

Quantum Circuit Learning on NISQ Hardware

Niclas Schillo and Andreas Sturm

Fraunhofer IAO, Fraunhofer-Institut für Arbeitswirtschaft und Organisation IAO, Nobelstraße 12, 70569 Stuttgart, Germany

Current quantum computers are small and error-prone systems for which the term noisy intermediate-scale quantum (NISQ) has become established. Since large scale, fault-tolerant quantum computers are not expected to be available in the near future, the task of finding NISQ suitable algorithms has received a lot of attention in recent years. 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 computers. An important class of variational quantum algorithms is the quantum circuit learning (QCL) framework. Consisting of a data encoding and a trainable, parametrized layer, these schemes implement a quantum model function that can be fitted to the problem at hand. For instance, in combination with the parameter shift rule to compute derivatives, they can be used to solve differential equations. QCL and related algorithms have been widely studied in the literature. However, numerical experiments are usually limited to simulators and results from real quantum computers are scarce. In this paper we close this gap by executing QCL circuits on a superconducting IBM quantum processor in conjunction with an analysis of the hardware errors. We show that exemplary QCL circuits with up to three qubits are executable on the IBM quantum computer. For this purpose, multiple functions are learned and an exemplary differential equation is solved on the quantum computer. Moreover, we present how the QCL framework can be used to learn

different quantum model functions in parallel, which can be applied to solve coupled differential equations in an efficient way.

1 Introduction

The current noisy intermediate scale quantum (NISQ) [1] era is characterized by quantum computers that are error-prone and limited in size. Since it is not expected that this era will be overcome in near future, quantum algorithms that can cope with the shortcomings of NISQ computers are of central interest. Starting with the variational quantum eigensolver (VQE) [2] variational quantum algorithms [3] have emerged as the most popular field of NISQ friendly algorithms. Promising applications include, among others, the finding of ground states with VQE, the solution of combinatorial optimization problems with the quantum approximate optimization algorithm (QAOA) [4] and machine learning tasks like image recognition [5, 6].

In this work, we consider variational quantum algorithms based on the quantum circuit learning (QCL) framework [7]. Quantum circuit learning is a loosely defined term that is used differently in the literature. In our consideration, QCL circuits are multi-qubit circuits that have one data encoding layer at the beginning where a variable is encoded into the quantum state using quantum feature map encoding [8]. This is followed by a variational layer consisting of parameterized quantum gates. Finally, an expectation value is measured. In this way, we obtain a parametrized function of the encoded variable. The types of functions that can be achieved with this method depend on the data encoding layer and the number of qubits.

One application of these circuits is to learn one dimensional functions by training the parameters with a classical optimizer. Building upon this idea, it is possible to use QCL in combination

Niclas Schillo: niclas.schillo@iao.fraunhofer.de
 Andreas Sturm: andreas.sturm@iao.fraunhofer.de

with the parameter shift rule to solve differential equations. The parameter shift rule is an approach to obtain gradients of a parameterized quantum circuit [7, 9].

The previously described circuits have already been extensively studied in the literature and their effectiveness has been demonstrated by classical simulations [7, 10]. Using these circuits to solve differential equations has also been analysed and classically simulated [11–14].

In this work, we build on these results and focus on the executability of QCL circuits on current quantum computers and investigate the resulting hardware errors. It is shown that different functions can be learned at a good quality with a three qubit QCL circuit on an superconducting IBM quantum computer with under one hundred optimization steps. However, if we increase the number of qubits the hardware errors increase significantly and an execution becomes infeasible. In addition, we successfully solve a differential equation with QCL circuits and the parameter shift rule on the IBM quantum computer. We observe that, due to the parameter shift rule, the errors are significantly higher than when learning functions directly. Furthermore, we present methods to use the multi-qubit character of the QCL circuit to efficiently learn multiple functions simultaneously or to solve coupled differential equations.

This work is organized as follows: Section 2 gives a brief introduction to the QCL circuits used in this work. Following this, Section 3 delves deeper into the method of function learning with QCL. We introduce numerical experiments and results from a statevector simulation. Subsequently, in Section 4, we execute the same experiments on the IBM quantum computer. In Section 5, the occurring hardware errors are analysed in detail. In addition, Section 6 examines whether it is possible to learn several functions with a single QCL circuit by measuring multiple qubits. It is also discussed if this can lead to an advantage in the optimization process. Finally, the possibility of solving differential equations in combination with the parameter shift rule is explored. In order to investigate how NISQ-friendly it is to solve differential equations with QCL circuits, the parameter shift rule is tested on the IBM quantum computer in Section 7.1. Following this, in Section 7.2, a simple differential equation is solved on the

quantum computer. Lastly, a coupled differential equation is solved with a single QCL circuit on a simulator in Section 7.3.

2 Quantum Circuit Learning

The general structure of QCL circuits is shown in Figure 1.

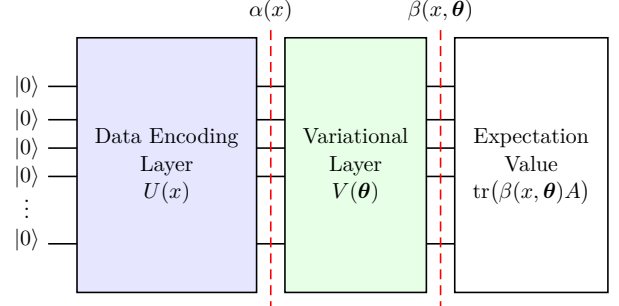


Figure 1: Structure of QCL circuits starting with the data encoding layer $U(x)$ followed by the parameterized variational layer $V(\theta)$ and the calculation of the expectation value of an observable A .

In the data encoding layer $U(x)$, the variable x is encoded into the quantum state with quantum feature map encoding. After the data encoding layer follows the variational layer $V(\theta)$, which consists of parameterized quantum gates with the parameters $\theta = (\theta_0, \theta_1, \dots)^\top$. In 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 form $\sin(nx)$ 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 change the coefficients c_i , the variational layer is introduced. The new coefficients, which we again denote with c_i , 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

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

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. The formulation of the cost function and the exact structure of the circuits is explained in Section 3.

In [7] 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 polynomial terms up to the order of x^N with additional $\sqrt{1-x^2}$ -terms.

3 QCL on a Simulator

In this section, we present the learning of different example functions with the help of a classical simulator. First, the cost function

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

is defined, where the quantum model function $f_{QC}(x, \theta) = \text{tr}(\beta(x, \theta)Z_0)$ is the Z expectation value of the first qubit, \sum_i sums over a number of training points x_i at which the cost function is evaluated and $f(x)$ is some function which

is to be learned. The cost function is classically minimized using the gradient-based sequential least squares programming (SLSQP) algorithm [15]. The starting parameters for the classical optimizer are chosen randomly. In the following example, a simple three qubit circuit with $R_Y(\arcsin(x))$ data encoding is used (see Figure 2).

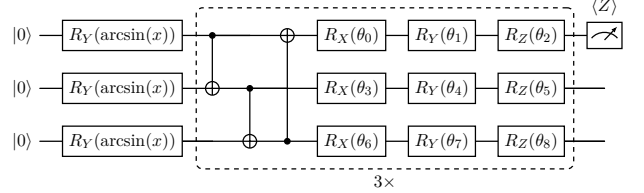


Figure 2: Three qubit QCL circuit with $R_Y(\arcsin(x))$ data encoding followed by a variational block consisting of three CNOT gates to achieve circular entanglement and θ -parameterized x-, y- and z-rotations. The variational block is repeated three times. The Z expectation value of the first qubit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

Since only the first qubit is measured in this example, it is necessary to use entanglement gates. They ensure that higher-order functions can be obtained by measuring the first qubit [7]. CNOT gates are chosen here, because they are part of the physical gate set of the IBM quantum computer, where this algorithm will be tested on in Section 4. The CNOT gates are applied in a linear chain with an additional gate between the first and the last qubit which is called circular entanglement. In general, a variety of different entanglement methods are possible. For example, the time evolution of an Ising Hamiltonian can be used to create a highly entangled state [7]. However, this work does not focus on the analysis of different entanglement methods.

The entanglement gates are followed by a θ -parameterized x-, y- and z-rotation with $\theta_i \in [0, 2\pi)$ on each qubit, which is sufficient to achieve any unitary single-qubit operation apart from a global phase [16]. The specific choice of the rotational gates allows several possibilities with the same generality. For example, it is also possible to use an x-, z- and additional x-rotation [7]. However, the arrangement described above has produced the best results.

The entangling gates and the θ -parameterized gates are repeated multiple times up to a certain depth D . The same parameters θ are used in every block. Therefore, the number of param-

eters stays the same for different depths and depends only on the qubit number. A greater depth results in a higher expressibility [17]. In the example in Figure 2, a depth of $D = 3$ is selected as this depth has proven to be suitable for representing complicated functions without creating a circuit that is too deep.

We now learn the functions $f_1(x) = x^3$, $f_2(x) = x^3 - x^2 + 1$ and $f_3(x) = \sin(2x)$ with the QCL circuit in Figure 2 on a statevector simulator without shot noise. The cost function is evaluated on 20 equidistant training points and is classically minimized using SLSQP. The SLSQP algorithm is executed with the default settings of the SciPy minimizer [18]. The final learned functions are plotted in Figure 3 (a)-(c) (green lines). The initial function resulting from the randomly selected start parameters at the beginning is also shown (black dashed lines in Figure 3 (a)-(c)). The qualitative behavior of the functions can be learned. However, the final result shows strong deviations which become clear in the error diagrams (green lines in Figure 3 (d)-(f)), where the absolute values of the respective errors $|f(x) - f_{QC}(x)|$ are plotted on a fine grid. The values of the cost function versus the number of cost function evaluations (green lines in Figure 3 (g)-(i)) shows that the optimization stagnates after just a few hundred cost function evaluations.

To increase the accuracy of the function learning, a post-processing parameter [7] called θ_{post} is multiplied by the value of the quantum model function $f_{QC}(x, \theta)$ and is also optimized with the classical optimizer. We call the modified quantum model function

$$f_{QC}^{\text{post}}(x, \theta) = f_{QC}(x, \theta) \cdot \theta_{\text{post}} \quad (8)$$

and define the cost function

$$L(\theta) = \sum_i |f(x_i) - f_{QC}^{\text{post}}(x_i, \theta)|^2. \quad (9)$$

The same example functions are now learned again with the post-processing parameter θ_{post} (red lines in Figure 3 (a)-(c)). The ability to learn functions is considerably improved by the additional post-processing parameter which can be clearly observed in the error diagrams (red lines in Figure 3 (d)-(f)). The values of the cost function are notably reduced even when comparing the same number of cost function evaluations (red lines in Figure 3 (g)-(i)). As an additional

degree of freedom, the post-processing parameter significantly increases the expressibility and leads to faster and more accurate learning [17]. The inclusion of this parameter also extends the value range to $f_{QC}^{\text{post}}(x, \theta) \in \mathbb{R}$ for $\theta_{\text{post}} \in \mathbb{R}$ and the quantum model functions are no longer limited by the value range of the Z expectation value $f_{QC}(x) \in [-1, 1]$.

4 QCL on 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 [10]. In this work, the full algorithm is executed on a quantum computer. This means that also every circuit evaluation during the optimization process is performed on the quantum computer. The QCL circuits that have been simulated in Section 3 are now executed on the 27-qubit IBM Quantum System One in Ehningen (ibmq_ehningen). A recent study on this system can be found in [19]. All experiments on ibmq_ehningen in this paper were performed with Twirled Readout Error eXtinction (TREX) [20]. Apart from this, no error mitigation techniques were applied.

Due to the hardware noise and particularly the shot noise, a gradient-based optimization is not possible. Hence, the previously used optimizer SLSQP is no longer used. Instead, the Constrained Optimization BY Linear Approximation (COBYLA) [21] proves to be suitable.

The same functions that have already been learned in Figure 3 on a simulator are now learned on ibmq_ehningen. The cost function is evaluated on 10 equidistant training points with 2000 shots for every circuit and is classically minimized using COBYLA (red markers in Figure 5 (a)-(c)). Additionally, the initial function resulting from the randomly selected start parameters at the beginning and $\theta_{\text{post}} = 1$ is plotted (black dashed lines in Figure 5 (a)-(c)). The functions can be learned with good accuracy on the quantum computer which becomes clear in the error diagrams (Figure 5 (d)-(f)), where the absolute values of the respective errors $|f(x) - f_{QC}(x)|$ are plotted. It can be observed that the two polynomials ($f_1(x) = x^3$, $f_2(x) = x^3 - x^2 + 1$) can be learned with higher accuracy than the trigonometric function ($f_3(x) = \sin(2x)$). This can be

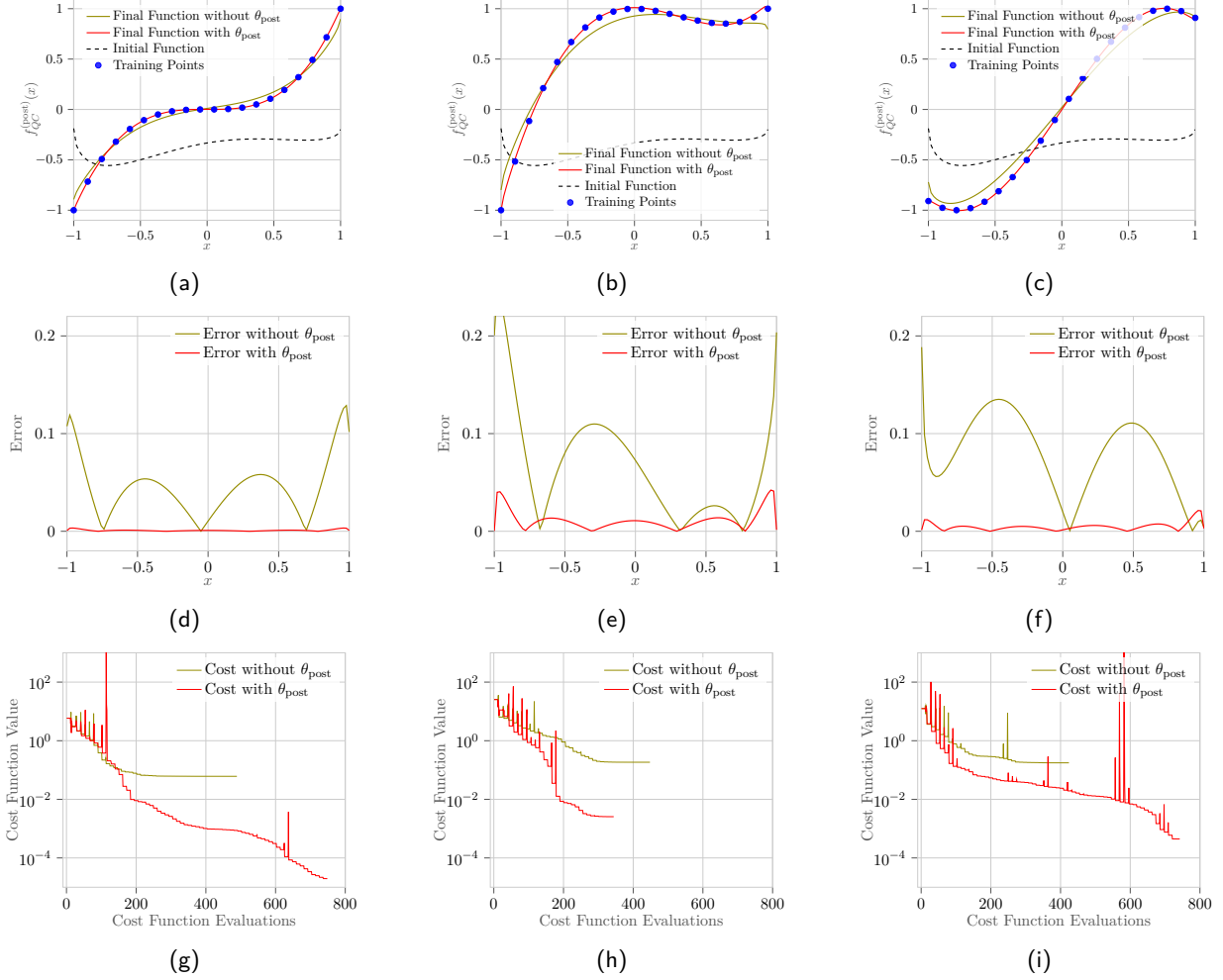


Figure 3: Three learned functions using QCL circuits on a statevector simulator with (red lines) and without (green lines) a post-processing parameter θ_{post} , $R_Y(\arcsin(x))$ data encoding, a qubit number of $N = 3$ and a depth of $D = 3$. The learned functions are $f_1(x) = x^3$ (a), $f_2(x) = x^3 - x^2 + 1$ (b) and $f_3(x) = \sin(2x)$ (c). The cost function is evaluated on 20 equidistant training points and is classically minimized using SLSQP. The initial function (dashed black line) shows $f_{QC}(x)$ with the randomly chosen starting parameters before the optimization process and $\theta_{post} = 1$. Additionally, in (d)-(f) the absolute values of the respective errors $|f(x) - f_{QC}(x)|$ 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.

explained by the fact that the selected data encoding layer generates polynomial-like functions (see Equation (4)) which are more suited to learn polynomial functions.

The values of the cost function versus the number of cost function evaluations (Figure 5 (g)-(i)) shows that the cost function is evaluated less than one hundred times during the optimization process. These results show that the QCL algorithm can already be executed on today's quantum computers and demonstrate the high relevance for the NISQ era.

5 Analysis of Hardware Errors

In Section 4, we showed that the QCL algorithm is able to learn various functions on a quantum computer. In this section, the effect of the physical hardware errors is examined in more detail. To this end, we want to investigate the errors in the execution of QCL circuits for different numbers of qubits. To adequately compare circuits of different sizes, all parameters θ_i are set to $\frac{\pi}{2}$, as shown in Figure 4 for a qubit number of $N = 3$ and a depth of $D = 3$.

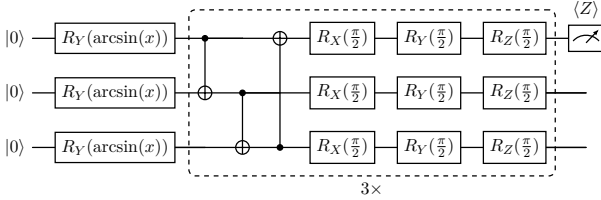


Figure 4: Three qubit QCL circuit with $R_Y(\arcsin(x))$ data encoding. The Z expectation value of the first qubit is measured.

This circuit gives expectation values of the form

$$f_{QC}(x) = (-1)^N \cdot x, \quad (10)$$

where N is the number of qubits. This means that, except for the sign, the expectation values are the same for all numbers of qubits. The circuit is executed on the IBM quantum computer for different numbers of qubits.

Figure 6 (a) shows the measured expectation values for different circuit sizes. 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 mea-

sured expectation values approaching

$$f_{QC}(x) \equiv 0. \quad (11)$$

Figure 6 (b) shows the corresponding mean absolute error

$$\text{MAE} = \frac{1}{T} \sum_{i=1}^T |f(x_i) - f_{QC}(x_i, \theta)|, \quad (12)$$

where T is the number of training points. The mean absolute error increases quickly and saturates early on. This is expected because $f_{QC}(x)$ approaches zero early on as the qubit number increases. The theoretical error for the case in Equation (11) is also shown as a black line in the Figure 6 (b).

In summary, QCL circuits on current quantum computers are only suitable for low numbers of qubits. For higher qubit numbers the hardware error causes the expectation values to approach zero.

6 Multi-Qubit 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

$$g_{QC}^{\text{post}}(x, \theta) = \langle Z_1 \rangle(x, \theta) \cdot \tilde{\theta}_{\text{post}}, \quad (13)$$

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

$$L(\theta) = \sum_i (|f(x_i) - f_{QC}^{\text{post}}(x_i, \theta)|^2 + |g(x_i) - g_{QC}^{\text{post}}(x_i, \theta)|^2), \quad (14)$$

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

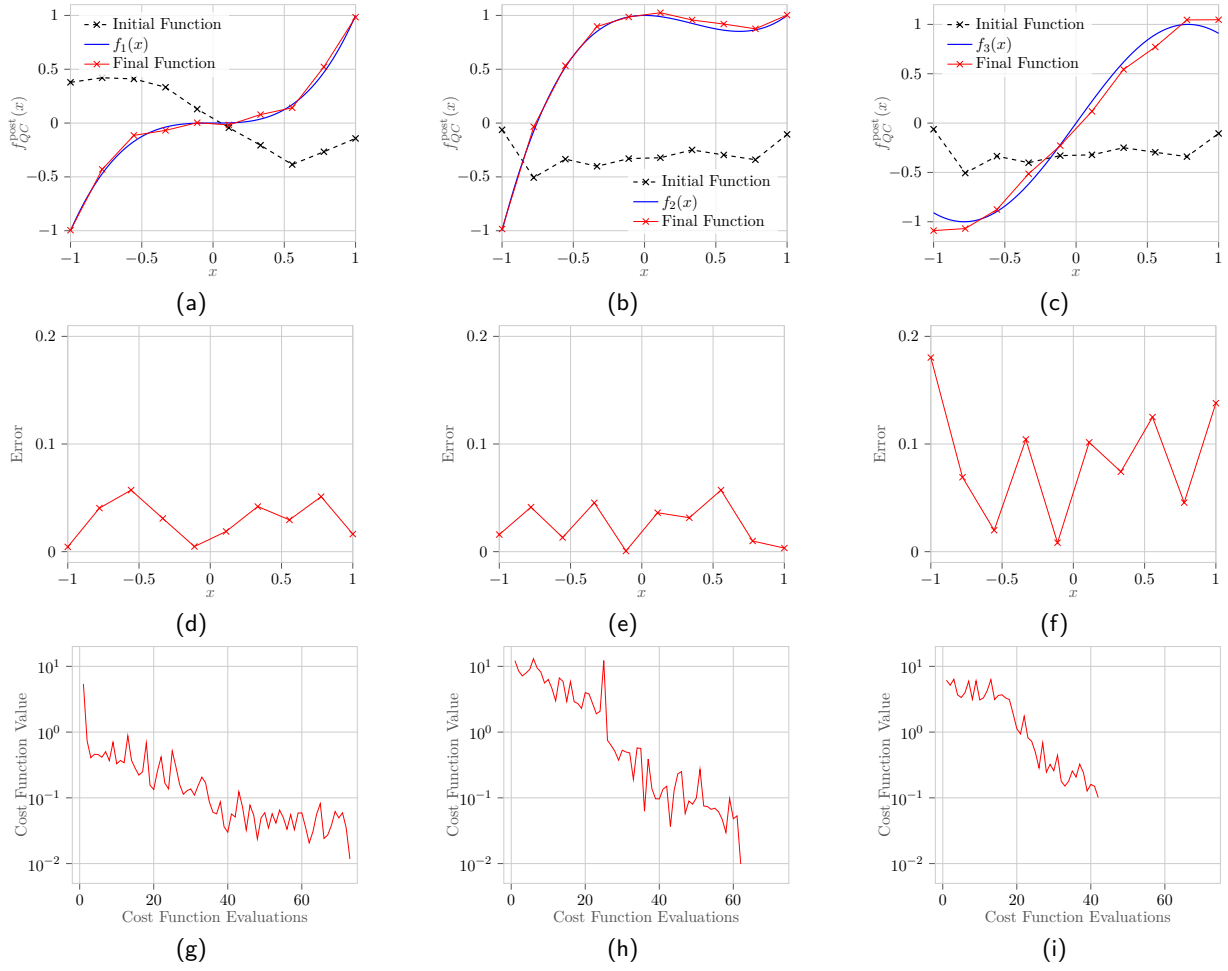


Figure 5: Three learned functions using QCL circuits on the `imbq_ehningen` with a post-processing parameter θ_{post} , $R_Y(\arcsin(x))$ data encoding and a qubit number of $N = 3$. The learned functions are $f_1(x) = x^3$ (a), $f_2(x) = x^3 - x^2 + 1$ (b) and $f_3(x) = \sin(2x)$ (c). The depth is $D = 2$ for $f_1(x)$ and $D = 3$ for $f_2(x)$ and $f_3(x)$. The cost function is evaluated on 10 equidistant training points with 2000 shots for every circuit and is classically minimized using COBYLA. The initial function (dashed black line) shows $f_{QC}^{post}(x)$ with the randomly chosen starting parameters before the optimization process and $\theta_{\text{post}} = 1$. Additionally, in (d)-(f) the absolute values of the respective errors $|f(x) - f_{QC}^{post}(x)|$ 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.

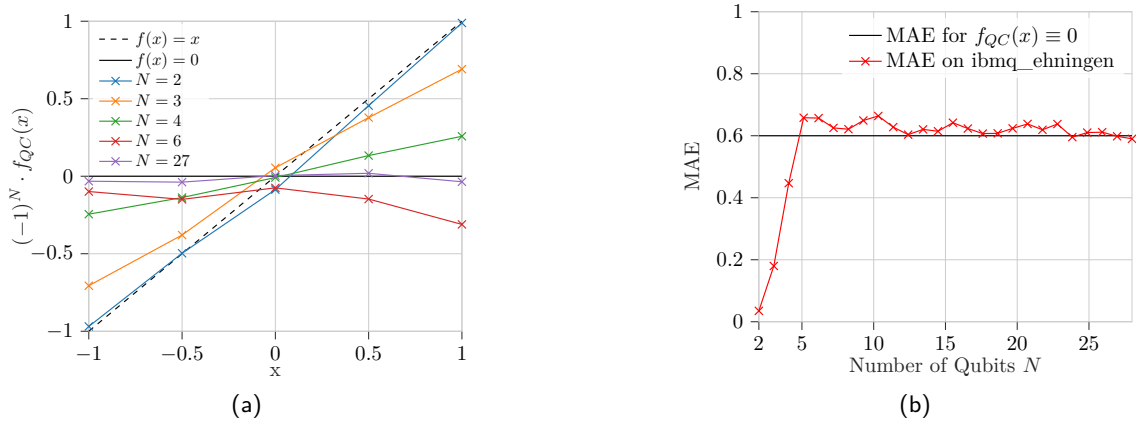


Figure 6: (a) $f_{QC}(x)$ of QCL circuits with $\theta_i = \frac{\pi}{2}$ for different numbers of qubits evaluated on 5 equidistant training points on the `imbq_ehningen`. For the function for $N = 3$ and $N = 27$ the sign is reversed so that it has the same sign as the other examples. (b) Mean average error for 5 training points using QCL on the `imbq_ehningen` (red line). The black line indicates the error that would occur if the Z expectation value would be 0 for all training points.

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 Figure 7) on a statevector simulator.

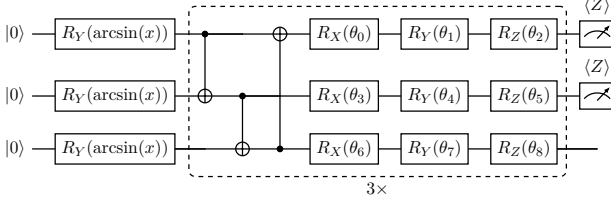


Figure 7: Three qubit QCL circuit with $R_Y(\arcsin(x))$ data encoding. The Z expectation values of the first and second qubit are measured.

The cost function is evaluated on 20 equidistant training points and is classically minimized using SLSQP (see Figure 8 (a)-(c)). In Figure 8 (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 Figure 3, in which only one function was learned. However, it is still so small that shot noise would have a much greater effect when executed on a real quantum computer.

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

$$f(x) = \sum_{i=0}^6 c_i x^i \quad (15)$$

where the coefficients c_i are randomly chosen with

$$\sum_{i=0}^6 c_i^2 \leq 1. \quad (16)$$

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 Figure 9 (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 func-

tion values and the number of function evaluations are divided by four for the multi-function case (blue line in Figure 9). In order to have the necessary parameters to learn multiple functions, unlike in the rest of the paper, 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.

7 Differential Equations

This section investigates the possibility of solving differential equations with QCL circuits in combination with the parameter shift rule (PSR).

7.1 Parameter Shift Rule

The PSR is a method to determine the exact derivative of a parameterized quantum circuit [7, 9]. 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 $\pm \frac{\pi}{2}$. 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 `imbq_ehningen`. For this purpose, the circuit and the optimized parameters from the learning of $f(x) = x^3$ in Figure 3 (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 Figure 10. The qualitative behavior of the derivatives can be determined

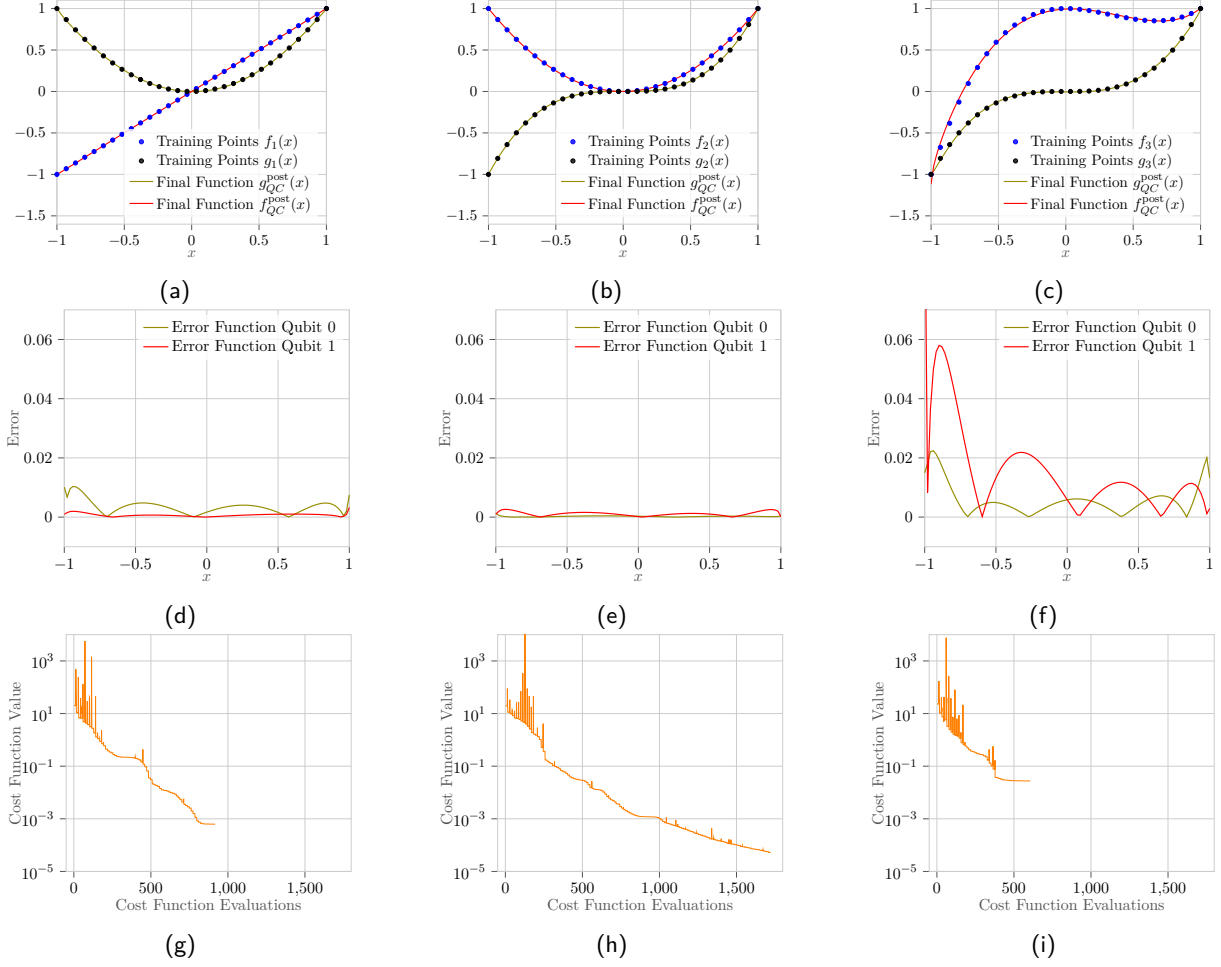


Figure 8: Learning two different functions per QCL circuit on a statevector simulator with a post-processing parameter $\theta_{\text{post},i}$, $R_Y(\arcsin(x))$ data encoding, a qubit number of $N = 3$ and a depth of $D = 3$. The learned functions are $f_1(x) = x$ and $g_1(x) = x^2$ (a), $f_2(x) = x^2$ and $g_2(x) = x^3$ (b) and $f_3(x) = x^3 - x^2 + 1$ and $g_3(x) = x^3$ (c). The cost function is evaluated on 30 equidistant training points and is classically minimized using SLSQP. Additionally, 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.

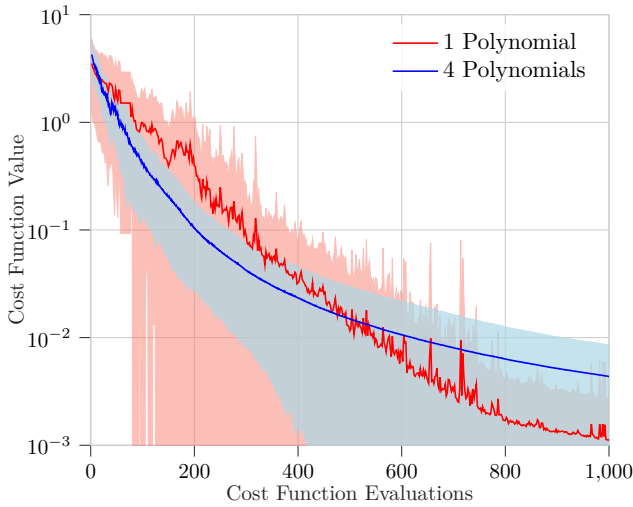


Figure 9: Average of the convergence curves of the training of 100 QCL circuits with $N = 6$ qubits and a depth of $D = 4$: For the red line 100 random polynomials are learned, i.e. one polynomial per circuit. For the blue line four polynomials are learned per circuit, which sums up to 400 random polynomials learned. We use 10 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

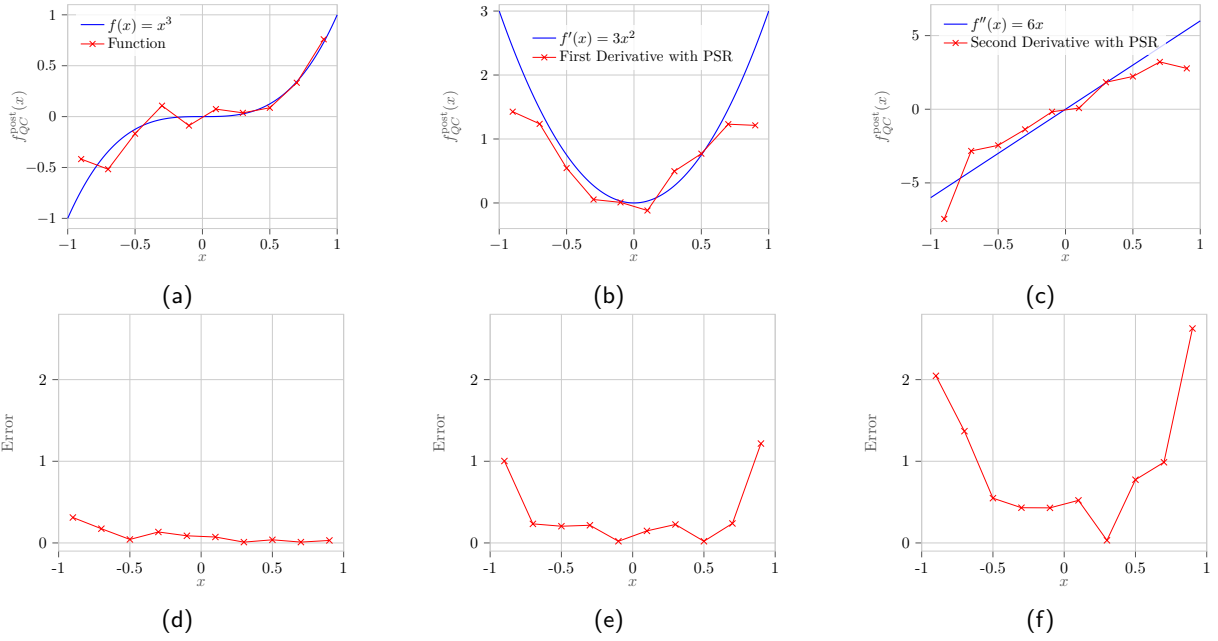


Figure 10: Resulting function values from executing the QCL circuit with trained/final parameters for $f(x) = x^3$, see Figure 3, on ibmq_ehningen with 10 equidistant training points (a). Values of the first (b) and second (c) derivative obtained with the parameter shift rule applied to aforementioned 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.

well. However, there are large errors, especially in the ranges 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.

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.

7.2 Differential Equation on IBM Quantum Computer

In this section, we aim to solve a differential equation with QCL circuits and the parameter shift rule on ibmq_ehningen. For this purpose, a simple example of a differential equation is considered to limit the required quantum resources. We focus on the differential equation

$$\begin{cases} f'(x) = 3x^2, \\ f(0) = 0. \end{cases} \quad (17)$$

The solution of this differential equation is

$$f(x) = x^3. \quad (18)$$

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

$$L(\theta) = \sum_i \left(\left| f_{QC}^{\text{post}}(x_i, \theta) - 3x_i^2 \right|^2 + \mu \left| f_{QC}^{\text{post}}(0, \theta) - 0 \right|^2 \right), \quad (19)$$

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 $\mu = 10$, 10 equidistant training points and 2000 shots is shown in Figure 11. 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 Figure 11 (d)-(f)) are higher than in the case of learned functions on ibmq_ehningen (compare Figure 5 (d)-(f)). This is because the derivatives are included in the cost function which results in significantly more circuit evaluations and the errors accumulate.

7.3 Coupled Harmonic Oscillator

In this section, we want to solve a coupled differential equation using a single QCL circuit in

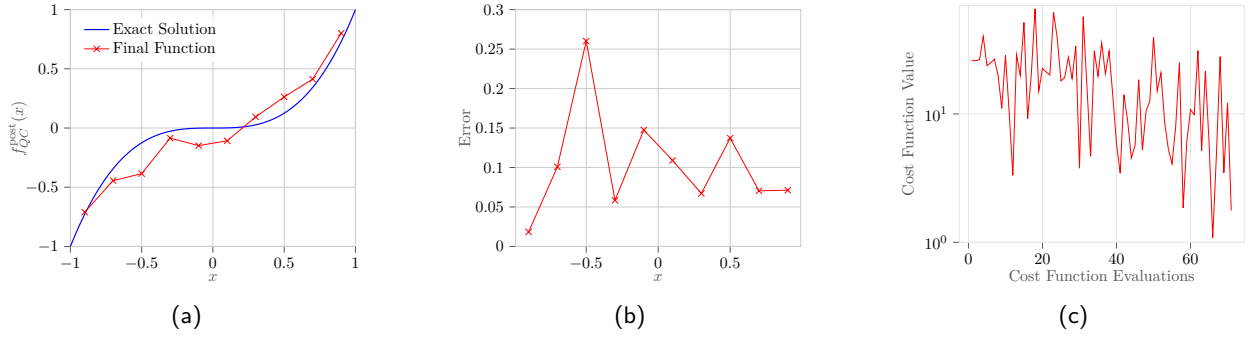


Figure 11: (a) Solution using QCL circuits of the differential equation (17) on `imbq_ehningen` with $\mu = 10$, $N = 3$, $D = 3$, 10 equidistant training points and 2000 shots with $R_Y(\arcsin(x))$ data encoding. The parameters are classically optimized using COBYLA. (b) The absolute values of the respective errors $|f(x) - f_{QC}^{\text{post}}(x)|$. (c) Values of the cost function versus the number of cost function evaluations during the optimization process.

combination with the parameter shift rule on a simulator. Coupled differential equations were solved with QCL circuits before, using a designated circuit for each equation [11]. We combine this idea with the result from Section 6, where we showed 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 Figure 12.

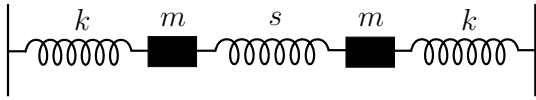


Figure 12: Example of the coupled harmonic oscillator with two identical masses m , two identical springs of spring strength k and one spring of spring strength s .

This system is described by the coupled differential equation

$$\mathbf{f}''(x) = \mathbf{S}\mathbf{f}(x), \quad \mathbf{f}'(0) = \mathbf{f}'_0, \quad \mathbf{f}(0) = \mathbf{f}_0, \quad (20)$$

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 $1/m$,

$$\mathbf{S} = \frac{1}{m} \begin{pmatrix} -k-s & s \\ s & -k-s \end{pmatrix}, \quad (21)$$

and \mathbf{f}'_0 and \mathbf{f}_0 are the initial velocity and initial displacement of the masses, respectively. For this paper we choose the initial conditions as $\mathbf{f}'_0 = (0, 0)$ and $\mathbf{f}_0 = (1, 0)$. Then, the solution to (20) is given by

$$\mathbf{f}(x) = \frac{1}{2} \begin{pmatrix} \cos(\omega_0 x) - \cos(\omega_1 x) \\ \cos(\omega_0 x) + \cos(\omega_1 x) \end{pmatrix}, \quad (22)$$

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 Figure 13, where the first and second qubit are measured.

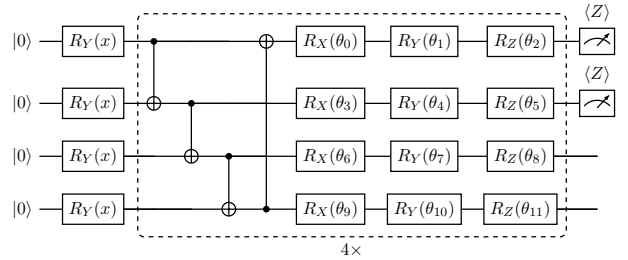


Figure 13: Four qubit QCL circuit with $R_Y(x)$ data encoding. The first and the second qubit are measured.

We chose the $R_Y(x)$ data encoding to obtain trigonometric functions, which are the appropriate choice to capture the periodic behavior one expects from an undamped system of oscillators. To solve the differential equation with the QCL circuit in Figure 13, the cost function

$$\begin{aligned} L(\boldsymbol{\theta}) = & \sum_i \left(\|\mathbf{f}_{QC}''(x_i, \boldsymbol{\theta}) - \mathbf{S}\mathbf{f}_{QC}(x_i, \boldsymbol{\theta})\|^2 \right. \\ & + \mu \|\mathbf{f}_{QC}'(0, \boldsymbol{\theta}) - \mathbf{f}'_0\|^2 \\ & \left. + \mu \|\mathbf{f}_{QC}(0, \boldsymbol{\theta}) - \mathbf{f}_0\|^2 \right) \end{aligned} \quad (23)$$

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

$$\mathbf{f}_{QC}(x, \boldsymbol{\theta}) = (f_{QC,0}^{\text{post}}(x, \boldsymbol{\theta}), f_{QC,1}^{\text{post}}(x, \boldsymbol{\theta})). \quad (24)$$

The derivatives in the cost function are calculated with the PSR. The result of the simulation

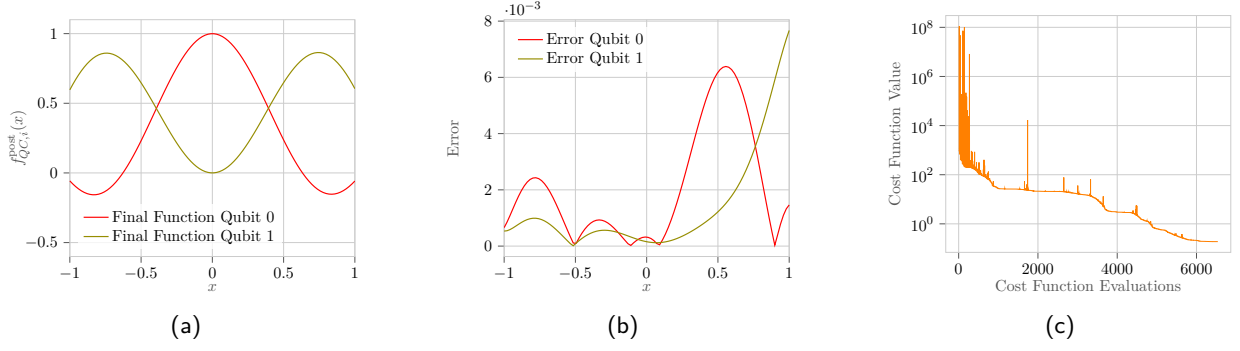


Figure 14: (a) Solution of the differential equation (20) on a statevector simulator with $\mathbf{f}'_0 = (0, 0)$, $\mathbf{f}_0 = (1, 0)$, $\omega_0^2 = 1$, $\omega_1^2 = 16$ and $\mu = 20$. The result is obtained with the QCL circuit in Figure 13 with $N = 4$, $D = 4$ and $R_Y(x)$ data encoding in combination with the parameter shift rule. The parameters are classically optimized using SLSQP. (b) The 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.

for $\omega_0^2 = 1$, $\omega_1^2 = 16$ and 30 equidistant training points is shown in Figure 14. It is possible to solve a coupled differential equation with just one circuit with low errors (see Figure 14 (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.

8 Conclusion

In this work, we have conducted a comprehensive investigation of the QCL framework and its executability on NISQ devices. At the beginning, multiple functions were successfully learned with a simulator. Different cost functions were compared and the importance of an additional post-processing parameter was shown. The QCL circuits were tested on an IBM quantum computer to investigate their NISQ-friendliness. Promising results were achieved here. For a number of functions, the qualitative behavior could be learned with good accuracy and few cost function evaluations. In addition, the hardware errors were examined in more detail. It was shown that QCL only works for low qubit numbers on the IBM quantum computer. 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.

Next, the ability to solve differential equations

with QCL circuits and the parameter shift rule was investigated. First, the parameter shift rule 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 subsequently solved to a certain extent on `ibmq_ehningen`. With the knowledge about multi-qubit measurements, the differential equation of a coupled harmonic oscillator was solved with only a single circuit on a simulator.

Overall, however, it was found that QCL requires many circuit evaluations, especially if derivatives are determined with the parameter shift rule. This makes the classical optimization computationally very intensive.

This problem applies not only to this algorithm 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 [22, 23]. Such methods give hope for a reduction of computation time and make the algorithm even more relevant.

Additionally, in [24], a physics-informed quantum algorithm was proposed, where quantum model functions can be trained and its derivative can be evaluated without independent sequential function evaluations on grid points. This approach could significantly reduce the number of circuit evaluations.

Acknowledgments

The authors would like to thank Vamshi Katukuri for insightful discussions and carefully proofreading this manuscript.

N. S. wants to thank Sungkun Hong for his advice and dedicated supervision of his Master’s thesis which built the starting point of this article.

The authors acknowledge funding from the Ministry of Economic Affairs, Labour and Tourism Baden-Württemberg in the frame of the Competence Center Quantum Computing Baden-Württemberg (project SEQUOIA End-to-End)

References

- [1] John Preskill. “Quantum computing in the nisy era and beyond”. *Quantum* **2**, 79 (2018).
- [2] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O’Brien. “A variational eigenvalue solver on a photonic quantum processor”. *Nature communications* **5**, 4213 (2014).
- [3] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. “Variational quantum algorithms”. *Nature Reviews Physics* **1**, 20 (2021).
- [4] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A quantum approximate optimization algorithm” (2014).
- [5] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. “An introduction to quantum machine learning”. *Contemporary Physics* **56**, 172–185 (2015).
- [6] Maxwell Henderson, Samriddhi Shakya, Shashindra Pradhan, and Tristan Cook. “Quantumvolutional neural networks: powering image recognition with quantum circuits”. *Quantum Machine Intelligence* **2** (2020).
- [7] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. “Quantum circuit learning”. *Physical Review A* **98** (2018).
- [8] Maria Schuld and Nathan Killoran. “Quantum machine learning in feature hilbert spaces”. *Physical Review Letters* **122** (2018).
- [9] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. “Evaluating analytic gradients on quantum hardware”. *Physical Review A* **99** (2019).
- [10] Kan Hatakeyama-Sato, Yasuhiko Igarashi, Takahiro Kashikawa, Koichi Kimura, and Kenichi Oyaizu. “Quantum circuit learning as a potential algorithm to predict experimental chemical properties”. *Digital Discovery* **2**, 165–176 (2023).
- [11] Oleksandr Kyriienko, Annie E. Paine, and Vincent E. Elfving. “Solving nonlinear differential equations with differentiable quantum circuits”. *Physical Review A* **103**, 052416 (2021).
- [12] Niklas Heim, Atiyo Ghosh, Oleksandr Kyriienko, and Vincent E. Elfving. “Quantum model-discovery” (2021).
- [13] Martin Knudsen and Christian B. Mendl. “Solving differential equations via continuous-variable quantum computers” (2020).
- [14] Annie E. Paine, Vincent E. Elfving, and Oleksandr Kyriienko. “Quantum quantile mechanics: solving stochastic differential equations for generating time-series”. *Advanced Quantum Technologies* **6**, 2300065 (2023).
- [15] Dieter Kraft. “A software package for sequential quadratic programming”. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR. (1988).
- [16] Michael A. Nielsen and Isaac L. Chuang. “Quantum computation and quantum information”. Cambridge University Press. (2010). 10th anniversary edition.
- [17] Niclas Schillo. “Quantum algorithms and quantum machine learning for differential equations” (2023).
- [18] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro,

- Fabian Pedregosa, and Paul van Mulbregt. “Scipy 1.0: fundamental algorithms for scientific computing in python”. *Nature methods* **17**, 261–272 (2020).
- [19] Andreas Sturm, Bharadwaj Mummaneni, and Leon Rullkötter. “Unlocking quantum optimization: a use case study on nisq systems” (2024).
- [20] Ewout van den Berg, Zlatko K. Mineev, and Kristan Temme. “Model-free readout-error mitigation for quantum expectation values” (2020).
- [21] Michael J. D. Powell. “A direct search optimization method that models the objective and constraint functions by linear interpolation”. In Susana Gomez and Jean-Pierre Hennart, editors, *Advances in Optimization and Numerical Analysis*. Springer Netherlands, Dordrecht (1994).
- [22] Shiro Tamiya and Hayata Yamasaki. “Stochastic gradient line bayesian optimization for efficient noise-robust optimization of parameterized quantum circuits”. *npj Quantum Information* **8** (2022).
- [23] Marco Wiedmann, Marc Hölle, Maniraman Periyasamy, Nico Meyer, Christian Ufrecht, Daniel D. Scherer, Axel Plinge, and Christopher Mutschler. “An empirical comparison of optimizers for quantum machine learning with spsa-based gradients” (2023).
- [24] Annie E. Paine, Vincent E. Elfving, and Oleksandr Kyriienko. “Physics-informed quantum machine learning: solving nonlinear differential equations in latent spaces without costly grid evaluations” (2023).