IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 10 March 2025, accepted 20 April 2025, date of publication 25 April 2025, date of current version 5 May 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3564392

RESEARCH ARTICLE

On the Challenges of Quantum Circuit Encoding Using Deep and Reinforcement Learning

PATRICK SELIG[1,2], NIALL MURPHY[3], DAVID REDMOND[3], AND SIMON CATON[1,2]
[1]School of Computer Science, University College Dublin, Dublin 4, D04 V1W8 Ireland
[2]UCD Centre for Quantum Engineering, Science and Technology, University College Dublin, Dublin 4, D04 V1W8 Ireland
[3]Equal1 Laboratories, Dublin 4, D04 V2N9 Ireland

Corresponding author: Patrick Selig (patrick.kappen@ucdconnect.ie)

This work was supported in part by Research Ireland under Grant 18/CRT/6183.

ABSTRACT Quantum computers may solve some computing tasks more efficiently than classical computers, but to do so requires the design of appropriate quantum circuits. However, it is hard to design even simple quantum programs. Recent results have shown that deep learning augmented search has the potential to discover good circuits for a problem, but it is not yet sufficiently understood how well deep learning is able to model such a complex domain: the search space and output space grow exponentially. This work explores the ability of neural networks to encode quantum circuits for tasks that require knowledge about the unitary representation of the circuit. To this end, we trained neural networks to directly learn to predict the unitary of a circuit and applied reinforcement learning to train neural networks to solve circuit based quantum state preparation. This work finds that encoding quantum circuits is quite difficult especially with regards to the amount of entanglement applied in both application domains. The use of intermediate states and structure in quantum circuits combined with a reasonable inductive bias, where applicable, can alleviate these problems to some extent. The use of intermediate states also greatly improves the results of Deep Reinforcement Learning for quantum state preparation. Based on these results, we discuss the next set of challenges to address if we are to design neural network-based approaches for the automatic generation of quantum circuits.

INDEX TERMS Quantum state preparation, quantum program synthesis, quantum architecture search, gate-based circuits, supervised machine learning, reinforcement learning, barren plateau.

I. INTRODUCTION

In recent years, developments in the building of quantum computers has made significant progress. There are lower errors in the results of quantum computations and more qubits available. Yet, there are still many challenges that need to be overcome: 1) quantum computers are difficult to apply to non-trivial tasks; 2) preparing a specific quantum state, to, for example, insert classical data in an appropriate encoding, is very challenging [1], [2]; and 3) computations are not error-free. There are enough application domains where quantum computers could be truly beneficial, such as cryptography (e.g. quantum key distribution), artificial intelligence and algorithmic design (e.g. solving optimization problems like the traveling salesman problem) and chemistry (e.g. molecule optimisation and drug discovery). However, in order to leverage these potential benefits, these challenges must be addressed to bring quantum computing forward. In this article, we explore different applications and architectures for deep learning and reinforcement learning that seek to aid in the design and development of quantum programs (circuits). We aim to shed light on different design considerations for researchers seeking to undertake ML-driven quantum architecture search.

There are many different ways to approach problems requiring the design of quantum circuits, for example, there exist different state preparation algorithms using numerical approaches or decomposition [3], [4], [5]. However, the challenge is that finding quantum algorithms for new tasks requires the exploration of an exponential search space.

The associate editor coordinating the review of this manuscript and approving it for publication was Sawyer Duane Campbell.

One potential approach to these challenges might be the use of various machine learning techniques (especially deep and reinforcement learning), as they have proven quite successful in a multitude of very complex, exponentially growing domains like the game of GO [6], protein folding [7] and language few-shot learning [8]. Neural network-based approaches have shown that they can consistently beat all other approaches by quite a margin and are still making progress. This might make them a promising candidate for current challenges in quantum computing. One might use a neural network to learn a distribution over circuits for different target states or use deep reinforcement learning to find quantum circuits for new tasks or for designing ansätze for variational quantum algorithms.

While the creation of quantum circuits using neural networks seems like a good fit, there are open questions regarding its usefulness when the neural network needs to learn an implicit model of a unitary representing a quantum circuit without specific structure. For example, results in [9] show that information encoded in general random unitaries is not necessarily on a lower dimensional plane. Thus, encoding such a unitary using neural networks is quite difficult, whereas unitaries with specific types of structure are easier to encode. This is the case when the task is to predict quantum circuit distributions without intermediate executions of the circuits for preparing arbitrary states as well as when using Reinforcement Learning to generate quantum circuits. It is important to know whether neural networks are able to encode general quantum circuits where their unitary is important and how well such approaches are able to do so.

It may be tempting for researchers to just "throw" high capacity (deep learning) models at the problem and hope that the architecture learns something useful. In this paper, we illustrate that there are quite complex and nuanced challenges in the applications of deep learning and reinforcement learning to automate aspects of quantum program design. Specifically, we highlight that it is difficult to learn the behavior of the quantum circuit directly. Thus, this work aims to explore how neural networks handle tasks that need to encode quantum circuits and their unitaries with a focus on creating quantum circuits automatically. To study the behavior of neural network-based search, two sets of experiments were conducted to highlight specific challenges and identify neural network design decisions of specific promise and importance. Using these experiments, we discuss different architectural decisions one may make, their impacts/limitations, and provide some remarks on potential architecture designs and aspects of the training regime that have shown some promise.

The first set of experiments investigates the ability of neural networks to predict the unitary implemented by a given quantum circuit. Since the unitary represents the action of a quantum circuit, this shows how well a neural network can encode a circuit's action. We keep this experiment (relatively) simple in order to analyze the ability of neural networks to learn how to encode quantum circuits. While there are many potential tasks that could be suited to illustrate this (e.g. quantum state preparation), the dependence between a circuit and its expected output might be more convoluted or may not strictly include information from the whole unitary.

The second set of experiments investigates the task of automating the state preparation process. State preparation is a common and important sub-step in quantum computing as it is used to bring a Quantum Computer (QC) into a specific state as part of an algorithm or workflow. Quantum state preparation is a suitable task to show how the difficulty of encoding quantum circuits affects tasks in practice. It is perhaps one of the most common and necessary tasks in quantum computing, and notably challenging: the approximation error of a target state depends directly on a part of the unitary of a circuit approximating it.

Tasks that require knowledge of the unitary (as implemented by a quantum circuit), such as quantum state preparation [10], unitary prediction [11], and when the input is restricted to a description of quantum circuits, are quite challenging to learn for neural networks. In such settings the training is very slow and when reinforcement learning is used often no real progress is made during the search for a good policy. Our results show that the biggest impact factor is the amount of entanglement in the quantum circuit, and also that the use of intermediate states helps the neural network significantly; making these kinds of tasks more feasible with neural networks.

The rest of the paper is structured as follows: section II discusses relevant related work. This is followed by a summary of relevant concepts in quantum computing and machine learning in section III. Next, open technical challenges (specifically the barren plateau problem [12], and issues of having to scale to 2^{2n_q}) are discussed in section IV, followed by the experiment designs (section V) and their results in section VI. Finally, we conclude the paper, highlight its main contributions and outcomes, and discuss future work in section VII.

II. RELATED WORK

This section will introduce relevant related work starting with deep reinforcement learning, which is necessary to train a deep neural network when no ground truth/labeled data for the intended behavior is available. Afterward, more specific related work is presented first by introducing program synthesis as general background information, followed by quantum program synthesis and state preparation.

A. DEEP REINFORCEMENT LEARNING

Deep Reinforcement Learning consists of a set of different algorithms that belong mainly to one of two classes [13]: model-based RL [14] and model-free RL [15]. Model-based RL uses a model to make predictions of the outcome of a given action while model-free RL does not do so and directly aims at finding a policy or value function that can be used to induce a policy to solve a specific task at hand. In practice, the borders between them can blur [16] and are not mutually exclusive [17], [18]. This work uses model-free RL, as it has proven highly successful in various search tasks such as Go [6].

Different deep model-free RL algorithms have been developed and successfully applied in various domains. Using a neural network to learn the optimal q function, [15] was able to play Atari games successfully without prior knowledge. Alternatively, policy optimization algorithms for optimizing a neural network representing the policy were developed in [19] with the Trust Region Policy Optimization (TRPO) approach and in [20] with the Proximal Policy Optimization (PPO) algorithm. However, model-free RL has also been deployed in quantum computing scenarios as well. Model-free Deep RL was applied successfully on various problems related to quantum computation. PPO was applied to optimize quantum circuits in [21] and for quantum compilation in [22]. Deep RL was used in [23] to control quantum systems and for state preparation in [24].

B. PROGRAM SYNTHESIS

The automatic design of a program is commonly called program synthesis [25]. It concerns itself with the automatic creation of a program given a description of it. A common description is a set of input-output pairs [26], consisting of inputs to the program and the corresponding outputs, but natural language descriptions are also used [27]. Program Synthesis has been applied in a variety of problem domains. It was applied to the generation of general programs in [28], [29], and [30] where the programs are represented using lambda calculus [31]. More domain-specific tasks include applying program synthesis to automatic string processing [32] and for the synthesis of rules for automated knowledge extraction in [33]. It has seen a wide application in computer graphics [34] where it was used to generate material graphs from images of materials in [35]. In [36] and [37] constructive solid geometry graphs were generated from images of the object.

Program synthesis in general requires solving a search problem with the goal of finding a program satisfying specific conditions, for example, reproducing a set of input output pairs. Approaches to solving the search problem are, for example, the use of evolutionary algorithms [25] as done in [38], [39], and [40] or genetic algorithms [41], [42]. Another direction of work creates programs using an expressive language and transforming it into a set of constraints that are then solved using a SAT solver [43] and an example for this approach is the sketch system [44], [45], [46]. The top-down search can be applied by starting at the output of the program and filling in missing parts using enumerative search [47] or probabilistic search [48]. DeepQPrep [10] used top-down search to find quantum circuits to prepare quantum states.

As the problem of search for a program given a set of potential instructions is exponential in nature, it is necessary to have a good heuristic to steer probabilistic search into a space of the solution space containing reasonable solutions. These heuristics can be automatically learned by using various machine learning techniques. The probability distribution of the appearance of each operation in the final program was learned to be predicted in [49]. Using an attention-based neural network model [50] predict the probability of each operation to be added to the program next. In [51] a probabilistic model of the program space was learned and in [52] it was viewed as an unsupervised learning problem that was solved using a combination of modeling with solver-based techniques. One very prominent technique of learning the heuristic is deep learning. Reference [53] used a Convolutional Neural Network (CNN) trained with supervised learning to predict programs from hand-drawn images and in [54] a RNN was used to encode the tree structure of the current program. RL was used to learn a neural network model, to guide the program search, without the need for ground truth data in [55]. In [30], deep learning was combined with automatic library discovery to learn a probabilistic model of the program space. Automatic library discovery allows the search complexity to be reduced by adding functional units containing multiple operations as one operation to be added to the program, done in [28] and [56]. Recently, large language models have become prevalent [8]. Although they are not necessarily trained to create programs, they have shown remarkable acuity at doing so [57] from natural language descriptions.

C. QUANTUM PROGRAM SYNTHESIS

Quantum Program Synthesis often, but not always, aims at generating a representation of a quantum program called a quantum circuit [58]; it can also be referred to as Quantum Architecture Search. One specific task of quantum circuit generation that absolutely requires the automatic generation of circuits is quantum state preparation as various algorithms [59], [60], [61], [62], [63], [64] require the quantum computer to be in a specific non-trivial state (based on classical data). In quantum state preparation, the goal is to find a quantum circuit that brings the QC into a specified state. Other applications include the creation of a quantum circuit based on a unitary matrix representing the desired computation as a $2^n \times 2^n$ matrix (also called quantum compilation). This matrix must be decomposed into the elementary gates available to the quantum computer which can be represented themselves as unitary matrices. Another class of quantum circuit synthesis tasks receives a form of high-level description of the task to be solved; [65] uses high-level descriptions to generate sets input output pairs describing the desired behavior of the system. Variational quantum circuits, a circuit parameterized by classical parameters, used in variational quantum machine learning and variational eigensolvers require the design of the circuit to be amenable to the optimization of its parameters [66]. This is necessary as the optimization of a quantum circuit can suffer from the barren plateau problem [12], making parameter optimization quite challenging.

These problems have been approached using classical techniques similar to those presented in the previous section. Genetic algorithms were used to generate quantum circuits in [67], and [68] used an evolutionary algorithm to design a quantum circuit for quantum state preparation. Bayesian optimization was used in [65] to generate a quantum circuit from a high-level description. Deep reinforcement learning was used in [22] and [69] to generate a quantum circuit from a unitary matrix that represents the whole computation. When the goal is quantum compilation, creating a circuit from a unitary matrix, matrix decomposition approaches can be used. These approaches decompose the unitary into smaller and smaller unitaries until elementary gates are reached. [3] uses the Householder transformation to decompose the matrix and [4] uses cosine-sine decompositions. The time complexity of these approaches was improved in [5] by combining fixed position and variable position unitary decompositions with numerical optimization techniques.

The synthesis of quantum circuits from a target unitary was performed using diffusion models [70] in [11]. A combination of deep learning and non-machine learning-based circuit synthesis can also be used to first propose potential circuits using a deep learning model and then to use bottom-up synthesis to improve those circuits, as was done in [9]. The paper also presents results on the ability of neural networks to encode unitaries which are related to the results of this work showing the difficulties of neural networks in encoding quantum circuits.

III. BACKGROUND

This section will go into the background of this work. First, subsection III-A will give an introduction to the necessary background in quantum computation followed by a short introduction to the necessary background in Deep Learning in subsection III-B. subsection III-C will give an explanation on how both concepts are combined in this work to do quantum circuit search.

A. QUANTUM PROGRAMS

Quantum programs can be represented in different ways, for example, as a form of pseudo code or high-level description which allows us to prove properties of the program and to get an understanding of the overall working of the algorithm. The issue with such a high-level approach is that it needs to be implemented using low-level primitive operations. The exact implementation of the algorithm depends on many factors, such as the number of qubits, the available operations on the QC and its physical properties. However, we consider a quantum program as a sequence of (unitary) operations, i.e. gates, acting on 1 or more qubits. We note that different configurations in the number and/or quality of qubits can require the creation of different quantum circuits even for similar tasks.

1) PROGRAM STRUCTURE AND OPERATIONS

A common representation of quantum programs is in the form of quantum circuits where qubits are represented as wires, and operations, referred to as gates, act on those wires (i.e., qubits). An example quantum circuit can be seen in Figure 1. Quantum programs are executed from left to right and represent the time evolution of a quantum system from an initial (or ground) state to an observed (or measured) set of outcomes. All gates are reversible operations (they are unitary transformations), except measurement, which collapses the quantum wave form into an observed value.

It is important to note that the number of wires at each gate that affects multiple qubits cannot increase or decrease. Quantum circuits may contain gates that affect one or more qubits, but this work restricts itself to gates acting only on one or two qubits. (It is possible to decompose n qubit gates into one and two qubit gates). Typically, multi-qubit gates involve some degree of entanglement. Entanglement is necessary to leverage multiple qubits in programs and generate increased expressivity in the functional form(s) a program represents.

Gates can be fixed in their operation or can be parameterized by a classical real value. All 1-qubit gates represent a rotation of the qubit around one or multiple of three orthogonal basis vectors, (e.g. the X, Y, and Z bases (see Figure 3)). As more gates are added to a circuit, its depth increases. The depth of a quantum circuit is defined as the number of gates on the longest path through the circuit. For example, the circuit in Figure 2 has a depth of 4.

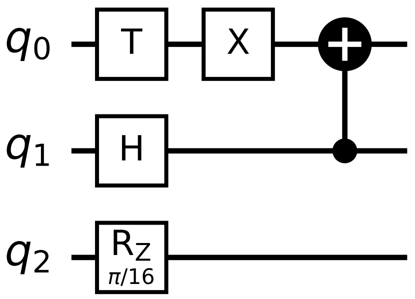

FIGURE 1. Example circuit with depth 3.

2) LAYERED UNITARY REPRESENTATION

Each gate or combination of gates can be represented as complex unitary matrices of the shape $2^{m_{act}} \times 2^{m_{act}}$ where m_{act} is equal to the number of qubits the gate acts on for a quantum circuit with m_{act} qubits. The unitaries of one-qubit gates therefore have a shape of 2×2. Multiple sequential one-qubit gates can be combined into one unitary matrix by multiplying them. This unitary can be represented by a sequence of three rotations with different angles (ignoring the global phase) [4]. A potential sequence is $R_z R_y R_z$. Each complex value v_{ij} in the unitary is constrained by $0 \leq |v_{ij}| \leq 1$ and the whole unitary matrix is normalized.

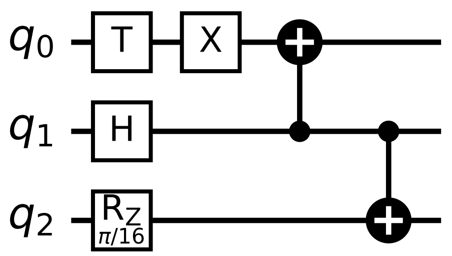

FIGURE 2. Example circuit with depth 4.

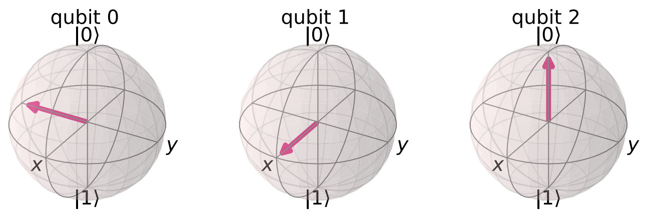

FIGURE 3. Visualization of a rotation about $\frac{\pi}{2}$ around the x,y and z-Axis for qubit 1, 2, and respectively.

Each entry in each 1-qubit unitary is a sum of the product of three values that depend on the different parameters of the rotations. As such, programs can be represented by interleaving layers of one qubit unitary gates followed by an arbitrary entanglement pattern, as can be seen in Figure 4. If a qubit does not have a unitary gate applied to it in a layer, it can simply be represented by the identity unitary, and it does not matter whether the entanglement gates or rotations are applied first in a layer. If there are no rotations before the first entanglement operations, the rotation layer can simply consist of identities. An entanglement layer at the end of the circuit is optional.

The total unitary implemented by a quantum circuit U_{circ} on m qubits can be calculated by computing the product of the unitaries of all N layers U_i with $0 < i \leq N$:

$$U_{circ} = U_N \dots U_2 U_1 \tag{1}$$

The unitary of each layer U_i is computed by:

$$U_{unit,i} = \bigotimes_{j=1}^{m} U_{i,j} \tag{2}$$

$$U_{ent,i} = \prod_{k=1}^{n_{cx,i}} C_{k,i} \tag{3}$$

$$U_i = U_{ent,i} U_{unit,i} \tag{4}$$

where the $U_{i,j}$ represent the single qubit unitaries in a layer and $C_{k,i}$ (of size $2^m \times 2^m$) is the kth CX gate between two qubits for a total of $n_{cx,i}$ CX gates that are added in layer i. For a more detailed introduction, see [71]. While it is clear that a unitary can be derived from a quantum program, so too is the potential value of being able to predict the unitary of an arbitrary program without executing it.

3) INITIAL STATE AND DATA LOADING

A quantum computer starts in a base state commonly represented by $|0\rangle$. To bring the quantum computer into a specific target state $|\psi_t\rangle$ a quantum circuit which represents a unitary U_{state} needs to be found such that:

$$U_{state} |0\rangle = |\psi_t\rangle \tag{5}$$

The quantum state vector represented by $|0\rangle$ in the standard basis contains a 1 in the first row and zeros in all other positions. This matrix multiplication means that the first column of the unitary represents the quantum state it prepares.

Many quantum algorithms and applications need the QC to be in a specific state that captures some input data (a process called state preparation). Depending on how input is passed into a quantum circuit, it might be necessary to put another quantum circuit in front of the computing algorithm (i.e., concatenate them) to bring the QC into the required initial state that encodes the input data. As shown in [10], automatically generating quantum programs to undertake this process is not straightforward. However, there has been some success in the applications of machine learning to do so.

B. DEEP LEARNING

The goal of Deep Learning is to train a multilayered neural network to minimize a loss l, a scalar function of inputs X, and some target Y. Neural networks are commonly optimized using stochastic gradients computed with backpropagation. The stochastic gradient is computed by taking batches of random subsets of the whole dataset. Neural networks use multiple layers of typically non-linear functions f_i of the input x_{in} and some parameter vector p_i.

$$x_{out} = f_i(x_{in}, p_i) \tag{6}$$

Depending on the structure of the input data X, different architectures are possible. This work views the quantum circuits as layers as discussed in subsection III-A. Therefore, it is a sequence of layers that can be approached using different architectures like recurrent neural networks or transformer neural networks. Recurrent neural networks compute the output at each step in the sequence in an iterative fashion, computing the results using the encoding of the last step and the next element in the sequence. Transformers take as input the whole sequence and perform a self-attention mechanism over the whole input for each input position.

C. QUANTUM CIRCUIT SEARCH

To create quantum circuits using deep learning, a search algorithm must be established. An obvious choice might be to build the circuit layer by layer as the content of each layer is well defined: a unitary gate for each qubit, including identity gates, and a set of CX-Gates between the different qubits. While this is certainly possible, it might lead to difficulties when applying RL to the task of quantum state preparation. Since a layer is, in a way, quite complex, it might be difficult to learn the influence of each of its parameters, as only a

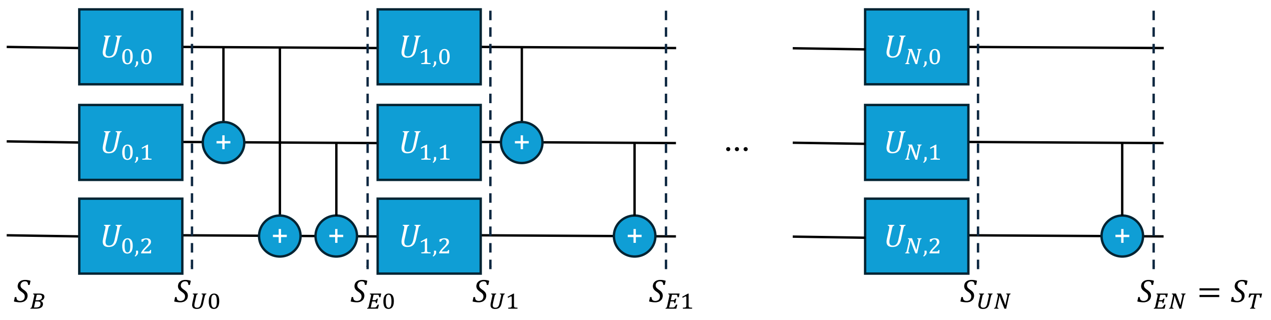

FIGURE 4. A circuit with N Layers. Each layer consists of a set of unitary gates followed by a set of entanglement gates.

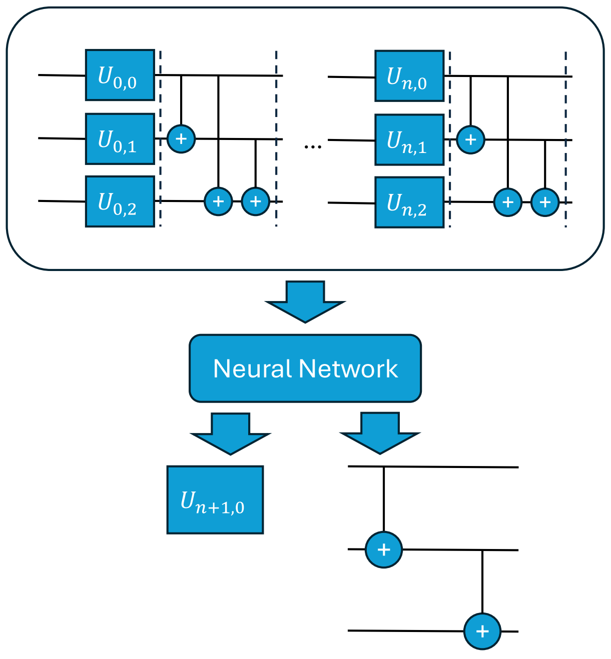

FIGURE 5. Overview of the neural network based search.

reward for the full layer is computed. An alternative would be to view each sub-part (each unitary gate and the entanglement pattern) as separate steps, thereby computing a reward at each step making it easier to learn about the influence of each sub-step.

Figure 5 gives an overview of the approach applied in this paper. In this approach, each time sub-step can be represented as a valid layered circuit at any sub-step. This is done by applying an identity to each unitary qubit that does not have a unitary added to it and applying no CX-Gates. A circuit that is not yet completed (all circuits before the final step during the search) is called a partial circuit, but there is in principle no difference between them, it is just used to signify that the circuit still has more gates that are going to be appended to it. As the circuit is built sub-step by sub-step and layer by layer the neural network needs to be able to predict the unitary to add to a qubit and the entanglement pattern. Since both parts have different shapes, it does so by having two different output layers. The first layer predicts a unitary to be added to one qubit. The second layer predicts the pattern of CX-Gates to add. The search algorithm will add the gates to the current partial circuit appropriately. The neural network receives information about the current step as a one-hot input vector with $n_{qubit} + 1$ elements: one for each qubit and the entanglement sub-step.

Instead of predicting a unitary, the neural network will predict three rotation values for three rotations around the z, y, and z axes. This combination of rotations is universal up to a global phase term, as discussed previously. During the search, two intermediate sets of information are used as input to the neural network: 1) the current partial circuit to which a new operation must be added; and 2) the state the QC is in after applying the current partial circuit or an output distribution of the measurements of it.

While the second input might make it easier to learn what to do for the neural network, it requires more intermediate executions of a circuit, which can be expensive. In contrast, the current partial circuit is readily available without much additional computational cost. Thus, the question arises of how well a neural network is able to encode and extract information from such a circuit in a manner that is conducive to its learning process.

IV. OPEN TECHNICAL CHALLENGES

Various tasks in quantum computing require knowledge of the unitary implemented by a quantum circuit. As an example, take circuit-based quantum state preparation as defined in Equation 5.

Finding a quantum circuit such that its unitary U prepares the target state $|\psi_t\rangle$ is not an easy task. One potential approach would be to use machine learning and especially deep learning to learn a distribution over potential circuits, as deep learning has proven quite successful in a wide range of very difficult domains [6], [7], [8] and in quantum state preparation [10]. For a task like state preparation without any intermediate states, the model needs to make a decision based on a current partial circuit. In order to be able to make a reasonable decision, the model would need to create an encoding of the circuit that contains important information, which in this case would be related to the unitary of the circuit (as it needs to know what the circuit computes in order to bring it closer to the target state). This is the case if there is little or no structure that can be assumed in the circuits that it needs to encode. An example for such a situation would be when using Reinforcement Learning for quantum state preparation and the initial policy shows a behavior that is close to a uniform distribution, which is often the case if no prior information is available.

We argue that there are two problems with the need to encode quantum circuits when there are no or few restrictions on the set of circuits. The first problem is related to the vanishing gradient in quantum circuits also called barren plateaus [12] and how it makes it difficult for the neural network to learn the relation between the gates in the circuits and the output especially for parameterized circuits. The second problem is the exponential growth of information needed to model the unitary when controlled operations are used.

A. VANISHING GRADIENTS

As was shown in [12], many parameterized quantum circuit ansätze have a high probability of having a gradient close to zero. Although this result mainly influences the training of variational quantum circuits using classical algorithms, this also has important implications for the training of neural networks that need to model the unitary implicitly. The main argument is that the mass of probability of states that fall outside zero decreases exponentially as the number of qubits increases. This means that the probability of the gradient being close to zero grows exponentially. This again results in a very weak coupling between the input (the gates) and the output as the number of qubits grows. As the neural network tries to learn connections between the circuit and its behavior, this becomes exceedingly hard.

Another way to analyze the situation would be to look at how the unitary is influenced by each of its gates. For this we use the representation of circuits as a sequence of layers as discussed in section III. Each layer consists of two parts:

- The application of a unitary to each qubit
- Addition of an arbitrary pattern of controlled operations

The unitary encoding for a rotation layer l of a circuit is computed by

$$U_l = \bigotimes^{n} R_z(\phi_q)R_y(\lambda_q)R_x(\theta_q) \tag{7}$$

Therefore each entry in the Unitary is a product of $2n$ values where n is equal to the number of qubits. The CX-gates in each layer essentially permute the unitary resulting from the unitary part, thereby creating a dependency between the involved qubits. To compute the total unitary of multiple layers, the product of the unitary of all the layers needs to be computed. This means that the resulting unitary is a sum of products. Each sum consists of 2^{nN_L} terms, where N_L is equal to the number of layers to be combined. Each of those terms consists of a product of $2nN_L$ values (see subsection III-A on the number of results in the unitary of each layer). The absolute value of each term is ≤ 1, so the terms decay exponentially to zero while the number of terms grows exponentially. The influence of each term of each element on the unitary layers is therefore small. This situation is exacerbated as the depth of the circuit increases, resulting in a shrinking gradient; similar to the vanishing gradient that occurs during neural network training [12]. This behavior gets even more complicated when CX-gates are taken into account as they change which unitary values influence which final unitary element, and how strongly.

We can view a quantum circuit as an object connecting input qubits to output qubits via different paths. A three-qubit circuit without any entanglement operations has only three paths that can be taken through it. The number of paths through the circuit grows exponentially as the entanglement gates are added and the qubits influence each other [12]. Through this, the number of interactions between the qubits grows exponentially with each path representing one combination of information, and thus the combinations of information grow exponentially. Overall, this behavior might make it very hard for a neural network to learn as each element has a very complex small influence on the unitary. The network would need to learn the behavior of something where the impact of each single input element can be fairly small but still important in the end.

B. EXPONENTIAL GROWTH OF INFORMATION

When the task requires implicit knowledge of the unitary, the neural network may need to realize (potentially implicitly) it. The issue with the need to realize the unitary lies in the fact that while the circuit might not contain exponentially many gates, the neural network still needs to periodically compute its unitary (a product of a unitary for each layer). Each unitary is of size 2^{2n}, thus the overall computation will require at least a feature representation of that size, or some data structure that allows the encoding of this object efficiently. If the set of quantum circuits is constrained and thus its unitaries as well, we can store reduced amounts of data. An example of a structure that would reduce the amount of data that needs to be encoded is the set of circuits that contain no gates that act on more than one qubit. In such a case, the total unitary of a circuit is computed as the Kronecker product of the independent one-qubit unitaries. Therefore, the encoding the neural network computes would only need to store information on each qubit separately, yielding n unitary complex unitaries of shape 2×2. In contrast, when there are multiple qubit gates combining all qubits, the whole unitary would need to be stored as all qubits depend on each other. Other sets of circuits that only create connections between a subset of qubits would result in unitaries with such a substructure. Thus, depending on the connectivity pattern between the qubits the size of the needed encoding changes, it grows exponentially in the number of connected qubits.

C. PRACTICAL IMPLICATIONS AND CONTRIBUTIONS

These effects (i.e. the barren plateau and exponential growth of information) have practical implications when learning circuits that have no clear structure. As an example, assume that the goal is to learn which gates to add to a partial circuit to prepare a specific state using Reinforcement Learning. If the neural network is trained from scratch without any imitation learning (e.g. without any circuit examples), it will generate random circuits but learning the connection between the gates added and the new resulting state is difficult. This means that the neural network only learns very slowly (if at all), making it extremely challenging to solve the problem. If the neural network would receive the state generated by the partial circuit and not the partial circuit, things might be different.

These implications mean that it is not possible to design a neural network architecture easily, but that research is needed to explore (even the most obvious) architectural decisions that may, or may not work. To this end, in the rest of this paper we show that we cannot just ''throw'' deep (reinforcement) learning at the problem and hope it works. Rather, it is difficult to learn the behavior of the quantum circuit directly. The combination of exponential growth of information and the barren plateau problem make a complex search space even harder to navigate: approaches like RL suffer greatly when attempting to explore a large space of circuits as the feedback gained from adding gates (or layers of gates) is basically zero, which makes the training very slow or even halt completely.

V. EXPERIMENT DESIGN

To explore the influence of the effects of an exponential growth of information and the barren plateau problem on the difficulty to learn to encode quantum circuits in practice, we performed two different sets of experiments: 1) focusing on predicting the unitary matrix of a given quantum circuit; and 2) focusing on using RL for the automated generation of state preparation circuits. We ensure that all neural networks have an encoding size bigger than the number of values in the unitary, that is, they could in principle store the whole unitary in the encoding of the circuit. This allows us to explore the difficulty to learn to encode a quantum circuit without the influence of the exponential scaling of the unitary of the circuit. We investigate two different neural network architectures commonly used for sequences (as a quantum circuit is a sequence of operations) and how different entanglement patterns affect the results: an LSTM and a transformer-based architecture.

Comparing these experiments shows how well the neural networks can encode quantum circuits with entanglement and how intermediate quantum state information can help improve results greatly for quantum state preparation. This, in turn, aids in the design of training regimes for neural network architectures, as we highlight easy to occur pitfalls in the training process, and discuss some potential remedies for these.

A. UNITARY PREDICTION

The first experiment is focused on learning to predict the unitary of a circuit given different entanglement patterns. The motivation to learn to predict unitaries is to explore how well neural networks are able to learn what operation (essentially its unitary) the quantum circuit performs without the influence of issues like the need for exploration, as is the case in reinforcement learning. Instead, we explore the influence of different circuit properties like the degree of entanglement (exercised through the number of CX-Gates).

As mentioned previously, the neural network has an encoding size that could in theory fit the whole unitary matrix into it more than once. This alleviates any issue arising from the encoding size being too small. Thus, this experiment highlights the difficulty of learning quantum circuits as the number of entanglement gates between qubits increases, while keeping the training itself ''simple(r)''.

To support this experiment, a dataset was created by sampling circuits consisting of a predefined number of layers and qubits at random. The dataset consists of 2 million circuits with up to 16 layers and 3 qubits and is available at [72]. The low number of qubits is sufficient for the difficulty of the task while at the same time keeping the resulting unitary reasonably small (8×8). For some experiments, CX-Gates were added only to specific qubits at specific layers to show their influence more clearly.

The angles in each layer are three sampled angles in $(0, 2\pi)$ and computing the unitary of a U-Gate which is a universal gate up to global phase [4]. The entanglement in each layer is created by uniformly sampling a subset from the set of possible entanglement patterns. We restrict the entanglement to one possible direct connection between two qubits going from the lower position qubit to the higher one. The entanglement is always applied in the order of the qubits. A set of potential patterns can be seen in Figure 4. The entanglement operation chosen in this work is the controlled X-Gate, also called CX-Gate. For 3 qubits there are three potential CX-Gates that can be added in each layer. After the circuit is sampled, we compute the unitary matrix of each layer in the circuit, and store the results; we also compute the unitary of the whole circuit.

The features of the neural network are: sequences of layers consisting of a $2 \times 2 \times 2$ matrix for each qubit (representing the real and imaginary parts of a unitary as a real value); and a binary encoding of whether a CX-Gate was selected or not.

The neural network architecture can be seen in Figure 6. The neural networks first combine the features in each layer into one encoding tensor in the Layer Encoder by applying one or multiple linear layers with the ReLU activation function to the unitary features and the entanglement binary encoding. These are then combined using another linear layer with the ReLU activation function. This results in a sequence of shape $n_l \times n_f$ where n_l is the number of layers and n_f the number of features. This sequence is then passed to either a LSTM or Transformer [73] in the Seq Encoder.[1]

The results are then decoded separately for each element in the sequence. This is done by applying two sets of linear layers, the Kron Predictor and the Full Predictor, with

[1]Note that we also experimented with a CNN-based architecture (not reported here) but its performance was worse than the LSTM and transformer-based architectures.

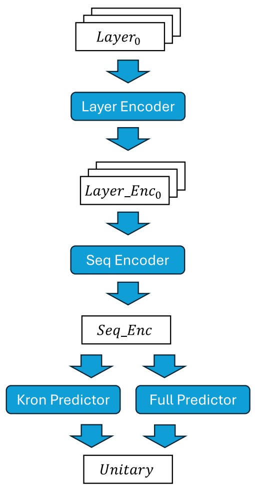

FIGURE 6. The neural network used for predicting unitaries from quantum circuits.

ReLU activation except the last which has a hyperbolic tangents activation. The Kron Predictor predicts 2×2 unitary matrices, one for each qubit, which are combined using the Kronecker product. The motivation behind the Kron Predictor is that 1 qubit unitaries are combined through the Kronecker product to a multi qubit unitary. The Kron Predictor essentially predicts 1 qubit unitaries which are then combined in the same way. The second set of linear layers directly predicts a potential unitary matrix of the shape 8×8. Both layers also predict a weight for the respective matrices. The results are combined by computing a weighted sum using the predicted weights and then normalized.

The optimization is performed using the Adam [74] optimizer with batches of size 256. The learning rate for the different sequence encoders used are 0.00025 for the LSTM model and 0.0005 for the transformer model. The models optimize the mean squared error between the ground truth unitary and the predicted unitary after each layer. Using results after each layer should give the neural network information for intermediate results helping the optimization. We found that only predicting the unitary after the last layer completly prevents learning when CX-Gates are present in the circuit. The encodings created by the neural network components had a size of 256 making it in theory possible to store the whole unitary at each step. The hyperparameters were found using probabilistic search using a uniform distribution over reasonable parameters.

B. DEEP RL FOR STATEPREPARATION

The second set of experiments focuses on finding a circuit that prepares a specific target state using deep reinforcement learning. State preparation is performed in two distinct manners (or design choices): 1) using intermediate states; 2) providing the circuit as input. These two design choices show that the need to learn to encode circuits in RL tasks is quite challenging. The results show that the difficulty of learning a circuit encoding – when the goal is to do sufficient exploration initially – hampers progress greatly. Thus, this experiment explores the potential benefit of using intermediate states to reduce the difficulty of encoding quantum circuits making training with RL more viable.

The experiment has practical relevance, but is influenced by the previously described effects (section IV) due to the need to explore the space of potential circuits sufficiently, and because a possible distribution of gates is close to a uniform distribution initially. However, any meaningful application of RL would need to be able to handle a wide range of circuits. The results are compared to results achieved in the same setting but with the use of the quantum state created by the circuit after each step.

In this experiment, circuits are built inductively by first adding a unitary to each qubit and then a set of CX-Gates to create a layer, as discussed in subsection III-C. A layer is not created in one step but each unitary is sampled separately followed by sampling the CX-Gates. The unitaries are created by predicting the three rotation angles of a U-Gate; a gate able to represent all 1-qubit unitaries [4]. The angles are samples from diagonal Gaussian distributions where the mean is predicted using the neural network. The CX pattern is chosen as in the previous sets of experiments and the neural network predicts the probability of adding a CX-gate in a binary probability distribution for each possible gate.

As mentioned, RL experiments were performed in two different settings: 1) using only the current partial circuit to which new gates must be added as input for the neural network; and 2) using the intermediate state generated by applying the current partial circuit to the base state as input for the neural network. The use of intermediate states instead of using the circuit removes the requirement to learn to encode the quantum circuit since it has information of the behavior of the current quantum circuit instead of needing to infer this information from the circuit. The goal here is to show that extracting such necessary information from the circuit is difficult and the use of intermediate states can reduce these issues.

Both neural networks, seen in Figure 7, first create an encoding of the input and then a fully connected 2-layer prediction neural network is applied to it that predicts the 3 angles of the U-Gate and the probability of each possible CX gates that can be added. Only one of those two options is used at each step, depending on whether a unitary is to be added to a qubit next or the CX-gates. The neural network that uses the current quantum circuit as input is the same as the LSTM model in subsection IV-A, but the last element of the output of the LSTM layer is used as the encoding of the input. The neural network that uses the current state as input uses a 3 layer feed-forward neural network to encode the state. The real and complex components of the state vector are combined so that the neural network receives real input data.

The target state vector of the reinforcement learning algorithm for which a distribution of circuits needs to be found is randomly sampled according to the uniform Haar

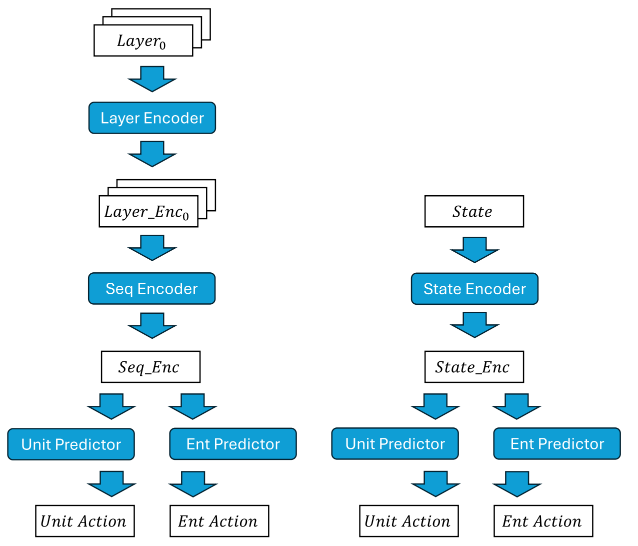

FIGURE 7. The neural network used for predicting unitaries to add to each qubit and entanglements from quantum circuits (left) and quantum states (right).

measure and are restricted to 3 qubits. The models were trained with the implementation of the PPO algorithm [20] from [75] using Stochastic Gradient Descent with a batch size of 256 and a learning rate of 0.0003. For both cases, the prediction network's first layer has a shape of 128×64 and the second $64 \times S_{out}$ where S_{out} represents the shape of the output equal to 3 angles plus the number of potential entanglement gates (for 3-qubits there are 3 possibilities). The activation function is Tanh in both cases.

The reinforcement learning training is performed for 10^6 search steps and after every 2048 search steps the model is trained for 10 epochs using the PPO algorithm. The entropy factor for the PPO algorithm is set to 0.05 in the clip range 0.2. λ and γ are set to 0.95 and 0.99 respectively. The reward is computed using the minimal similarity as proposed in [10] after each action. The motivation is that the neural network should learn to improve the circuit with each gate it adds to the circuit overall.

VI. RESULTS

This section will first show the results achieved for unitary prediction (subsection V-A) followed by the results achieved when doing state preparation using reinforcement learning (subsection V-B).

A. UNITARY PREDICTION

The goal of the experiment is to train neural networks to predict unitaries implemented by a specific quantum circuit. In these settings, the capacity of the feature vector, its dimension, is ensured to be sufficient for the storage of the whole unitary (to ensure that this is not a performance inhibitor). The experiments are undertaken with special interest on the influence of entangling gates (i.e., CX-Gates), as they significantly increase the complexity of the unitary.

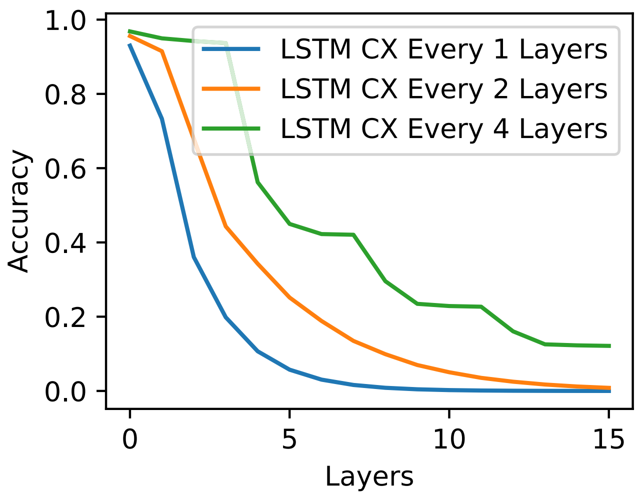

FIGURE 8. Results achieved using a LSTM based Neural Network architecture for applying a CX-Gate in every 1st, 2nd and 4th layer.

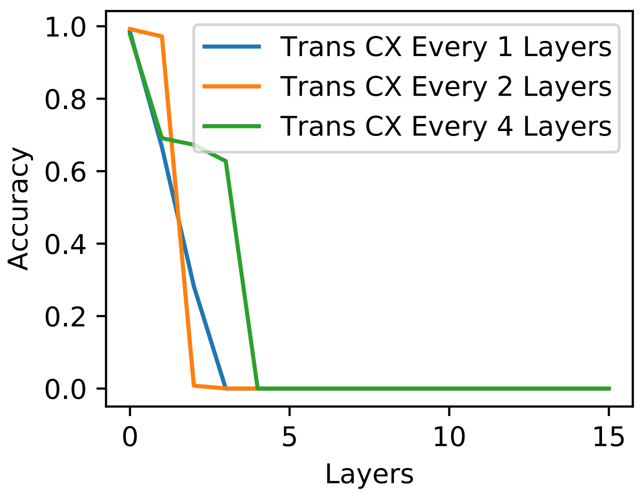

FIGURE 9. Results achieved using a transformer based Neural Network architecture for applying a CX-Gate in every 1st, 2nd and 4th layer.

The prediction results for different layers for the experiment using completely random entanglement patterns after training can be seen in Figure 8 and Figure 9 for LSTM and Transformer based neural network architectures (in blue) after training for 400000 steps. It can be seen that both fail to produce reasonable results for more than a few layers. The prediction accuracy quickly drops to a value of zero, which is equal to a prediction of the mean value in the unitary. The accuracy of the LSTM neural network drops to below half the maximum value after just 3 layers while the transfomer neural network sharply drops to zero after 3 layers. This behavior remains consistent for different samples of the data set in general. The LSTM based neural network is clearly able to make good predictions for one and two layers, but after that, results degrade exponentially.

Figure 8 and Figure 9 show the resulting accuracy when one entanglement gate is added every 2 (orange) and 4 (green) layers. Between these entanglement layers, the results

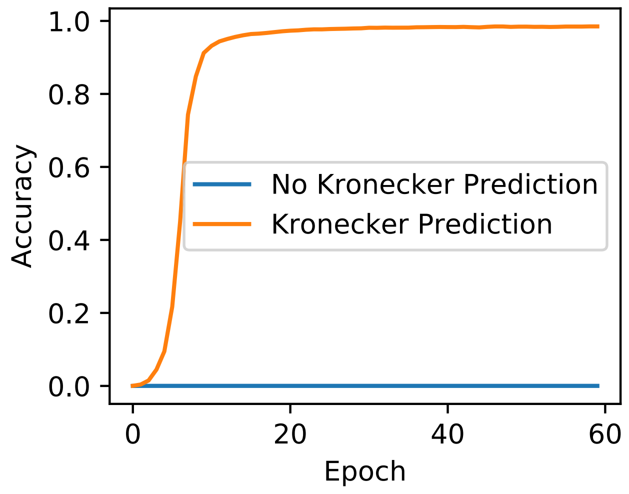

FIGURE 10. Squared difference achieved training LSTM neural networks for quantum state preparation without entanglement gates using a Kronecker product of predicted parts(blue) and predicting the Unitary directly(orange).

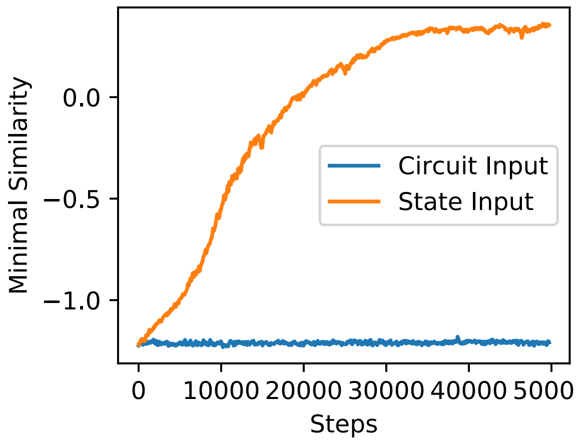

FIGURE 11. Results achieved training neural network for quantum state preparation using the current quantum state (blue) or the current circuit as input (orange).

degrade less overall. This becomes especially clear for the LSTM neural network trained on circuits that have a CX-Gate added every 4 layers (Figure 8 green). The results first decline a bit after adding a CX-Gate but then tend to plateau. This means that the neural networks can make use of the fact that they can be decomposed for each qubit in these layers. The transformer struggles to learn anything beyond the first layer that has a CX-Gate added to it. The transformer-based architecture performs quite poorly overall. This might be due to its attention mechanism, which focuses on specific parts in the input, but in quantum circuits (as designed for this set of experiments), every gate and layer is of importance making it harder to focus on one specific part.

Figure 10 shows results when no entanglement is applied using a Kronecker prediction layer and directly predicting the unitary for the LSTM neural network. In these cases, it becomes apparent that the neural networks can use the fact that the problem can be decomposed when a Kronecker predictor, a strong inductive bias, is used as it learns to predict the unitary with very high accuracy. In contrast, it is not able to learn to predict the unitaries at all when no strong inductive bias is present in the neural network through the Kronecker prediction layer.

The results indicate that it is challenging for neural networks to learn to predict unitaries as more controlled operations (CX-Gates) are added or no inductive bias through the use of the Kronecker prediction is present even for a small number of qubits.

B. DEEP RL STATE PREPARATION

The goal of the second set of experiments is to learn to generate circuits, using RL, for quantum state preparation using intermediate quantum states or quantum circuits as input to the neural network.

Figure 11 shows the results of using PPO to learn a distribution of quantum circuits to prepare a specific quantum state. As can be seen, the neural network that receives the current state as input is learning to predict a more suitable gate at each time step. It achieves a value of roughly 0.25 out of 1.0. In contrast, the neural network trained using the current partial circuit as input only shows a slight improvement at the beginning, which then quickly reaches a plateau at which point no further improvement can be seen. This clearly indicates that not much learning is taking place for the neural network that receives the current partial circuit as input. Figure 12 shows that the neural network trained to use the current state places a higher probability on the gates that yield a higher reward compared to the neural network trained on the current circuit. The neural network trained on the circuit only puts a higher probability for good gates in the first gate, and then the results degrade further. The results also show a drop every 4 gates for the state input neural network which is probably due to the addition of controlled gates which seem to be harder to learn in terms of what is "good" or "bad".

Although there is definitely an influence due to the amount of data for shallower circuits (the space of states is smaller for shallower circuits), since the neural network-augmented search for quantum state preparation was shown to be successful in [10] for a simpler set of circuits, it still clearly shows that the neural network trained on the partial circuits struggles a lot more than the one trained on the current state. This aligns with the results observed in subsection VI-A which show that neural networks seem to struggle when encoding deep circuits. This issue takes place in this reinforcement learning task since the created circuits do not have any specific structure, therefore making learning an encoding model of them quite hard. It is important to note that training the neural network using RL can be harder since the learned target is not fixed for each example but behaves on the trajectories through the state space of the problem domain potentially making already hard learn domains even harder. Overall, both approaches could be combined by getting the quantum state only every few layers and using the circuit

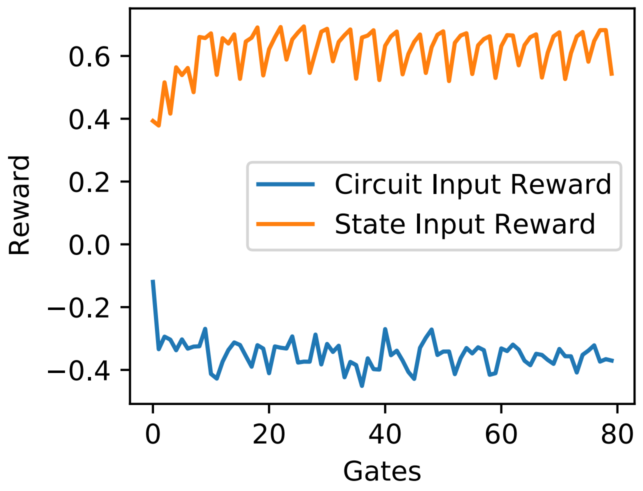

FIGURE 12. Comparison of the reward achieved with the number of gates achieved on the x-Axis after training.

input for the layers in between. Preliminary experiments (not reported here) have shown that this does not really seem to work even for just two layers, as the improvement compared to only using the circuit as input is marginal. This might be related to the fact that just predicting unitaries without any strong inductive bias, the Kronecker product in the predictor, for example, in the neural network, is hard.

VII. CONCLUSION

In this paper, we have explored the ability of neural network based approaches to learn unitary related properties given a corresponding circuit encoding through two sets of experiments: 1) predicting the unitary of a circuit, and 2) performing neural network driven state preparation (or quantum architecture search). We discussed open challenges and specifically why training is difficult. We have also highlighted that learning to predict the unitary implemented by a circuit is quite difficult. Thus, a great deal of domain knowledge is needed in designing neural network architectures for quantum circuit encoding and that it is not possible to (simply) ''apply'' deep learning to the problem.

Our simulations have shown that the interaction between the input and output can be extremely difficult when no decomposition of the problem is possible or when the neural network does not contain a sufficiently strong inductive bias. We have shown that a Kronecker predictive layer could be an appropriate addition to the neural architecture to achieve sufficient inductive bias in the model. An additional outcome is the realization that we can reduce the need to encode deep circuits by using intermediate states.

While the use of intermediate states alleviates some of the challenges presented, there is the issue that using those intermediate states increases the cost of the approach greatly as each state would need to be computed using quantum computer evaluations and be processed by the neural network.

Our results show that overall the quality of predictions decreases exponentially in the number of controlled operations (i.e., as the degree of entanglement increases). This occurs even for a small number of qubits and when the capacity of the neural network should be high enough to store the whole unitary matrix. For automating quantum state preparation, this effect is of significant importance when reinforcement learning is used to train the neural network. However, this effect was alleviated when intermediate states were used.

From our experiments and results, we can highlight the following conclusions, insights, and remarks on the design of neural network architectures:

- The complex dependencies in quantum circuits make the learning of unitary related properties potentially very slow and difficult, i.e. it is difficult to learn the behavior of the quantum circuit directly. This is largely due to the barren plateau problem.
- Increases in the degree of entanglement greatly exacerbate the difficulty of training.
- Quantum state preparation is difficult to solve using RL when the circuit is the input, but intermediate quantum states can alleviate problems in RL for quantum state preparation.
- It is important to maintain a strong inductive bias for unitary prediction tasks; we highlight a Kronecker prediction layer as an enabler of inductive bias.
- Despite the challenges highlighted in this paper, we have shown that a small number of layers (for example, in an ansatz) are easily achievable with neural network-based approaches.
- Approaches in reinforcement learning suffer greatly from the issues raised (low gradient variance, and exponentially growing representations) as they need to explore a large space of circuits, but have little to no information gained from individual gate additions.

Potential ways to alleviate these issues might be to select datasets more carefully and create more structured circuits according to some specific problem domains. Using intermediate information of the state created by the circuit might alleviate this problem, and (as others have shown, e.g. [24]) performance issues could potentially be overcome when information of the state is used and the target domain is narrower. Based on our results, more research on the optimal ratio of circuit depth and computation of intermediate states is needed. Studying and designing other neural network architectures might also provide further insights and solutions to the challenges highlighted in this work. Another potential solution might be to learn and combine ''local'' layer-wise information. Finally, it might be useful to use a gradient through circuit simulation with respect to the rotations.

Since the intermediate states are scaling exponentially in the number of qubits, it might be important to study whether partial information from the quantum state might be enough to reduce the impact of exponential growth in the state. Further work might look at using only the parts of the quantum state that deviate the most from a given target state.

Based on our results, more work is needed to show the influence on other quantum computing related tasks that require at least a partial knowledge of the computed unitary. A more formal mathematical analysis of these problems would be beneficial in giving more insight into what classes of tasks might be possible and which not.

ACKNOWLEDGMENT

This publication has emanated from research conducted with the financial support of Taighde Éireann – Research Ireland under Grant 18/CRT/6183. For the purpose of Open Access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

REFERENCES

[1] X.-M. Zhang, T. Li, and X. Yuan, "Quantum state preparation with optimal circuit depth: Implementations and applications," *Phys. Rev. Lett.*, vol. 129, no. 23, Nov. 2022, Art. no. 230504, doi: 10.1103/physrevlett.129.230504.

[2] J. Iaconis, S. Johri, and E. Y. Zhu, "Quantum state preparation of normal distributions using matrix product states," *npj Quantum Inf.*, vol. 10, no. 1, p. 15, Jan. 2024, doi: 10.1038/s41534-024-00805-0.

[3] T. G. de Brugière, M. Baboulin, B. Valiron, and C. Allouche, "Quantum circuits synthesis using householder transformations," *Comput. Phys. Commun.*, vol. 248, Mar. 2020, Art. no. 107001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465519303388

[4] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum logic circuits," in *Proc. Conf. Asia South Pacific Design Autom. (ASP-DAC)*. New York, NY, USA: Association for Computing Machinery, 2005, p. 272, doi: 10.1145/1120725.1120847.

[5] E. Younis, K. Sen, K. Yelick, and C. Iancu, "QFAST: Conflating search and numerical optimization for scalable quantum circuit synthesis," in *Proc. IEEE Int. Conf. Quantum Comput. Eng. (QCE)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2021, pp. 232–243, doi: 10.1109/QCE52317.2021.00041.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[7] J. Jumper et al., "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, Aug. 2021, doi: 10.1038/s41586-021-03819-2.

[8] T. B. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[9] M. Weiden, E. Younis, J. Kalloor, J. Kubiatowicz, and C. Iancu, "Improving quantum circuit synthesis with machine learning," in *Proc. IEEE Int. Conf. Quantum Comput. Eng. (QCE)*. Los Alamitos, CA, USA: IEEE Computer Society, Sep. 2023, pp. 1–11, doi: 10.1109/qce57702.2023.00093.

[10] P. Selig, N. Murphy, D. Redmond, and S. Caton, "DeepQPrep: Neural network augmented search for quantum state preparation," *IEEE Access*, vol. 11, pp. 76388–76402, 2023.

[11] F. Fürrutter, G. Muñoz-Gil, and H. J. Briegel, "Quantum circuit synthesis with diffusion models," *Nature Mach. Intell.*, vol. 6, no. 5, pp. 515–524, May 2024, doi: 10.1038/s42256-024-00831-9.

[12] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes," *Nature Commun.*, vol. 9, no. 1, p. 4812, Nov. 2018, doi: 10.1038/s41467-018-07090-4.

[13] P. Swazinna, S. Udluft, D. Hein, and T. Runkler, "Comparing model-free and model-based algorithms for offline reinforcement learning," *IFAC-PapersOnLine*, vol. 55, no. 15, pp. 19–26, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896322010138

[14] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning," 2019, *arXiv:1907.02057*.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[16] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep RL for model-based control," *CoRR*, vol. abs/1802.09081, Aug. 2018. [Online]. Available: http://arxiv.org/abs/1802.09081

[17] Y. Chebotar, K. Hausman, M. Zhang, G. S. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, Aug. 2017, pp. 703–711.

[18] T. Che, Y. Lu, G. Tucker, S. Bhupatiraju, S. Gu, S. Levine, and Y. Bengio, "Combining model-based and model-free RL via multi-step control variates," 2018. [Online]. Available: https://openreview.net/forum?id=HkPCrEZ0Z

[19] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2015, pp. 1889–1897.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[21] T. Fösel, M. Y. Niu, F. Marquardt, and L. Li, "Quantum circuit optimization with deep reinforcement learning," 2021, *arXiv:2103.07585*.

[22] L. Moro, M. G. A. Paris, M. Restelli, and E. Prati, "Quantum compiling by deep reinforcement learning," *Commun. Phys.*, vol. 4, no. 1, p. 178, Aug. 2021, doi: 10.1038/s42005-021-00684-3.

[23] M. Y. Niu, S. Boixo, V. N. Smelyanskiy, and H. Neven, "Universal quantum control through deep reinforcement learning," *npj Quantum Inf.*, vol. 5, no. 1, p. 33, Apr. 2019, doi: 10.1038/s41534-019-0141-3.

[24] R. Porotti, A. Essig, B. Huard, and F. Marquardt, "Deep reinforcement learning for quantum state preparation with weak nonlinear measurements," *Quantum*, vol. 6, p. 747, Jun. 2022, doi: 10.22331/q-2022-06-28-747.

[25] D. Sobania, D. Schweim, and F. Rothlauf, "A comprehensive survey on program synthesis with evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 27, no. 1, pp. 82–97, Feb. 2023.

[26] J. K. Feser, S. Chaudhuri, and I. Dillig, "Synthesizing data structure transformations from input-output examples," *ACM SIGPLAN Notices*, vol. 50, no. 6, pp. 229–239, Jun. 2015, doi: 10.1145/2813885.2737977.

[27] A. Desai, S. Gulwani, V. Hingorani, N. Jain, A. Karkare, M. Marron, and S. Roy, "Program synthesis using natural language," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. (ICSE)*. New York, NY, USA: Association for Computing Machinery, May 2016, pp. 345–356, doi: 10.1145/2884781.2884786.

[28] M. L. Bowers, T. Olausson, L. Wong, G. Grand, J. B. Tenenbaum, K. Ellis, and A. Solar-Lezama, "Top-down synthesis for library learning," in *Proc. ACM Program. Lang.*, vol. 7, Jan. 2023, pp. 1182–1213, doi: 10.1145/3571234.

[29] A. Nazari, Y. Huang, R. Samanta, A. Radhakrishna, and M. Raghothaman, "Explainable program synthesis by localizing specifications," in *Proc. OOPSLA*, vol. 7, Oct. 2023, pp. 2171–2195. [Online]. Available: https://www.microsoft.com/en-us/research/publication/explainable-program-synthesis-by-localizing-specifications/

[30] K. Ellis, C. Wong, M. Nye, M. Sable-Meyer, L. Cary, L. Morales, L. Hewitt, A. Solar-Lezama, and J. B. Tenenbaum, "Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning," *Philos. Trans. Royal Soc. A, Math., Phys. Eng. Sci.*, vol. 381, no. 2251, 2023, Art. no. 20220050. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2022.0050

[31] N. G. de Bruijn, "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-Rosser theorem," *Indagationes Mathematicae (Proceedings)*, vol. 75, no. 5, pp. 381–392, 1972. [Online]. Available: https://www.sciencedirect.com/science/article/pii/1385725872900340

[32] S. Gulwani, "Automating string processing in spreadsheets using input-output examples," in *Proc. 38th Annu. ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang.*, Jan. 2011, pp. 317–330.

[33] E. Noriega-Atala, R. Vacareanu, G. Hahn-Powell, and M. A. Valenzuela-Escárcega, "Neural-guided program synthesis of information extraction rules using self-supervision," in *Proc. 1st Workshop Pattern-Based Approaches NLP Age Deep Learn.*, Oct. 2022, pp. 85–93. [Online]. Available: https://aclanthology.org/2022.pandl-1.10

[34] D. Ritchie, P. Guerrero, R. K. Jones, N. J. Mitra, A. Schulz, K. D. D. Willis, and J. Wu, "Neurosymbolic models for computer graphics," *Comput. Graph. Forum*, vol. 42, no. 2, pp. 545–568, May 2023, doi: 10.1111/cgf.14775.

[35] P. Guerrero, M. Hašan, K. Sunkavalli, R. Měch, T. Boubekeur, and N. J. Mitra, "MatFormer: A generative model for procedural materials," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 1–12, Jul. 2022, doi: 10.1145/3528223.3530173.

[36] T. Du, J. P. Inala, Y. Pu, A. Spielberg, A. Schulz, D. Rus, A. Solar-Lezama, and W. Matusik, "InverseCSG: Automatic conversion of 3D models to CSG trees," *ACM Trans. Graph.*, vol. 37, no. 6, pp. 1–16, Dec. 2018.

[37] G. Sharma, R. Goyal, D. Liu, E. Kalogerakis, and S. Maji, "CSGNet: Neural shape parser for constructive solid geometry," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 5515–5523.

[38] T. Perkis, "Stack-based genetic programming," in *Proc. 1st IEEE Conf. Evol. Comput. IEEE World Congr. Comput. Intell.*, Jun. 1994, pp. 148–153.

[39] L. Spector, D. M. Clark, I. Lindsay, B. Barr, and J. Klein, "Genetic programming for finite algebras," in *Proc. 10th Annu. Conf. Genetic Evol. Comput.*, Jul. 2008, pp. 1291–1298.

[40] L. Spector, J. Klein, and M. Keijzer, "The Push3 execution stack and the evolution of control," in *Proc. 7th Annu. Conf. Genetic Evol. Comput.*, Jun. 2005, pp. 1689–1696.

[41] E. Pantridge, T. Helmuth, and L. Spector, *Comparison of Linear Genome Representations for Software Synthesis*. Cham, Switzerland: Springer, 2020, pp. 255–274, doi: 10.1007/978-3-030-39958-0_13.

[42] T. Helmuth, N. F. McPhee, and L. Spector, "Program synthesis using uniform mutation by addition and deletion," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2018, pp. 1127–1134.

[43] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds., Berlin, Germany: Springer, 2004, pp. 502–518.

[44] A. Solar-Lezama, "Program synthesis by sketching," Ph.D. dissertation, Comput. Sci. Division, Univ. California, Berkeley, CA, USA, 2008. [Online]. Available: http://people.csail.mit.edu/asolar/papers/thesis.pdf

[45] A. Solar-Lezama, "Program sketching," *Int. J. Softw. Tools Technol. Transf.*, vol. 15, nos. 5–6, pp. 475–495, Oct. 2013, doi: 10.1007/s10009-012-0249-7.

[46] J. P. Inala, N. Polikarpova, X. Qiu, B. S. Lerner, and A. Solar-Lezama, "Synthesis of recursive ADT transformations from reusable templates," in *Proc. TACAS ETAPS*, Jan. 2017, pp. 247–263, doi: 10.1007/978-3-662-54577-5_14.

[47] J. K. Feser, S. Chaudhuri, and I. Dillig, "Synthesizing data structure transformations from input-output examples," in *Proc. 36th ACM SIGPLAN Conf. Program. Lang. Design Implement.*, 2015, pp. 229–239, doi: 10.1145/2737924.2737977.

[48] P.-M. Osera and S. Zdancewic, "Type-and-example-directed program synthesis," in *Proc. 36th ACM SIGPLAN Conf. Program. Lang. Design Implement.*, Jun. 2015, pp. 619–630, doi: 10.1145/2737924.2738007.

[49] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "DeepCoder: Learning to write programs," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Nov. 2016. [Online]. Available: https://openreview.net/forum?id=ByldLrqlx

[50] J. Devlin, J. Uesato, S. Bhupatiraju, R. Singh, A. Mohamed, and P. Kohli, "RobustFill: Neural program learning under noisy I/O," in *Proc. 34th Int. Conf. Mach. Learn.*, Aug. 2017, pp. 990–998. [Online]. Available: http://proceedings.mlr.press/v70/devlin17a.html

[51] W. Lee, K. Heo, R. Alur, and M. Naik, "Accelerating search-based program synthesis using learned probabilistic models," in *Proc. 39th ACM SIGPLAN Conf. Program. Lang. Design Implement.* New York, NY, USA: Association for Computing Machinery, Jun. 2018, pp. 436–449, doi: 10.1145/3192366.3192410.

[52] K. Ellis, A. Solar-Lezama, and J. B. Tenenbaum, "Unsupervised learning by program synthesis," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, Dec. 2015, pp. 973–981. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/b73dfe25b4b8714c029b37a6ad3006fa-Paper.pdf

[53] K. Ellis, D. Ritchie, A. Solar-Lezama, and J. B. Tenenbaum, "Learning to infer graphics programs from hand-drawn images," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, vol. 31. Red Hook, NY, USA: Curran Associates Inc., Feb. 2018, pp. 6059–6068.

[54] I. Polosukhin and A. Skidanov. (2018). *Neural Program Search: Solving Data Processing Tasks From Description and Examples*. [Online]. Available: https://openreview.net/forum?id=B1KJJf-R-

[55] R. Simmons-Edler, A. Miltner, and S. Seung, "Program synthesis through reinforcement learning guided tree search," 2018, *arXiv:1806.02932*.

[56] G. Claire, S. Sudhakaran, E. Najarro, and S. Risi, "Open-ended library learning in unsupervised program synthesis," in *Proc. Artif. Life Conf. Ghost Mach.*, Jul. 2023, p. 72. [Online]. Available: https://doi.org/10.1162/isal_a_00685

[57] S. Zhang, Z. Chen, Y. Shen, M. Ding, J. B. Tenenbaum, and C. Gan, "Planning with large language models for code generation," 2023, *arXiv:2303.05510*.

[58] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

[59] P. Rebentrost, M. Mohseni, and S. Lloyd, "Quantum support vector machine for big data classification," *Phys. Rev. Lett.*, vol. 113, no. 13, Sep. 2014, Art. no. 130503. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.113.130503

[60] S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum principal component analysis," *Nature Phys.*, vol. 10, no. 9, pp. 631–633, Sep. 2014, doi: 10.1038/nphys3029.

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

[62] M. Ben-Dov, D. Shnaiderov, A. Makmal, and E. G. D. Torre, "Approximate encoding of quantum states using shallow circuits," *npj Quantum Inf.*, vol. 10, no. 1, p. 65, Jul. 2024, doi: 10.1038/s41534-024-00858-1.

[63] J. Zylberman and F. Debbasch, "Efficient quantum state preparation with Walsh series," *Phys. Rev. A, Gen. Phys.*, vol. 109, no. 4, Apr. 2024, Art. no. 042401, doi: 10.1103/physreva.109.042401.

[64] K. Gui, A. M. Dalzell, A. Achille, M. Suchara, and F. T. Chong, "Spacetime-efficient low-depth quantum state preparation with applications," *Quantum*, vol. 8, p. 1257, Feb. 2024, doi: 10.22331/q-2024-02-15-1257.

[65] Y. Xiao, S. Nazarian, and P. Bogdan, "A stochastic quantum program synthesis framework based on Bayesian optimization," *Sci. Rep.*, vol. 11, no. 1, Jun. 2021, Art. no. 13138. [Online]. Available: https://doi.org/10.1038/s41598-021-91035-3

[66] P. Selig, N. Murphy, D. Redmond, and S. Caton, "A case for noisy shallow gate-based circuits in quantum machine learning," in *Proc. Int. Conf. Rebooting Comput. (ICRC)*, Nov. 2021, pp. 24–34.

[67] F. T. Miranda, P. P. Balbi, and P. C. S. Costa, "Synthesis of quantum circuits with an island genetic algorithm," Tech. Rep., 2021.

[68] F. M. Creevey, C. D. Hill, and L. C. L. Hollenberg, "GASP: A genetic algorithm for state preparation on quantum computers," *Sci. Rep.*, vol. 13, no. 1, Jul. 2023, Art. no. 11956. [Online]. Available: https://doi.org/10.1038/s41598-023-37767-w

[69] M. Weiden, E. Younis, J. Kalloor, J. Kubiatowicz, and C. Iancu, "Improving quantum circuit synthesis with machine learning," 2023, *arXiv:2306.05622*.

[70] J. Ho, A. N. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, vol. 33, J. Ho, A. Jain, and P. Abbeel, Eds., Curran Associates Inc., Jan. 2020, pp. 6840–6851. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf

[71] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

[72] P. Selig, N. Murphy, and S. Caton, "Quantum unitary dataset," Tech. Rep., 2025, doi: 10.5281/zenodo.14864254.

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, Jun. 2017, pp. 5998–6008. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[74] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[75] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, Jan. 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

PATRICK SELIG received the B.S. degree in physics and the M.Sc. degree in IT security from Ruhr-University Bochum, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree with the ML-Laboratories Program, University College Dublin. He was a Research Scientist with Technical University Dresden, from 2017 to 2018, and the Fraunhofer Institute for Material and Beam Technology IWS, till 2020. His research interests include machine learning, program induction, and quantum computing.

NIALL MURPHY received the B.Sc. degree (Hons.) in computer science and software engineering and the Ph.D. degree in theoretical computer science from Maynooth University, Ireland, in 2004 and 2010, respectively. He was a Postdoctoral Researcher with Universidad Politécnica de Madrid, Spain, from 2011 to 2013. He then was a Contract Scientist with the Microsoft Research Cambridge, U.K., from 2013 to 2018, and a Postdoctoral Researcher with the Sainsbury Laboratory, University of Cambridge, U.K., from 2015 to 2018. He was a Computational Biologist with Nuritas, Dublin, Ireland, from 2018 to 2020. He is currently a Quantum Algorithm Developer with Equal1 Laboratories, Dublin. His research focuses on studying the computational power of non-standard forms of computation, such as biological computing and quantum computing.

DAVID REDMOND received the B.E.E. degree from University College Cork, in 1988, and the M.Sc. degree in mathematics from University College Dublin, in 2017. He was with the Semiconductor Industry for more than 30 years developing high-volume consumer and industrial products from handset 2/3/4G RF transceivers to SerDes and Gigabit Ethernet products for such companies, such as AT&T Lucent, S3, Motorola, Freescale, Analog Devices operating at various levels of corporate research and development and product development. He has been with VP AI and Silicon for Equal1 Laboratories, since 2020. His research interests include machine learning and statistics.

SIMON CATON received the B.Sc. (Hons.) and Ph.D. degrees in computer science in 2005 and 2010, respectively. He was a Postdoctoral Researcher with Karlsruhe Institute of Technology, Germany, from 2010 to 2014, and as a Lecturer in data analytics with the National College of Ireland, from 2014 to 2019. He is currently an Assistant Professor with the School of Computer Science, University College Dublin, and a Steering Committee Member with the UCD Centre for Quantum Engineering, Science, and Technology (C-QuEST). His research interests include the applications of machine learning across the domains of social media, quantum computing, and parallel and distributed computing.