Quantum Architecture Search via Deep Reinforcement Learning

En-Jui Kuo,[1,2,*] Yao-Lung L. Fang,[3,†] and Samuel Yen-Chi Chen[3,‡]

[1] *Department of Physics, University of Maryland, College Park, MD 20742, USA*

[2] *Joint Quantum Institute, NIST/University of Maryland, College Park, MD 20742, USA*

[3] *Computational Science Initiative, Brookhaven National Laboratory, Upton, NY 11973, USA*

(Dated: April 19, 2021)

Abstract

Recent advances in quantum computing have drawn considerable attention to building realistic application for *and* using quantum computers. However, designing a suitable quantum circuit architecture requires expert knowledge. For example, it is non-trivial to design a quantum gate sequence for generating a particular quantum state with as fewer gates as possible. We propose a quantum architecture search framework with the power of deep reinforcement learning (DRL) to address this challenge. In the proposed framework, the DRL agent can only access the Pauli-X, Y, Z expectation values and a predefined set of quantum operations for learning the target quantum state, and is optimized by the advantage actor-critic (A2C) and proximal policy optimization (PPO) algorithms. We demonstrate a successful generation of quantum gate sequences for multi-qubit GHZ states without encoding any knowledge of quantum physics in the agent. The design of our framework is rather general and can be employed with other DRL architectures or optimization methods to study gate synthesis and compilation for many quantum states.

* kuoenjui@umd.edu

† leofang@bnl.gov

‡ ychen@bnl.gov

I. INTRODUCTION

Recently, reinforcement learning (RL) [1] has found tremendous success and demonstrated a human- or superhuman- level of capabilities in a wide range of tasks, such as mastering video games [2–5] and even the game of Go [6, 7]. With such success, it is natural to consider applying such techniques to scientific areas that require sophisticated control capabilities. Indeed, RL has been used to study quantum control [8–14], quantum error correction [15–19] and the optimization of variational quantum algorithms [20–23].

RL has also been applied to automatically building a deep learning architecture for a given task. This is the so-called *neural architecture search* [24] and has been proven possible in a wide variety of machine learning (ML) tasks [25–31]. The core idea is to train an RL agent to sequentially put in different deep learning components (e.g., convolutional operations, residual connections, pooling and so on) and then evaluate the model performance. Although the concept is simple, several recent studies have reported reaching a state-of-the-art performance [32] and beating the best human-crafted DL models.

Quantum computing has promised exponential speedups for several hard computational problems otherwise intractable on a classical computer [33, 34], such as factorizing large integers [35] and unstructured database search [36]. Recent studies in variational quantum algorithms (VQA) have applied quantum computing to many scientific domains, including molecular dynamical studies [37], quantum optimization [38, 39] and various quantum machine learning (QML) applications such as regression [40–42], classification [41, 43–57], generative modeling [58–62], deep reinforcement learning [63–69], sequence modeling [40, 70, 71], speech recognition [72], metric and embedding learning [73, 74], transfer learning [47] and federated learning [75]. However, designing a quantum circuit to solve a specific task is non-trivial, as it demands domain knowledge and sometimes extraordinary insights.

In this study, we investigate the potential of training an RL agent to search for a *quantum circuit architecture* for generating a desired quantum state. In this work, we present a new quantum architecture search framework powered by deep reinforcement learning (DRL). As shown in Figure 1, the proposed framework includes an RL agent interacting with a quantum computer or quantum simulator. The RL agent will sequentially generate an output action, which is a candidate of the quantum gate or operation placed on the circuit. The built circuit is evaluated against certain metrics, such as the *fidelity*, to check if it actually reaches the

goal. The reward is calculated based on the fidelity and sent back to the RL agent. The procedure is carried out iteratively to train the RL agent.

Our contributions are the following:

- Provide a framework for the study of quantum architecture search.
- Demonstrate building a quantum circuit step-by-step via deep reinforcement learning without any knowledge in physics.

The paper is organized as follows. In Section II, we introduce the RL background knowledge used in this work. In Section III we introduce the quantum architectures that our agent will search. In Section IV, we describe the experimental procedures and results in details. Finally we discuss the results in Section V and conclude in Section VI.

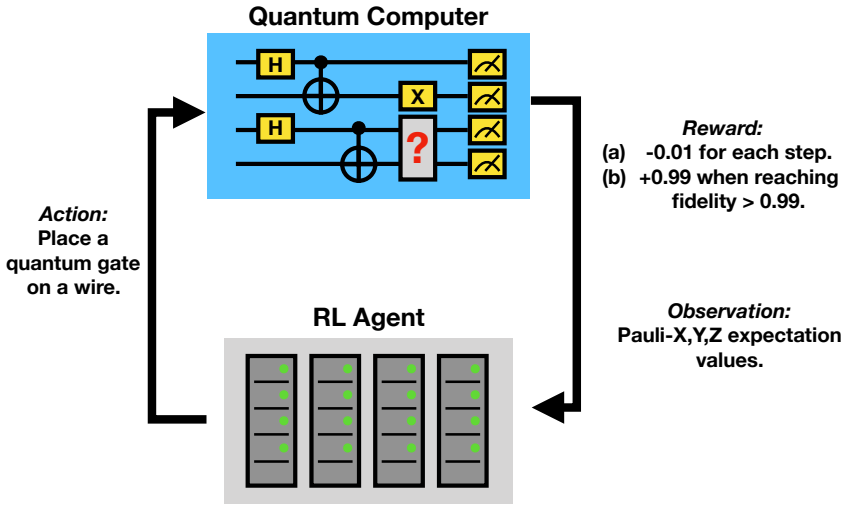


FIG. 1: **Overview of DRL for our quantum architecture search framework.** The proposed quantum architecture search framework consists of two major components. First is a quantum computer or quantum simulator. In this work, we use a quantum simulator with and without noise. Second is an RL agent interacting with the quantum computer. In each time step, the RL agent will generate an action for the quantum computer. The action specifies a quantum operation to be added to the system. Then the fidelity of the quantum circuit is evaluated to determine the *reward* to be sent back to the agent. In addition, Pauli-X, Y and Z expectation values are also fed back to the agent. The RL agent will then be updated based on these information.

II. REINFORCEMENT LEARNING

Reinforcement learning (RL) is a machine learning paradigm in which an *agent* learns how to make decisions via interacting with the environments [1]. Concretely speaking, the *agent* interacts with an *environment* $\mathcal{E}$ over a number of discrete time steps. At each time step t, the agent receives a *state* or *observation* s_t from the environment $\mathcal{E}$ and then chooses an *action* a_t from a set of possible actions $\mathcal{A}$ according to its *policy* π. The policy π is a function which maps the state or observation s_t to action a_t. In general, the policy can be stochastic, meaning that given a state s, the action output can be a probability distribution $\pi(a_t|s_t)$ conditioned on s_t. After executing the action a_t, the agent receives the state of the next time step s_{t+1} and a scalar *reward* r_t. The process continues until the agent reaches the terminal state or a pre-defined stopping criteria (e.g. the maximum steps allowed). An *episode* is defined as an agent starting from a randomly selected initial state and following the aforementioned process all the way through the terminal state or reaching a stopping criteria.

We define the total discounted return from time step t as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where γ is the discount factor that lies in $(0, 1]$. In principle, γ is provided by the investigator to control how future rewards are weighted to the decision making function. When a large γ is considered, the agent weighs the future reward more heavily. On the other hand, with a small γ, future rewards are quickly ignored and immediate reward will be weighted more. The goal of the agent is to maximize the expected return from each state s_t in the training process. The *action-value function* or *Q-value function* $Q^\pi(s, a) = \mathbb{E}[R_t|s_t = s, a]$ is the expected return for selecting an action a in state s based on policy π. The optimal action value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ gives a maximal action-value across all possible policies. The value of state s under policy π, $V^\pi(s) = \mathbb{E}\left[R_t|s_t = s\right]$, is the agent's expected return by following policy π from the state s. Various RL algorithms are designed to find the policy which can maximize the value function. The RL algorithms which maximize the value function are called *value-based* RL.

A. Policy Gradient

In contrast to the *value-based* RL, which learns the value function and use it as the reference to generate the decision on each time-step, there is another kind of RL method called *policy gradient*. In this method, the policy function $\pi(a|s;\theta)$ is parameterized with the parameters θ. The θ will then be subject to the optimization procedure which is *gradient ascent* on the expected total return $\mathbb{E}[R_t]$. One of the classic examples of policy gradient algorithm is the REINFORCE algorithm [76]. In the standard REINFORCE algorithm, the parameters θ are updated along the direction $\nabla_\theta \log \pi\left(a_t|s_t;\theta\right) R_t$, which is the unbiased estimate of $\nabla_\theta \mathbb{E}\left[R_t\right]$. However, the policy gradient method suffers from large variance of the $\nabla_\theta \mathbb{E}\left[R_t\right]$, making the training very hard. To reduce the variance of this estimate and keep it unbiased, one can subtract a learned function of the state $b_t(s_t)$, which is known as the *baseline*, from the return. The result is therefore $\nabla_\theta \log \pi\left(a_t|s_t;\theta\right)\left(R_t - b_t\left(s_t\right)\right)$.

B. Advantage Actor-Critic (A2C)

A learned estimate of the value function is a common choice for the baseline $b_t(s_t) \approx V^\pi(s_t)$. This choice usually leads to a much lower variance estimate of the policy gradient. When one uses the approximate value function as the baseline, the quantity $R_t - b_t = Q(s_t, a_t) - V(s_t)$ can be seen as the *advantage* $A(s_t, a_t)$ of the action a_t at the state s_t. Intuitively, one can see this advantage as "how good or bad the action a_t compared to the average value at this state $V(s_t)$." For example, if the $Q(s_t, a_t)$ equals to 10 at a given time-step t, it is not clear whether a_t is a good action or not. However, if we also know that the $V(s_t)$ equals to, say 2 here, then we can imply that a_t may not be bad. Conversely, if the $V(s_t)$ equals to 15, then the advantage is $10 - 15 = -5$, meaning that the Q value for this action a_t is well below the average $V(s_t)$ and therefore that action is not good. This approach is called *advantage actor-critic* (A2C) method where the policy π is the actor and the baseline which is the value function V is the critic [1].

C. Proximal Policy Optimization (PPO)

In the policy gradients method, we optimize the policy according to the *policy loss* $L_{\text{policy}}(\theta) = \mathbb{E}_t[-\log \pi\left(a_t \mid st;\theta\right)]$ via gradient descent. However, the training itself may suf-

fer from instabilities. If the step size of policy update is too small, the training process would be too slow. On the other hand, if the step size is too high, there will be a high variance in the training. The proximal policy optimization (PPO) [77] fixes this problem by limiting the policy update step size at each training step. The PPO introduces the loss function called *clipped surrogate loss function* that will constraint the policy change a a small range with the help of a clip. Consider the ratio between the probability of action a_t under current policy and the probability under previous policy $q_t(\theta) = \frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})}$. If $q_t(\theta) > 1$, it means the action a_t is with higher probability in the current policy than in the old one. And if $0 < q_t(\theta) < 1$, it means that the action a_t is less probable in the current policy than in the old one. Our new loss function can then be defined as $L_{\text{policy}}(\theta) = \mathbb{E}_t[q_t(\theta)A_t] = \mathbb{E}_t[\frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})}A_t]$, where $A_t = R_t - V(s_t;\theta)$ is the advantage function. However, if the action under current policy is much more probable than in the previous policy, the ratio q_t may be large, leading to a large policy update step. To circumvent this problem, the original PPO algorithm [77] adds a constraint on the ratio, which can only be in the range 0.8 to 1.2. The modified loss function is now $L_{\text{policy}}(\theta) = \mathbb{E}_t[-min(q_tA_t, clip(q_t, 1-C, 1+C)A_t)]$ where the C is the clip hyperparameter (common choice is 0.2). Finally, the value loss and entropy bonus are added into the total loss function as usual: $L(\theta) = L_{\text{policy}} + c_1L_{\text{value}} - c_2H$ where $L_{\text{value}} = \mathbb{E}_t[\|R_t - V(s_t;\theta)\|^2]$ is the value loss and $H = \mathbb{E}_t[H_t] = \mathbb{E}_t[-\sum_j \pi(a_j \mid s_t;\theta)\log(\pi(a_j \mid s_t;\theta))]$ is the entropy bonus which is to encourage exploration.

III. PROBLEM SETUP

Below we describe in detail the problem we aim to solve using DRL. Given an initial state $|0\cdots0\rangle$ and the target state, the goal is to produce a quantum circuit which transforms the initial state to the target state within certain error tolerance. We use the Pauli measurements as observations, a natural choice in quantum mechanics. We then use various RL algorithms to achieve our goal. The overall scheme is shown in Figure 1. Specifically, the environment $\mathcal{E}$ is the quantum computer or quantum simulator. In this work, we use a quantum simulator since currently it is not yet practical to train tens of thousands of episodes on a cloud-based quantum device. The RL agent, hosted on a classical computer, interacts with the environment $\mathcal{E}$. In each time step, the RL agent chooses an action a from the possible set of actions $\mathcal{A}$, which consists of different quantum operations (one- and two- qubit gates).

After the RL agent updates the quantum circuit with the chosen action, the environment $\mathcal{E}$ executes the new circuit and calculates the *fidelity* to the given target state. If the fidelity reaches a pre-defined threshold, the episode ends and a large positive reward is given to the RL agent. Otherwise, the RL agent receives a small negative reward. The states or observations which the environment $\mathcal{E}$ returns to the RL agent are Pauli measurements on each qubit, so for an n-qubit system the dimension of the observations is $3n$. The procedure continues until the agent reaches either the desired threshold or the maximum allowed steps. RL algorithms like A2C and PPO are employed to optimize the agent. Next, we discuss in detail the mathematical setting of our problem.

A. Mathematical formulation of the problem

Suppose we are given the number of qubits $n \in \mathbb{N}$, the initial quantum state $|0\rangle^{\otimes^n}$, the target state $|\psi\rangle$, the tolerance error $\epsilon \geq 0$, and a set of gates $\mathbb{G}$. Our goal is to find a quantum circuit $\mathcal{C} : |0\rangle^{\otimes^n} \to |\psi\rangle$ so that our DRL architecture serves as a function $\mathcal{F}$:

$$\mathcal{F} : (|0\rangle^{\otimes^n}, |\psi\rangle, \epsilon, \mathbb{G}) \to \mathcal{C} \tag{1}$$

such that $1 \geq D(|\psi\rangle, \mathcal{C}(|0\rangle^{\otimes^n})) \geq 1 - \epsilon$, where $\mathcal{C}$ is composed of gates $g \in \mathbb{G}$ and D is a distance metric between two quantum states (larger is better). In this paper, we use the fidelity [78] to be our distance D. Given two density operators ρ and σ (see also Sec. IV A 3), the fidelity is generally defined as the quantity $F(\rho, \sigma) = \left[\mathrm{tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right]^2$. In the special case where ρ and σ represent pure quantum states, namely, $\rho = |\psi_\rho\rangle\langle\psi_\rho|$ and $\sigma = |\psi_\sigma\rangle\langle\psi_\sigma|$, the definition becomes the inner product of two states: $F(\rho, \sigma) = |\langle\psi_\rho|\psi_\sigma\rangle|^2$.

B. Multi-qubit entangled states as target

To validate that the proposed DRL pipeline can be applied to quantum architecture search, it is best to check if multi-qubit entanglement can be generated as expected. To this end, we target the generation of two kinds of quantum states: Bell state and Greenberger–Horne–Zeilinger (GHZ) state.

A Bell state reaches maximal two-qubit entanglement,

$$|\mathrm{Bell}\rangle = \frac{|0\rangle^{\otimes 2} + |1\rangle^{\otimes 2}}{\sqrt{2}} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{2}$$

To generate a Bell state, we pick the observation to be the expectation values of Pauli matrices on each qubits $\{\langle\sigma_j^i\rangle \,|\, i \in \{0,1\}, j \in \{x,y,z\}\}$. The action set $\mathbb{G}$ is

$$\mathbb{G} = \bigcup_{i=1}^{n} \left\{ U_i\left(\pi/4\right), X_i, Y_i, Z_i, H_i, CNOT_{i,(i+1)(mod2)} \right\}, \tag{3}$$

where $n = 2$ (for two qubits), $U_i(\theta) = \left(\begin{smallmatrix} 1 & 0 \\ 0 & \exp(i\theta) \end{smallmatrix}\right)$ is the single qubit rotation about the Z-axis applied to qubit i, $X_i \equiv \sigma_x^i$ is the Pauli-X gate and likewise for Y_i and Z_i, H_i is the Hadamard gate, and $CNOT_{i,j}$ is the CNOT gate with the i-th qubit as control and j-th qubit as target, so we have 12 actions in total. A textbook example for creating a Bell state is shown in Fig. 2.

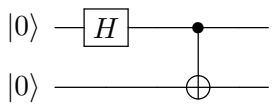


FIG. 2: **Quantum circuit for the Bell state.**

A GHZ state is a multi-qubit generalization of the Bell state, in which an equal superposition between the lowest and the highest energy states is created. For 3 qubits it is given by

$$|\mathrm{GHZ}\rangle = \frac{|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3}}{\sqrt{2}} = \frac{|000\rangle + |111\rangle}{\sqrt{2}} \tag{4}$$

To generate the 3-qubit GHZ state, we again use the expectation values of individual qubit's Pauli matrices, leading to 9 observables in total. For the actions, we pick the same single-qubit gates as in Eq. (3), and six $CNOT$ gates as two-qubit gates, so total we have 21 actions. In this fashion, for general n-qubit cases there will be $5n + n(n-1) = \Omega(n^2)$ actions, increasing only quadratically instead of exponentially in n. An example for creating a 3-qubit GHZ state is shown in Fig. 3.

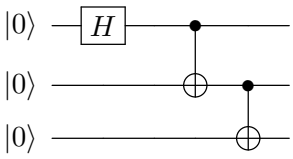


FIG. 3: **Quantum circuit for the GHZ state.**

IV. EXPERIMENTS AND RESULTS

A. Experimental Settings

1. *Optimizer*

We apply the gradient-descent method to optimize the RL policy. There are a wide variety of gradient-descent methods which are demonstrated highly successful [79–81]. In this work, we use the Adam [81] optimizer for training the RL agent in both the A2C and PPO cases. Adam is one of the gradient-descent methods which computes the *adaptive learning rates* for each parameter. In addition, Adam stores both the exponentially decaying average of gradients g_t and its square g_t^2,

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t \tag{5a}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2 \tag{5b}$$

where β_1 and β_2 are hyperparameters. We use $\beta_1 = 0.9$ and $\beta_2 = 0.999$ in this work. The m_t and v_t are adjusted according to the following formula to counteract the biases towards 0,

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{6a}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{6b}$$

The parameters θ_t in the RL model in the time step t are then updated according to the following formula,

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{7}$$

We use the Adam optimizer provided in the Python package PyTorch [82] to perform the optimization procedures.

2. *Quantum Noise in Quantum Simulator*

Here we introduce the error schemes we use in this study. We consider two forms of errors, *gate errors* and *measurement errors*. The gate error refers to the imperfection in any quantum operation we perform, whereas the measurement error refers to the error

that occurs during quantum measurement. For the gate error, we consider the depolarizing noise which replaces the state of any qubit with a random state of probability p_{gate}. For the measurement error, we consider a random flip between 0 and 1 with probability p_{meas} immediately before the actual measurement. We use the following noise configuration in the simulation software to test our deep RL agents:

- error rate (both p_{gate} and p_{meas}) = 0.001
- error rate (both p_{gate} and p_{meas}) = 0.005

For the simulation of quantum circuits in both noise-free and noisy environments, we use the software package Qiskit from IBM [83].

3. *Density Matrix of Quantum States*

The general form of a *density matrix* ρ of a quantum state under the basis $\{|\psi_i\rangle\}$ is,

$$\rho = \sum_j p_j \left|\psi_j\right\rangle \left\langle\psi_j\right| \tag{8}$$

where p_j represents the probability that the quantum system is in the pure state $|\psi_j\rangle$ such that $\sum_j p_j = 1$. For example, the density matrix of the Bell state considered in this study is $|\text{Bell}\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$. Its corresponding density matrix ρ is then given by

$$|\text{Bell}\rangle \langle\text{Bell}| = \frac{1}{2}\left(|00\rangle \langle 00| + |00\rangle \langle 11| + |11\rangle \langle 00| + |11\rangle \langle 11|\right) \tag{9}$$

The density matrix is used in calculating the state fidelity F as mentioned earlier.

4. *Quantum State Tomography*

Quantum state tomography is a procedure to reconstruct the density matrix associated with a quantum state from a set of complete measurements. Expanding the density matrix in the Pauli basis of N qubits,

$$\rho = \frac{1}{2^N} \sum_{i_1,\cdots,i_N=0}^{3} \rho_{i_i,\cdots,i_N} \sigma_{i_1} \otimes \cdots \otimes \sigma_{i_N}, \tag{10}$$

it can be seen that to fully determine ρ requires $4^N - 1$ measurement operations (minus one due to the conservation of probability, $\text{Tr}(\rho) = 1$). More generally, measurements with

$4^N - 1$ linearly independent projective operators can uniquely determine the density matrix, for which Eq. 10 is a special case with the projectors being the Pauli operators. As a result, the number of measurements grows exponentially in the qubit number N, posing a significant challenge in verifying multi-qubit quantum states in any experiments, and with a finite number of shots the expectation values for $\{\rho_{i_1,\cdots,i_N}\}$ can only be measured within certain accuracy. For the purpose of this work, however, we perform the quantum state tomography simulations using IBM's Qiskit software package [83].

5. *Customized OpenAI Gym Environment*

We build a customized OpenAI Gym [84] environment to facilitate the development and testing of this work. In this package, users can set the target quantum state, threshold of fidelity and the quantum computing backend (real device or simulator software). In addition, it is also possible to customize the noise pattern. We construct the testing environments with the following settings:

- **Observation:** As mentioned the agent receives Pauli-X, Y Z expectation values on each qubit. For general n-qubit systems, the number of observations will be $3 \times n$.
- **Action:** The RL agent is expected to select a quantum gate operating on the specific qubit as given in Eq. (3).
- **Reward:** For each step before successfully reaching the goal, the agent will receive a -0.01 reward to encourage the shortest path. When reaching the goal, the agent will receive a reward of value $(F - 0.01)$.

6. *Hyperparameters*

In this work, we employ the neural network models (shown in TableI) as our DRL agents: We consider two DRL algorithms in this work, their hyperparameters are:

- A2C: learning rate $\eta = 10^{-4}$, discount factor $\gamma = 0.99$
- PPO: learning rate $\eta = 0.002$, discount factor $\gamma = 0.99$, epsilon clip parameter $C = 0.2$, update epoch number $K = 4$

	Linear	Tanh	Linear	Tanh	Linear
Input	state dim		64		64
Output	64		64		action dim (actor) or 1 (critic)

TABLE I: The neural network for A2C and PPO. The structure is the same for both the actor and critic. The only difference is that in the actor, there is a softmax at the end of the network.

B. Results: Noise-Free Environments

1. 2-*qubit Bell state*

Here we consider the application of DRL to generate the 2-qubit Bell state from scratch under the noise-free environment. The result is in the Figure 5. We can observe that both A2C and PPO methods can successfully train the DRL agent to synthesize the Bell state. It is demonstrated that, with the same neural network architecture, the PPO method reaches optimal results faster and the result is more stable compared to the A2C method. In Figure 4 we provide the quantum circuit for Bell state generated by the DRL agent.

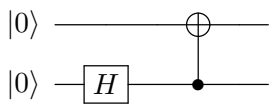
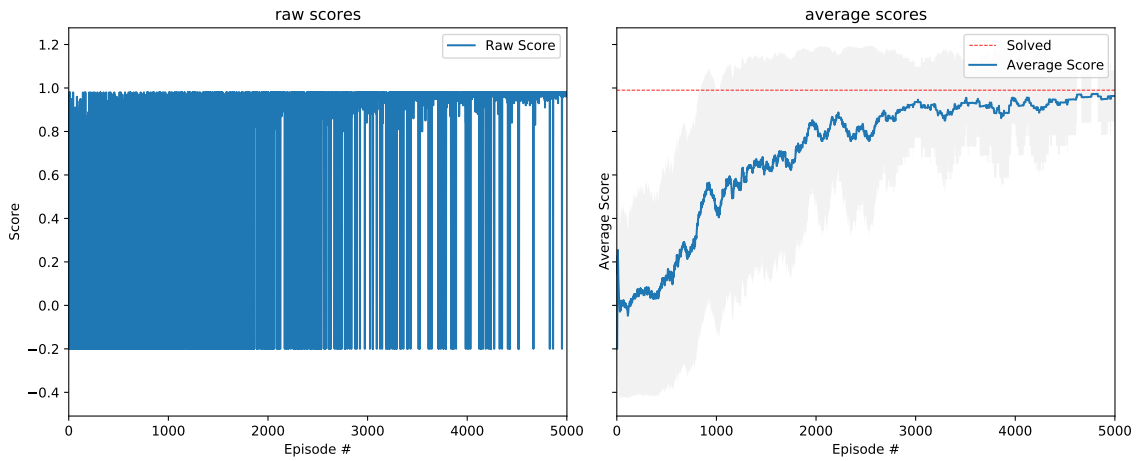


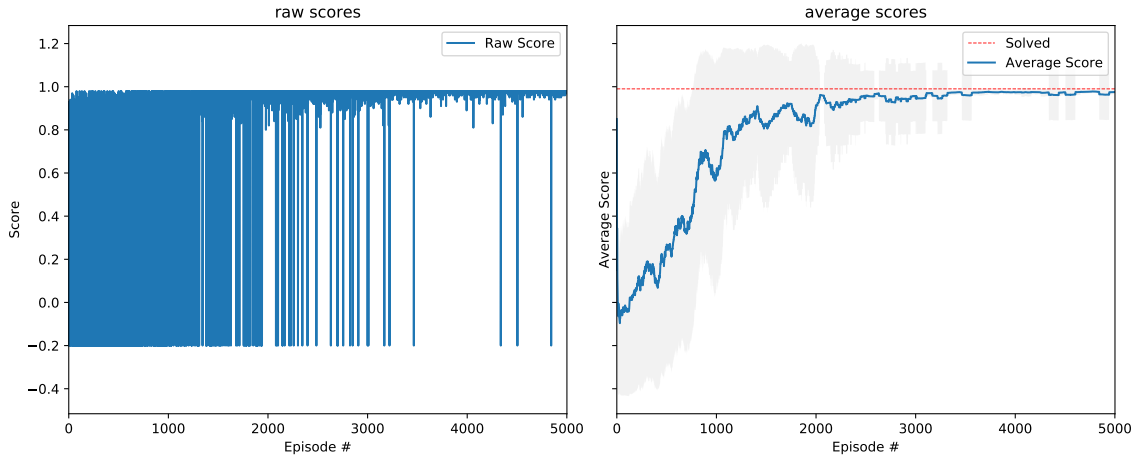
FIG. 4: **Quantum circuit for the Bell state generated by the DRL(PPO) agent.**

2. 3-*qubit GHZ state*

Here we consider the application of DRL to generate the 3-qubit GHZ state from scratch under the noise-free environment. The result is in the Figure 7. We can observe that both A2C and PPO methods can successfully train the DRL agent to synthesize the GHZ state. It is demonstrated that, with the same neural network architecture, the PPO method reaches optimal results faster and the result is more stable compared to the A2C method. Notably, the advantage of PPO over A2C is much more significant compared to the 2-qubit case. In Figure 6 we provide the quantum circuit for GHZ state generated by the DRL agent.



(a) **A2C for noise-free two-qubit system.**



(b) **PPO for noise-free two-qubit system.**

FIG. 5: **Deep Reinforcement Learning for Two-Qubit system.** In the synthesis of the Bell state with noise-free simulation environment, we set the total number of training episodes to be 5000. The left panels of the figure show the raw scores of the DRL agents. The gray area in the right panels of the figure represents the standard deviation of reward in each training episode. We observe that, given the same neural network architecture, PPO performs better than the A2C in terms of the convergence speed and the stability.

C. Results: Noisy Environments

In the previous section, we observe that the RL training based on PPO algorithm converges faster. In the noisy scenario, we only use the PPO and not the vanilla A2C since

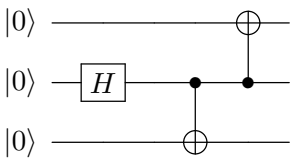
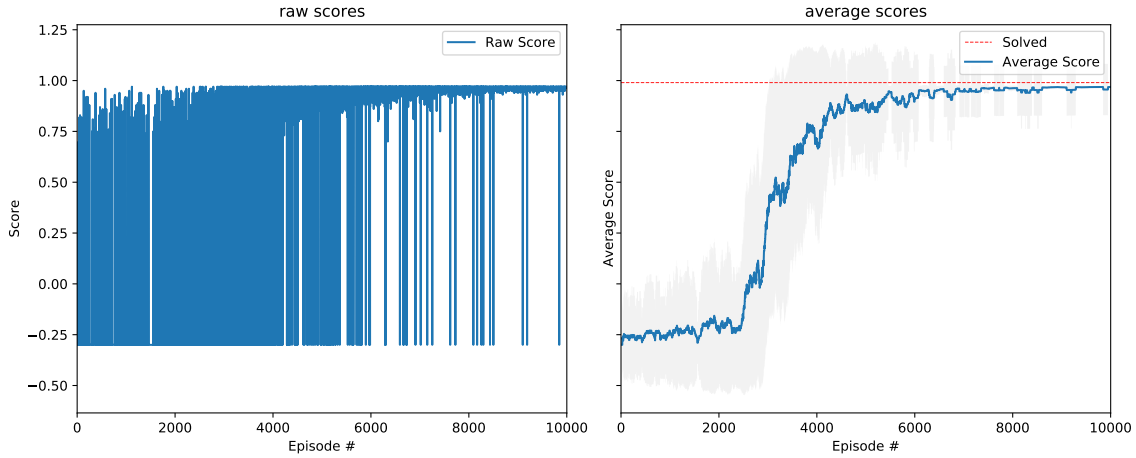
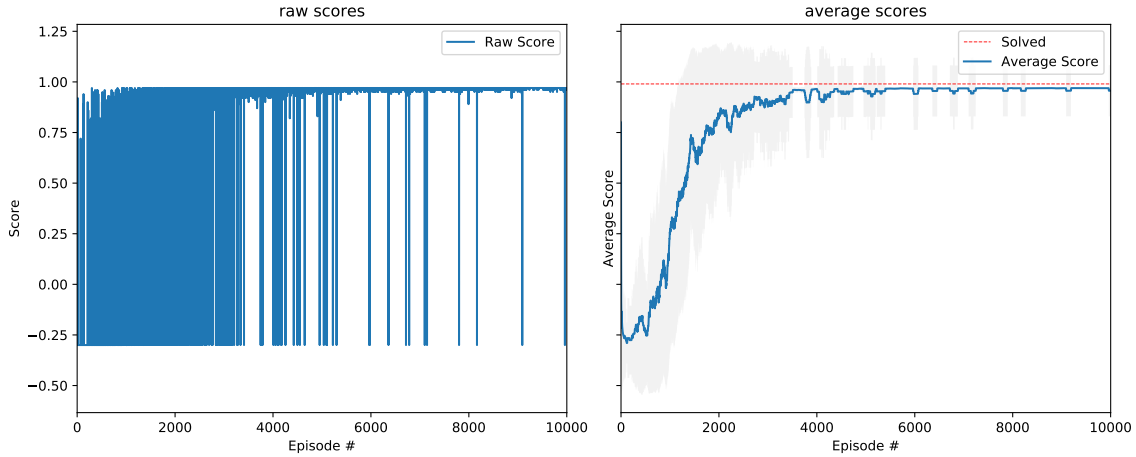


FIG. 6: **Quantum circuit for the GHZ state generated by the DRL(PPO) agent.**



(a) **A2C for noise-free three-qubit system.**



(b) **PPO for noise-free three-qubit system.**

FIG. 7: **Deep Reinforcement Learning for Three-Qubit system.** In the synthesis of the GHZ state with noise-free simulation environment, we set the total number of training episodes to be 10000. We observe that, given the same neural network architecture, PPO performs significantly better than the A2C in terms of the convergence speed and the stability. The result is consistent with the 2-qubit case.

that the noisy environment is considered harder than the noise-free one. Here we study the case of applying DRL agent to synthesize the 2-qubit Bell state under noisy environment. The first case we consider is with single-qubit error rate = 0.001 and the fidelity threshold = 0.95. Similar to the previous noise-free 2-qubit experiments, the agent gets a negative reward −0.01 at each step to encourage the shortest path. The maximum steps an agent can try in an episode is still 20. If the agent can reach fidelity beyond the threshold 0.95, then the agent will receive a positive reward (fidelity−0.01). Otherwise it will only receive reward = −0.01 when the episode ends. The result is shown in Figure 8a. Compared to the setting with the same single-qubit error rate and fidelity threshold = 0.99 (shown in Figure 8b), we observe that the one with fidelity threshold = 0.95 performs better, with much more stable score (smaller standard deviation).

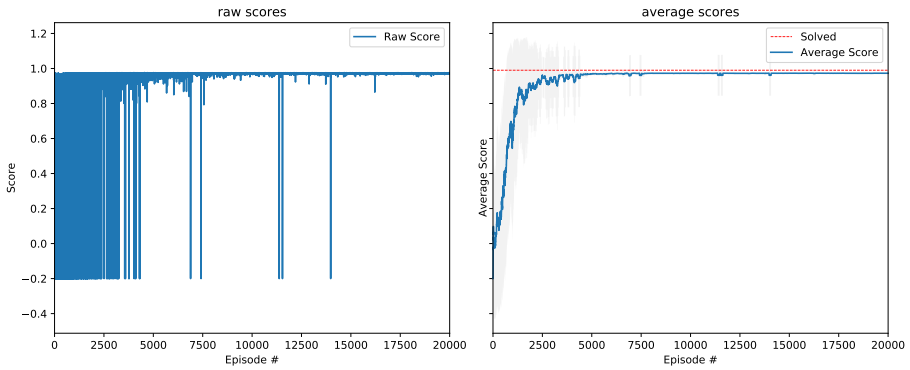
Here we need to point out that the fidelity threshold is to define whether the agent reaches a minimum goal. The agent is still trained to maximize the overall return, and the final fidelity which the agent can achieve is not limited to this threshold. A potential explanation is that, under the setting of fidelity threshold = 0.95, the agent would receive more guidance in the training phase. If the threshold is high, say 0.99, then the agent will stop after the maximum attempts and get no information about the fidelity in many of the training episodes. On the other hand, if the fidelity threshold is lower, the agent would receive positive reward in more training episode, which will in turn help the agent to adjust its model parameters.

Finally we compare the performance between single-qubit error rate = 0.001 and 0.005, both with the fidelity threshold = 0.95. We observe that both cases (shown in Figure 8a and Figure 8c) converge quickly. However, the final converged fidelity in the case with higher error rate is a bit lower.

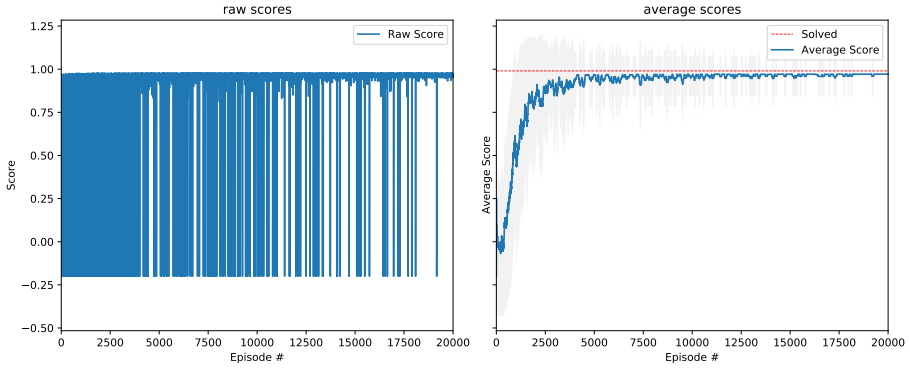
V. DISCUSSION

A. Relevant Works

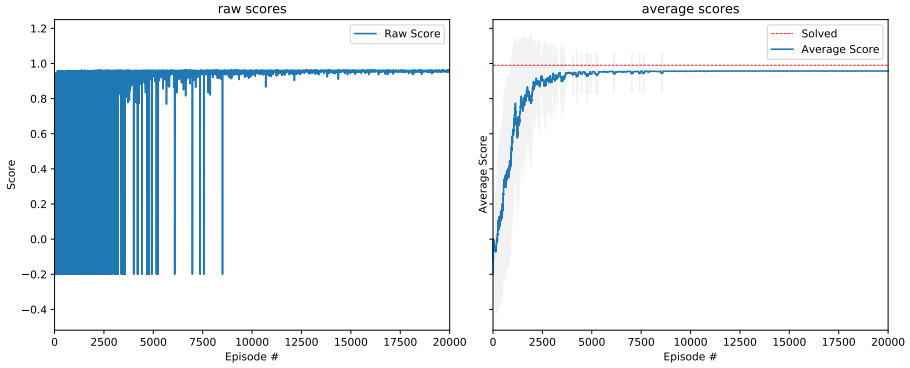
Deep reinforcement learning techniques have been applied in the investigation of quantum computing technologies. There are three main categories: quantum error correction, quantum optimal control and quantum architecture search. Early works on the quantum



(a) **PPO for noisy two-qubit system with single-qubit error rate** 0.001 **and fidelity threshold** 0.95**.**



(b) **PPO for noisy two-qubit system with single-qubit error rate** 0.001 **and fidelity threshold** 0.99**.**



(c) **PPO for noisy two-qubit system with single-qubit error rate** 0.005 **and fidelity threshold** 0.95**.**

FIG. 8: **Deep Reinforcement for Noisy Two-Qubit system.** Synthesis of the Bell state.

architecture search based on the heuristic search methods [85]. Recent works focus on the application of machine learning techniques [86–90]. Our methods differ from these. For example, in the work [87, 89], the authors proposed frameworks to optimize the existing quantum circuit, reducing the number of quantum gates. In our method, there is no existing quantum circuit to be optimized. The quantum circuit is to be generated from scratch. Our method is also different from [88] as we do not directly sample from a distribution of quantum circuits. Our method is also different from [86] as we are not to generate a batch of circuits in each time step nor using random search, instead, we would let the circuit grow incrementally. Recent work on optimizing parameterized quantum structures [91] indicates potential direction of extending AI/ML method to a more general setting (e.g. optimizing the quantum circuit architecture and its parameters simultaneously).

B. More Complex Problems

One may wonder how efficient this approach can be extended to large quantum circuits. In general, this should be very difficult and the complexity scales exponentially in N, the number of qubits. However, given a universal set of one and two-qubit quantum gates, in principle to approximate any quantum state (up to an error tolerance) it only requires a finite number of gates. So our approach is still valid for arbitrary large qubit size but computationally hard. There is no free lunch we can get. But our method is still useful to construct the general density matrix. It is interesting to see the difficulty in the training related to the complexity of the target density matrix. There are various ways to define the complexity of the density matrices [92, 93], the connection to which we leave as future work.

C. Noisy Environments

In this work we investigate the potential of applying deep reinforcement learning in the quantum gate search under simple noisy configurations. Our proposed software toolkit and framework is possible to be extended into other more complex noise models. Therefore, the results of this work is a good choice of testbed for a variety of future studies concerning different noise or error schemes. For example, recent works suggest that ML models can be used to learn the quantum circuit architecture under the noise effects [94]. We expect our

framework can be incorporated with such techniques. In addition, real quantum computers have different hardware topologies and these indeed have influences on the circuit design, we leave these topology-aware quantum architecture search as future work.

D. Real Quantum Computers

It is interesting to ask whether one can use real quantum computers to realize our algorithms. Our platform is based on Qiskit. Therefore, one can easily connect our module by connecting the real IBM quantum computer. Thousand of training episodes are required in our experiment, however, real quantum computers do not have so much resource to do. So it is interesting to investigate this problem when quantum computing resources are more accessible. We leave it as future work.

E. Other Quantum States

In this work, we consider the cases of two-qubit Bell state and three-qubit GHZ state for demonstration. However, the framework of the testing environment and RL agents are rather general. It is possible to investigate the quantum architecture search problem with other target quantum states and different noise configurations. In addition, it is also very convenient to test the performance of different reinforcement learning algorithms on quantum architecture search via the standard OpenAI Gym interface.

F. Extension of the Environment

In this work, we pre-defined a set of gates or operations which can be used to generate a desired quantum state. This is not a limitation of our framework. The testing environment itself can be extended or modified to fit the quantum computing devices that are of interest. For example, it is interesting to build customized training environments with available operations from a specific quantum hardware.

G. Circuit Optimization

One relevant question is that once we give a circuit C to produce a particular quantum state. Can one optimize this circuit getting a new circuit C' by reducing its depth and circuit complexity? The answer is yes, recently, there is a paper using reinforcement learning for given a circuit representation [95] and optimize the circuit depth. So [95] can be viewed as the next step or the useful tool to optimize our circuits. However, we are building quantum circuit from scratch. So our goal is different from theirs. One can easily see from the complexity point of view, solving our task efficiently does not imply solving their task efficiently and vice versa. One can indeed combine our work and their work to form a pipeline to solve the following: given a target state ψ and then try to find an efficient circuit C such that using C to create the target state. There is another related paper [96] using reinforcement learning and variational quantum circuit to find the ground state.

VI. CONCLUSION

In this work, we demonstrate the application of deep reinforcement learning (DRL) to automatically generate the quantum gates sequence from the density matrix only. Our results suggest that with the currently available deep reinforcement learning algorithms, it is possible to discover the near-optimal quantum gate sequence with very limited physics knowledge encoded into the RL agents. We also present the customized OpenAI Gym environment for the experiments, which is a valuable tool for exploring other related quantum computing problems.

ACKNOWLEDGMENTS

This work is supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics program under Award Number DE-SC-0012704 and the Brookhaven National Laboratory LDRD #20-024.

Appendix A: RL Algorithms

Here we provide the details of the RL algorithms used in this work.

Algorithm 1 Advantage Actor-Critic (A2C) for quantum architecture search

Define the number of total episode M

Define the maximum steps in a single episode S

for episode $= 1, 2, \ldots, M$ **do**

Reset the testing environment and initialise state s_1

Initialise trajectory buffer $\mathcal{T}$

Initialise the counter t

Initialise episode reward $R_E = 0$

for step $= 1, 2, \ldots, S$ **do**

Select the action a_t from the policy $\pi\left(a_t \mid s_t; \theta_\pi\right)$

Execute action a_t in emulator and observe reward r_t and next state s_{t+1}

Record the transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{T}$

Episode reward $R_E \leftarrow R_E + 1$

if reaching terminal state **or** reaching maximum steps M **then**

Calculate the value targets R_t for each state s_t in the trajectory buffer $\mathcal{T}$

Calculate the values $V(s_t, \theta_v)$ of each state s_t from the model $V\left(s_t, \theta_v\right)$

Calculate the value loss $L_{\text{value}} = \mathbb{E}_t \left\|V(s_t, \theta_v) - R_t\right\|^2$

Calculate the entropy term $H = \sum_t H_t = \sum_t \left[-\sum_j \pi\left(a_j \mid s_t; \theta_\pi\right) \log(\pi\left(a_j \mid s_t; \theta_\pi\right))\right]$

Calculate the advantage $A_t = R_t - V(s_t, \theta_v)$

Calculate the policy loss $L_{\text{policy}} = \mathbb{E}_t \left[-\log \pi\left(a_t \mid s_t; \theta_\pi\right) A_t\right]$

Total loss $L = L_{\text{value}} + L_{\text{policy}} - 0.001 \times H$

Update the agent policy parameters θ_π and θ_v with gradient descent on the loss L

end if

end for

end for

Algorithm 2 PPO for quantum architecture search

Define the number of total episode M

Define the maximum steps in a single episode S

Define the update timestep U

Define the update epoch number K

Define the epsilon clip C

Initialise trajectory buffer $\mathcal{T}$

Initialise timestep counter t

Initialize two sets of model parameters θ and θ_{old}

for episode $= 1, 2, \ldots, M$ **do**

Reset the testing environment and initialise state s_1

for step $= 1, 2, \ldots, S$ **do**

Update the timestep $t = t + 1$

Select the action a_t from the policy $\pi\left(a_t \mid s_t; \theta_{\text{old}}\right)$

Execute action a_t in emulator and observe reward r_t and next state s_{t+1}

Record the transition $\left(s_t, a_t, \log \pi\left(a_t \mid s_t; \theta_{\text{old}}\right), r_t\right)$ in $\mathcal{T}$

if $t = U$ **then**

Calculate the discounted rewards R_t for each state s_t in the trajectory buffer $\mathcal{T}$

for $k = 1, 2, \ldots, K$ **do**

Calculate the log probability $\log \pi\left(a_t \mid s_t; \theta\right)$, state values $V(s_t, \theta)$ and entropy H_t.

Calculate the ratio $q_t = \exp\left(\log \pi\left(a_t \mid s_t; \theta\right) - \log \pi\left(a_t \mid s_t; \theta_{\text{old}}\right)\right)$

Calculate the advantage $A_t = R_t - V(s_t, \theta)$

Calculate the $surr_1 = q_t \times A_t$

Calculate the $surr_2 = clip(q_t, 1 - C, 1 + C) \times A_t$

Calculate the loss $L = \mathbb{E}_t[-min(surr_1, surr_2) + 0.5 \left\|V(s_t, \theta) - R_t\right\|^2 - 0.01 H_t]$

Update the agent policy parameters θ with gradient descent on the loss L

end for

Update the θ_{old} to θ

Reset the trajectory buffer $\mathcal{T}$

Reset the timestep counter $t = 0$

end if

end for

end for

Appendix B: Code samples

Consider the case of noise-free two-qubit system, the OpenAI Gym environment setting is as follows,

```
import gym
import gym_twoqubit
target = np.asarray([0.70710678+0.j,0.        +0.j,0.        +0.j, 0.70710678+0.j])
env = gym.make('BasicTwoQubit-v0', target = target)
```

where we import relevant packages and set the target of the quantum state that we want the RL agent to learn. The target is used to initialize the gym environment. Consider the case of noisy two-qubit system, the OpenAI Gym environment setting is as follows,

```
import gym
import gym_twoqubit
from qiskit.providers.aer.noise import NoiseModel
from qiskit.providers.aer.noise.errors import pauli_error, depolarizing_error

def get_noise(p_meas,p_gate):
        error_meas = pauli_error([('X',p_meas), ('I', 1 - p_meas)])
        error_gate1 = depolarizing_error(p_gate, 1)
        error_gate2 = error_gate1.tensor(error_gate1)

        noise_model = NoiseModel()
        # measurement error is applied to measurements
        noise_model.add_all_qubit_quantum_error(error_meas, "measure")
        # single qubit gate error is applied to x gates
        noise_model.add_all_qubit_quantum_error(error_gate1, ["x"])
        # two qubit gate error is applied to cx gates
        noise_model.add_all_qubit_quantum_error(error_gate2, ["cx"])
```

```
18
19          return noise_model
20
21  def generate_backend_noise_info(backend):
22          device_backend = backend
23          coupling_map = device_backend.configuration().coupling_map
24          noise_model = NoiseModel.from_backend(device_backend)
25          basis_gates = noise_model.basis_gates
26
27          backend_noise_info = {
28          "noise_model": noise_model,
29          "coupling_map": coupling_map,
30          "basis_gates": basis_gates,
31          }
32
33          return backend_noise_info
34
35  noise_model = get_noise(0.001,0.001)
36  backend_noise_info = backend_noise_info = {
37  "noise_model": noise_model,
38  "coupling_map": None,
39  "basis_gates": None,
40  }
41  target = np.asarray([0.70710678+0.j,0.        +0.j,0.        +0.j, 0.70710678+0.j])
42  env = gym.make('NoisyTwoQubit-v0',
43  target = target,
44  backend_noise_info = backend_noise_info,
45  verbose = True,
46  fidelity_threshold = 0.99)
```

where we import relevant packages and set the target of the quantum state that we want

the RL agent to learn. In addition, we use functions from Qiskit package to define the noise model and the quantum simulation backend. The target and backend setting are then used to initialize the gym environment. We adopt the code for generating noise model from IBM qiskit textbook [97].

---

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] 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, pp. 529–533, 2 2015.

[3] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *arXiv preprint arXiv:1911.08265*, 2019.

[4] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," *arXiv preprint arXiv:2003.13350*, 2020.

[5] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *International conference on learning representations*, 2018.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[8] M. Bukov, A. G. Day, D. Sels, P. Weinberg, A. Polkovnikov, and P. Mehta, "Reinforcement learning in different phases of quantum control," *Physical Review X*, vol. 8, no. 3, p. 031086, 2018.

[9] T. Fösel, P. Tighineanu, T. Weiss, and F. Marquardt, "Reinforcement learning with neural networks for quantum feedback," *Physical Review X*, vol. 8, no. 3, p. 031084, 2018.

[10] M. Y. Niu, S. Boixo, V. N. Smelyanskiy, and H. Neven, "Universal quantum control through deep reinforcement learning," *npj Quantum Information*, vol. 5, no. 1, pp. 1–8, 2019.

[11] Z. An and D. Zhou, "Deep reinforcement learning for quantum gate control," *EPL (Europhysics Letters)*, vol. 126, no. 6, p. 60002, 2019.

[12] X.-M. Zhang, Z. Wei, R. Asad, X.-C. Yang, and X. Wang, "When does reinforcement learning stand out in quantum control? a comparative study on state preparation," *npj Quantum Information*, vol. 5, no. 1, pp. 1–7, 2019.

[13] P. Palittapongarnpim, P. Wittek, E. Zahedinejad, S. Vedaie, and B. C. Sanders, "Learning in quantum control: High-dimensional global optimization for noisy quantum dynamics," *Neurocomputing*, vol. 268, pp. 116–126, 2017.

[14] H. Xu, J. Li, L. Liu, Y. Wang, H. Yuan, and X. Wang, "Generalizable control for quantum parameter estimation through reinforcement learning," *npj Quantum Information*, vol. 5, no. 1, pp. 1–8, 2019.

[15] P. Andreasson, J. Johansson, S. Liljestrand, and M. Granath, "Quantum error correction for the toric code using deep reinforcement learning," *Quantum*, vol. 3, p. 183, 2019.

[16] D. Fitzek, M. Eliasson, A. F. Kockum, and M. Granath, "Deep q-learning decoder for depolarizing noise on the toric code," *Physical Review Research*, vol. 2, no. 2, p. 023230, 2020.

[17] A. Olsson and G. Lindeby, "Distributed training for deep reinforcement learning decoders on the toric code," 2020.

[18] H. P. Nautrup, N. Delfosse, V. Dunjko, H. J. Briegel, and N. Friis, "Optimizing quantum error correction codes with reinforcement learning," *Quantum*, vol. 3, p. 215, 2019.

[19] L. D. Colomer, M. Skotiniotis, and R. Muñoz-Tapia, "Reinforcement learning for optimal error correction of toric codes," *Physics Letters A*, vol. 384, no. 17, p. 126353, 2020.

[20] M. M. Wauters, E. Panizon, G. B. Mbeng, and G. E. Santoro, "Reinforcement learning assisted quantum optimization," *arXiv preprint arXiv:2004.12323*, 2020.

[21] J. Yao, M. Bukov, and L. Lin, "Policy gradient based quantum approximate optimization algorithm," *arXiv preprint arXiv:2002.01068*, 2020.

[22] G. Verdon, M. Broughton, J. R. McClean, K. J. Sung, R. Babbush, Z. Jiang, H. Neven, and M. Mohseni, "Learning to learn with quantum neural networks via classical neural networks," *arXiv preprint arXiv:1907.05415*, 2019.

[23] M. Wilson, S. Stromswold, F. Wudarski, S. Hadfield, N. M. Tubman, and E. Rieffel, "Opti-

mizing quantum heuristics with meta-learning," *arXiv preprint arXiv:1908.03185*, 2019.

[24] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[25] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.

[26] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," *arXiv preprint arXiv:1707.04873*, 2017.

[27] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

[28] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2423–2432, 2018.

[29] M. Schrimpf, S. Merity, J. Bradbury, and R. Socher, "A flexible approach to automated rnn architecture generation," *arXiv preprint arXiv:1712.07316*, 2017.

[30] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[31] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," *arXiv preprint arXiv:1806.02639*, 2018.

[32] T. Elsken, J. H. Metzen, F. Hutter, *et al.*, "Neural architecture search: A survey.," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[33] A. W. Harrow and A. Montanaro, "Quantum computational supremacy," *Nature*, vol. 549, no. 7671, pp. 203–209, 2017.

[34] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[35] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[36] L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack," *Physical review letters*, vol. 79, no. 2, p. 325, 1997.

[37] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik,

and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature communications*, vol. 5, p. 4213, 2014.

[38] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, "Quantum approximate optimization algorithm: performance, mechanism, and implementation on near-term devices," *arXiv preprint arXiv:1812.01041*, 2018.

[39] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[40] S. Y.-C. Chen, S. Yoo, and Y.-L. L. Fang, "Quantum long short-term memory," *arXiv preprint arXiv:2009.01783*, 2020.

[41] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, "Quantum circuit learning," *Physical Review A*, vol. 98, no. 3, p. 032309, 2018.

[42] O. Kyriienko, A. E. Paine, and V. E. Elfving, "Solving nonlinear differential equations with differentiable quantum circuits," *arXiv preprint arXiv:2011.10395*, 2020.

[43] M. Schuld, A. Bocharov, K. Svore, and N. Wiebe, "Circuit-centric quantum classifiers," *arXiv preprint arXiv:1804.00633*, 2018.

[44] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, no. 7747, pp. 209–212, 2019.

[45] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors," *arXiv preprint arXiv:1802.06002*, 2018.

[46] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, "Parameterized quantum circuits as machine learning models," *Quantum Science and Technology*, vol. 4, no. 4, p. 043001, 2019.

[47] A. Mari, T. R. Bromley, J. Izaac, M. Schuld, and N. Killoran, "Transfer learning in hybrid classical-quantum neural networks," *arXiv preprint arXiv:1912.08278*, 2019.

[48] Z. Abohashima, M. Elhosen, E. H. Houssein, and W. M. Mohamed, "Classification with quantum machine learning: A survey," *arXiv preprint arXiv:2006.12270*, 2020.

[49] P. Easom-McCaldin, A. Bouridane, A. Belatreche, and R. Jiang, "Towards building a facial identification system using quantum machine learning techniques," *arXiv preprint arXiv:2008.12616*, 2020.

[50] A. Sarma, R. Chatterjee, K. Gili, and T. Yu, "Quantum unsupervised and supervised learning on superconducting processors," *arXiv preprint arXiv:1909.04226*, 2019.

[51] S. A. Stein, B. Baheri, R. M. Tischio, Y. Chen, Y. Mao, Q. Guan, A. Li, and B. Fang, "A hybrid system for learning classical data in quantum states," *arXiv preprint arXiv:2012.00256*, 2020.

[52] S. Y.-C. Chen, C.-M. Huang, C.-W. Hsing, and Y.-J. Kao, "Hybrid quantum-classical classifier based on tensor network and variational quantum circuit," *arXiv preprint arXiv:2011.14651*, 2020.

[53] S. Y.-C. Chen, T.-C. Wei, C. Zhang, H. Yu, and S. Yoo, "Quantum convolutional neural networks for high energy physics data analysis," *arXiv preprint arXiv:2012.12177*, 2020.

[54] S. L. Wu, J. Chan, W. Guan, S. Sun, A. Wang, C. Zhou, M. Livny, F. Carminati, A. Di Meglio, A. C. Li, *et al.*, "Application of quantum machine learning using the quantum variational classifier method to high energy physics analysis at the lhc on ibm quantum computer simulator and hardware with 10 qubits," *arXiv preprint arXiv:2012.11560*, 2020.

[55] S. A. Stein, Y. Mao, B. Baheri, Q. Guan, A. Li, D. Chen, S. Xu, and C. Ding, "Quclassi: A hybrid deep neural network architecture based on quantum state fidelity," *arXiv preprint arXiv:2103.11307*, 2021.

[56] S. Y.-C. Chen, T.-C. Wei, C. Zhang, H. Yu, and S. Yoo, "Hybrid quantum-classical graph convolutional network," *arXiv preprint arXiv:2101.06189*, 2021.

[57] B. Jaderberg, L. W. Anderson, W. Xie, S. Albanie, M. Kiffner, and D. Jaksch, "Quantum self-supervised learning," *arXiv preprint arXiv:2103.14653*, 2021.

[58] P.-L. Dallaire-Demers and N. Killoran, "Quantum generative adversarial networks," *Physical Review A*, vol. 98, no. 1, p. 012324, 2018.

[59] S. A. Stein, B. Baheri, R. M. Tischio, Y. Mao, Q. Guan, A. Li, B. Fang, and S. Xu, "Qugan: A generative adversarial network through quantum states," *arXiv preprint arXiv:2010.09036*, 2020.

[60] C. Zoufal, A. Lucchi, and S. Woerner, "Quantum generative adversarial networks for learning and loading random distributions," *npj Quantum Information*, vol. 5, no. 1, pp. 1–9, 2019.

[61] H. Situ, Z. He, L. Li, and S. Zheng, "Quantum generative adversarial network for generating discrete data," *arXiv preprint arXiv:1807.01235*, 2018.

[62] K. Nakaji and N. Yamamoto, "Quantum semi-supervised generative adversarial network for enhanced data classification," *arXiv preprint arXiv:2010.13727*, 2020.

[63] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, "Variational

quantum circuits for deep reinforcement learning," *IEEE Access*, vol. 8, pp. 141007–141024, 2020.

[64] O. Lockwood and M. Si, "Reinforcement learning with quantum variational circuit," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, pp. 245–251, 2020.

[65] S. Jerbi, L. M. Trenkwalder, H. P. Nautrup, H. J. Briegel, and V. Dunjko, "Quantum enhancements for deep reinforcement learning in large spaces," *PRX Quantum*, vol. 2, no. 1, p. 010328, 2021.

[66] C.-C. CHEN, K. SHIBA, M. SOGABE, K. SAKAMOTO, and T. SOGABE, "Hybrid quantum-classical ulam-von neumann linear solver-based quantum dynamic programing algorithm," *Proceedings of the Annual Conference of JSAI*, vol. JSAI2020, pp. 2K6ES203–2K6ES203, 2020.

[67] S. Wu, S. Jin, D. Wen, and X. Wang, "Quantum reinforcement learning in continuous action space," *arXiv preprint arXiv:2012.10711*, 2020.

[68] A. Skolik, S. Jerbi, and V. Dunjko, "Quantum agents in the gym: a variational quantum algorithm for deep q-learning," *arXiv preprint arXiv:2103.15084*, 2021.

[69] S. Jerbi, C. Gyurik, S. Marshall, H. J. Briegel, and V. Dunjko, "Variational quantum policies for reinforcement learning," *arXiv preprint arXiv:2103.05577*, 2021.

[70] J. Bausch, "Recurrent quantum neural networks," *arXiv preprint arXiv:2006.14619*, 2020.

[71] Y. Takaki, K. Mitarai, M. Negoro, K. Fujii, and M. Kitagawa, "Learning temporal data with variational quantum recurrent neural network," *arXiv preprint arXiv:2012.11242*, 2020.

[72] C.-H. H. Yang, J. Qi, S. Y.-C. Chen, P.-Y. Chen, S. M. Siniscalchi, X. Ma, and C.-H. Lee, "Decentralizing feature extraction with quantum convolutional neural network for automatic speech recognition," *arXiv preprint arXiv:2010.13309*, 2020.

[73] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac, and N. Killoran, "Quantum embeddings for machine learning," *arXiv preprint arXiv:2001.03622*, 2020.

[74] N. A. Nghiem, S. Y.-C. Chen, and T.-C. Wei, "A unified classification framework with quantum metric learning," *arXiv preprint arXiv:2010.13186*, 2020.

[75] S. Y.-C. Chen and S. Yoo, "Federated quantum machine learning," *arXiv preprint arXiv:2103.12010*, 2021.

[76] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforce-

ment learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

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

[78] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[79] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[80] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning, 2012.

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

[82] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32* (H. Wallach and H. Larochelle and A. Beygelzimer and F. d'Alché-Buc and E. Fox and R. Garnett, ed.), pp. 8024–8035, Curran Associates, Inc., 2019.

[83] A. Cross, "The ibm q experience and qiskit open-source quantum computing software," in *APS Meeting Abstracts*, 2018.

[84] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[85] C. P. Williams and A. G. Gray, "Automated design of quantum circuits," in *NASA International Conference on Quantum Computing and Quantum Communications*, pp. 113–125, Springer, 1998.

[86] M. Pirhooshyaran and T. Terlaky, "Quantum circuit design search," *arXiv preprint arXiv:2012.04046*, 2020.

[87] Y. Du, T. Huang, S. You, M.-H. Hsieh, and D. Tao, "Quantum circuit architecture search: error mitigation and trainability enhancement for variational quantum solvers," *arXiv preprint arXiv:2010.10217*, 2020.

[88] S.-X. Zhang, C.-Y. Hsieh, S. Zhang, and H. Yao, "Differentiable quantum architecture search," *arXiv preprint arXiv:2010.08561*, 2020.

[89] X.-C. Wu, M. G. Davis, F. T. Chong, and C. Iancu, "Optimizing noisy-intermediate scale

quantum circuits: A block-based synthesis," *arXiv preprint arXiv:2012.09835*, 2020.

[90] S.-X. Zhang, C.-Y. Hsieh, S. Zhang, and H. Yao, "Neural predictor based quantum architecture search," *arXiv preprint arXiv:2103.06524*, 2021.

[91] M. Ostaszewski, E. Grant, and M. Benedetti, "Structure optimization for parameterized quantum circuits," *Quantum*, vol. 5, p. 391, 2021.

[92] M. B. Plenio, "Logarithmic negativity: a full entanglement monotone that is not convex," *Physical review letters*, vol. 95, no. 9, p. 090503, 2005.

[93] G. Vidal and R. F. Werner, "Computable measure of entanglement," *Physical Review A*, vol. 65, no. 3, p. 032314, 2002.

[94] L. Cincio, K. Rudinger, M. Sarovar, and P. J. Coles, "Machine learning of noise-resilient quantum circuits," *PRX Quantum*, vol. 2, no. 1, p. 010324, 2021.

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

[96] M. Ostaszewski, L. M. Trenkwalder, W. Masarczyk, E. Scerri, and V. Dunjko, "Reinforcement learning for optimization of variational quantum circuit architectures," *arXiv preprint arXiv:2103.16089*, 2021.

[97] "Learn Quantum Computation using Qiskit." `https://qiskit.org/textbook/ch-quantum-hardware/error-correction-repetition-code.html`, 2020.