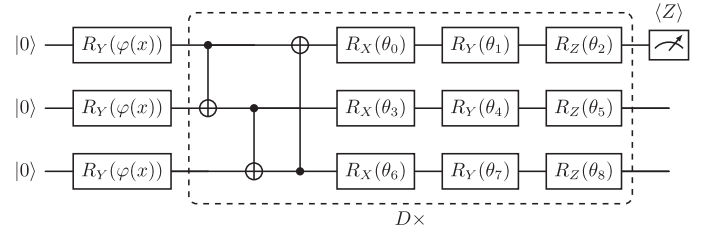


FIGURE 2. Three-qubit QCL circuit with $R_Y(\varphi(x))$ data encoding followed by a variational block consisting of three CNOT gates to achieve circular entanglement and θ -parameterized x -, y -, and z -rotations. The variational block is repeated D times. The Z expectation value of the first qubit is measured.

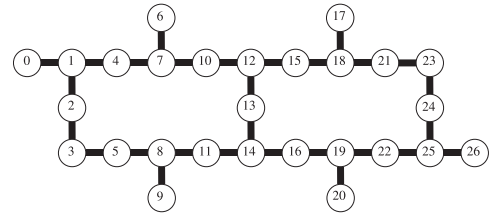


FIGURE 3. Coupling map of `ibmq_ehningen`.

with an additional gate between the first and last qubits, forming a circular entanglement pattern. This entanglement is used to generate functions up to the order N on the first qubit. Afterward, θ -parameterized x -, y -, and z -rotations are applied to each qubit, allowing for any unitary single-qubit operation, excluding global phase. The entanglement and parameterized rotation layers are repeated D times, using the same parameters θ in each block. These repetitions increase the expressivity of the circuit without increasing the number of parameters [17]. From now on, we use $A = Z_0$, i.e., we use the Z expectation value of the first qubit to read out our quantum model function. The circuit structure for the example of $N = 3$ is illustrated in Fig. 2.

III. HARDWARE IMPLEMENTATION DETAILS

The QCL circuits will be executed on the `ibmq_ehningen` quantum computer. This system is a 27-qubit superconducting IBM quantum computer based on transmon qubits of the Falcon chip class. One important characteristic of real quantum computers is that their operations are subject to errors. For `ibmq_ehningen`, we show exemplary error rates in Appendix C. Furthermore, they only have a certain set of native gates that can be executed directly (for `ibmq_ehningen`: CNOT, I, RZ, \sqrt{X} , X), and they have limited connectivity between the different qubits as seen in the coupling map in Fig. 3 for `ibmq_ehningen`.

Despite the limited connectivity, arbitrary circuits can be realized by applying additional SWAP gates. Since a SWAP gate between connected qubits consists of three CNOT gates, this increases the circuit depth, which makes an optimized transpilation to the specific hardware especially important.

Our QCL circuits in the form given in Fig. 2 cannot be executed directly on `ibmq_ehningen` as they do not meet the

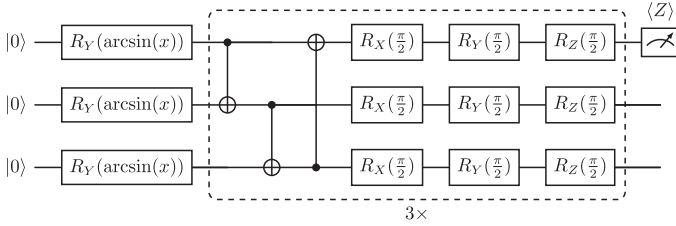


FIGURE 4. Three-qubit QCL circuit with $R_Y(\arcsin(x))$ data encoding and a variational layer with $\theta_i = \frac{\pi}{2}$ for all i .

hardware constraints with regard to the coupling map and the native gate set of this back end. To transform the circuits into a compatible form, we use the Qiskit transpiler [18] with the `optimization_level = 2` setting. This medium optimization level is designed to strike a balance between improving the circuit performance and maintaining reasonable transpilation times. At this level, the transpiler applies the following key transformations to optimize the circuit for execution on the IBM system. First, the transpiler searches for an initial qubit layout that minimizes the need for SWAP gates when mapped to the coupling map of the device. Then, the circuit is unrolled to the basis gates supported by the hardware. Finally, optimizations are performed in the form of commutative gate cancellation and redundant reset removal to minimize the circuit depth.

In addition, all experiments on `ibmq_ehningen` are performed with Twirled Readout Error eXtinction (TRES) [19]. TRES is a technique that mitigates readout errors in quantum computations by randomly applying Pauli X gates to qubits just before measurement, effectively diagonalizing the noise channel of the measurement. This randomization allows for easier characterization and correction of readout errors without requiring knowledge about the error model. Apart from this, no error mitigation techniques are applied.

IV. PRIOR INVESTIGATIONS

Before we run the resource-intensive quantum–classical variational workflow to learn functions by optimizing the parameters θ , we first select fixed parameters to make statements about the scalability of the circuits and the number of shots required. For this purpose, all parameters θ_i are set to $\pi/2$, as shown in Fig. 4 for a qubit number of $N = 3$, a depth of $D = 3$, and $R_Y(\arcsin(x))$ data encoding.

It is easy to check that this circuit yields

$$f_{\text{QC}}(x) = (-1)^N \cdot x \quad (9)$$

where N is the number of qubits.

To transpile the circuit, we use the Qiskit transpiler endowed with the coupling map and basis gates of `ibmq_ehningen`. As an initial layout, we choose the qubits $0, \dots, N - 1$. The stochastic components of the optimization process are made reproducible by choosing a fixed transpilation seed (that we chose as 123). In order to determine a suitable setting for our later experiments on the real IBM quantum back end, we run the transpiled circuit on a simulator with three different noise models. Our first noise model

(noise model 1) has perfect quantum operations, and the only error source stems from finite sampling, i.e., from shot noise. For the second one (noise model 2), we randomly and independently insert Pauli operators after gates and before measurements with probabilities determined by the gate [20]. Single-qubit gates are modified with probability $p_1/3$ by one of the nontrivial Pauli operators $\{X, Y, Z\}$, and two qubit gates are followed by a nonidentity Pauli product, i.e., one of $\{P_1 \otimes P_2 \mid P_1, P_2 \in \{I, X, Y, Z\}\} \setminus \{I \otimes I\}$, with probability $p_2/15$. Moreover, a binary measurement in the Z basis has the wrong outcome with probability p_r in this noise model. The error rates are taken as $p_1 = 0.00024$, $p_2 = 0.0075$, and $p_r = 0.012$. These error rates correspond to the median error rates as reported in the calibration data from the `ibmq_ehningen` quantum computer on 30 January 2024 (see Appendix C). Finally, our third noise model (noise model 3) is derived from the `ibmq_ehningen` calibration data via the method `NoiseMode.from_backend_properties` of the Qiskit Aer package [21] and includes, in addition to single-gate, two-qubit-gate, and readout errors, a depolarizing error and a thermal relaxation error.

A. NUMBER OF SHOTS

First, we investigate how the number of shots affects the result in order to determine a suitable number for the subsequent experiments. To do this, we use the circuit in Fig. 4 with $N = 3$ and the x values $x \in \{-1, -0.5, 0, 0.5, 1\}$. We simulate these circuits to obtain f_{QC} with the noise models introduced before and a range of different shot numbers. For each noise model and shot number, we repeat the simulation 20 times. Fig. 5 shows the different values of $f_{\text{QC}}(x)$ for the selected x values as x markers and the exact values as horizontal lines. For noise model 1 (only shot noise), the results fluctuate around the exact values and diminish as the number of shots increases. For noise models 2 and 3, there is a noticeable drift of $f_{\text{QC}}(x)$ toward zero, independent of the number of shots. This drift is more pronounced in noise model 3 (derived from calibration data) compared to noise model 2 (Pauli noise). Importantly, even at high shot numbers, this deviation persists, indicating that it is an effect of the hardware errors and cannot be compensated with a higher shot number. Based on these results, we will use 2000 shots for our subsequent experiments. This number of shots provides a good balance between accuracy and computational efficiency. At 2000 shots, the results for all noise models show low fluctuations, while still maintaining reasonable execution times for the subsequent hardware experiments.

B. NUMBER OF QUBITS

We now analyze how the error scales with increasing qubit count, providing insights into the practical limitations of the circuit size for `ibmq_ehningen`. For this purpose, circuits with the same structure but different qubit numbers are simulated. In addition, the circuits are also executed on the real `ibmq_ehningen` back end. Fig. 6 shows $(-1)^N f_{\text{QC}}(x)$ for different circuit sizes on a simulator with noise model

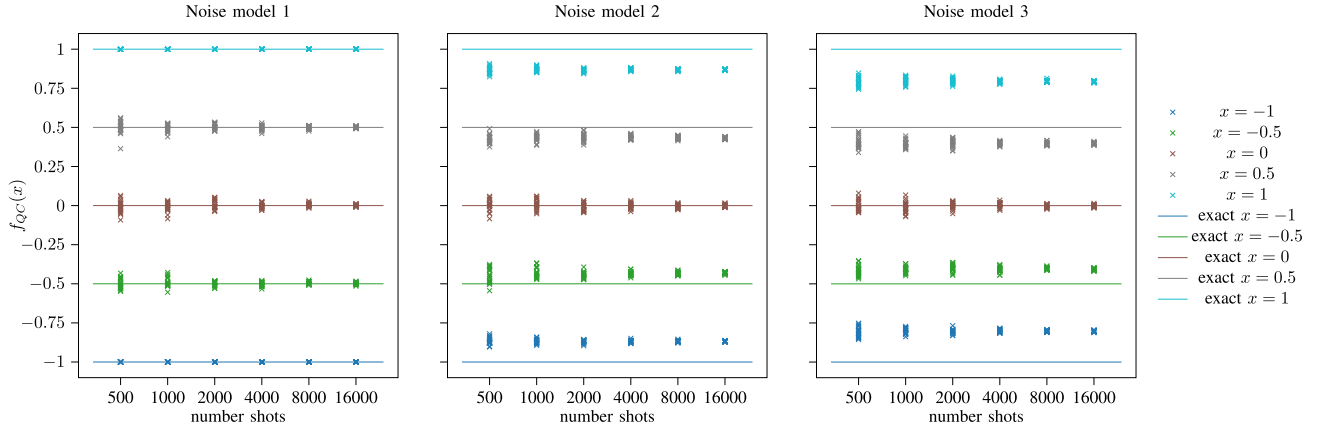


FIGURE 5. $f_{\text{QC}}(x)$ of QCL circuit structured as given in Fig. 4 evaluated at the points $x \in \{-1, -0.5, 0, 0.5, 1\}$, plotted against the number of shots. Three noise models are compared: noise model 1 (only shot noise), noise model 2 (Pauli noise), and noise model 3 (derived from calibration data).

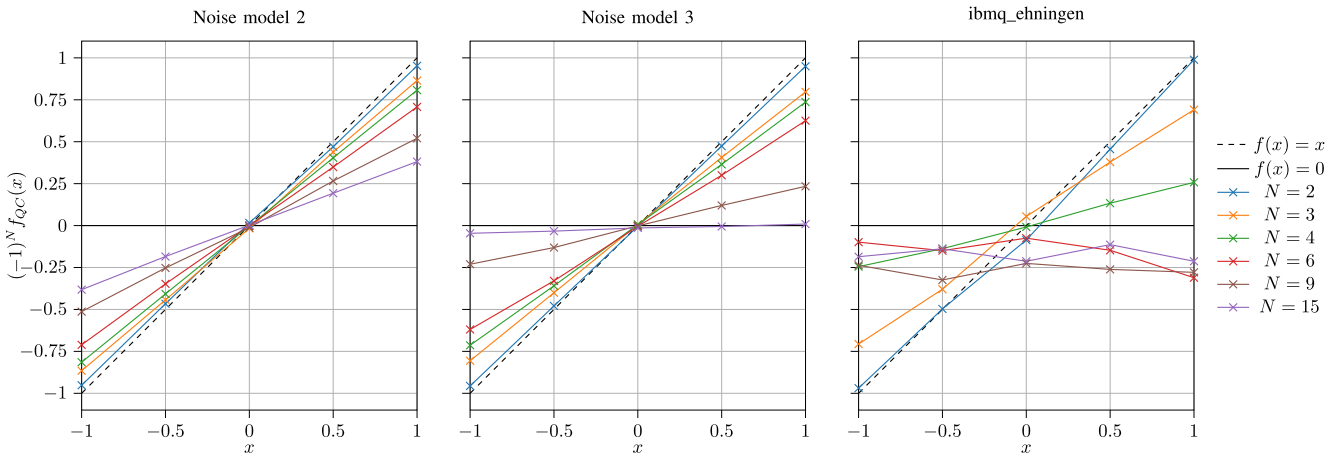


FIGURE 6. $(-1)^N f_{\text{QC}}(x)$ of QCL circuits structured as given in Fig. 4 for different numbers of qubits evaluated at the points $x \in \{-1, -0.5, 0, 0.5, 1\}$ on a simulator with noise model 2 (Pauli noise) and noise model 3 (derived from calibration data), as well as on the `ibmq_ehningen`.

2 (Pauli noise) and noise model 3 (derived from calibration data) as well as on `ibmq_ehningen`. Moreover, the plot shows $f(x) = x$, which corresponds to $(-1)^N f_{\text{QC}}(x)$ for an exact statevector simulation. Due to the increasing number of CNOT gates, the circuit depth also increases significantly with the number of qubits. We observe that the increasing depth and therefore increasing hardware errors lead to the measured expectation values approaching

$$f_{\text{QC}}(x) \equiv 0. \quad (10)$$

Among the simulated error models, the noise model 2 (Pauli noise) shows a slower convergence toward zero, which is to be expected since this model assumes fewer sources of error. On the real `ibmq_ehningen` back end, the expectation values approach $f_{\text{QC}}(x) \equiv 0$ more rapidly than in the simulations, indicating a faster accumulation of errors. The discrepancy between the real back end and the error models illustrates the limitations of simplified noise models in representing the error dynamics in the actual quantum hardware. The error models chosen here can reflect the general behavior, but are not detailed enough for a more accurate representation. We

will discuss possible reasons for this discrepancy after examining the scaling of the error in more detail.

For a more comprehensive picture, we analyze the mean absolute error (MAE)

$$\text{MAE} = \frac{1}{T} \sum_{i=1}^T |f(x_i) - f_{\text{QC}}(x_i, \theta)| \quad (11)$$

where T is the number of training points, for different number of qubits N . On real quantum devices, every qubit and every gate has an individual error rate, and we expect that these error rates have a strong influence on MAE. Thus, we now consider two transpilation of our underlying circuit (see Fig. 4) that use different qubits of `ibmq_ehningen`. We use the qubits $0, \dots, N-1$ (first qubits) and, in addition, the qubits $26-N+1, \dots, 26$ (last qubits) as the initial layouts for the transpiler (see Fig. 3). Fig. 7 shows the MAE for qubit numbers from $N=2$ to $N=20$ for the three different noise models and the two initial layouts on a simulator. Moreover, we add the MAE for `ibmq_ehningen` when the transpiler selected the best qubits.

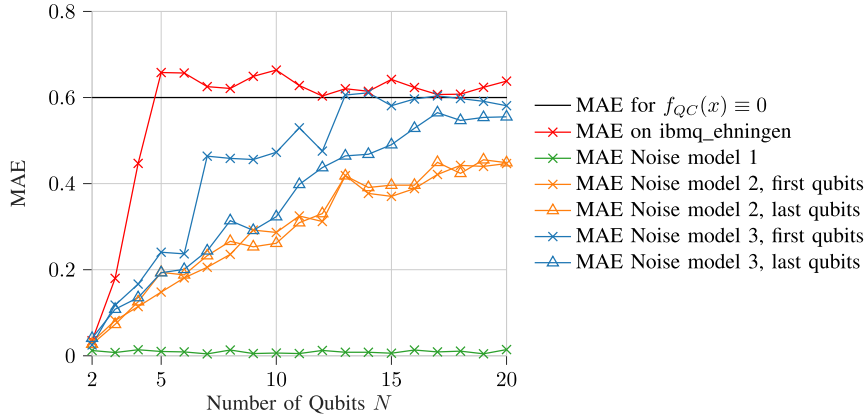


FIGURE 7. MAE of QCL circuits structured as given in Fig. 4 for different numbers of qubits evaluated at the points $x \in \{-1, -0.5, 0, 0.5, 1\}$ on a simulator with noise model 1 (only shot noise), noise model 2 (Pauli noise), and noise model 3 (derived from calibration data) and on *ibmq_ehningen*. In addition, two different sets of qubits of *ibmq_ehningen* are used (first and last qubits). The black line indicates the error of MAE = 0.6, which corresponds to the error in the case $f_{QC}(x) \equiv 0$.

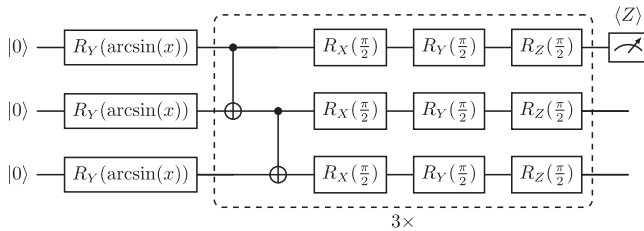


FIGURE 8. Three-qubit QCL circuit with linear entanglement with $R_Y(\arcsin(x))$ data encoding. The Z expectation value of the first qubit is measured.

Clearly, the MAE for noise model 1 is not affected by the qubit number since it uses perfect qubits and gates. For noise model 2, we see a linear increase in MAE with the number of qubits. We will discuss this in more detail in the next section. For our most advanced noise model 3, we observe a higher MAE compared to the second noise model. Moreover, we can see that the transpilation with the first initial qubit layout suffers from a much higher MAE than the transpilation with the second layout for $N \geq 7$ qubits. This can be explained since beginning with this number of qubits, the first transpilation has to use the CNOT gate between qubits 4 and 7, which has by far the highest error rate (see Table 2 in Appendix C), whereas the second transpilation can use CNOT gates with lower errors. The MAE from the real quantum computer *ibmq_ehningen* increases even more strongly and already plateaus for $N = 5$ qubits on MAE = 0.6. This can be expected since this is the MAE of $f_{QC}(x) \equiv 0$ and, as we have seen in Fig. 6, *ibmq_ehningen* returns approximately this quantum function for this qubit number regime. We see that that the error models provide better results than the real experiments, which indicates that they cannot capture all error sources present in *ibmq_ehningen*. For example, a study on the *ibmq_ehningen* processor revealed significant impacts of crosstalk on gate fidelities, leading to correlated errors between simultaneously executed quantum gates on neighboring qubits [22]. This research highlights the

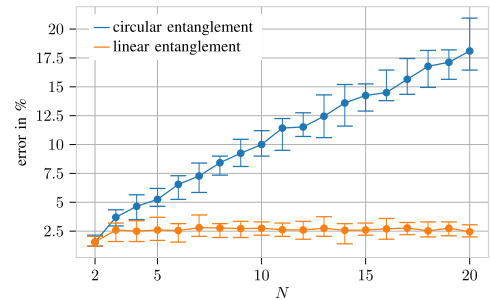


FIGURE 9. Error at $x = 1$ averaged over 20 repetitions using QCL circuits structured as given in Fig. 4 on a simulator with 2000 shots and a CNOT error of 1%. The errors for different qubit numbers with circular entanglement (blue line) and linear entanglement (orange line) are compared.

importance of considering crosstalk in quantum system modeling. Another study focusing on NISQ systems for quantum optimization uncovered time-dependent errors in IBM quantum computers, showing varying results at different times [23]. These findings demonstrate some of the error sources in superconducting quantum hardware. The discrepancy between the error models used in this work and real experiments can be attributed, among other things, to these factors.

1) ROLE OF CIRCULAR ENTANGLEMENT

One additional factor, apart from the pure error rates of the quantum operations, is the resilience in the design of the circuit against the spread of errors. To investigate this, we compare the results from our previous circuit design with circular entanglement (see Fig. 4) with a design that uses only linear entanglement, as given in Fig. 8. The two circuits agree in all gates except that the circuit with linear entanglement lacks the last CNOT in the variational layer connecting the first and the last qubit.

In order to study the spread of errors, we use our second noise model without single-gate and readout errors, $p_1 = 0$ and $p_r = 0$, and 1% two qubit error rate $p_2 = 0.01$. We set $x = 1$ so that the exact values of f_{QC} are given by $f_{QC}(1) =$

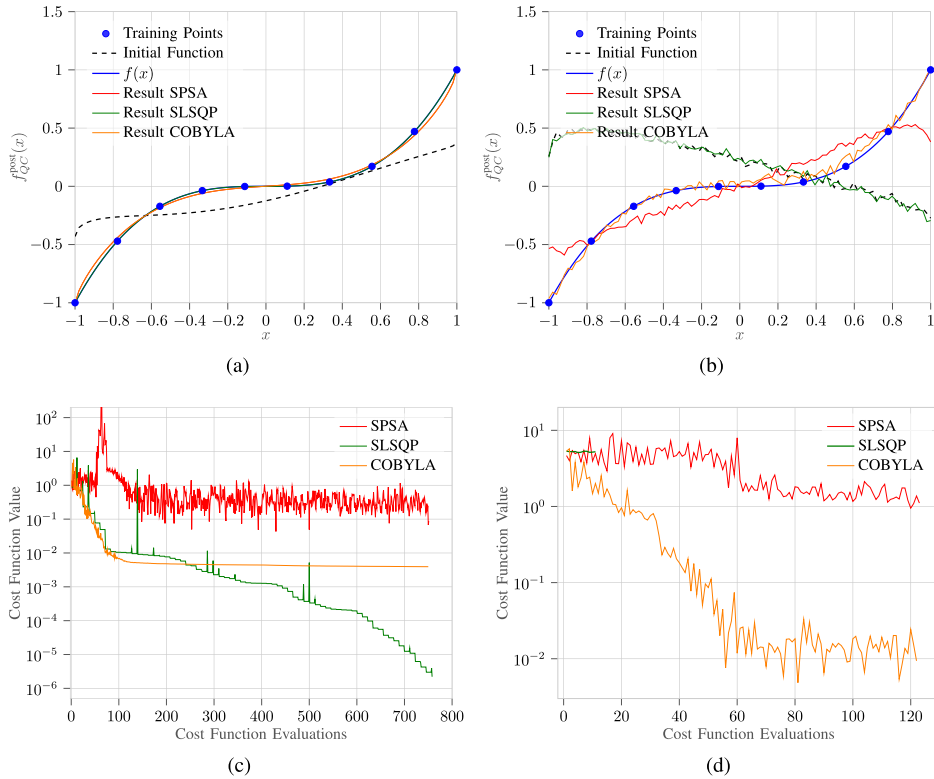


FIGURE 10. $f(x) = x^3$ learned using QCL circuits with a postprocessing parameter θ_{post} , $R_Y(\arcsin(x))$ data encoding, and a qubit number of $N = 3$ on (a) a statevector simulator and (b) a simulator with noise model 3 (derived from calibration data) and 2000 shots. The cost function is evaluated on ten equidistant training points and is classically minimized using different optimizers. The initial function (dashed black line) shows $f_{QC}^{post}(x)$ with the randomly chosen starting parameters before the optimization process and $\theta_{post} = 1$. In (c) and (d), the respective values of the cost function versus the number of cost function evaluations during the optimization process are plotted.

$(-1)^N$ and $f_{QC}(1) = 1$ for the case of circular and linear entanglement, respectively. Fig. 9 shows the errors for both circuit designs. We see that the linear entanglement design is resilient against the spread of errors, whereas in the circular entanglement design, we observe a linear increase of the error with the qubit number. This phenomenon arises from the additional pathways for error propagation that circular entanglement introduces. While both circular and linear entanglement schemes allow errors to propagate through the chain of qubits, circular entanglement creates an extra connection that increases the error measured at the first qubit. This is the same behavior that we observed in Fig. 7 for the MAE of our Pauli noise model. At this point, we emphasize that the more resilient design of the linear entanglement circuit comes with severe disadvantages in solving our original problem of learning functions and solving differential equations. As we explain in Appendix B, we need circular entanglement to be able to learn arbitrary functions of degree N . Therefore, in the following experiments, we will use circular entanglement but limit ourselves to a qubit number of $N = 3$.

C. COMPARISON OF CLASSICAL OPTIMIZERS

As a last step before implementing QCL for different examples, we want to investigate which classical optimizer is best suited to optimize the parameters in the case of exact statevector simulations and for noise model 3. To do this,

we use the circuit in Fig. 2 with $R_Y(\arcsin(x))$ data encoding and the cost function introduced in (8) to learn the function $f(x) = x^3$. The cost function is evaluated on ten equidistant training points and is classically minimized using the simultaneous perturbation stochastic approximation (SPSA) [24], sequential least squares programming (SLSQP) [25], and constrained optimization by linear approximation (COBYLA) [26]. The SPSA algorithm is run with the default Qiskit settings, while the SLSQP and COBYLA algorithms are executed with the default SciPy minimizer settings [27]. The final learned functions are plotted in Fig. 10(a) and (b). The initial function resulting from the randomly selected start parameters is also shown [black dashed lines in Fig. 10(a) and (b)]. The values of the cost function versus the number of cost function evaluations are shown in Fig. 10(c) and (d).

In the case of exact statevector simulations [see Fig. 10(a) and (c)], the SLSQP algorithm demonstrates the best performance in terms of final accuracy. While its convergence speed may be comparable to that of COBYLA, SLSQP achieves a notably higher precision in approximating the target function $f(x) = x^3$. This superior accuracy can be attributed to SLSQP's ability to use gradient information, which can be precisely estimated in noiseless simulations. However, the situation changes when considering shot noise and hardware noise [see Fig. 10(b) and (d)]. In this more realistic scenario, SLSQP no longer works effectively and

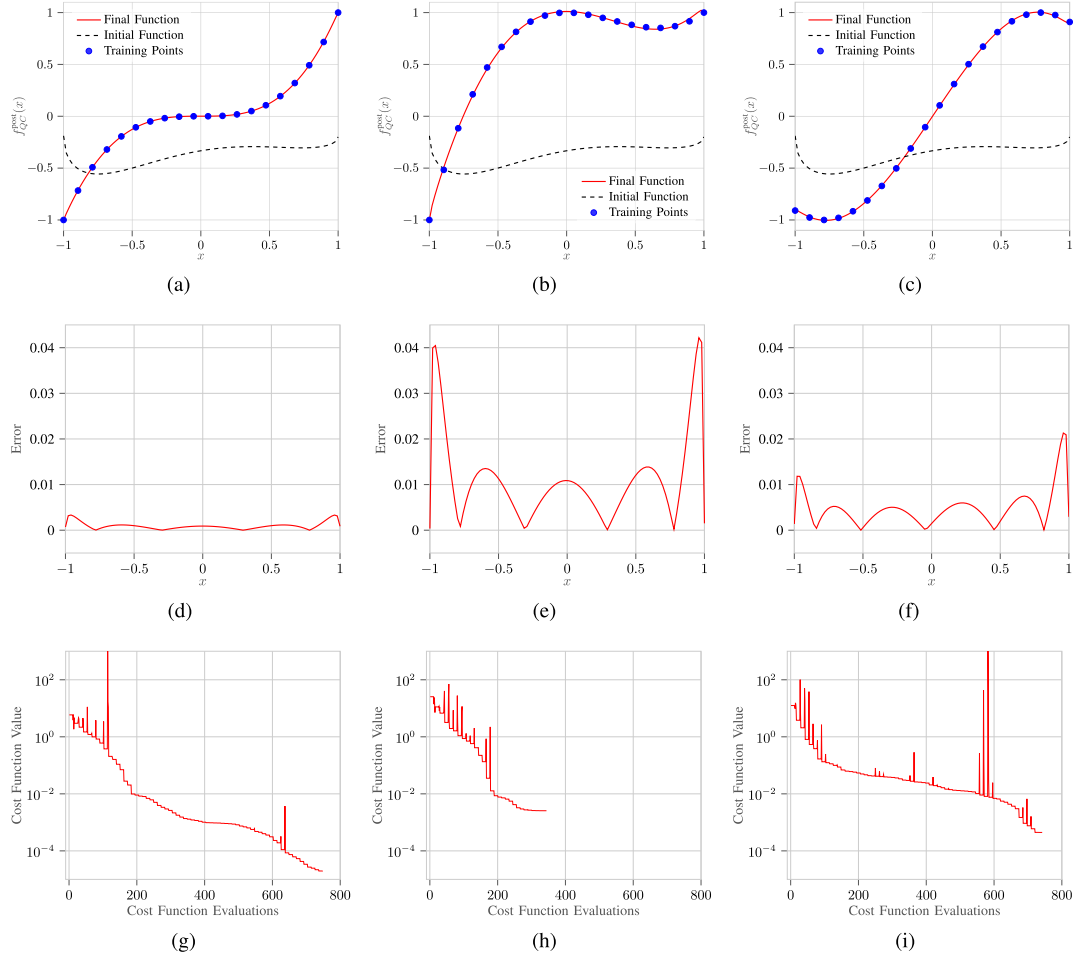


FIGURE 11. Three learned functions using QCL circuits on a statevector simulator with a postprocessing parameter θ_{post} , $R_Y(\arcsin(x))$ data encoding, a qubit number of $N = 3$, and a depth of $D = 3$. The learned functions are (a) $f_1(x) = x^3$, (b) $f_2(x) = x^3 - x^2 + 1$, and (c) $f_3(x) = \sin(2x)$. The cost function is evaluated on 20 equidistant training points and is classically minimized using SLSQP. The initial function (dashed black line) shows $f_{\text{QC}}(x)$ with the randomly chosen starting parameters for the optimization process and $\theta_{\text{post}} = 1$. In addition, in (d)–(f), the absolute values of the respective errors $|f(x) - f_{\text{QC}}^{\text{post}}(x)|$ are shown on a fine grid. In (g)–(i), the respective values of the cost function versus the number of cost function evaluations in the optimization process are plotted.

the optimization breaks. This is primarily due to the optimizer’s reliance on gradient calculations, which become very unreliable in the presence of noise. The stochastic nature of shot noise and the additional hardware errors introduce fluctuations that can mislead gradient-based optimizers. In contrast, the COBYLA optimizer is more effective under noisy conditions. COBYLA does not rely on gradient information, making it more robust to noise. Hence, COBYLA achieves a better approximation of the target function and maintains a stable convergence trajectory in the presence of noise. SPSA is a popular optimizer since it determines the gradients with fewer measurements and is, therefore, more stable in the presence of noise. In our example, however, it is outperformed by SLSQP in the noise-free case, while COBYLA proves to be the better choice in the noisy scenario.

V. FUNCTION LEARNING

In this section, we present the learning of different example functions with a simulator as well as on `ibmq_ehningen`.

We use the cost function in (8) and the circuit in Fig. 2 with $R_Y(\arcsin(x))$ data encoding and a depth of $D = 3$ as this depth has proven to be suitable for representing complicated functions without creating a circuit that is too deep.

A. SIMULATOR

We now learn the functions $f_1(x) = x^3$, $f_2(x) = x^3 - x^2 + 1$, and $f_3(x) = \sin(2x)$ on the interval $[-1, 1]$ with the help of an exact statevector simulator without shot noise. The cost function is evaluated on 20 equidistant training points and is classically minimized using SLSQP with the default settings of the SciPy minimizer. The final learned functions are plotted in Fig. 11(a)–(c) (red lines). The initial function resulting from the randomly selected start parameters at the beginning is also shown [black dashed lines in Fig. 11(a)–(c)]. We see that the target function can be approximated on a simulator. The next step is to reproduce these results on actual quantum hardware, in particular on `ibmq_ehningen`, before investigating the possibility of solving differential equations.

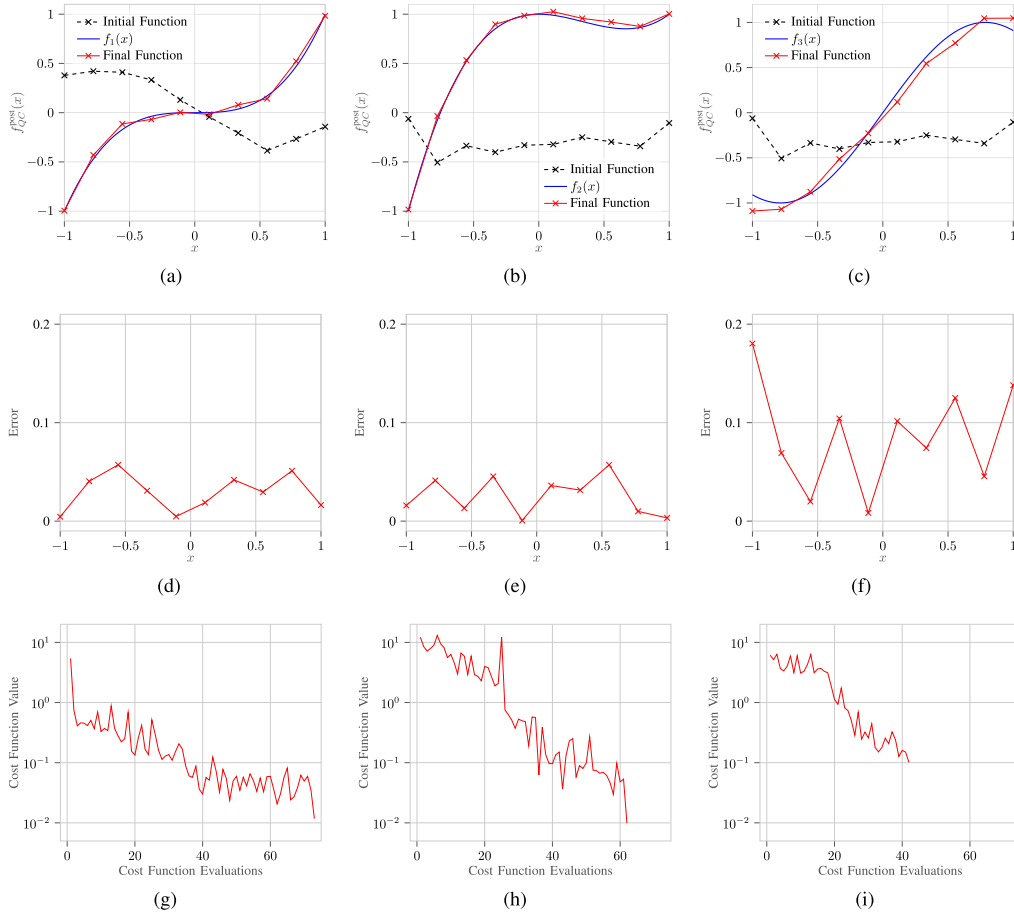


FIGURE 12. Three learned functions using QCL circuits on the `ibmq_ehningen` with a postprocessing parameter θ_{post} , $R_Y(\arcsin(x))$ data encoding and a qubit number of $N = 3$. The learned functions are (a) $f_1(x) = x^3$, (b) $f_2(x) = x^3 - x^2 + 1$, and (c) $f_3(x) = \sin(2x)$. The cost function is evaluated on ten equidistant training points with 2000 shots for every circuit and is classically minimized using COBYLA. The initial function (dashed black line) shows $f_{\text{QC}}^{\text{post}}(x)$ with the randomly chosen starting parameters for the optimization process and $\theta_{\text{post}} = 1$. In addition, in (d)–(f), the absolute values of the respective errors $|f(x) - f_{\text{QC}}^{\text{post}}(x)|$ are shown. In (g)–(i), the respective values of the cost function versus the number of cost function evaluations during the optimization process are plotted.

B. IBM QUANTUM COMPUTER

In the literature, QCL has been executed entirely on a simulator, or only the circuit with the final optimized parameters has been tested on a quantum computer [12]. In this work, the full algorithm is executed on a quantum computer. This means that every circuit evaluation during the optimization process is also performed on the quantum computer. The QCL circuits that have been simulated in Section V-A are now executed on `ibmq_ehningen`, which was introduced in Section III. As we analyzed in Section IV-C, due to the hardware and shot noise, a gradient-based optimization runs into problems, and the gradient-free COBYLA optimizer proves to be suitable. The same functions that have already been learned in Section V-A on a simulator are now learned on `ibmq_ehningen`. The cost function is evaluated on ten equidistant training points with 2000 shots for every circuit and is classically minimized using COBYLA [red markers in Fig. 12(a)–(c)]. In addition, the initial function resulting from the randomly selected start parameters at the beginning of the optimization and $\theta_{\text{post}} = 1$ is plotted [black dashed lines in Fig. 12(a)–(c)]. The functions can

be learned with good accuracy on the quantum computer, which becomes clear in the error diagrams [see Fig. 12(d)–(f)], where the absolute values of the respective errors $|f(x) - f_{\text{QC}}^{\text{post}}(x)|$ are plotted. It can be observed that the two polynomials ($f_1(x) = x^3$ and $f_2(x) = x^3 - x^2 + 1$) can be learned with higher accuracy than the trigonometric function ($f_3(x) = \sin(2x)$). This can be explained by the fact that the selected data encoding layer generates polynomial-like functions [see (4)], which are more suited to learn polynomial functions.

The values of the cost function versus the number of cost function evaluations [see Fig. 12(g)–(i)] show that the cost function is evaluated less than 100 times during the optimization process.

If we compare the results presented here with those in Section IV, we can see that the error is smaller than expected from the investigation with fixed parameters. We assume that this is due to the fact that systematic hardware errors are learned by the variational algorithm, and thus, the results are closer to the target function compared to the experiments without variational optimization.

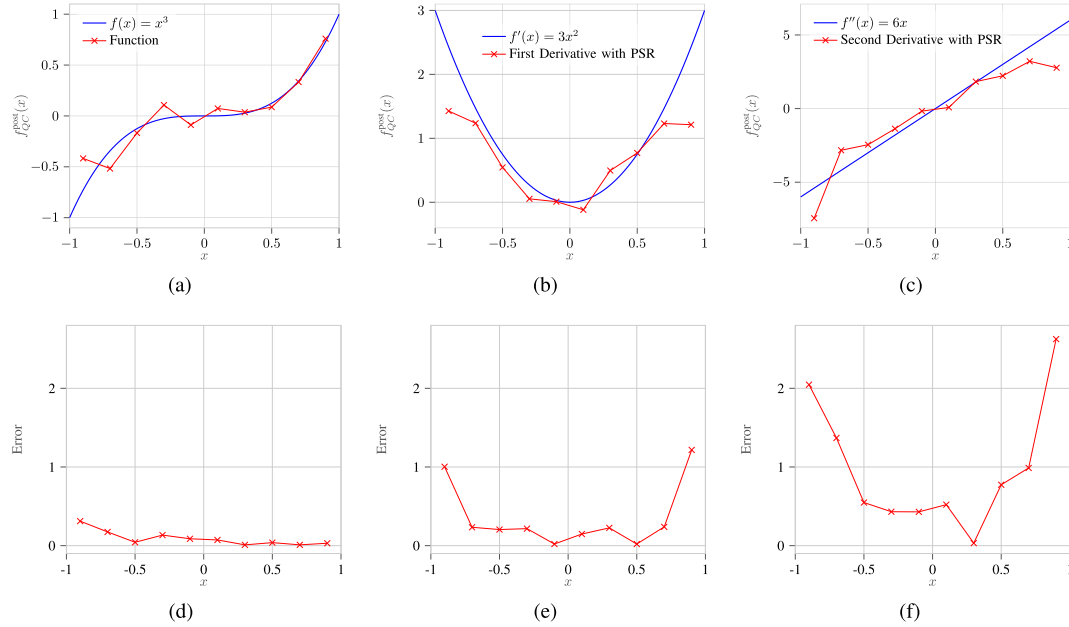


FIGURE 13. (a) Resulting function values from executing the QCL circuit with trained parameters for $f(x) = x^3$ (see Fig. 11) on `ibmq_ehningen` with ten equidistant training points. Values of (b) the first and (c) second derivatives obtained with the PSR applied to aforementioned circuit and executed on `ibmq_ehningen`. The shot number is chosen as 1024 in all cases. In (d)–(f), we provide the absolute value of the errors between the QCL model and the exact functions on the grid points.

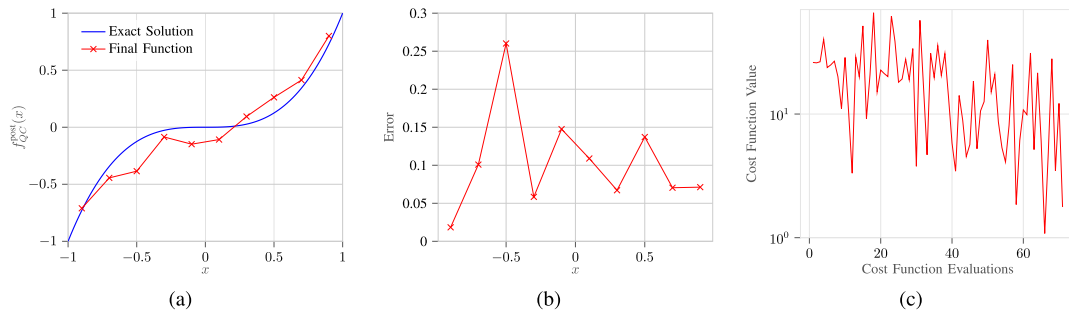


FIGURE 14. (a) Solution of the differential (12) using QCL circuits on `ibmq_ehningen` with $\mu = 10$, $N = 3$, $D = 3$, ten equidistant training points, and 2000 shots with $R_Y(\arcsin(x))$ data encoding. The parameters are optimized using COBYLA. (b) Absolute values of the respective errors $|f(x) - f_{QC}^{post}(x)|$. (c) Values of the cost function versus the number of cost function evaluations during the optimization process.

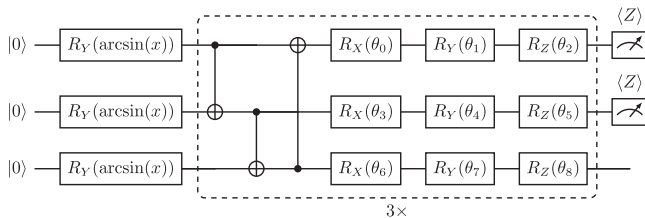


FIGURE 15. Three-qubit QCL circuit with $R_Y(\arcsin(x))$ data encoding. The Z expectation values of the first and second qubits are measured.

VI. DIFFERENTIAL EQUATIONS

This section investigates the possibility of solving differential equations with QCL circuits in combination with the PSR.

A. PARAMETER SHIFT RULE

The PSR is a method to determine the exact derivative of a parameterized quantum circuit [9], [11]. To obtain the first

derivative w.r.t. x of our QCL circuits, the PSR boils down to evaluating $2N$ expectation values. As the name suggests, in each of these expectation values, the variable x in one of the data encoding gates is shifted by $\pm\pi/2$. Also note that, due to the chain rule, the derivative of the inner function $\varphi'(x)$ also enters in the calculation. Higher derivatives are obtained similarly but require additional evaluations of the expectation value. For example, we need to execute $4N^2 - 2N$ QCL circuits for the second derivative.

To investigate the applicability of the PSR for QCL circuits on current quantum hardware, it is tested on `ibmq_ehningen`. For this purpose, the circuit and the optimized parameters from the learning of $f(x) = x^3$ in Fig. 11(a) are used, and the derivatives w.r.t. x are calculated with the PSR. Since the derivative of the inner function $\varphi(x) = \arcsin(x)$ is required, which diverges for $x = -1$ and $x = 1$, a value range of $x \in [-0.9, 0.9]$ is selected in this example. The results can be seen in Fig. 13.

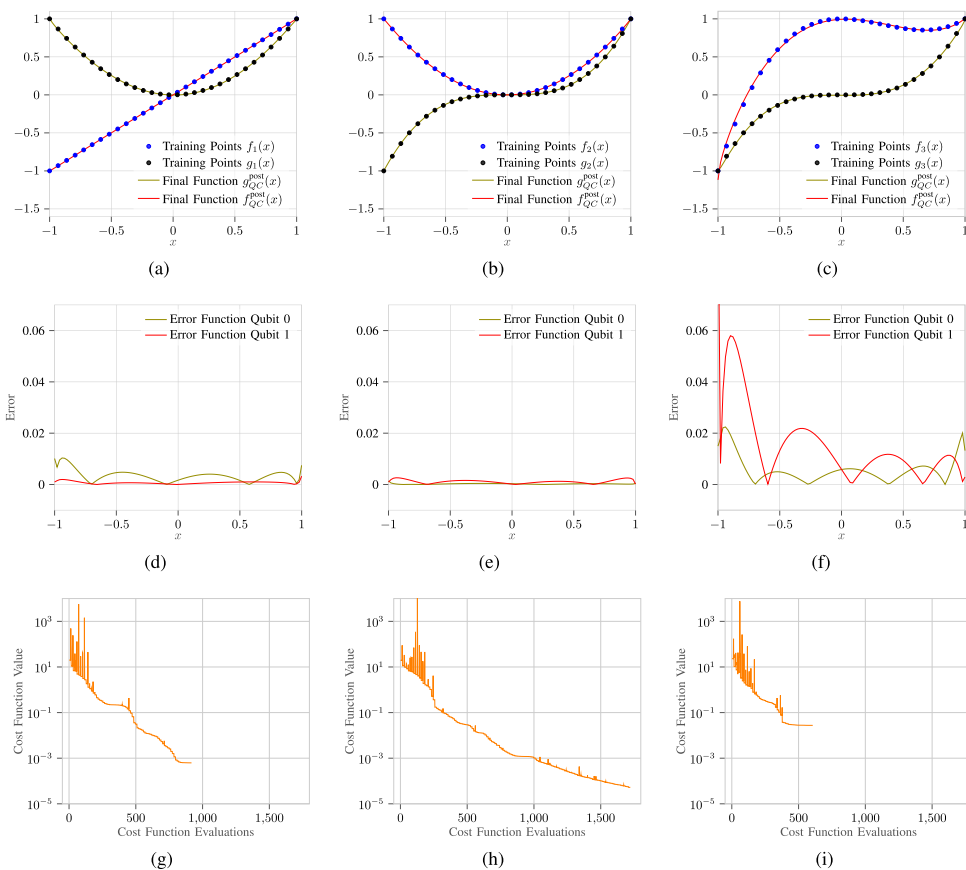


FIGURE 16. Learning two different functions per QCL circuit on a statevector simulator with $R_Y(\arcsin(x))$ data encoding, a qubit number of $N = 3$, and a depth of $D = 3$. The learned functions are (a) $f_1(x) = x$ and $g_1(x) = x^2$, (b) $f_2(x) = x^2$ and $g_2(x) = x^3$, and (c) $f_3(x) = x^3 - x^2 + 1$ and $g_3(x) = x^3$. The cost function is evaluated on 30 equidistant training points and is classically minimized using SLSQP. In addition, in (d)–(f), the absolute values of the respective errors are shown. In (g)–(i), the respective values of the cost function versus the number of cost function evaluations are plotted.

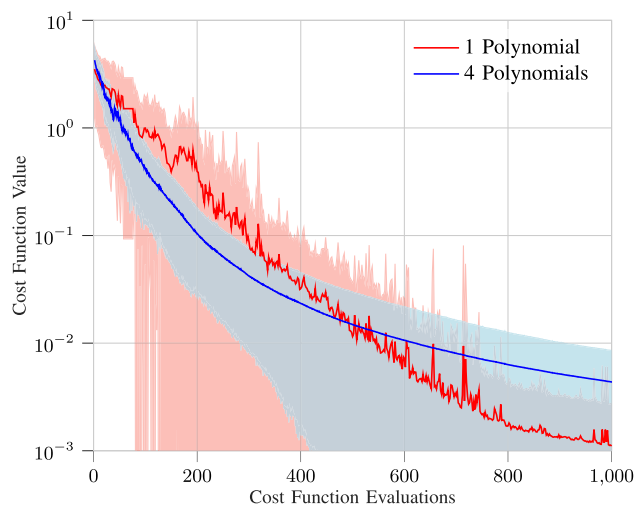


FIGURE 17. Average of the convergence curves of the training of 100 QCL circuits with $N = 6$ qubits and a depth of $D = 4$. For the red line, 100 random polynomials are learned, i.e., one polynomial per circuit. For the blue line, four polynomials are learned per circuit, which sums up to 400 random polynomials learned. We use ten equidistant training points and the classical optimizer COBYLA. For a fair comparison, the value of the cost function and the number of function evaluations is divided by the number of polynomials learned in parallel. The shaded areas indicate the respective standard deviations.

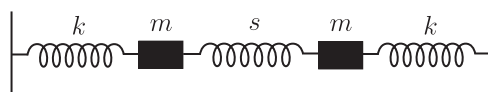


FIGURE 18. Example of the coupled harmonic oscillator with two identical masses m , two identical springs of spring strength k , and one spring of spring strength s .

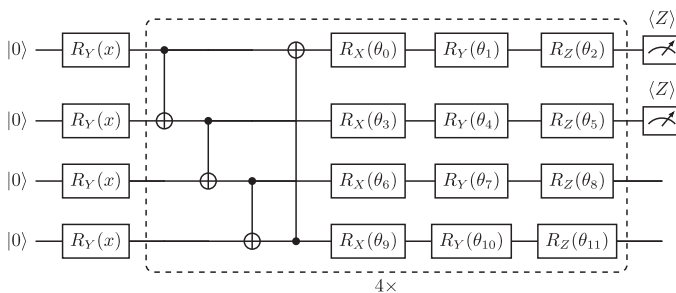


FIGURE 19. Four-qubit QCL circuit with $R_Y(x)$ data encoding. The first and second qubits are measured.

The qualitative behavior of the derivatives can be determined well. However, there are large errors, especially near $x = -1$ and $x = 1$, which go far beyond shot noise. The errors are mainly hardware errors that accumulate due to the high number of circuits that need to be evaluated.

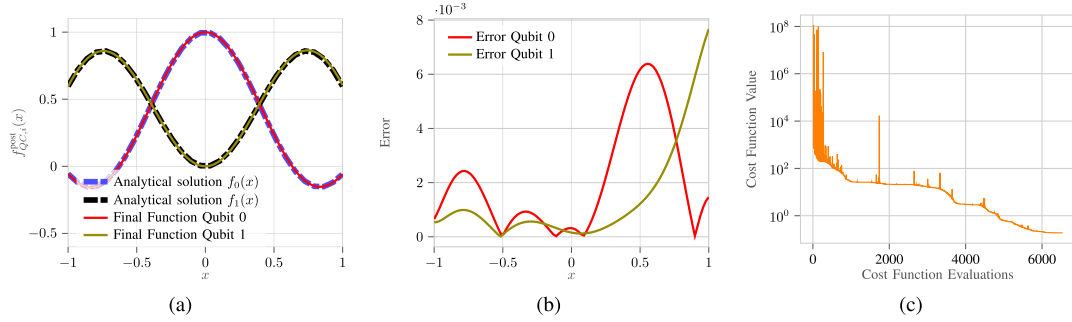


FIGURE 20. (a) Solution of the differential equation (19) on a statevector simulator with $\mathbf{f}_0 = (0, 0)$, $\mathbf{f}_1 = (1, 0)$, $\omega_0^2 = 1$, $\omega_1^2 = 16$, and $\mu = 20$. The result is obtained with the QCL circuit in Fig. 19 with $N = 4$, $D = 4$, and $R_Y(x)$ data encoding in combination with the PSR. The parameters are classically optimized using SLSQP. (b) Absolute values of the respective errors $|f_i(x) - f_{QC,i}^{\text{post}}(x)|$ are shown. (c) Values of the cost function versus the number of cost function evaluations.

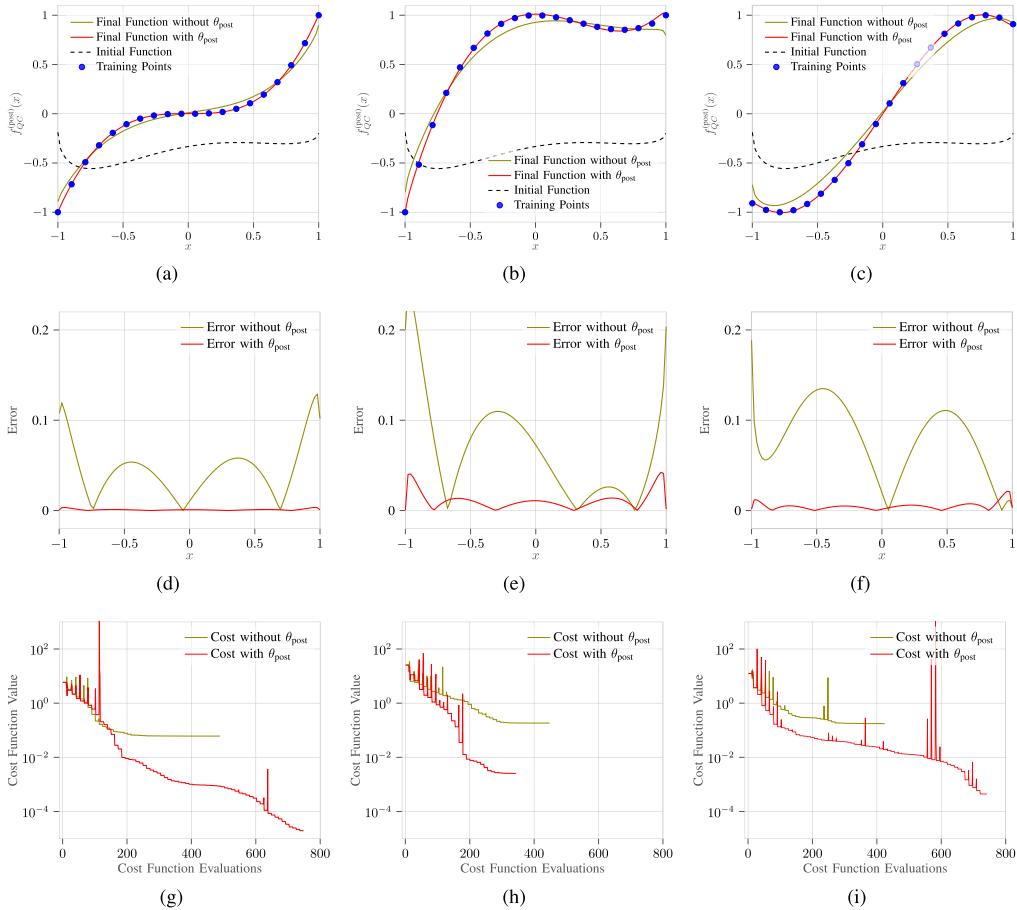


FIGURE 21. Comparative analysis of QCL performance with (red lines) and without (green lines) the postprocessing parameter θ_{post} for three functions: (a) $f_1(x) = x^3$, (b) $f_2(x) = x^3 - x^2 + 1$, and (c) $f_3(x) = \sin(2x)$. The dashed black lines show the initial functions with randomly chosen starting parameters and $\theta_{\text{post}} = 1$. (d)–(f) show the absolute errors on a fine grid. (g)–(i) display the cost function values versus the number of cost function evaluations during optimization.

In the next section, we focus on differential equations where these derivatives become crucial. Solving these differential equations on a real quantum computer is, therefore, associated with large errors.

B. DIFFERENTIAL EQUATION ON THE IBM QUANTUM COMPUTER

In this section, we aim to solve a differential equation with QCL circuits and the PSR on `ibmq_ehningen`. For

this purpose, a simple example of a differential equation is considered to limit the required quantum resources, namely

$$\begin{cases} f'(x) = 3x^2 \\ f(0) = 0. \end{cases} \quad (12)$$

The solution of this differential equation is

$$f(x) = x^3. \quad (13)$$

To be able to solve it with QCL circuits, we define the cost function

$$L(\theta) = \sum_i \left(\left| f'_{QC}{}^{\text{post}}(x_i, \theta) - 3x_i^2 \right|^2 + \mu \left| f_{QC}{}^{\text{post}}(0, \theta) - 0 \right|^2 \right) \quad (14)$$

where μ is a weight factor and the derivatives are determined with the PSR.

The result for a circuit with $N = 3$, $D = 3$, a weight factor $\mu = 10$, ten equidistant training points, and 2000 shots is shown in Fig. 14. The value range is chosen to be $x \in [-0.9, 0.9]$ because the $R_Y(\arcsin(x))$ encoding is used. It can be observed that the differential equation can be solved to a certain extent in the sense that the qualitative behavior can be reproduced approximately. The errors [see Fig. 14(d)–(f)] are higher than in the case of learned functions on `ibmq_ehningen` [compare Fig. 12(d)–(f)]. This is because the derivatives are included in the cost function, which results in significantly more circuit evaluations and the errors accumulate.

VII. MULTIQUBIT MEASUREMENTS

So far, we have only measured a single qubit in all our QCL circuits. More specifically, we have used the Z expectation value of the first qubit to learn different functions. However, we found that it is possible to use the expectation values of different qubits to learn different functions simultaneously with the same circuit. In this section, this idea will be investigated in more detail. For this purpose, simulations are carried out where the expectation value of the first qubit and the expectation value of the second qubit are used to learn two different functions simultaneously. The quantum model function of the first qubit is $f_{QC}{}^{\text{post}}(x, \theta)$, as before. The quantum model function of the second qubit is

$$g_{QC}{}^{\text{post}}(x, \theta) = \langle Z_1 \rangle(x, \theta) \cdot \tilde{\theta}_{\text{post}} \quad (15)$$

where $\langle Z_1 \rangle(x, \theta)$ is the Z expectation value of the second qubit and $\tilde{\theta}_{\text{post}}$ is a separate postprocessing parameter. We define the cost function

$$L(\theta) = \sum_i \left(|f(x_i) - f_{QC}{}^{\text{post}}(x_i, \theta)|^2 + |g(x_i) - g_{QC}{}^{\text{post}}(x_i, \theta)|^2 \right) \quad (16)$$

where $f(x)$ and $g(x)$ can be two different functions.

Three pairs of exemplary functions ($f_1(x) = x$ and $g_1(x) = x^2$; $f_2(x) = x^2$ and $g_2(x) = x^3$; $f_3(x) = x^3 - x^2 + 1$ and $g_3(x) = x^3$) are now learned with a three-qubit QCL circuit with a depth of $D = 3$ and $R_Y(\arcsin(x))$ data encoding (see Fig. 15) on a statevector simulator. The cost function is evaluated on 20 equidistant training points and is classically minimized using SLSQP [see Fig. 16(a)–(c)]. In Fig. 16(d)–(f), the absolute values of the respective errors $|f_i(x) - f_{QC,i}{}^{\text{post}}(x, \theta)|$ are plotted. These errors are higher than in the previous example in Fig. 11, in which only one function was learned.

To investigate whether it can be advantageous to use multiple qubits for different functions, a six-qubit QCL circuit

with a depth of $D = 4$ is used to learn polynomials of the form

$$f(x) = \sum_{i=0}^6 c_i x^i \quad (17)$$

where the coefficients c_i are randomly chosen with

$$\sum_{i=0}^6 c_i^2 \leq 1. \quad (18)$$

These polynomials are first learned individually by measuring only the first qubit. This is done for 100 random polynomials, and the average of the resulting convergence curves of the cost function is shown in Fig. 17 (red line). In addition, four different random polynomials are simultaneously learned by measuring four different qubits. This is also repeated 100 times with four different polynomials (400 polynomials in total), and the average of the resulting convergence curves is plotted. In order to adequately compare the convergence curves for the two cases, the cost function values and the number of function evaluations are divided by 4 for the multifunction case (blue line in Fig. 17). In order to have the necessary parameters to learn multiple functions, unlike in the rest of this article, no parameters are repeated here with increasing depth, but new parameters are introduced.

In the beginning of the optimization, the functions can be learned faster if multiple qubits of the same circuit are used. However, the maximum accuracy that can be achieved with just one function is higher. Therefore, the approach presented here could be valuable when aiming to quickly learn the qualitative behavior of several functions. We only considered one example, and there could be further advantages when approximating a higher number of functions.

A. COUPLED HARMONIC OSCILLATOR

In this section, we want to solve a coupled differential equation using a single QCL circuit in combination with the PSR on a simulator. Coupled differential equations were solved with QCL circuits before, using a designated circuit for each equation [13]. We combine this problem with the finding that multiple functions can be learned with a single circuit.

For this purpose, a coupled harmonic oscillator with two masses m , two springs of spring strength k , and one spring of spring strength s is considered. The arrangement can be seen in Fig. 18.

This system is described by the coupled differential equation

$$\mathbf{f}''(x) = \mathbf{S}\mathbf{f}(x) \quad \mathbf{f}'(0) = \mathbf{f}'_0 \quad \mathbf{f}(0) = \mathbf{f}_0 \quad (19)$$

where the variable x describes the time, $\mathbf{f}(x) = (f_0(x), f_1(x))$ collects the displacements of the two masses, \mathbf{S} is the stiffness matrix scaled by $1/m$,

$$\mathbf{S} = \frac{1}{m} \begin{pmatrix} -k - s & s \\ s & -k - s \end{pmatrix} \quad (20)$$

and \mathbf{f}'_0 and \mathbf{f}_0 are the initial velocity and initial displacement of the masses, respectively. For this article, we choose the initial conditions as $\mathbf{f}'_0 = (0, 0)$ and $\mathbf{f}_0 = (1, 0)$. Then, the solution to (19) is given by

$$\mathbf{f}(x) = \frac{1}{2} \begin{pmatrix} \cos(\omega_0 x) - \cos(\omega_1 x) \\ \cos(\omega_0 x) + \cos(\omega_1 x) \end{pmatrix} \quad (21)$$

with frequencies $\omega_0^2 = k/m$ and $\omega_1^2 = k/m + 2s/m$. The differential equation is now solved with the four-qubit QCL circuit shown in Fig. 19, where the first and second qubits are measured.

We chose the $R_Y(x)$ data encoding to obtain trigonometric functions, which are the appropriate choice to capture the periodic behavior expected from an undamped system of oscillators. To solve the differential equation with the QCL circuit in Fig. 19, the cost function

$$L(\theta) = \sum_i \left(\|\mathbf{f}'_{\text{QC}}(x_i, \theta) - \mathbf{Sf}_{\text{QC}}(x_i, \theta)\|^2 + \mu \|\mathbf{f}'_{\text{QC}}(0, \theta) - \mathbf{f}'_0\|^2 + \mu \|\mathbf{f}_{\text{QC}}(0, \theta) - \mathbf{f}_0\|^2 \right) \quad (22)$$

is defined, where μ is a problem-specific weight factor and

$$\mathbf{f}_{\text{QC}}(x, \theta) = \left(f_{\text{QC},0}^{\text{post}}(x, \theta), f_{\text{QC},1}^{\text{post}}(x, \theta) \right). \quad (23)$$

The derivatives in the cost function are calculated with the PSR. The result of the simulation for $\omega_0^2 = 1$, $\omega_1^2 = 16$, and 30 equidistant training points is shown in Fig. 20. It is possible to solve a coupled differential equation with just one circuit with low errors [see Fig. 20(b)]. This method could bring considerable advantages, in particular for very complicated systems with even more coupled equations, since only one circuit with the corresponding parameters has to be optimized, and not a set of parameters for each equation.

VIII. CONCLUSION

In this work, we have conducted an investigation of the QCL framework and its executability on NISQ devices. In the beginning, simulations with different noise models and hardware experiments on `ibmq_ehningen` were performed to investigate the scalability of QCL circuits on NISQ devices. It was discovered that these circuits can be executed on NISQ hardware, but only a fraction of the available qubits of `ibmq_ehningen` can be effectively used before the errors become too high. Following this, several functions were successfully learned with three-qubit QCL circuits on a simulator and subsequently also on `ibmq_ehningen`. Next, the ability to solve differential equations with QCL circuits and the PSR was investigated. First, the PSR was tested on the IBM quantum computer to investigate the NISQ applicability. It was possible to determine the derivative of functions. However, the resulting errors were very high. Nevertheless, a simple differential equation could be solved to a certain extent on `ibmq_ehningen`.

Furthermore, it was shown that multiple functions can be learned with a single QCL circuit when multiple qubits are measured, which results in a faster learning in the early stages of the optimization process for some examples. Following this idea, the differential equation of a coupled harmonic oscillator was solved with only a single circuit on a simulator.

Overall, it should be mentioned that the optimization process on the classical computer can be computationally intensive and takes up a large part of the computing time.

This problem applies not only to the QCL framework but to most variational quantum algorithms. Therefore, a lot of research focuses on this problem, for example in the expansion and development of more suitable classical optimizers [28], [29]. Such methods give hope for a reduction of computation time and make the algorithm even more relevant.

APPENDIX A A IMPACT OF POSTPROCESSING PARAMETER

In this appendix, we demonstrate the crucial role of the postprocessing parameter θ_{post} in improving the accuracy of function learning using QCL. We present a comparison of QCL performance with and without this parameter for three example functions. We use the circuit depicted in Fig. 2 with a depth of $D = 3$, $R_Y(\arcsin(x))$ data encoding, and a qubit number of $N = 3$. We use a statevector simulator without shot noise, evaluate the cost function on 20 equidistant training points, and minimize the cost function using SLSQP with default SciPy minimizer settings. Fig. 21 shows the results for $f_1(x) = x^3$, $f_2(x) = x^3 - x^2 + 1$, and $f_3(x) = \sin(2x)$. The results clearly demonstrate the limitations of QCL without the postprocessing parameter. While the qualitative behavior of the target functions is captured, significant deviations are observed without θ_{post} [green lines in Fig. 21(d)–(f)]. The absolute errors $|f(x) - f_{\text{QC}}(x)|$ are substantially higher without θ_{post} [green lines in Fig. 21(a)–(c)]. Without θ_{post} , the optimization process progresses more slowly and stagnates earlier.

In conclusion, the inclusion of θ_{post} leads to more accurate function approximations, significantly reduced errors, faster convergence, and lower final cost values. It also extends the value range to $f_{\text{QC}}^{\text{post}}(x, \theta) \in \mathbb{R}$ for $\theta_{\text{post}} \in \mathbb{R}$.

APPENDIX B NECESSITY OF CIRCULAR ENTANGLEMENT

In this appendix, we demonstrate the importance of circular entanglement in our QCL approach, despite its potential drawbacks in terms of hardware efficiency and error resilience, as discussed in Section IV-B1. Fig. 22 compares the performance of QCL circuits with circular and linear entanglement on a simulator that incorporates hardware noise (noise model 3 from Section IV). The function $f(x) = x^3$ is learned using both entanglement strategies with $R_Y(\arcsin(x))$ data encoding and a qubit number of $N = 3$. The cost function is evaluated on

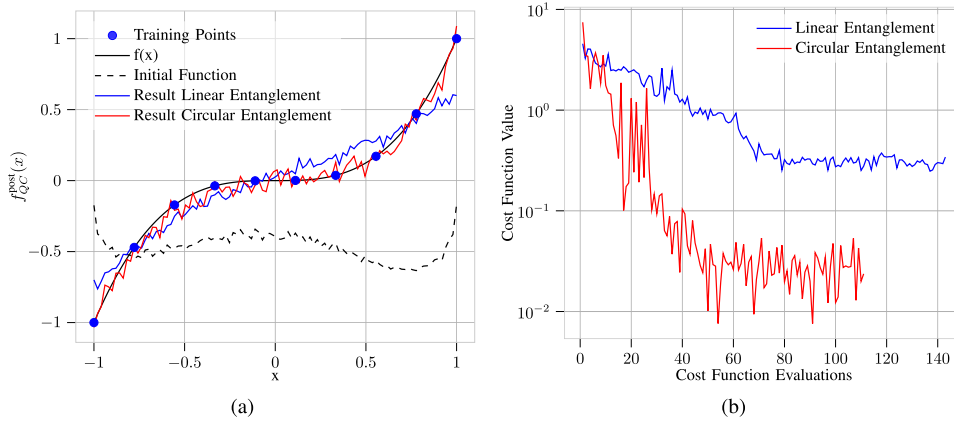


FIGURE 22. $f(x) = x^3$ learned using QCL circuits on a simulator with noise model 3 with 2000 shots, $R_Y(\arcsin(x))$ data encoding, and a qubit number of $N = 3$. The cost function is evaluated on ten equidistant training points and is classically minimized using COBYLA. The results for QCL with circular entanglement and linear entanglement are compared (a). The initial function (dashed black line) shows $f_{QC}^{post}(x)$ with the randomly chosen starting parameters before the optimization process and $\theta_{post} = 1$. In (b), the respective values of the cost functions versus the number of cost function evaluations during the optimization process are plotted.

TABLE 1. X Errors of `ibmq_ehningen` From 30 January 2024, 8:42:06+01:00 in Descending Order

Qubit	X error
19	0.00013591302669571659
16	0.00014569737631580065
24	0.0001651366443341561
6	0.00016680730700449124
3	0.00019475108276179376
25	0.00019511898972169597
12	0.0001977410765303121
23	0.0002005349223274833
13	0.00020272128722526137
9	0.0002079151356677378
26	0.0002107794975931981
0	0.00021184286817880178
1	0.0002263078174650456
15	0.00023837868227682494
11	0.0002477045424814508
22	0.0002549839225068226
18	0.00027892545750661713
21	0.0002979112936644023
5	0.00031537329438427003
2	0.00033257814929062255
4	0.0003574360500279698
20	0.0003953408174393563
14	0.0004337410173584769
8	0.0004944174096551563
17	0.0005776839080296207
10	0.001741461094155414
7	0.004221992895832456
median	0.00023837868227682494

TABLE 2. CNOT Errors of `ibmq_ehningen` From 30 January 2024, 8:42:06+01:00 in Descending Order

Qubit pair	CNOT error
(23, 24)	0.003766111620421869
(22, 25)	0.005100573458733021
(1, 4)	0.005725284943773834
(18, 21)	0.0059052091403848095
(19, 22)	0.006018110357534884
(0, 1)	0.006244572705932261
(14, 16)	0.006699047143990389
(16, 19)	0.006816483127679324
(2, 3)	0.00686035596001125
(19, 20)	0.006865771449287628
(8, 9)	0.007161408838511407
(21, 23)	0.007217990657473805
(5, 8)	0.007345102646509005
(1, 2)	0.007464918811527332
(3, 5)	0.0074783630145963675
(12, 15)	0.007548664291686519
(24, 25)	0.008734138864550017
(15, 18)	0.009328357231174117
(13, 14)	0.009752602982268627
(11, 14)	0.010083426759830982
(25, 26)	0.01018025088966229
(8, 11)	0.01066070025821142
(17, 18)	0.013317040648282985
(6, 7)	0.01607606156008279
(10, 12)	0.017207769362116293
(12, 13)	0.021712709764238475
(7, 10)	0.043634415742329125
(4, 7)	0.06896314743392529
median	0.00747164091306185

ten equidistant training points and optimized using the COBYLA algorithm. Fig. 22(a) clearly shows that the circuit with circular entanglement (red line) achieves a significantly better approximation of the target function compared to the circuit with linear entanglement (blue line). Circular entanglement allows the QCL circuit to learn the cubic nature of the function more accurately. The convergence plot in Fig. 22(b) further supports this observation. The linear entanglement circuit (blue line) struggles to reduce the cost function beyond a certain point, suggesting a limitation in its expressivity. These results demonstrate that, despite the advantages of linear entanglement in terms of hardware

efficiency and error resilience (see Section IV-B1), circular entanglement is necessary to achieve the required expressivity in our experiments.

APPENDIX C CALIBRATION DATA OF `IBMQ_EHNINGEN`

This appendix presents the calibration data of `ibmq_ehningen` from 30 January 2024, 8:42:06+01:00, which forms the basis for the noise models used in our simulations. The individual x, CNOT, and readout errors are given in Tables 1–3. The median T1 and T2 times are 113 and 96 μ s, respectively. It is noteworthy that

TABLE 3. Readout Errors of Ibmq_ehningen From 30 January 2024, 8:42:06+01:00 in Descending Order

Qubit	readout error
21	0.00669999999999928
13	0.00730000000000084
14	0.00739999999999962
16	0.00770000000000004
23	0.00770000000000004
24	0.00869999999999993
15	0.00900000000000008
4	0.00940000000000075
9	0.00940000000000075
25	0.00970000000000042
19	0.01019999999999987
1	0.01049999999999954
2	0.01119999999999988
0	0.01150000000000066
12	0.01259999999999945
18	0.01280000000000034
26	0.01339999999999967
6	0.01380000000000034
8	0.01400000000000012
10	0.01530000000000091
5	0.01669999999999937
22	0.01760000000000006
20	0.01819999999999994
7	0.02000000000000018
3	0.02049999999999963
11	0.0232
17	0.02859999999999996
median	0.01150000000000066

there is significant variation in error rates across different qubits and qubit pairs. For example, the X gate errors range from 0.00014 to 0.00422, while the $CNOT$ gate errors range from 0.00377 to 0.06896. This high variability underscores the importance of using a detailed noise model for accurate simulations.

ACKNOWLEDGMENT

The authors thank Vamshi Katukuri for insightful discussions and carefully proofreading this article. The author Niclas Schillo also thanks Sungkun Hong for his advice and dedicated supervision of his master’s thesis, which built the starting point of this article.

REFERENCES

[1] D. W. Berry, “High-order quantum algorithm for solving linear differential equations,” *J. Phys. A: Math. Theor.*, vol. 47, no. 10, 2014, Art. no. 105301, doi: 10.1088/1751-8113/47/10/105301.

[2] A. Montanaro and S. Pallister, “Quantum algorithms and the finite element method,” *Phys. Rev. A*, vol. 93, no. 3, 2016, Art. no. 032324, doi: 10.1103/PhysRevA.93.032324.

[3] D. W. Berry, A. M. Childs, A. Ostrander, and G. Wang, “Quantum algorithm for linear differential equations with exponentially improved dependence on precision,” *Commun. Math. Phys.*, vol. 356, no. 3, pp. 1057–1081, 2017, doi: 10.1007/s00220-017-3002-y.

[4] F. Gaitan, “Finding flows of a Navier–Stokes fluid through quantum computing,” *npj Quantum Inf.*, vol. 6, no. 1, 2020, Art. no. 2, doi: 10.1038/s41534-020-00291-0.

[5] A. Y. Kitaev, “Quantum measurements and the Abelian stabilizer problem,” *Electron. Colloq. Comput. Complex.*, vol. TR96-003, 1996. [Online]. Available: <https://eccc.weizmann.ac.il/eccc-reports/1996/TR96-003/index.html>

[6] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, “Quantum machine learning,” *Nature*, vol. 549, no. 7671, pp. 195–202, 2017, doi: 10.1038/nature23474.

[7] S. Lloyd et al., “Quantum algorithm for nonlinear differential equations,” 2020, *arXiv:2011.06571*, doi: 10.48550/arXiv.2011.06571.

[8] J. Preskill, “Quantum computing in the NISQ era and beyond,” *Quantum*, vol. 2, 2018, Art. no. 79, doi: 10.22331/q-2018-08-06-79.

[9] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, “Quantum circuit learning,” *Phys. Rev. A*, vol. 98, no. 3, 2018, Art. no. 032309, doi: 10.1103/PhysRevA.98.032309.

[10] M. Schuld and N. Killoran, “Quantum machine learning in feature Hilbert spaces,” *Phys. Rev. Lett.*, vol. 122, 2018, Art. no. 040504, doi: 10.48550/arXiv.1803.07128.

[11] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, “Evaluating analytic gradients on quantum hardware,” *Phys. Rev. A*, vol. 99, 2019, Art. no. 032331, doi: 10.48550/arXiv.1811.11184.

[12] K. Hatakeyama-Sato, Y. Igarashi, T. Kashikawa, K. Kimura, and K. Oyazui, “Quantum circuit learning as a potential algorithm to predict experimental chemical properties,” *Digit. Discov.*, vol. 2, no. 1, pp. 165–176, 2023, doi: 10.1039/D2DD00090C.

[13] O. Kyriienko, A. E. Paine, and V. E. Elfving, “Solving nonlinear differential equations with differentiable quantum circuits,” *Phys. Rev. A*, vol. 103, no. 5, 2021, Art. no. 052416, doi: 10.48550/arXiv.2011.10395.

[14] N. Heim, A. Ghosh, O. Kyriienko, and V. E. Elfving, “Quantum model-discovery,” 2021, *arXiv:2111.06376*, doi: 10.48550/arXiv.2111.06376.

[15] M. Knudsen and C. B. Mendl, “Solving differential equations via continuous-variable quantum computers,” 2020, *arXiv:2012.12220*, doi: 10.48550/arXiv.2012.12220.

[16] A. E. Paine, V. E. Elfving, and O. Kyriienko, “Quantum quantile mechanics: Solving stochastic differential equations for generating time-series,” *Adv. Quantum Technol.*, vol. 6, no. 10, 2023, Art. no. 2300065, doi: 10.1002/quete.202300065.

[17] N. Schillo, “Quantum algorithms and quantum machine learning for differential equations,” master’s thesis, Faculty Math. Phys., Univ. Stuttgart, Stuttgart, Germany, 2023, doi: 10.18419/opus-13866.

[18] IBM, “Qiskit transpiler.” Accessed: Dec. 13, 2024. [Online]. Available: <https://docs.quantum.ibm.com/api/qiskit/compiler#transpile>

[19] E. van den Berg, Z. K. Mineev, and K. Temme, “Model-free readout-error mitigation for quantum expectation values,” *Phys. Rev. A*, vol. 105, Mar. 2022, Art. no. 032620, doi: 10.1103/PhysRevA.105.032620.

[20] E. Knill, “Quantum computing with realistically noisy devices,” *Nature*, vol. 434, no. 7029, pp. 39–44, 2005, doi: 10.1038/nature03350.

[21] IBM, “Qiskit Aer noise model.” Accessed: Dec. 13, 2024. [Online]. Available: https://qiskit.github.io/qiskit-aer/stubs/qiskit_aer.noise.NoiseModel.html#qiskit_aer.noise.NoiseModel.from_backend_properties

[22] A. Ketterer and T. Wellens, “Characterizing crosstalk of superconducting transmon processors,” *Phys. Rev. Appl.*, vol. 20, 2023, Art. no. 034065, doi: 10.1103/PhysRevApplied.20.034065.

[23] A. Sturm, B. Mummaneni, and L. Rullkötter, “Unlocking quantum optimization: A use case study on NISQ systems,” 2024, *arXiv:2404.07171*, doi: 10.48550/arXiv.2404.07171.

[24] J. C. Spall, “An overview of the simultaneous perturbation method for efficient optimization,” *Johns Hopkins Apl Tech. Dig.*, vol. 19, pp. 482–492, 1998.

[25] D. Kraft, *A Software Package for Sequential Quadratic Programming* (ser. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt. Köln, Germany: Wiss. Berichtswesen d. DFVLR, 1988.

[26] M. J. D. Powell, “A direct search optimization method that models the objective and constraint functions by linear interpolation,” in *Advances in Optimization and Numerical Analysis*, S. Gomez and J.-P. Hennart, Eds. Dordrecht, The Netherlands: Springer, 1994, doi: 10.1007/978-94-015-8330-5_4.

[27] P. Virtanen et al., “SciPy 1.0: Fundamental algorithms for scientific computing in Python,” *Nat. Methods*, vol. 17, no. 3, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.

[28] S. Tamiya and H. Yamasaki, “Stochastic gradient line Bayesian optimization for efficient noise-robust optimization of parameterized quantum circuits,” *npj Quantum Inf.*, vol. 8, no. 1, 2022, Art. no. 90, doi: 10.1038/s41534-022-00592-6.

[29] M. Wiedmann et al., “An empirical comparison of optimizers for quantum machine learning with SPSA-based gradients,” in *Proc. 2023 IEEE Int. Conf. Quantum Comput. Eng.*, Bellevue, WA, USA, 2023, pp. 450–456, doi: 10.1109/QCE57702.2023.00058.