Efficient quantum circuit compression using Reinforcement Learning

Noah Berthusen[1,2]

[1] *Ames Laboratory, Ames, Iowa 50011, USA*
[2] *Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa 50011, USA.*

(Dated: November 16, 2021)

Computations by the current generation of noisy intermediate scale quantum (NISQ) computers are often plagued by errors such as decoherence and cross talk. Such errors severely limit the depth of NISQ quantum circuits, yet many quantum algorithms that show promise of a quantum speedup require deep circuits and prolonged coherence times. In this work, we propose leveraging Reinforcement Learning (RL) to intelligently build quantum circuits that can recreate given target states, given no information about the circuit used to construct them. The RL agent learns about the hidden system by receiving rewards based on local observables, calculated using the target state, and the fidelity of the final state. By constraining the depth of the circuits built by the agent, we hypothesize that this approach allows us to compress the depth of quantum circuits necessary to create the target state. One important application of our method is dynamic quantum simulation, where the target state is a time-evolved state using a given Hamiltonian and a Trotterized quantum circuit. Our method promises quantum simulations out to longer final times than are currently feasible on NISQ devices.

I. INTRODUCTION

While the advent of fault tolerant quantum computers seems to offer exponential speedups in some computational problems, a sufficiently effective error correcting scheme capable of mitigating the numerous sources of error on current devices has not yet been developed. Nonetheless, recent research is aiming to find speedups possible on Noisy Intermediate-Scale Quantum (NISQ) computers in the domain of quantum simulation of physics and chemistry [1]. For correlated electron materials, the exponential growth of the Hilbert space means classical simulations become intractable at only tens of qubits. Indeed, Richard Feynman famously quoted "Nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical..."

Numerous algorithms have been proposed for simulating electronic structure hamiltonians including the phase estimation algorithm (PEA) [2] and the variational quantum eigensolver (VQE) [3]. Due to the very long circuits and coherence times required by PEA, VQE has recently become the defacto method for doing digital quantum simulation. VQE is a hybrid quantum-classical algorithm where a quantum subroutine is run inside of a classical optimization loop. On the quantum computer, a parameterized ansatz $|\Psi(U(\vec{\boldsymbol{\theta}})\rangle$ is prepared and the expectation value measured. The classical computer then alters the parameters $\vec{\boldsymbol{\theta}}$ to attempt to minimize some loss function. Often, the ground state energy E_0 of a hamiltonian is the goal, although variational methods have also been developed to target excited states $E_i,\ i > 0$ or time-evolved states $e^{-iHt}|\Psi_0\rangle$ [4].

In the VQE algorithm, the quality of the simulation mainly depends on the choice of ansatz. Several options have been investigated, including static ansätze [5],[6] and adaptive methods [7],[8]. One work of specific interest for this paper utilizes Reinforcement Learning (RL) as the machinery behind classical optimization [9]. In that work, the structure of the ansatz circuit was fixed beforehand and only the parameters were altered. We make the distinction that in this work, the ansatz circuit is not predetermined; instead, it is constructed by the RL agent using a given set of gates. Much like the adaptive VQE techniques, dynamically building an ansatz for a given problem has the possibility to minimize useless/redundant features in the circuit and decrease circuit depth overall.

For the application of simulating the time-evolution of a quantum system, the Trotter formula provides a way to approximate $\exp(-iH_jt)$ for each non-commuting part of the hamiltonain $H = \sum_j H_j$.

$$e^{-iHt} = \lim_{n\to\infty} \Big(\prod_j e^{-iH_j t/n} \Big)^n \tag{1}$$

While accurate for large Trotter order and small step size, this results in circuits that are too long to be feasibly run on NISQ devices. A potential use for VQE as well as the technique described in this paper is to compress a variable length Trotter evolution into a fixed length circuit. In that case, more Trotter steps could be applied to the compressed state without reaching a problematic circuit depth.

In this paper, we will go through the mathematics of RL and see how it can be applied to preparing quantum states of a few qubits. With the groundwork laid, we will see how this technique can be applied to Trotter compression.

II. MARKOV DECISION PROCESSES

The machinery behind RL as a whole lies in Markov Decision Processes (MDPs) [10]. MDPs are represented as a four-tuple, $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$. At discrete time steps t, an

agent finds itself in a state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of all reachable states. In this state, the agent can choose to take an action $a_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of possible actions (potentially for that given state). Taking an action a_t in state s_t causes the agent to transition to state s_{t+1} with probability $\mathcal{P}(s_{t+1}|s_t, a_t)$. An environment that follows this transition probability rule is said to have the *Markov property*; that is, previous states and actions have no effect in determining s_{t+1} (and r_{t+1}). The reward $r_t \in \mathbb{R}$, is given to the agent after transitioning from s_t to s_{t+1}, $\mathcal{R}(s_{t+1})$.

In general, the goal of the agent in a MDP is to maximize the rewards it earns in the long run. This cumulative reward is defined as the expected *discounted reward* $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, where $\gamma \in [0, 1]$. Discounting the future reward by increasing γ makes the agent think more about the long-term effects of its actions instead of choosing actions that always result in the largest immediate reward r_{t+1}. More specifically, the agent wants to discover the optimal policy $\pi_*(a_t = a|s_t = s)$ that maximizes the *value function* $V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s]$. We consider the policy to be optimal if $V_*(s) \geq V_\pi(s)\ \forall s \in \mathcal{S}$ and policies π.

This value function can take a recursive form that allows it to be used in reinforcement learning and dynamic programming. In this form it is known as the *Bellman equation* for V_π. Note: the expectation value for a random variable X is defined as $\mathbb{E}(X) = \sum X \cdot Pr(X)$.

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi\Big[G_t\Big|s_t = s\Big] \\ &= \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\Big|s_t = s\Big] \\ &= \mathbb{E}_\pi\Big[R_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}\Big|s_t = s\Big] \\ &= \mathbb{E}_\pi\Big[R_{t+1}\Big] + \gamma\mathbb{E}_\pi\Big[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2}\Big|s_t = s\Big] \\ &= \sum_a \pi(a|s)\sum_{s',r}\mathcal{P}(s', r|s, a)\mathcal{R}(s', s, a) + \\ &\qquad \gamma\sum_a \pi(a|s)\sum_{s',r}\mathcal{P}(s', r|s, a)V_\pi(s') \\ &= \sum_a \pi(a|s)\sum_{s',r}\mathcal{P}(s', r|s, a)\Big(\mathcal{R}(s', s, a) + \gamma V_\pi(s')\Big) \end{aligned} \tag{2}$$

In this form, it is easy to see that the value function is indeed just an expectation value; the probability of a state transition occurring is multiplied with the immediate reward plus the discounted value of state s'. Summing over all possible state transitions gives $V_\pi(s)$. We can similarly define the *action-value function* $Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a]$ which describes the value of taking action a in state s. If $\mathcal{P}(s', r|s, a)$ is known $\forall s, a$, then a system of $|\mathcal{S}|$ with $|\mathcal{S}|$ variables can be solved to determine $V_\pi(s)$. However, as the size of state spaces grow, this becomes computationally infeasible. So we turn to *iterative policy evaluation*, an algorithm that uses the Bellman equation to update $V_\pi(s)$ and eventually converge as $k \to \infty$. It is important to note that this algorithm does not determine the optimal policy π_*, it just determines the value of each state with respect to the current policy.

$$\begin{aligned} V_{k+1}(s) &= \mathbb{E}[R_{t+1} + \gamma V_k(s_{t+1}|s_t = s)] \\ &= \sum_a \pi(a|s)\sum_{s',r}\mathcal{P}(s', r|s, a)\Big(\mathcal{R}(s', s, a) + \gamma V_k(s')\Big) \end{aligned} \tag{3}$$

An example of the policy evaluation algorithm is shown in Fig. 2. Policy evaluation does not give us any insight into the changes that we should make to the current policy in order to maximize $V_\pi(s)\ \forall s \in \mathcal{S}$ and policies π. The question to ask is then would it be better to choose a different action $a \neq \pi(s)$ for some state s. Formally, this is asking whether $Q_\pi(s, \pi'(s)) \geq V_\pi(s)$ for policies π and π', where π' is an updated policy. If this previous statement holds, then the *policy improvement theorem* says that $V_{\pi'}(s) \geq V_\pi(s)\ \forall s \in \mathcal{S}$; that is, the policy π' is equal, or better than, π. This process is called *policy improvement* and is described below

$$\begin{aligned} \pi'(s) &= \text{argmax}_a Q_\pi(s, a) \\ &= \text{argmax}_a \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(s_{t+1})|s_t = s, a_t = a] \\ &= \text{argmax}_a \sum_{s',r}\mathcal{P}(s', r|s, a)\Big(\mathcal{R}(s', s, a) + \gamma V_\pi(s')\Big) \end{aligned} \tag{4}$$

Here argmax_a describes the action for which $Q_\pi(s, a)$ is at its maximum. By looping over all states and actions and greedily selecting the action that maximizes the action-value function in the short run, we ensure that the policy improvement theorem is satisfied. We alternate iterative policy evaluation and policy improvement in an algorithm called *policy iteration* in order to find the optimal policy. We continue alternating these two steps until $|V_{k+1}(s) - V_k(s)| < \epsilon$ for some $\epsilon > 0$ and $\pi'(s) = \pi(s)\ \forall s \in \mathcal{S}$. When this occurs, we know that $V_{k+1}(s) = V_*(s)$ and $\pi'(s) = \pi_*(s)$.

To apply this framework in the context of the dynamics of a single qubit, certain thought has to be given to ensure that we can use policy evaluation and policy iteration to determine the optimal policy. By carefully defining the MDP four-tuple, $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, we will see that finding optimal gate sequences to approximate a goal state can be modeled and solved as a MDP [11].

III. SINGLE-QUBIT DYNAMICS

A. MDP Specification

Pure single-qubit states are continuous in $\mathbb{C}^2$, and while there are methods of dealing with continuous state spaces in MDPs, we choose to discretize the state space instead.

There is a one-to-one mapping from $\mathbb{C}^2$ to $\mathbb{R}^3$, where the points in $\mathbb{R}^3$ are commonly represented by the Bloch sphere. For any pure single-qubit state $|\Psi\rangle = a|0\rangle + b|1\rangle$, where a, b are complex coefficients satisfying $|a|^2+|b|^2 = 1$, we can represent the same state as

$$|\Psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle \tag{5}$$

where $0 \leq \theta \leq \pi$ and $-\pi \leq \phi \leq \pi$. Thus we only need two real numbers to completely and uniquely describe a pure single-qubit state. For any arbitrary a, b that make up a quantum state(vector), it is simple to determine the corresponding θ, ϕ. With a and b in polar form, factor out a's phase so that a is entirely real. Then $\theta = 2 \cdot \arccos(a)$ and $\phi = \arctan(b_{\text{imag}}/b_{\text{real}})$, the complex argument of b. To discretize the state space, we break up θ and ϕ into intervals with lengths of $\epsilon = \pi/k$, where k is some integer. The state space $\mathcal{S}$ is then made up of $\sim 2k^2$ states that are discretized as $n\epsilon < \theta \leq (n+1)\epsilon,\ 0 \leq n \leq k$ and $(m-k)\epsilon < \phi \leq (m-k+1)\epsilon,\ 0 \leq m \leq 2k$. We also distinguish the two poles as distinct states; that is, a qubit is in $|0\rangle$ when $\theta < \epsilon$, regardless of ϕ. Similarly, a qubit is in the $|1\rangle$ state when $\theta < \pi - \epsilon$, independent of ϕ.

For the action space $\mathcal{A}$, we choose a number of non-parameterized unitaries that can be applied to a single qubit. Parameterized gates such as $RX(\theta)$ and $RY(\theta)$ are defined by the continuous variable θ. There are also ways to deal with continuous action spaces in MDPs, such as discretizing the parameter, but we choose the discrete set {H, T, I} for ease of computation.

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\ T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}\ I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{6}$$

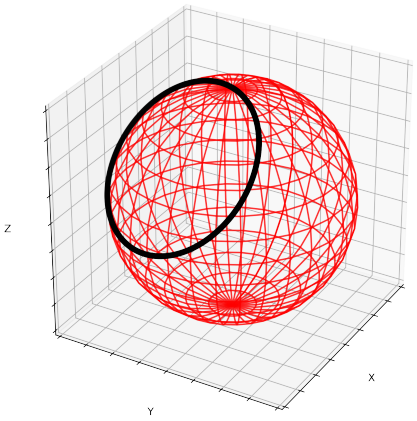

FIG. 1. Rotations $R_{\hat{n}}(\alpha)$ caused by $(HT)^n|0\rangle$.

Since we are finding π_* using policy iteration, we must know $\mathcal{P}$, $\mathcal{P}(s', r|s, a)\ \forall s, a$. In general, it is not the case that you know the exact dynamics of the system, but in this case the it is easy to simulate the environment and to create a probability distribution. Applying unitaries to a quantum state is a deterministic action, but applying a string of unitaries is not a Markovian process. For a single-qubit system to have the Markov property, i.e. only the current state and action determine the next state, we have to 'shuffle' the qubit statevector within the current MDP state before applying an action. Since we discretized the state space, applying a unitary to a qubit in an arbitrary state s—by choosing some k, m, n—can result in a variety of next states s'. So to find $\mathcal{P}(s', r|s, a)$, all we have to do is randomly choose a number of $s \in \mathcal{S}$, apply each action (unitary) to the state and record which state it ends up in. If we sample enough random qubit states, we can create an adequate probability distribution that allows value and policy iteration to converge to π_*.

The last piece of the MDP to define is the reward structure $\mathcal{R}$. One measure of accuracy that represents the closeness of two quantum states $|\Psi\rangle, |\Phi\rangle$ is the fidelity $|\langle\Psi|\Phi\rangle|^2$. While an intuitive choice, it is expensive to calculate on real devices, and it is overkill for our MDP. We opt to use the simple reward function $\mathcal{R}(s) = 0$ if s is the goal state, and $\mathcal{R}(s) = -1$ for all other states. This means that $V_*(s) = 0$ for the goal state, and $V_*(s) < 0$ for all other states.

B. Optimal Quantum Circuits

With the MDP defined in the previous section, we can specify the goal state and available gateset and perform policy iteration to find the optimal policy. In this case, this would be the shortest quantum circuit that accurately approximates the goal state. We target the state $(HT)^n|0\rangle$ for some $n > 0$. This circuit results in a rotation of the Bloch sphere about an axis $\vec{n} = (\cos\frac{\pi}{8}, \sin\frac{\pi}{8}, \cos\frac{\pi}{8})$ through an angle $\theta = \cos^2\frac{\pi}{8}$ as shown in Fig. 1. This angle θ is irrational, so $(HT)^n \neq (HT)^m,\ \forall m, n$. Interestingly, this circuit can be used to approximate any single qubit unitary operator [2]. After running policy iteration until convergence, we use the following algorithm to find the optimal gate sequence: Start in a random quantum state within the $|0\rangle$ MDP state. Then follow the actions given by π_* until we reach the goal region. In case this process does not converge to the goal region, we shuffle intermediate quantum states within their respective MDP states until convergence. In Table. I we investigate n up to 10^{10} and find that it accurately recreates the goal state in far fewer gates. The fidelity of the recreation is constrained by the discretization size we specified in Sec. III A, but in most cases we find the MDP to find the goal state to be approximated almost perfectly. This approximation could be improved even further by increasing k, which would effectively increase the minimum fidelity between any two random quantum states within one MDP state.

We apply the same process to recreating random single qubit states generated by random unitaries applied to the $|0\rangle$ state [12]. We find that an average fidelity $|\langle\psi_g|\psi_f\rangle|^2$ of 0.997 is able to be reached using a sequence of 8.64 gates from our gateset. For some cases, applying the

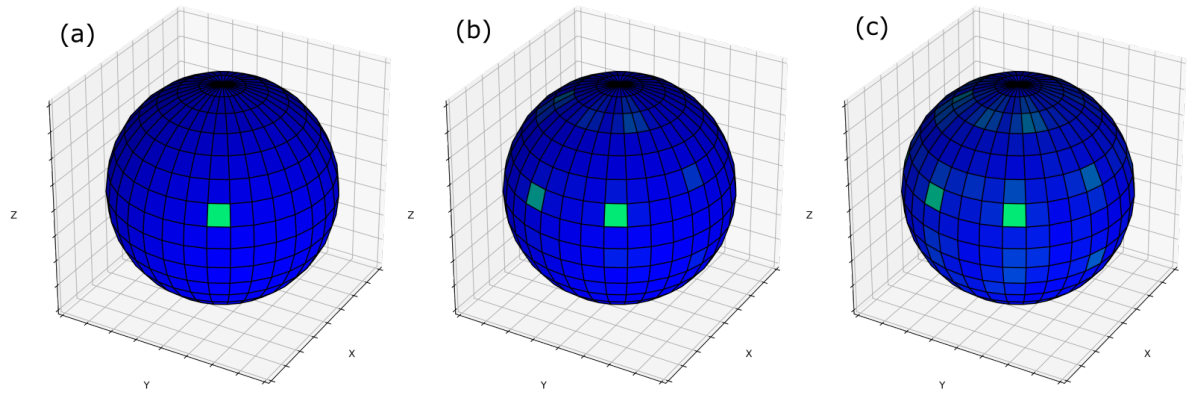

FIG. 2. Policy evaluation for π_* when targeting $(HT)^n|0\rangle$, $n = 10^2$. (a) Policy evaluation after $k = 1$ iteration. The highlighted state is the goal state with a value of 0; all other states have values around -1. The brighter the color of the state, the higher its value is and the more advantageous it is to be in that state under the current policy. (b) Policy evaluation after $k = 5$ iterations. By now, states from which the goal state is reachable in a few action have an increased value. (c) Policy evaluation after $k = 62$ evaluations. Here the algorithm has converged to V_*.

sequences to the pure $|0\rangle$ state was unable to reach the goal region, so in those situations a random quantum state within $|0\rangle$ MDP state was chosen.

n	Gate sequence	$\lvert\langle\psi_g\rvert\psi_f\rangle\rvert^2$
10^2	HTHTH	0.992
10^3	HTH	0.988
10^4	HTH	0.992
10^5	HTH	0.995
10^6	HTTHT	0.998
10^7	HTHTHTHTTTH	0.990
10^8	I	0.999
10^9	I	0.996
10^{10}	HTHTHTH	0.972

TABLE I. Optimal gate sequences to prepare $(HT)^n|0\rangle$. Gate sequences are applied from right to left. The fidelity $|\langle\psi_g|\psi_f\rangle|^2$ is found by applying the gate sequence to $|0\rangle$, as opposed to a random state in the northern pole.

IV. MULTI-QUBIT DYNAMICS

While this methodology works well for the single-qubit case, scaling up to multiple qubits is difficult. For one, there is no Bloch sphere representation for more than one qubit, so we must find another way to parameterize the Hilbert space. One idea is to represent the state space as the group of $\mathbf{SU}(N)$ matrices, the *special unitary group*. $\mathbf{SU}(N)$ is the group of $N \times N$ unitary matrices U with $\det U = 1$.

It is possible to parameterize $\mathbf{SU}(N)$. In general, the number of parameters is polynomial in N; that is, $N^2 - 1$ matrices are needed to form a basis for $\mathbf{SU}(N)$, and each of these matrices has a corresponding parameter. Each of the $N^2 - 1$ parameters is between 0 and π, but many are bounded above by a factor of π. If we impose a discretization of size $\epsilon = \pi/k$, then we have k^{N^2-1} as an upper bound on the number of MDP states. Even for the two qubit case, $\mathbf{SU}(4)$, $|\mathcal{S}| \approx k^{15}$, an unmanageable number of states for which to create transition matrices and solve using policy iteration. Another option when designing the state space is to represent the density matrix or unitary $U \in \mathbf{SU}(N)$ as two matrices containing the real and imaginary parts. While in this method the matrix scales exponentially with the number of qubits, it is straightforward to implement and feasible for a small number of qubits. As such, we will focus on this latter representation as we look at learning multi-qubit dynamics. With this state space, there are again too many states to apply policy iteration; however, we have approximate solution methods that allow us to learn the optimal policy for a MDP.

A. Proximal Policy Optimization

As state and action spaces get larger, it becomes infeasible or impossible to know and store the information needed to know the transition probabilities. Because of this, tabular methods such as policy iteration become unusable. This means that we no longer have the ability to query a simply array to find π_* for a given state. However, generalization methods exist that allow us to create *function approximations* of the value or action-value functions. For a set of parameters $\theta \in \mathbb{R}^d$, the policy function now becomes $\pi(a|s,\theta) = P\{a_t = a|s_t = s, \theta_t = \theta\}$. These parameters can parameterize an arbitrary function, like a simple linear regressor or a more complex model such as a convolutional neural network (CNN). We will focus on applying the latter to the problem of multi-qubit state

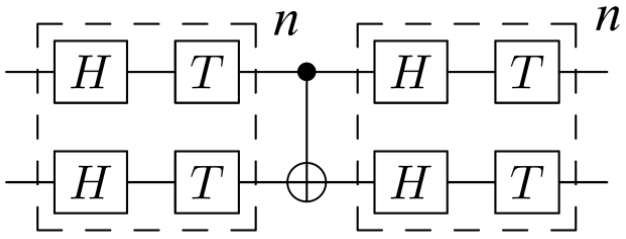

FIG. 3. Circuit that includes entanglement for testing PPO.

preparation. We can apply Proximal Policy Optimization (PPO) to approximate the value function and inform the agent of the best action to take in a given state. This algorithm is implemented in the Gym package for Python, so we can create an environment to represent the action of gates on qubits and train an agent. Other packages such as Qiskit or Pyquil can be used for this purpose, but we chose to write our own quantum simulator that does the matrix multiplication. A difference from the single qubit MDP is the specification of the reward function; in the multi-qubit case, there is no goal state for the agent to enter into. So we exchange the reward function for simply the fidelity of the goal state and the state the agent is currently in, $|\langle\psi_g|\psi_f\rangle|^2$. We use the same gate set as before on each qubit, but we also include a CNOT gate between each pair of qubits.

We test the method on three example circuits to recreate. The first circuit we test is the Bell state for n qubits. This can be implemented in only n gates—a Hadamard on the first qubit and CNOTs between it and the rest. As such, the agent can easily find the correct circuit for 2, 3, and 4 qubits. A slightly more complicated test is the circuit $[(HT)^m]^{\otimes n}$. This is essentially the same problem as the single-qubit case, but done in parallel for each qubit. After running the PPO algorithm for 100,000 time steps, we find that the agent can recreate a state with ≥ 0.80 fidelity with the goal circuit for all m tested. Finally, we look at the circuit depicted in Fig. 3. This is similar to the circuit in the second example, but we add a CNOT to introduce entanglement between the two qubits. Adding this entanglement appears to substantially increase the difficulty of the problem, as the agent was unable to recreate any state with a fidelity much higher than 0.50 unless the solution was trivial. These tests were only for the two qubit case, so extending the investigation further to include three or four qubits aims to yield even poorer results.

V. CONCLUSION

In this report we have looked at the feasibility of using RL as a way to recreate arbitrary quantum states and compress them into shorter circuits. Using the machinery of MDPs, we saw that a simple model can accurately specific circuits as well as random unitaries. As we looked into multi-qubit dynamics, we found that the PPO algorithm generally has difficult in training an agent to recreate even semi-complex quantum states. This could be a limitation of the algorithm itself, or its description of the problem. For one, the agent might learn better if it had more information about its environment, such as entanglement entropy or the closeness of density matrices. Adding this information to the agent can be a focus of future work. With these results, we cannot recommend the use of RL for this task. Other variational methods such as VQE have been shown to be more robust and capable for similar uses. This is not to say that RL is useless in the field of quantum computation; uses such as controlling hardware [13] have been shown to be practical. In any case, as we continue developing the the NISQ era, the more hybrid quantum-classical algorithms we can find, the better. Find the code from this report here.

[1] J. Preskill, Quantum **2**, 79 (2018).
[2] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, 2000).
[3] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, New Journal of Physics **18**, 023023 (2016).
[4] Y.-X. Yao, N. Gomes, F. Zhang, T. Iadecola, C.-Z. Wang, K.-M. Ho, and P. P. Orth, "Adaptive variational quantum dynamics simulations," (2020), arXiv:2011.00622 [quant-ph].
[5] D. Wecker, M. B. Hastings, and M. Troyer, Physical Review A **92** (2015), 10.1103/physreva.92.042303.
[6] R. Wiersema, C. Zhou, Y. de Sereville, J. F. Carrasquilla, Y. B. Kim, and H. Yuen, PRX Quantum **1**, 020319 (2020).
[7] H. R. Grimsley, S. E. Economou, E. Barnes, and N. J. Mayhall, Nature Communications **10** (2019), 10.1038/s41467-019-10988-2.
[8] H. L. Tang, V. O. Shkolnikov, G. S. Barron, H. R. Grimsley, N. J. Mayhall, E. Barnes, and S. E. Economou, "qubit-adapt-vqe: An adaptive algorithm for constructing hardware-efficient ansatze on a quantum processor," (2020), arXiv:1911.10205 [quant-ph].
[9] A. Bolens and M. Heyl, "Reinforcement learning for digital quantum simulation," (2020), arXiv:2006.16269 [quant-ph].
[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. (The MIT Press, 2018).
[11] M. S. Alam, "Quantum logic gate synthesis as a markov decision process," (2019), arXiv:1912.12002 [quant-ph].
[12] F. Mezzadri, (2006), arXiv:math-ph/0609050.
[13] M. Y. Niu, S. Boixo, V. Smelyanskiy, and H. Neven, "Universal quantum control through deep reinforcement learning," (2018), arXiv:1803.01857 [quant-ph].